ACCESS CONTROL BY BOOLEAN EXPRESSION EVALUATION

Donald V. Miller Robert W. Baldwin

Tandem Computers, Inc.
10600 N. Tantau - Loc 247
Cupertino, CA, 95014-0788

Abstract

An access control mechanism based on boolean expression evaluation, BEE, is presented. This mechanism allows the implementation of customer-specified rather than vendor-specified security policies. The mechanism enables one to easily implement such conventional mechanisms as access control lists, named access control lists, user groups, user attributes, user capability lists, and user roles. Additional access restrictions based on time, day, date, location, load average, or any customer-supplied function can be incorporated into access decisions. This mechanism can directly express Clark-Wilson triples and it can easily implement policies that are difficult or impossible to implement using the Bell-LaPadula model.

Conventional discretionary access control mechanisms have several problems. Traditionally, granting, revoking, and denying access of a user to an object is based on either Capability Lists (C-Lists) as shown in Figure 1A, or Access Control Lists (ACLs) as shown in Figure 1B. These mechanisms are cumbersome to administer when there is a large number of subjects and objects. It is hard to maintain the security configuration when the nature of subjects and objects change (e.g., people change jobs or data is reclassified). Both mechanisms have difficulty representing security policies that are based on attributes like time, day of week, location, program name, roles, or system load average. These mechanisms tend to restrict one's thinking about access control policies to just the abilities of the mechanisms rather than to the desired system behavior. Customers are forced to suffer with a security system that is either to restrictive or too loose because the system cannot represent their true security policy.

The mechanism discussed in this paper, BEE, is presented schematically in Figure 1C. The figure makes it clear that BEE could implement either ACLs or C-Lists, because it can make access decisions based on information from either the object or the subject. However, the key feature of BEE is the kind of information that it associates with subjects and objects. If capabilities are stored with subjects and lists of individuals are stored with objects, then BEE will have all the problems of ACLs and C-Lists.

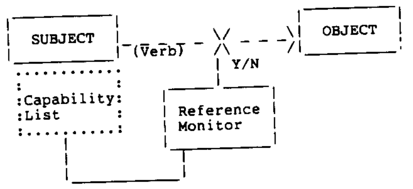

Fig. 1A - Capability Lists

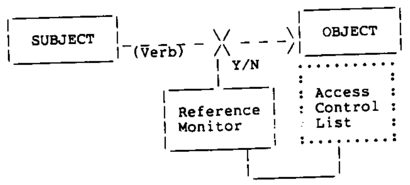

Fig. 1B - Access Control Lists

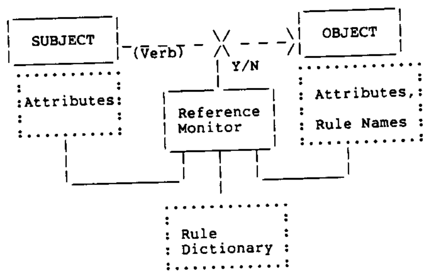

Fig. 1C - General Access Control

To avoid the problems of traditional mechanisms, BEE stores descriptive attributes of the subjects and objects. The idea is to explicitly represent the reason that some subject is allowed to perform a given action on a given object. ACF2 [1] has a similar approach with the use of its 24-character user ID string. Each character position can represent some characteristic of the user. BEE generalizes this notion by allowing attributes to be associated with both the subject and the object, and it allows the naming of access control rules, which ACF2 does not.

BEE can be thought of a policy-based access control rather than identity-based access control or mandatory access control (in this sense MAC refers to a vendor-chosen rule that governs all access decisions). BEE is a form of discretion access control that is based on attributes that may or may not be changeable at the discretionary of the users. For example, read access to an object may be allowed because the subject is working on the same project as the object's creator. In this example, the subject and object both have attributes that indicate which project they are associated with, and the access rule, which might be called "same-project-read-access", compares these attributes. Other rules would govern the set of projects that a subject can be part of. In Figure 1C, the subject has a project attribute (perhaps more than one) and the object has a project attribute (which might have been derived from the primary project of the subject that created this object). The object also names the rule that governs read access to the object. In this case the rule is a simple boolean expression that compares the project attributes. Access is allowed if the attributes are equal.

In many ways BEE can thought of as a generalization of lattice-based security models like BLP [2]. In that model subjects and objects are labelled and access decisions are based on a simple boolean expression comparing levels and category sets. A key feature of the BLP model is that there are only two access rules (read and write), which are immutable and apply to all access decisions. In contrast, BEE allows a customer to define his own access rules and some users will have the discretion to choose the rule that will govern a particular object or set of objects. The set of attributes that BEE examines can also be defined by the customer. The attributes are not limited to levels and categories. They can be arbitrary name-value pairs. However, like the BLP model, BEE assumes that ordinary users will not be able to create new attribute names or access rules, and that they will have only limited abilities to set the values of their own attributes.

BEE simplifies the administration of the security system by clearly modularizing knowledge about subjects, objects, and access rules. When a person is promoted, the set of jobs he performs changes and the corresponding access controls need to change. When someone is promoted in an ACL based system, all ACLs must be reviewed and/or updated to make sure they reflect the new jobs performed by that person. In a C-List based system, that subject's entire C-List (which is likely to be quite large) must be reviewed. With BEE, however, only a few attributes of the subject must be changed to reflect his new position. The access rules on objects are already expressed in terms of these attributes, so there is no need to examine all the objects in the system. Similarly, when an object is reclassified (e.g., a draft quarterly report becomes an official quarterly report) only the attributes and rules that govern that object need to change. In an ACL-based system, reclassifying an object would require creating a new list of individuals and groups of individuals that should have access to the official report. Creating such a list is such an error prone process that true ACL-based systems (e.g., VMS and Multics) include a feature that copies ACLs from an existing object (perhaps a template) into a new object. Such systems run into problems when the template ACL changes. It can be quite hard to find all the objects that have copies of the template ACLs. BEE's ability to use both subject and object attributes and to refer to rules indirectly by name can be used to create a modularized security system that is easy to maintain and extend.

There are other aspects of access control where there is little standardization or consensus, so vendors make haphazard choices that the customers must live with. Several examples concern the rights and responsibilities of an object's creator. Should the disk space occupied by the object be charged to the creator, or can it be charged to a designated owner? Should the creator be the only subject who is allowed to purge (delete) an object, or should that privilege be shared with the super-user and the creator of the directory in which the object is named? Customers with different security policies could have different answers to these questions. Unfortunately, vendors often pick specific policies and compile them into the security system. A general access control mechanism should allow the customer to pick the policies that govern his system.

Many of the foregoing issues were pointed out by the NCSC [3,4] and discussed extensively by T.F. Lunt [5]. To date, a single general purpose access control mechanism that can enforce customer supplied policies has not been presented. Customers must be content with a "semantic gap" between the concepts in their security policies and the features provided by the access control mechanism. A generalized mechanism helps customers bridge that semantic gap. A direct result of a reduced semantic gap will be higher assurance that the desired policy is the one that has been expressed to the security system.

In this paper we first discuss the requirements of a general purpose access control mechanism. Then we present a general mechanism that can enforce a wide range of site-specified access control policies in a single general-purpose system. We show how this mechanism can implement traditional ACLs and C-Lists and more recent ideas like the Clark-Wilson model. Although the mechanism was not designed for the BLP model, a brief discussion is included about how this approach can be extended to cover the mandatory features of BLP.

Generalized Access Control

Because so many alternative formulations of discretionary access control policies are possible, it is unnecessarily limiting to have to "wire" a specific policy into a system. It is appealing to envision a system that could enforce any of a number of access control policies, where each installation would choose the particular policy to be enforced.

Discretionary access mechanisms are in the most reduced sense binary decisions; a subject is either allowed or is not allowed to perform some action on some object. This concept can be stated as: the sentence "Subject may Verb Object" is either true or false. A general discretionary access control mechanism, therefore, should have subjects, verbs, and objects as inputs to a boolean expression evaluator. To be completely general, subjects, verbs, and objects should be defined to the system along with subject attributes, object attributes, and verb attributes, allowing easy grouping of subjects and objects.

To ease the task of administering a system, a set of rules should also be defined to the system. It is our contention that a small number (say less than 100) of these rules should be capable of expressing the security policy of a production system. The rules are parameterized by the values of the attributes of the subject and object involved in the access, so it is possible to write general rules that apply to a wide range of situations. For example, one rule might be true when every the subject is an fulltime employee (specifically not a contractor or consultant). This rule could control read access to the online corporate newsletter and it could also control execute access to the office supply ordering system (i.e., consultants cannot order diskettes). A carefully defined set of rules will greatly simplifying the task of administering the security of a system.

Expressing the security policy in a few boolean expressions provides greater assurance that the true policy is being enforced by the system. It is much easier to examine a few rules rather than to examine perhaps millions of access control lists, each containing perhaps hundreds of individual users. However, if the rule definition language is not powerful enough, there will be an explosion in the number of rules, and assurance will be lost.

Thus, the elements of a general purpose access control mechanism are those depicted in figure 2.

```
Object Name \  |________________|
Object Type  |--| Lookup Rule(s) |--+
Verb   Name /  |________________|  |
                                    |
Subject Attributes     \            |
 - user name/ID         |           |
 - user role            |           |
 - active groups        |--+        |
 - user location        |  |        |
 - program name         |  |   _____V____
 - process name        /   |  |          |
                           +-->|          |
Object Attributes      \       | Evaluate |
 - object name          |----->|          |
 - object attributes   /       | Rule     |--> Yes/No
                           +-->|          |
Global Attributes      \   |   |__________|
 - time                 |  |
 - date                 |--+
 - named lists          |
 - named constants     /
```

Figure 2
Generalized Access Control Mechanism

ARF - a Demonstration Program

A discretionary access control mechanism based on the evaluation of boolean expressions has been implemented in a demonstration program which runs in a window environment on a personal computer. This program, called ARF (Access Restriction Facility) is being used to test the feasibility of the concepts presented here. ARF's user interface consists of six windows: a user window, a verb window, an object window, a definition window, and a rule window, and a hierarchical window. Given a selected Subject, Verb, and Object, the program determines whether of not access is allowed.

ARF can demonstrate generalized access control to objects such as files, processes, devices, tables, rows, columns, fields, commands, etc. Access can allowed or denied on the basis of any user name, role, group, location, hair color, temperature, phase of the moon -- indeed anything that can be expressed as a character string.

We now proceed to describe the windows of the demonstration program and then to demonstrate how various access control policies can be expressed in terms of the boolean expressions used by ARF.

ARF Windows

Below we illustrate the various ARF window types.

Subject Window:

```
 ______________________________________________________________
|                                                              |
|S.GP.NAME    LVL ROLE        GROUPS    PROGRAM                |
|=========    === ==========  ========  ========               |
|A.HW.NABER    3  PROGRAMMER  G1        EDITOR                 |
|A.FI.CHOU     4  MANAGER     G2 B D    ADMIN                  |
|B.SW.CROOK    5  PROGRAMMER  B A C     MAIL                   |
|A.SA.LIDDY    6  SEC_ADMIN   B AA      ARFCOM                 |
|B.AR.NASH     2  ACCT_REP    BANK1     EDITOR                 |
|A.SW.WILSON   3  DESIGNER    C         SQL                    |
|C.QA.JULIAN   7  QA TESTER   D         SQL                    |
|A.SA.FELIX    1  SALESMAN    S         MAIL                   |
|A.GR.WHITE    1  ARTIST      D         EDITOR                 |
|______________________________________________________________|
```

Fig. 3 - Example of the SUBJECT window

The subject window is used to display and update subject names and subject attributes. In this example there is a one-to-one correspondence between subjects (processes) and individual users (people). Columns in this window can be used to specify the user's group(s), role(s), or general attributes.

The number and names of fields appearing in the subject window and the type of information they contain are configurable by the system security officer. No field in the subject record has any intrinsic meaning. In the example of Fig. 3, "LVL" can refer to a level of trust, or perhaps the number of freckles on the user's arm. Fields have meaning only when coupled by the Evaluator to information in the other windows. Hence, there can be any number of "attributes" associated with a particular subject such as system, group, name, level, role, group membership, and active program, as depicted in Fig. 3, but other attributes are possible such as hair color, location, etc. In other words, the data appearing in the fields of the subject records are not assumed to have any meaning; they are just strings.

ARF itself can be used to enforce a strict separation of duties. Different individuals or groups could be authorized to insert/update information into the Subject database based on other ARF rules stored in the Rule database.

Verb Window:

```
 ______________________________________________
|                                              |
|NAME:DEFAULT RULE                             |
|==============================================|
|C: 0                                          |
|R: 0                                          |
|W: 0                                          |
|E: 0                                          |
|P: 0                                          |
|SING: 1                                       |
|PLAY: 1                                       |
|ADD_GROUP: 0                                  |
|______________________________________________|
```

Fig. 4 - Example of the VERB window

The verb window is used to display and update verb names and their associated default rules. Default rules are used when an explicit rule for a particular object does not exist. The default rule itself may reference other rules as defined in the rule window. These default rules serve to provide higher level of access control policies. For instance, setting the default to 0 means that the system is a "closed" system - access is denied unless explicitly allowed, whereas setting the default to 1 means that the system is an "open" system, allowing access unless explicitly denied. In the example of Fig. 4, Read (R), Write (W), Execute (E), and Purge (P) are defined with a closed system rule, so that explicit authorization is required in order to use them.

This definition of verbs is a distinguishing feature of ARF. There are no pre-defined verbs as there are in access control mechanisms currently provided by vendors. This feature is extremely valuable when one considers that a system needs to understand verbs such as Insert, Update, Drop, etc. from SQL; system commands such as Backup; subsystem commands such as Add Line, Down, Alter Priority, etc.; or verbs for new products.

Object Window:

```
NAME                           RULES
============== ==================================
$SYS.#MAIL    : R: 'M' IN SUBJ.GROUPS ; W: 1
$ABC.DON.A    : W: G AND (WEEKEND OR TEMP>71)
$ABC.SQL.DATA : R: SUBJ.PROGRAM='SQL'
$XYZ.SVB.F1   : W: MEMBER_OF_A AND SUBJ.NAME <> 'NABER'
$XYZ.SVB.F2   : W: SUBJ.NAME IN ACL_1
MOVIES        : E: DAY IN WEEKEND OR NOT WRK_HRS
BUSINESS_LUNCH: E: TEMP>74 OR SUBJ.ROLE='SALESMAN'
COFFEE_BREAK  : E: BREAK_TIME OR SUBJ.ROLE='PROGRAMMER'
OPERA         : SING: 'OPERA' IN SUBJ.GROUPS
ROCK          : SING: SUBJ.ROLE = 'ROCKER'
BANK          : ADD_GROUP: SUBJ.GROUPS V BANK_LIST = ''
BANK1_DATA    : R: ACCT_REP AND BANK1_REP
```

Fig. 5 - Example of the OBJECT window

The object window is used to display and update object names and rules for defined verbs. In Fig. 5, information preceding the first colon is either the name of the object or an attribute of the object, the semantics being defined via the definition window. Everything after the colon is considered to be a rule list. Rules for specific verbs are designated as:

```
<VerbName>: <Rule>
```

Rules for the verbs must be separated by semicolons as shown in the line for $SYS.#MAIL.

In this window, rule names (defined in the rule window) are prefixed by a colon and verb names (defined in the verb window) are suffixed by a colon.

Rules can refer to other rules. For example, the rule for $XYZ.SVB.F1 is shown below.

```
W: MEMBER_OF_A AND SUBJ.NAME<>'NABER';
```

The verb W (WRITE) is defined in the verb window, and the rule MEMBER_OF_A is defined in the rule window. This means that the verb W can be invoked for object $XYZ.SVB.F1 only if the rule MEMBER_OF_A satisfied. The other part of the rule, " AND SUBJ.NAME <> 'NABER'", means that even if the rule MEMBER_OF_A is satisfied, write access by NABER is explicitly denied.

Another form of indirection is the named ACL. The write rule for object "$XYZ.SVB.F2" indicates that the subject must be on the access control list ACL_1, which is specified in the definition window.

Definition Window:

```
NAME      VALUE
======== ==========
SYSTEM:  [1,1]
GROUP:   [3,2]
NAME:    [6,6]
LVL:     [13,3]
ROLE:    [17,10]
GROUPS:  [28,8]
PROGRAM: [38,8]
VOL:     [1,5]
DATE:    EXTERN
TIME:    EXTERN
HOUR:    [1,2]
MIN:     [4,2]
TEMP:    EXTERN
DAY:     EXTERN
WEEKEND: SAT SUN
ACL_1:   CHOU WILSON CROOK
ACL_2:   WILSON JULIAN
BANK_LIST: BANK1 BANK2 BANK3
TRUSTED: EXTERN
```

Fig. 6 - Example of the DEFINITION window

The definition window is used to display and update field definitions, external function declarations, and strings.

Definitions are themselves character strings, which may be one of the following:

- Field designator:

  <DefName>: [<StartPos>,<NumChars>]

- External function declaration:

  <DefName>: EXTERN

  For external functions, the caller must provide a function named "STD_FCN" with a string argument of <name> which returns a general character string.

- Lists:

  <DefName>: <List>

In the example of Fig. 6, the definition "VOL: [1,5]", when applied to a string, extracts the first five characters of the string (starting at character 1, for five characters). Thus, when applied to an object name in the examples shown in the object window, .VOL extracts the volume name for the object. For example, the volume name for object "$VXYZ FILE_2" is $VXYZ.

In the example of Fig. 6, DATE, TIME, TEMP, and DAY are external functions. Commonly used functions may in practice be implemented as intrinsic functions which need not be explicitly defined.

The lists ACL_1 and ACL_2 are used to define access control lists.

Rule Window:

```
NAME            VALUE
============    ==============================
ANYBODY:        1 ! UNIVERSAL ACCESS
NOBODY:         0
WEEKDAY:        NOT DAY IN WEEKEND
WRK_HRS:        TIME.HOUR >= 8 AND TIME.HOUR<=17
RULE_85:        MEMBER_OF_A
                 AND (PROGRAMMERS OR MANAGERS)
BREAK_TIME:     TIME.MIN >= 28 AND TIME.MIN <= 40
PROGRAMMERS:    SUBJ.ROLE = 'PROGRAMMER'
MANAGERS:       SUBJ.ROLE = 'MANAGER'
MEMBER_OF_A:    'A' IN SUBJ.GROUPS
ACCT_REP:       SUBJ.ROLE = 'ACCT_REP'
BANK1_REP:      SUBJ.GROUPS V BANK_LIST = 'BANK1'
CLARK_WILSON:   SUBJ.ROLE = 'ACCOUNTANT'
                   AND PROGRAM = 'ACCTS_PAY'
```

Fig. 7 - Example of the RULE window

The rule window is used to display and update named boolean expressions to be used by the expression evaluator. Rules defined in the rule window override the default rules defined in the verb window. Everything after an exclamation mark (!) in a rule is a comment.

Rules are boolean expressions which must yield a true or false result. Rules may reference other rules, definitions, or subject attributes (fields).

Hierarchical Window:

ARF includes a window that is used to define who can create and modify users, rules, definitions, and objects. The details are beyond the scope of this paper, but the key idea is to treat rules and definitions as first class objects that have attributes and a collection of verbs that are performed on them. As first class objects, it is possible to use ARF to control them. The possibility of infinite regression - who controls the rules on rules, etc. - is avoided by having some objects compiled into the system. Those objects are the source of all authority to modify or create other objects.

Emulation of Various Mechanisms

The following are examples of using ARF to express various access control mechanisms.

Access Control Lists

Access control lists can be specified using ARF's object window as follows:

```
W: SUBJ.NAME IN 'USER_A USER_B USER_D'
```

This rule grants write access if the user's name is in the set (USER_A USER_B USER_D).

Named Access Control Lists

Named access control lists can be specified using ARF's object window as follows:

```
W: SUBJ.NAME IN ACL_X
```

where ACL_X is defined as follows (using the definition window):

```
ACL_X: USER_A USER_B USER_D
```

Exclusionary Access Control Lists

Exclusionary access control lists can be expressed using ARF's object window as follows.

```
W: NOT (SUBJ.NAME IN LIST1
   OR SUBJ.DEPT IN LIST1)
```

This means that the users whose usernames and departments are specified in LIST1 are explicitly denied write authorization for the object, all other users are allowed access. Notice that a negative clause like the one above can be ANDed with a positive clause to express a policy that allow access to all people in some set, except for certain individuals. The resulting policy is unambiguously represented by the boolean expression.

Capability Lists

Capability lists can be specified using ARF's user window by defining one of the user attributes (fields) in that window to be a list of objects for which the user has authorization.

Suppose that a field called _E_LIST exists for each user. With a default E (Execute) rule of OBJECT.NAME IN SUBJ._E_LIST, we can control which programs can be executed without having specific rules on the objects themselves, although such rules would not be prohibited.

For example, USER_A's, _E_LIST could contain

```
OBJ_X OBJ_Y OBJ_Z
```

allowing this USER_A to execute OBJ_X, OBJ_Y, or OBJ_Z.

Access by Time

```
8 <= HOUR AND HOUR <= 17
```

where HOUR is defined in the definition window to be EXTERN (an external function). To simplify the administration of the rules and to ensure a consistent definition of working hours, a named rule could be created like:

```
WRK_HRS: 8 <= HOUR AND HOUR <= 17
```

Chinese Wall Security Policy

The Chinese Wall [6] is an interesting policy since it can only with difficulty be implemented by the Bell-LaPadula model. The policy combines discretionary with mandatory controls and is required in the operation of many financial services organizations. It is, as significant to the financial world as the Bell-LaPadula model is to the military.

The policy can be briefly explained by means of the following example:

> Suppose a security company deals with three banks. A representative of the company can freely choose one of the three banks in order to offer advice to them. However, once a person chooses a bank, that person is denied access to information about either of the other two banks.

Of course this policy could be enforced via categories or compartments, each manufacturer occupying one compartment, but some agent external to the Bell-LaPadula model would be required to ensure that one and only one compartment existed for each representative of the institution.

Because of this separation of policy from the model, we feel that there would be little assurance that the policy is being enforced. All that the Bell-LaPadula model will do is to ensure that the user attempting access is authorized for that compartment, not that the other two compartments are not authorized.

Implementation of this policy via ARF is rather simple. The example ARF windows illustrate how to do this.

The object BANK has the rule

```
   CHOOSE_BANK: SUBJ.GROUPS V BANK_LIST = ''

   (where 'V' means 'intersection',
    i.e. a set theoretic 'and')
```

allowing the user to choose a bank only if they have not chosen one already.

The object BANK1_DATA has the rule

```
   BANK1_DATA:  R: ACCT_REP AND BANK1_REP
```

The rules referenced by this rule, namely

```
   ACCT_REP:   SUBJ.ROLE = 'ACCT_REP'
   BANK1_REP:  SUBJ.GROUPS V BANK_LIST = 'BANK1'
```

allow read access to users who have chosen BANK1 and only BANK1 and who are account reps.

Bell-LaPadula Model

ARF was not designed to handle the BLP model, but it can be extended to incorporate that model. The primary extension is the ability to specify a collection of rules that are ANDed with any more specific access rule. The default rules serve the opposite purpose of applying only when there is not a more specific rule. All the verbs would have to be divided into read-class and write-class operations and the appropriate rule would apply to each class.

The next extension would be to define attributes for objects and subjects that would serve the purpose of security levels and categories. This would handle the labelling requirement for existing objects. Newly created objects would have to be labeled by the attributes of the subject that created them (in order to prevent possible information flows between levels or classes). The labelling could be static in the sense that each subject would operate in exactly one access class (level plus categories), or dynamic in which case the process and file system mechanism would have to maintain floating access class information that would be derived from the attributes of objects that have been accessed by the subject.

This design is not being proposed as a realistic implementation of the BLP model, but it is interesting to view BLP as just one example of an attribute based security policy.

Clark-Wilson Triples

Clark-Wilson triples [9] are rather trivial using ARF.

The object FILE1 could be protected by the rule

```
   UPDATE: SUBJ.ROLE = 'ACCOUNTANT'
     AND PROGRAM = '$SYSTEM.SYSTEM.ACCTS_PAY'
```

allowing Update access only if the user is an accountant and the program being executed is $SYSTEM.SYSTEM.ACCTS_PAY.

Securing access properly on $SYSTEM.SYSTEM would ensure that the program ACCTS_PAY is authentic.

Administration

An issue not addressed by the Orange Book but of great concern to our customers is that of administration. Since administration of a set of subjects and users in a computer system becomes more difficult as their number increases, a goal of an access control mechanism should be to minimize the amount of work required by security administration personnel. This can be accomplished in several ways with ARF. One of the ways has already been described - defining a few rules which describe the security policy of a computer system and basing access authorization on this small set of rules. The effort required for administration can be further minimized, and the security of the system can be increased, by using ARF to enforce the concept of separation of duties among several sets of security administration personnel.

This concept of a hierarchy of administration and evaluation is what we refer to as HDAC, Hierarchical Discretionary Access Control. With HDAC, a security group can partition subjects into domains of administration and can set rules for the system independent of the rules set by the creators of the actual objects. This is in essence a form of non-discretionary access control which many commercial customers consider to be adequate and preferable to MAC (Mandatory Access Control). Hierarchies of administration and evaluation are discussed in the next two sections.

Subject Administration

Using ARF, a hierarchy of administration can exist for subjects. Certain sets of users can be responsible for adding, deleting, altering subjects of a specified set.

example:

Suppose the following two files exist:

```
File Name:    $SYS.ARFADMIN.RULES
Access Rules:
  READ:  SUBJ.ROLE = 'SEC_ADMIN'
          OR PROGRAM_NAME = $SYS.ARF.ADMIN;
  WRITE: SUBJ.ROLE = 'SEC_ADMIN';
  PURGE: NOBODY
```

```
 ____________________________________________
|                                            |
|SWDEV.*   : SUBJ.ROLE = 'SWDEV_SEC'         |
|FINANCE.* : SUBJ.ROLE = 'FINANCE_SEC'       |
|____________________________________________|
```

This file specifies who can change which user records and can only be altered by SEC_ADMIN personnel.

```
File Name:    $SYS.SYS.USERID
Access Rules:
  WRITE: PROGRAM_NAME = '$SYS.ARF.ADMIN';
  PURGE: NOBODY;
```

```
 ____________________________________________
|              |                             |
|user_name     | user attributes, etc.       |
|--------------------------------------------|
|                                            |
|SWDEV.JOHN    \ administered by SWDEV_SEC   |
|SWDEV.MARY    /                             |
|                                            |
|FINANCE.ALICE \ administered by FINANCE_SEC |
|FINANCE.BOB   /                             |
|____________________________________________|
```

The ARF administration program, $SYS.ARF.ADMIN, uses the rules in $SYS.ARFADMIN.RULES to determine who can change which user records in $SYS.SYS.USERID, thus providing a clear separation of duties between SWDEV security personnel and FINANCE security personnel. The entries in $SYS.SYS.USERID will then be used to enforce access restriction to the system and entities within the system as described previously.

Object Administration

Given that objects are protected via ARF, whose decisions are based on a set of subject and object attributes, a set of rules, and a set of definitions, administration is made considerably easier if a hierarchy of rules is implemented and enforced. That is, a security administrator should be able to say "All disk objects on volume $A adhere to Rule_A. All disk objects on $A.X adhere to Rule_A_X, etc. Therefore, specific rules need not be applied to each specific object and the rules can be changed without altering the objects affected by the Rules.

Control over who can change the rules can be enforced via ARF. There can be a set of Rules specifying Rule_Name and the Rule for altering specified Rules.

example:

Suppose the following two files exist:

```
FileName:     $SYS.ARFADMIN.RULES
AccessRules:
  READ:  SUBJ.ROLE = 'SEC_ADMIN'
         OR PROGRAM_NAME = $SYS.ARF.ADMIN;
  WRITE: SUBJ.ROLE = 'SEC_ADMIN';
  PURGE: NOBODY;
 ___________________________________________
|                                           |
|$FINANCE.ACCTPAY.*: SUBJ.ROLE              |
|               = 'ACCTS_PAY_SECURITY'      |
|                                           |
|$FINANCE.ACCTRCV.*: SUBJ.ROLE              |
|               = 'ACCTS_RCV_SECURITY'      |
|___________________________________________|
```

This file specifies who can change which rules and can only be altered by SEC_ADMIN personnel.

```
File Name:    $SYS.ARF.RULES
Access Rules:
  READ:  ANYBODY;
  WRITE: PROGRAM_NAME = '$SYS.ARF.ADMIN';
  PURGE: NOBODY;
 ___________________________________________
|                                           |
|$FINANCE.ACCTPAY.*: R: 'ACCTS_PAY'         |
|                       IN SUBJ.GROUPS;     |
|                                           |
|$FINANCE.ACCTRCV.*: R: 'ACCTS_RCV'         |
|                       IN SUBJ.GROUPS;     |
|___________________________________________|
```

This file specifies who can access which files and can only be altered by someone running the program $SYS.ARF.ADMIN.

The ARF administration program, $SYS.ARF.ADMIN, uses the rules in $SYS.ARFADMIN.RULES to allow/disallow changes to the rules in $SYS.ARF.RULES. The rules in $SYS.ARF.RULES are the rules used by the reference monitors for the authorization checks when users try to access objects.

As defined, only users whose role is Accounts Receivable Security can alter the rules for files in subvolume $FINANCE.ACCTRCV and only users whose role is Accounts Payable Security can alter the rules for files for the subvolume $FINANCE.ACCTPAY.

For accessing the files, users in group ACCTS_PAY can read files in subvolume $FINANACE.ACCTPAY while members of the group ACCTS_RCV can read files in subvolume $FINANCE.ACCTRCV. Note that this may or may not include the people who specified the access rules.

Only a user whose role is 'SEC_ADMIN' can alter the rules contained in $SYS.ARFADMIN, and the object authorization rules in $SYS.ARF.RULES can only be altered by running the program $SYS.ARF.ADMIN. Access to this program could have further restrictions.

This allow a clear separation of security administration duties, but ultimate security responsibility does rest on those users whose ROLE = 'SEC_ADMIN'. They're the ones who set the rules.

Groups

Security policies are concerned not only with which subject may obtain access to which objects, but also with the granting, revoking, and denying of authorizations to and from users and groups. Given the set of authorizations for users and groups, some rule must be applied for deriving authorization for subjects.

In the general case, a user may belong to more than one group. In assigning privileges to subjects acting on behalf of a user, one can choose to:

1. Have the subject operate with the union of privileges of all the groups to which the user belongs, as well as all of his or her individual privileges;
2. Have the subject operate with the privilege of only one group at a time;
3. Allow the subject to choose whether to operate with its user's privileges or with the privileges of one of the groups to which its user belongs; and
4. Implement some other policy.

Roles

Some applications may require that discretionary access controls be specified on the basis of user roles. Many systems have some built-in roles (e.g., system administrator, database administrator, system security officer). However, some applications require that user job access control requirements be formalized in rules (for example, the Secure Military Message System [11]). Thus, a generic capability for application-defined roles is desirable.

The example ARF windows illustrate the concept of both Roles and Groups. As illustrated, the difference is quite clear. Roles are single entities whereas Groups are sets. Testing for groups membership is less restrictive than testing for a specific role. This distinction allow users to operate either with the union of all the privileges of the Groups or only in a particular Role. Rules allowing the user to activate particular Roles, whether they be elements of the Groups set or not allows complete flexibility regarding the operation of the system. No assumptions about Roles and Groups are built into ARF - indeed ARF does not know about Roles and Groups until they are defined.

Explicit Denial

In the higher evaluation classes of the Criteria [3], users must be able to specify which users and groups are authorized for specific modes of access to named objects, as well as which users and groups are explicitly denied authorization for particular named objects. Note that explicit denial of authorization is not the same as simple lack of authorization. For example, the set of users and groups authorized for an object might be implemented as an ACL (access control list) and the set of users and groups explicitly denied authorization as an XACL (**exclusionary** access control list), as in the Naval Surveillance Model [7]. Because the set of users and groups authorized for an object may be independent of the set of users and groups denied authorization, there may be apparent conflicts between the two sets. For example, consider the following ACL and XACL for, say, write access to an object:

ACL: U1, U2, G1, G2

XACL: U1, G2, U3, G3

Now consider the following questions:

Is U1 authorized for R ?

If U2 is in G3,
is U2 authorized for R ?

If U1 is in G1,
is U1 authorized for R ?

If U1 is in G1 & U1 is in G2,
is U1 authorized for R?

If U3 is in G1, U3 is not in G2,
and U3 is not in G3,
is U3 authorized for R ?

The answers to questions such as these are not provided by the Criteria.

Specific choices have been made by particular systems designed for these evaluation classes; however, such choices are arbitrary and may not be suitable for all applications. Lunt [2,8] goes into detail about a number of specific alternative approaches to these questions. However, it is probable that choices about the meaning of denial and about how to reconcile the authorizations granted to users individually and as members of groups are application dependent.

The confusion and questions that result from these ACLs and XACLs serves to illustrate the limitations of ACLs. In these examples, there is no clear mapping from the ACLs to policy - indeed, we contend there cannot be because the policy is at a level of abstraction above ACLs.

It is quite probable that a customer has an access control policy such as:

"Access to relation R is allowed if the user is in group G1, but not if he is also in group G2"

The ARF rule would be:

```
('G1' IN SUBJ.GROUPS)
 AND NOT ('G2' IN SUBJ.GROUPS)
```

If the customer then decided to have a list of trusted users to whom this policy did not apply, the rule would be:

```
(('G1' IN SUBJ.GROUPS)
  AND NOT ('G2' IN SUBJ.GROUPS))
OR SUBJ.NAME IN TRUSTED_LIST
```

There is no ambiguity in such an expression.

Propagation of Authorization

Several database systems, for example Oracle, use grantflags to control the propagation of authorizations. In Oracle, grantflags are specified for each user for a relation or view and access mode. The grantflag can have the value "grant" or "nogrant." The "grant" flag allows a user to grant and revoke the corresponding access mode. In addition, a user with a "grant" flag for a relation or view R and mode m can give and rescind that grantflag.

In SeaView, the propagation of access modes is controlled through the access modes "grant" and "give-grant". If a user U is authorized the "grant" access mode for a relation R, then U can grant or revoke any access mode other than "grant" and "give-grant" for R. A user U that is authorized the "give-grant" access mode for R can additionally grant and revoke the "grant" and "give-grant" access modes for R.

SeaView's inclusion of the give-grant mode enables a greater degree of control over the propagation of authorizations than does Oracle's grantflag approach. In Oracle, if user A grants user B mode m for relation or view R with the grantflag, then A cannot prevent B from granting the grantflag to other users. In SeaView, however, A could grant B the "grant" mode while withholding the "give-grant" mode.

There are other means such as ownership policies for controlling the propagation of authorizations. The choice, however, is likely to be application-specific.

With ARF, the capabilities of users with respect to the grant option could be easily controlled by rules which might vary according to the sensitivity of the relations, physical location, etc. Thus, rather than a "button, button, who's got the button" policy, the customer could say exactly what the policy should be.

Conclusions

This paper has presented a single general access control mechanism that can implement a wide range of security policies. This flexibility means that a customer's true security policy can be implemented by the mechanism. The benefits of flexibility are:

- Policy enforced by vendor mechanism

  ==> Higher assurance of security.

- No excessive restrictions

  ==> Improve usability
  ==> Mechanism will be used and not bypassed

The mechanism represents customer policies by boolean expressions (rules) based on the attributes of subjects, verbs and objects. These rules can be named and referenced by other rules, so layered abstractions can be created to implement the overall security policy. This representation allows the security administrator to clearly express the desired behavior. With ARF, there is only a small semantic gap between the customer's security policies and the features supported by the access control mechanism. The benefits of a small semantic gap are:

- Ease of initial security setup
- Ease of maintenance of security configuration

A key feature of the mechanism is its extensibility. The mechanism can provide access control for operating system objects and application objects. The set of verbs is not fixed. It can handle simple Read and Write operations on files, or Insert and Select operations on DBMS tables, and even application defined transactions like Transfer_Funds that operate on objects defined by an application (e.g., a collection of tables). The benefits are extensibility are:

- Uniform application and OS security
- Simplified security administration

While the flexibility of this mechanism could be used to create obscure and practically unmaintainable security configurations, it is our contention that customers can quickly learn how to avoid such problems. We see the design of the set of attributes and rules to be very similar to the design of database tables. They are set up once by a single knowledgeable person and then rarely changed. In fact, part of the design includes the rules about who can define new rules and attributes. Ordinary users would rarely have to decide how an object should be protected and when they do, the decision is made at a high level. For example, a user might have to choose between the rule that allows read-only access to all people working on the Stealth project (e.g. the Stealth-Interest-Group rule), or the rule that allows read-write access to the managers of the Stealth project (e.g., the Stealth-status-report rule). The user does not need to produce a detailed list of the individuals and groups of individuals who have read, write, execute, and delete access to the file. When used properly, the flexibility will greatly enhance the usability and security of the system.

The difficulties presented by the richness of this mechanism are offset by the fact that customers can think about security policies in an abstract sense. They do not need to be confined by the restrictions of traditional mechanisms. Customers will be able to implement security policies which are closer to their true security policies. For many customers, it will be the first time they have thought in terms of general policies rather than specific implementations.

References

[1] Computer Associates. ACF2 Reference Manual. 1986.

[2] L.J.LaPadula, D.E.Bell, "Secure Computer Systems: A Mathematical Model", MITRE Technical Report 2547, May 1973.

[3] National Computer Security Center, Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, Dec 1985.

[4] National Computer Security Center, A Guide to Understanding Discretionary Access Control, NCSC-TG-0003 Version 1, Sept 1987

[5] T.F.Lunt. Access Control Policies: Some Unanswered Questions, Computer Science Laboratory, SRI International, Menlo Park, California, 1988.

[6] D.F.C.Brewer, M.J.Nash, "The Chinese Wall Security Policy", 1989 IEEE Computer Society Symposium on Security and Privacy, IEEE Computer Society Press, May, 1989.

[7] R.D.Graubart and J.P.L.Woodward. "A Preliminary Naval Surveillance DBMS Security Model". Proc. 1982 Symposium on Security and Privacy, pages 21-37, April 1982.

[8] D.E.Denning, T.F.Lunt, P.G.Neumann, R.R.Schell, M.Heckman and W.R.Shockley. Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System. Computer Science Laboratory, SRI International, Menlo Park, California, 1986.

[9] D.D.Clark, D.R.Wilson, "Evolution of a Model for Computer Integrity", Ernst & Whinney, 1988.

[10] D.E.Denning, T.F.Lunt, R.R.Schell, M.Heckman and W.R.Shockley. The SeaView Formal Security Policy Model. Computer Science Laboratory, SRI International, Menlo Park, California, 1987.

[11] C.E.Landwehr, C.L.Heitmeyer, and J.McLean. "A Security Model for Military Message Systems". ACM Transactions on Computer Systems, August, 1984.

[12] T.F.Lunt. "Access Control Policies for Database Systems". Proceedings of the IFIP WG 11.3 Workshop on Database Security, Kingston, Ontario, October, 1988, forthcoming.