A New Security Policy for Distributed Resource Management and Access Control

Steven J. Greenwald

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375
United States of America

(greenwald@itd.nrl.navy.mil)

Abstract

The common security policies of access control and resource management are based upon a central managing authority, called the system administration, that is ultimately responsible for the management of a usage policy. This management is usually done with some combination of mandatory access control and discretionary access control methods, where each user is granted (or denied) privileges and resources depending on the usage policies enforced at that particular system. System administration includes the management of system resources, user accounts, and user privileges. This security policy is typified by an operating system such as UNIX, and it introduces several difficulties when working in a distributed computing environment.

This paper addresses distributed resource management and access control. It proposes a new version of the "Distributed Compartment Model" (DCM), first developed by Greenwald in his 1994 doctoral dissertation. DCM consists of two major components: Handles, a method for role based access control, and Distributed Compartments, a method allowing users to manage resources within a distributed system across administrative domain boundaries. This new version of DCM discussed in this paper is a refinement of the original formal security policy model. This paper concentrates on the history and background of the problems motivating the creation of DCM, describes the informal security policy, proposes some example scenarios as to how DCM could be used, and concludes with a discussion of the results of this research.

1996 ACM New Security Paradigm Workshop Lake Arrowhead CA
0-89791-878-9/96/09 ..$3.50

0 INTRODUCTION

The security policy currently used on most distributed systems is an old one, dating back to simpler times when most computer systems were centralized. This security policy is based on the idea that there is a central managing authority, called the *system administration*, that is ultimately responsible for the management of computer security within an administrative domain [19]. In this security policy, system administration includes the management of system resources, user accounts, and user privileges. This security policy is typified by an operating system such as UNIX [1]. This paper shall refer to this older security policy as the *Jurassic Age Security Policy* (JASP) since it apparently dates back to the time when huge dinosaur computers were kept in air-conditioned pens, lazily grazing on their data, before faster, leaner machines wiped them out.[1]

JASP introduces some difficulties when working in a distributed computing environment. The solution proposed here is a new security policy consisting of two components: *handles* and *distributed compartments*. This new security policy, the *Distributed Compartment Model* (DCM) [8, 9] is a new paradigm for the man-

[1]The author is obviously open to suggestions for more appropriate terminology, and is also interested in exactly when JASP first came into existence.

agement of resources and the control of user access on distributed computer systems. DCM was specifically created to rectify some of the resource management and access control problems and deficiencies of the common security policies [6].

The remainder of this paper is organized as follows. Section 1 introduces the problem statement that motivated the creation of DCM. Section 2 introduces the DCM security policy. Section 3 consists of some examples as to how DCM could be used. Finally, section 4 concludes this paper with a discussion of the results of this research.

1 Problem Statement

1.1 Introduction

Most of the computer systems in use are based on JASP. This paper is specifically concerned with the management of system resources and the management of access control in a distributed computing environment. Several systems and models have been created to facilitate versions of security in a distributed environment, such as Guard [28], the NRL Military Message Experiment (Sigma) [27], the NRL Military Message System [14], Project ADMIRAL [25], CMU's Andrew [22], Amoeba [16], the Centralized-Parallel-Distributed model [3], and the IEEE Mass Storage Reference Model [10] to name but a few. However, these systems and models still contain some undesirable elements of JASP [9], leading to the following problems.

1.2 Resource Management and Access Control Problems with Distributed Systems

JASP presents the following problems when working in a distributed environment.

1. User-names are often duplicated across name-space domains in a distributed system. For example, two different users may have the same user-name on two different hosts within a distributed system. This presents a problem when using groupware[2]: how can each user be unambiguously identified? Attaching a host computer system identifier to each user-name (*e.g.*, the RFC822 Standard for ARPA Internet Text Messages) works, but is often cumbersome.

2. Location transparency (where a user is not restricted to a single administrative domain in order to access applications) may not be possible. In an application where location transparency is a goal, using a user-name and host-identifier combination is unacceptable. For mobile users who often change hosts, the combination of user-name and host-identifier fails to uniquely identify the user (*e.g.*, "biff@host1.foobar.org" and "biff@host2.foobar.org" may be the same user, resulting in multiple aliases for the same user). A related problem is that one user may have two (or more) different user-names at different locations (*e.g.*, are "greenwald@itd.nrl.navy.mil" and "sjg@cis.ufl.edu" the same user?). Even worse, two users in different administrative domains may have the same user-name.

3. Unique user identifiers based on user-name and host-name combinations may be redundant to groupware collaborators. In many cases of collaboration the users do not care about which computer systems their colleagues are using. For example, two researchers at different universities collaborating on the same research paper are not particularly interested in cumbersome host computer names—they are only interested in collaborating with one other, and unambiguously identifying each other in a convenient manner.

4. There exists a "weak link in the chain" effect. This means that security is a problem since the security of the entire distributed system depends upon the security of the individual hosts that are being used within a group of administrative domains. One lax system administration can compromise an entire distributed system by allowing access to unauthorized users, sharing of user-names, *etc.* This results in the system with the weakest security setting the maximum security standard for the entire distributed system.

5. In many installations the system administration is reluctant to permit a single user to have multiple user-names (this is not a criticism of system administrations—they often have good reasons for this policy). This makes it difficult for users to test and use groupware. The reason this point is important is that for many groupware applications (*e.g.*, DCS [18]) a particular user may need to assume different roles (in DCM a role is a labeled set of capabilities that a user can activate). Roles, as opposed to protection groups, are generally considered to be a form of mandatory access control (*e.g.*, a subject generally cannot allocate permissions to a role or determine membership [20]). For example, a user may wish to simultaneously assume the

[2]Groupware (computer supported cooperative work) is commonly defined as software designed to support groups of people, often at different physical locations, working together.

roles of "professor" and "chairman" for a particular groupware session. However, it is typical that all processes started by a user have the same privileges [1]. There are many references for the interested reader on role-based access control (RBAC) [13, 14, 26, 20, 21].

6. It may be difficult to share resources with other users on other computer systems without getting permission from the system administrations involved. For example, two users subject to different system administrations who wish to share a file with each other may find it impossible without using cumbersome methods (*e.g.*, File Transfer Protocol, electronic mail, World Wide Web) that are unsuited for real-time applications.

7. Foreign user accounts are often necessary to correct the previous problem. This places a management burden on the system administration because it has to manage users from a foreign environment. In addition, there is the very serious difficulty of the system administration initially verifying the identity of these foreign users, who are often not physically present. In addition, foreign user accounts present the potential problem of giving the foreign user too many permissions.

8. Military chain of command systems and corporate hierarchical systems may be difficult to model and implement because their structure clashes with the "flat" structure of the omnipotent-system-administrator approach of JASP.

1.3 Comments

The above problems result because most of the security paradigms in use are outmoded. They are based on the assumption of a centralized access control mechanism dating from the days when centralized time-sharing mainframes dominated the field. This naturally resulted in centralized management of system resources, and the implicit condition of location dependency for users and resources. Historically, these conditions were not seen as problems because the security systems were designed for these single stand-alone systems.

2 THE DCM SECURITY POLICY

2.1 Philosophy of the Security Policy

The philosophical justifications and a narrative description of version two of the DCM security policy are presented here. The original security policy was presented in my dissertation [8] and a subsequent technical report [9].

What follows can be thought of as an organizational (informal) security policy. I would like to emphasize that the philosophy behind the solution to the problem statement was not motivated by the traditional view of a multilevel secure (MLS), categorized system, such as that presented in the famous "Orange Book" [5]. It is not an MLS system such as the often-cited BLP model [2].

There are many ways to solve the above problem statement. I chose to adopt a libertarian (classical liberal) philosophy that maximizes the freedom of users while limiting system administration intervention to only necessary functions. Philosophically, this should have the benefits of allowing users as much flexibility in managing their affairs as possible, while eliminating much of the drudgery commonly associated with system administration. However, this should not be construed as a "lax" security policy. Within their limits, users can be as regimented as a military chain of command system if they so choose. Hopefully, this is a good compromise between authoritarian control and anarchy. In fact, as we shall later see, one of the advantages of this security policy is that it can represent authoritarian hierarchical command structures as used in the military, government, or business worlds.

The solution to the problem statement has two parts. The first is a role based access control method I term *handles*. The second is a method for allowing users to manage resources within a distributed system, across administrative domain boundaries, with a measure of independence from any system administrations, that I term *distributed compartments* (*discom* for short).[3]

2.2 Handles

The central ideas behind handles is that for groupware applications, user-names that depend on operating systems and networks should be eliminated as a means of identification and access control.

The proposed solution is the concept of a *handle*, which is a type of role-based access control that spans administrative domains. A handle is a labeled set of capabilities, where capabilities are actions that subjects can perform upon objects. This allows a groupware application to use a handle as an identifier for users, and as a way to assign those users a set of capabilities. A user joining a groupware session is queried for a handle, and is then authenticated by the groupware's security manager process. Authentication can be an entirely in-

[3]This paper will freely associate the terms "distributed compartment" and "discom."

dependent operation (*e.g.*, user knowledge, physical attributes, possession of security objects) using a method such as Kerberos [24, 7, 23]. This keeps user access to that application as separate as possible from the operating system. For example, a handle could have the label "Author" and have a set of capabilities that allow operations upon a set of objects such as "File-1," "File-2," *etc.*

Under this method, an individual user would first need to gain access to a particular host in the distributed system through JASP by having a valid user-name and (perhaps) a password.[4] The user would then need a valid handle and would then need to be authenticated by the groupware in order to be allowed access to the application. Once this is complete, the user has activated a role (a set of capabilities) for that application. It should be noted that the relationship between users and handles can be many-to-many. Additionally, once a user has activated a role, auditing can still take place (*i.e.*, if necessary, the user can still be identified while the role is activated).

This role based, operating system independent approach, has the following advantages.

1. Security policy dependencies are reduced because the security policy is not entirely dependent on an operating system, system administration, bureaucracy, *etc.* As things now stand, even one lax system administration can disrupt security in groupware applications. Some examples of these security policy violations follow.

    - Users might be permitted to share accounts with one another. Under this situation it is impossible to verify the identity of the actual user by the use of user-names. Of course, users could still do this with handles, but it is hoped that the impetus to do so (*e.g.*, user frustration with the system administration) will decrease or disappear. In certain cases, it may be beneficial for users to share handles (*e.g.*, this is useful in some RBAC paradigms).
    - Users who have no authorization to use a particular groupware application may have to be given permission to access files and applications that they have no need for.
    - Users who need access to a particular application, yet lack the permission to access the application, may have to wait a long time for the (usually overworked) system administration to grant the proper permission.

2. Handles can be more descriptive than user-names. For example, a user-name of "sjg" does not convey much information. With handles there could be a more descriptive name such as "Steve," "Greenwald," "Third Programmer," "Referee 2," *etc.* The label associated with each handle can be much more descriptive to users of groupware than (often cryptic) user-names.

3. Multiple handles can be permitted for the same user. The advantages of allowing this follow.

    - The testing of groupware applications becomes much easier. One user can easily assume the roles of many subjects by having several handles.
    - Anonymity is possible.
    - Multiple roles for individual users becomes possible. Different handles can be used for different user roles. A user needing to change roles just needs to use the appropriate handle.
    - Intervention by the system administration is limited. The system administration does not have to be concerned with creating multiple accounts for the same user.

4. More than one user may share the same handle. This allows multiple users to share the same role.

5. Security policy mechanisms can be implemented relatively independently of any underlying operating system. Keeping security matters within a particular groupware application facilitates development of whatever higher-level operating system paradigm is wanted, independent of the actual operating system. For example, a new operating system could be simulated while using an old one, incorporating separate user accounts. Testing of experimental security mechanisms becomes safer since the testing environment is analogous to a tightly isolated "glove box."

6. Management of handles can be made part of the groupware application, allowing different security methods to be implemented. For example, security can be partitioned in hierarchical compartments with different users maintaining the handles of their own compartments (this idea will be discussed in the next section).

One area specifically not covered is authentication. There are a variety of authentication methods available (*e.g.*, passwords, physical attributes, possession of objects). Specific authentication methods are beyond the scope of this research, and are separate from DCM. However, one important point is that the system administrations will not be responsible for the authentication

[4] It should be possible to implement DCM as an actual operating system, thereby eliminating this first step.

of handles. That will be the responsibility of the particular groupware application. This frees the system administration from the burden of managing the handles, and frees the groupware managers from the necessity of having to contact the system administration every time maintenance is needed for access control.

2.3 Distributed Compartments

Handles need a framework in which to operate. This is also true of groupware applications. This motivated the creation of DCM.

A *distributed compartment* is a group (*i.e.*, a tuple) of entities that is not restricted to a single physical computer system.

More formally, a discom is defined as a tuple,

$$D = (S,\ O,\ P,\ R,\ H),$$

where S is a set of subjects representing user processes or dæmons, O is a set of objects representing such things as files, hardware devices, programs, users, [2, 4, 19], and (recursively) child discoms [8], P is a set of privileges, R is a set of resources (atomic units of computer systems used by software) such as file space, memory, or CPU time allocation, and H is a set of handles that the users of the discom D can activate in order to perform actions upon the objects. Objects in a discom are composed of some or all of the resources in the discom's resource pool. It is important to note that the elements of R do not necessarily come from the same administrative domain. Therefore, a discom may span administrative domains (and vice-versa).

Discoms are partially ordered as the nodes of rooted trees called *empires*. A discom may be thought of as similar to a directory in a standard hierarchical directory structure. However, as noted above, it does not necessarily reside on a single computer system. The root discom is termed the *empire discom*.

Each discom must have at least one distinguished member of its subject set S termed a *governor*. By definition, governors have all the capabilities for that discom (in other words, we can assume for the sake of simplicity that governors can activate a special handle that has all the capabilities in a discom). Governors are an important part of the security policy because one of their properties is that they are also governors of any descendant discoms. This allows a hierarchical command structure to be created without the use of a system administrator (there are other variations on the security policy which do not have this property - the interested reader is referred to [9]).

The *privileges* of a discom consist of at least 24 operations termed the *initial privileges* (they are termed initial because it is possible in the model to have additional user-defined privileges):

1. create a new object from the resource pool;
2. destroy an existing object, returning its resources to the resource pool;
3. modify an existing object by adding or removing resources from the resource pool;
4. merge two existing objects into a single object;
5. split an existing object into two objects;
6. create a child discom from the resource pool;
7. destroy a child discom returning its resources to the resource pool;
8. merge two child discoms into a single child discom;
9. split a child discom into two child discoms;
10. destroy an existing empire;
11. merge two existing empires into a single empire;
12. split an existing empire into two empires;
13. create a new subject;
14. destroy an existing subject;
15. create a new handle;
16. destroy a handle;
17. grant governorship to a subject;
18. rescind a governorship from a subject;
19. add a resource to the resource pool of a discom (from the resource pool of its parent discom);
20. remove a resource from the resource pool of a discom (returning it to the resource pool of its parent discom);
21. grant a subject the right to activate a handle;
22. rescind from a subject the right to activate a handle;
23. make a subject a member of a child discom;
24. remove a subject as a member of a child discom.

This combination of subjects, objects, and privileges, makes it possible to create a system similar to an access control matrix.

DCM has a set of security properties. Their informal definitions follow.

Divine Right Axiom A subject can create an empire discom only if given that privilege by the administrators of the system administrations involved. This is the only area where the system administration need be involved in the management of discoms. The rationale for this is that the system administration is ultimately responsible for the use of its system. It should be given the right to restrict the creation of empire discoms.

Creator Property The creator of a discom automatically becomes a governor of that discom. The rationale for this is that if this property was not present, then it would be possible to create discoms that were inaccessible to all the subjects in an empire.

Government Property The governor of a discom may grant and revoke handle activation privileges to non-governor subjects of that discom. The rationale for this is that someone has to do this or nothing will get done. The purpose of the security policy is to eliminate, as much as possible, involvement by the system administration. If all subjects had the power to do this, then anarchy would be the result. Therefore, only governors or those subjects they give this privilege to may have this special status.

Nova Property A non-governor subject may access a descendant discom only if made a member of that discom by a governor of an ancestor discom (conditional down access). This is similar in concept to the BLP *-property ("no write down"), except reversed, hence the name (a nova is an exploding star).[5] The rationale for this is that governors control their resources and may allocate them as they wish, in the military chain of command sense.

Ceiling Property A subject may not access an ancestor discom without being a subject of that discom. The ceiling property does not allow any access at all of an ancestor discom without membership. The rationale for this is that the governors of the ancestor discoms are allowed to manage their resources as they see fit. They may not wish even write-only access from descendant discoms since that might use up resources (*e.g.*, disk space) and cause (for example) a denial of service problem, or a covert channel [12, 17].

Cordon Property Discoms must be isolated from other discoms. Isolation means that the only things that any two discoms can share are members of their subject sets. The rationale for this is that if discoms weren't isolated from one another, then information could flow between them in unrestricted ways, possibly violating the nova property and the ceiling property. Allowing only subjects to act as information flow channels between discoms makes trusted subjects possible (and perhaps necessary).

Demesne Property[6] The governor of a discom always has unrestricted access to any descendant discoms. The rationale for this is that a governor, by definition, controls the resources of the discom. Since any descendant discoms that exist are part of the resources of a discom, the governor should not be prohibited from any accesses to them. In addition, if a governor could not always access descendant discoms, completely autonomous discoms could result, creating a potential need for the intervention of the system administrations involved, if the governor ever needed access again after the autonomous discoms prohibited access to the governor.

The intent of these properties is to create a system where the management of distributed compartments is not done by the local system administration, but ultimately by the individual users who are governors of empire discoms (by the divine right axiom). It should be noted that some mechanism is probably needed to serialize the transactions that would occur in an actual implementation to prevent problems such as deadlocks and race conditions.

2.4 Empire Example

An example empire might consist of three discoms arranged as in figure 1. Each discom D_i consists of a set of subjects S_i, a set of objects O_i, a set of privileges P_i, a resource pool R_i, and a set of handles H_i. D_1 is the empire discom, and has 2 child discoms, D_2 and D_3. Each discom has at least one governor subject. Suppose that subject s_1 is a member of S_1 and is also a governor. That means (due to the demesne property) that s_1 is also a member of S_2 and S_3. However, the objects, privileges, resources and handles of a discom are never shared with other discoms.

[5] This is similar in concept to McLean's †-property [15].

[6] "Demesne" is usually pronounced similarly to "domain."

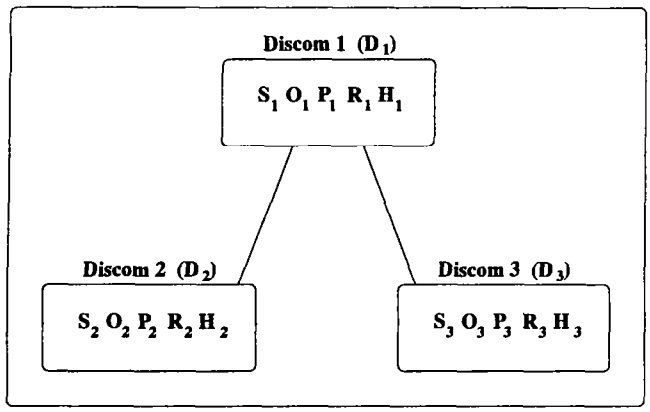

Figure 1. An Empire with Three Discoms.

In this example, D_1 might be the discom of a professor, and accessible only to the professor (*i.e.*, D_1 has only one subject, s_1, who is also a governor) with a set of resources, some objects (perhaps representing files), some privileges, and a set of handles (perhaps only 1 handle giving the governor the maximum privileges for the discom). Since the professor, s_1 is the governor of the empire discom, s_1 must also belong to the subject sets of D_2 and D_3 (because of the demesne property).

Discom D_2 might be a discom that is used by the professor and a teaching assistant (s_2) to grade papers. The teaching assistant need not have the maximum privileges for D_2. For example, s_2 might be permitted to activate a handle that allows it to read certain files, but not write them. Some of the resources in the resource pool R_2 initially came from R_1 at the time that D_2 was created. Of course, since s_1 is a governor, it could modify the handle activation rights of s_2 or even make s_2 a governor (due to the government property).

Discom D_3 might be a discom that is used by the professor to collaborate with a colleague s_3 from another university in another administrative domain. The colleague may have contributed resources from the "foreign" university's administrative domain, such that R_3 is composed of resources from two administrative domains. Suppose that the two subjects are both governors of D_3, and could create, destroy, and modify objects (for example, they might be using a groupware application to write a paper together, with the paper represented as an object in O_3).

In this example, s_2 would not be allowed access to D_1 (due to the ceiling property) or D_3 (because of the cordon property since s_2 is not a member of D_3), and s_3 would not be allowed access to D_1 (again, because of the ceiling property) or D_2 (again, because of the cordon property since s_3 is not a member of D_2). Since s_1 is the governor of the empire discom, s_1 would have unrestricted access to all of the discoms in this empire. Of course, another discom (say D_4) might be created as a child of D_3, which would mean that s_1 and s_3 would both automatically be governors of D_4 due to the demesne property (additionally, if there were another subject in D_3 that actually caused the creation of D_4, that subject would be a governor of D_4 due to the creator property). Conversely, if another discom were to be created as a child of D_2, s_2 would not automatically become a member of it, since s_2 is not a governor subject.

Notice that because of the nova property, if a non-governor subject were created in D_3, it could not access D_4 unless it were made a member of D_4.

How was this empire initially created? By use of the divine right axiom. This is the only point at which a system administrator would be necessary in the entire process outlined above. Once the empire discom is created, its governor can proceed with the creation of descendant discoms, objects, *etc.*, without any more administration intervention.

2.5 Subject Instantiation Example

In DCM before users can do work, they must be instantiated as subjects. The following are the 6 basic operations that a subject must go through, in order to perform "work" in DCM ("work" is used in the context of executing non-DCM applications within DCM).

1. *Instantiation.* The user must obtain a valid subject identifier and must then be authenticated. Once this occurs, an instantiated subject corresponding to an actual user exists. This allows auditing of the user or subject to take place.

2. *Joining.* The instantiated subject chooses a discom to join. The request must be validated by the discom.

3. *Activation.* The subject, joined to a discom, chooses a handle to activate. The request must be validated by the discom. Once a handle is activated, "work" can be performed.

4. *Release.* An activated subject releases an activated handle.

5. *Departure.* A subject joined with a particular discom departs from that discom.

6. *Passivation.* The subject is passivated, and the user exits the system.

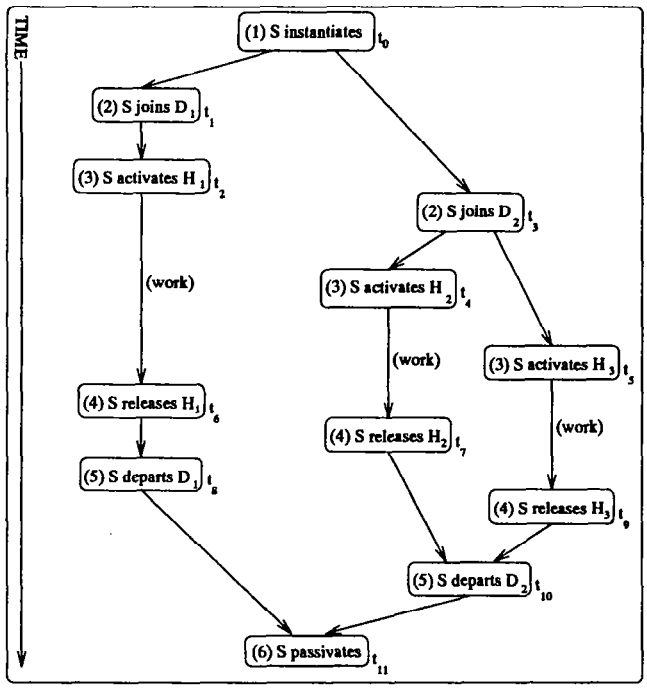

Figure 2. An Example Subject Instantiation Graph.

These steps are nested, in the sense that (1) must take place before (6), (2) before (5), and (3) before (4).

An illustrated example of how this might work is given in the subject instantiation graph of figure 2, where a particular subject is designated by S. There are 2 discoms D_1 and D_2. There are 3 handles: in D_1 there is H_1, and in D_2 there are H_2 and H_3. Time is denoted by $t_0 < t_1 < \cdots < t_{11}$. Note that these are not units of time; instead these symbols are merely used to indicate that an event happened before (or after) some other event. In addition, the label "(work)" in the graph is used to denote when a subject can execute non-DCM applications (as noted above).

Figure 2 illustrates how a subject can be joined to multiple discoms during the same time interval, and also how a subject can activate multiple handles during the same time interval. The following is a walk-through of the graph during the different times, and also mentions how the DCM system might appear to a user.

0. A user instantiates a subject S. This might occur by the user running an application program to enter a DCM system. Once instantiated, the user "view" of the system might present a selection menu that lists the discoms that the user may join.

1. S joins D_1. At this time, a list of handles that S is allowed to activate in D_1 might appear in a selection menu.

2. S activates H_1. At this time, S has activated H_1 in D_1. Work can now be performed in D_1.

3. S joins D_2. Note that at this time, S is still joined to D_1, and H_1 is still activated. At this time, a list of handles the S is allowed to activate in D_2 might appear in a selection menu.

4. S in D_2 activates H_2. Work can now be performed in D_2. Note that S is also still activating H_1 in D_1, and performing work in D_1 concurrently.

5. S in D_2 activates H_3. At this time, S has activated a total of 3 handles: H_1 in D_1, H_2 in D_2, and H_3 in D_2. Work can be performed concurrently on 3 paths in the graph at this time.

6. S in D_1 releases H_1. At this time, S can no longer perform work in D_1.

7. S in D_2 releases H_2. At this time, S can no longer perform work in D_2 using H_2. Note that H_3 is still activated by S.

8. S departs D_1. S is still joined with D_2, and is still performing work using H_3 in D_2.

9. S in D_2 releases H_3. At this point, S can perform no work whatsoever, even though S is still a member of D_2.

10. S departs D_2. S is still instantiated at this point.

11. S passivates. The user is no longer using the DCM system.

2.6 Conclusions

The combination of handles and distributed compartments is a reasonable solution to the earlier problem statement. Combining these two ideas reveals the following areas of concern.

1. How should system resources be allocated within the discoms? It should be possible to implement a system where the resources of a discom are restricted according to a subject's privileges within that discom (with something analogous to an access control matrix). For example, limiting the CPU time of a discom member might prove to be a very valuable thing in a real-world application.

2. What is the best way to manage the distribution of the resources of the discoms over a distributed computer system?

3. What is the best way to manage the name-space that will occur with this system? For example, it might be desirable if each discom had its own name-space. This would allow handles unique to each discom.

4. There is a profound difference in the way subjects are destroyed and governorship is revoked. Since subjects can exist in a rooted tree, they must be deleted from the "bottom up." Governors also can exist in a rooted tree, however the demesne principle prohibits rescinding from the bottom up. Instead, governors must be rescinded from the top down (effectively splitting a tree if one exists at some point in the revocation).

3 Usage Examples

This section contains some example scenarios as to how DCM might be put to use in the real world in order to better clarify the ideas presented. All labels are given in uppercase for clarity.

3.1 Electronic Mail Between Discoms

One common application is electronic mail. In order to exchange electronic mail between discoms (or any other type of information) a subject would be used as an intermediary. Of course, the subject would have to be a member of both discoms. It could read the mail from one discom, and act as an information flow channel while writing to the other discom. Of course, the subject would need to have the necessary handles and privileges in both discoms in order to accomplish this.

3.2 "Simple" File Creation

A file is called "simple" if the resources from which it is composed come from only one administration. A file would be composed of one or more objects in a particular discom. Each object would contain only resources from the same administration. The resources involved might be blocks of disk storage.

3.3 "Complex" File Creation

A file is called "complex" if the resources from which it is composed come from more than one administration. Such a file would be composed of one or more objects in a particular discom. The resources of these objects would come from more than one administration. This would amount to sharing the file across multiple system administration boundaries. The resources involved might be blocks of disk storage from the different system administrations involved.

3.4 Replicated Fault Tolerant Files

Replicated fault tolerant files are possible within a discom by making several files that do not share resources from the same system administration. For example, if there were two files, each an object: o_a and o_b in a particular discom, information stored in them could be replicated (*i.e.*, the two files would always contain the same data under ideal circumstances). However, to implement a replicated fault tolerant file structure across administrative domains, a constraint would be added that the two files must not contain resources from administrative domains in common with each other. This would be an extremely easy way to implement replicated fault tolerant files across administrative domain boundaries, with a high degree of fault tolerance since the two files do not share any resources in common from the same administrative domains.

3.5 CPU Resource Access

In DCM, resources can be allocations of CPU time. For example, a particular resource r might represent up to 10 seconds of CPU time of supercomputer access (if less than 10 seconds were used, all of the resource would be expended since resources are atomic in this security policy). An object incorporating r could use up that resource depending on the actions of the subject who controls the object. This is an example of an expendable resource.

3.6 Distributed Conferencing

A groupware system such as the Distributed Conferencing System (DCS) [18] is designed to facilitate group work. Such a system can be used for *conferencing* where the participants are not physically present in the same location. It would be useful for each conference to have associated with it one (or more) discoms in which objects could reside, and in which distributed handles could be used to identify the conference participants by roles.

For example, consider two conferences named "EXAM" and "REFEREE." Each conference holds regular meetings with the participants (subjects) using DCS, who do not meet face to face. Each conference is an application running in its own discom, and each discom has its own set of handles.

In the EXAM conference, members are concerned with creating, administering, and grading one subject area of a Ph.D. comprehensive examination given periodically in a computer science department. Profes-

sor Ada Algol is a subject that can activate the handle "CHAIRPERSON." She also has certain privileges associated with this subject/handle (for example, controlling the floor-passing[7] mechanism of certain DCS applications). It is clear from her handle what role she is playing (although the mnemonic value of the handle may not be required, or may be non-existent since there is no way to enforce the requirement that all handles have a mnemonic value).

The members of the REFEREE conference are concerned with evaluating papers submitted to a journal. All the members are anonymous referees for the journal. Professor Bert Basic is a subject given the handle "CHAIRPERSON." He also has certain privileges associated with this subject/handle. In this case however, his identity remains a secret due to the sensitive nature of the task the conference is performing.

In both conferences there exist two handles with the same label (and the same mnemonic value), yet used by different subjects and unique to their respective discoms. Additionally, as time goes on and as duties rotate, it is easy to change the identity of CHAIRPERSON to another member.

In addition, Professor Bert Basic may be a member of the EXAM conference with the handle "BERT," and also a member of the REFEREE conference with the anonymous handle "REFEREE-3."

3.7 Grading Projects

A typical problem is that of a professor who assigns a programming project to students that is due by a specified date. One of the problems is how the students can submit their program code in a secure way, without other students having access to it, but allowing the graders easy access. Currently, in the UNIX environments in common use in academia, the following are some of the methods used for on-line submission.

1. A "security through obscurity" approach where the graders allow everyone execute and write access to an individual directory created for each student in the graders' own directory tree. The graders keep the names of the specific directories hidden from everyone but their respective students. By allowing execute access but not read access, it is unlikely that someone will stumble upon the name (but not impossible).

2. The graders have each student create a special directory in the student's home directory. The system administration then creates a special protection group to which the only members are the graders, and then changes the group ID of each student's special directory to the new group. The student must then allow group access to the directory. The student may then place files in the directory that can be read by the graders.

3. A special "setuid" program is created that, when invoked by the students will copy their programs to a directory owned by the graders.

All of these methods are unsatisfactory for a variety of reasons. They are potentially insecure, rely on the system administration, are cumbersome, or even potentially dangerous (in the case of using a setuid program).

Using distributed compartments and handles, the graders could create a discom called "PROJECTS" to which only the graders have access. One child discom per student could then be created in PROJECTS with each student a subject in their child discom with their own handle (*e.g.*, "GREENWALD"). Each student would be allowed membership to their own discom, but would be prohibited from access to any other discom in PROJECTS. Students could then be given the necessary privileges to read and write to their own discoms during the submission period.

Due to the creator property, the graders would automatically be governors of discom PROJECTS. Due to the cordon property, the students would be prohibited from accessing each other's discoms as long as the privileges were properly maintained. Students would not be allowed access above their own discom due to the ceiling property. Since the graders are governors of the PROJECTS discom, the demesne property allows them unrestricted access to all descendant discoms, and therefore the student's discoms. When the submission period expires, the graders can simply revoke the students' privileges to their discoms, and thereby prohibit late submissions.

All of this would take place in a secure environment, and with no intervention required from the system administration.

3.8 Paper Collaboration

Two researchers at separate universities with separate system administrations might want to cooperate on the same paper. The researchers might want access to materials at each other's locations (*e.g.*, source code, word processing files, *etc.*). Some of the current ways to do this in a UNIX environment follow.

1. Share accounts. This is generally considered a very Bad Thing.

2. Use File Transfer Protocol (FTP) to periodically exchange files. The researchers would have to ar-

[7] "Floor-passing" is a groupware mechanism whereby control of some application can be passed from user to user.

range this with their respective system administrations, and could not operate in real-time.

3. Use electronic mail to send versions of the files back and forth. If the files are binary, they would have to be uuencoded and uudecoded (uuencoding is a method for exchanging binary data over text systems, such as electronic mail, that prohibit certain control characters). Again, they could not operate in real-time.

4. Petition the system administrations at the respective sites for accounts for the foreign researchers, and the creation of a security group to which they could both belong. This will probably result in the "foreign" users having too many privileges.

5. Use miscellaneous dangerous methods (because they give too many privileges to the collaborators) such as modifications to their ".rhosts" file in certain network environments. This is effectively the same as sharing accounts.

These methods are not particularly satisfactory for obvious reasons. However, using distributed compartments and handles, one of the researchers could create a distributed compartment and make the other researcher a subject (or even a governor), with access using handles through a proxy server. Additionally, they could share resources from each other's administrative domains.

3.9 Location Transparency

A traveling business person for a large corporation with many branch offices might need to have access to a "home" computer system while visiting the different branches. If the computer systems at the various branches are separately administered, but connected by a computer network, this would require the business person to have multiple user-names and passwords, to access the home system via the network.

One solution to this problem is to have a guest user-name at each location with the same user-name and password, restricted to the group of users who travel. Normally this is a bad practice, because historically, guest accounts have been abused. However, this account can be made with severely limited privileges, allowing access only to the client process that manages DCM over the network.

The business person needs only to remember one user-name and password, and can use handles to gain access to the distributed compartments that are required.

3.10 Hierarchical Chain of Command Systems

Current resource sharing security policies are "flat," not hierarchical, with one or more system administrators in an administrative domain that each have the maximum privileges for their administrative domain. Organizations such as the military, government, and business are usually organized in hierarchies. Ideally, a resource sharing security policy should reflect this fact.

For example, an organization might be structured in a hierarchy with a CEO at the top of the hierarchy, then at the next level there might be two departments, Payroll and Accounts Payable each with their own department manager. Under the Payroll department might be a number of Payroll clerks, each with separate duties and privileges. For example, a Payroll clerk might have the duty to enter employee time-card data into the computer system that prints employee checks. If this Payroll clerk went on vacation, it would be desirable to assign another clerk to that role simply by changing handle activation privileges. In addition, in the event of the Payroll department manager leaving, a replacement manager can simply be given the handle activation rights for that position.

4 CONCLUSIONS

It is hoped that DCM is the beginning of the evolution of a new paradigm towards a system of resource allocation and access control that will better meet the needs of distributed computer systems and groupware applications for many different types of systems. A software implementation of this security policy would probably help to further refine these ideas.

There have been objections to this security policy (usually from system administrators) on the grounds that it would allow unscrupulous users to give their resources to any number of other users, leading to abuse of computer system resources. However, this situation already exists in current operating systems (for example, it is unfortunately common for users to share their accounts with others by giving them their user-names and passwords). I feel that in an implementation of this security policy, better control over resources is possible, since it would be relatively easy for an individual user to give others highly *controlled* access to resources in the form of discoms without compromising the security of the existing systems. In other words, it is possible to limit the rights of users, within the context of DCM, so that in a worst case scenario (such as an unscrupulous governor) no damage can be done to the systems outside of the empire to which that user belongs.

There are many results that have emerged from this

research that show how the problem statement has been solved. Some results from the research follow.

1. Handles do not have to exist for users alone. Any entity that needs to access or modify discom objects can have an associated subject and handle. Dæmons, data transformations, probes, processes, *etc.*, can be accommodated by the security policy.
2. This security policy does not have to be centralized (and should not be). It can be implemented in a distributed fashion, using classic client-server techniques, object-oriented techniques such as CORBA [11], *etc.*
3. Security that is compartmentalized and multilevel is possible. Each subject could have associated with it a group of discoms of which it is a member. User-defined privileges would allow subject-object access to agree with many different security policies, with a different policy possible for each discom. For example, BLP could be implemented in one discom, while a less restrictive version of it could be implemented in another. This gives the advantage of containment within a discom in the event of a problem with a new security policy.
4. Since this security policy is distributed, fault tolerance can be added by using replicated data base methods. This should be particularly easy to implement (see the usage example in the previous chapter).
5. A particular discom can span multiple system administrations if the objects in that discom are composed of resources from those system administrations.
6. The initial governors of an empire would probably come from the users who provide resources from their particular system administrations in order to create their empire discom.
7. An administrative domain can participate in more than one empire.
8. Objects can be contained by a set of administrative domains (*i.e.*, objects can span administrative domains).
9. Discoms can be contained by a set of administrative domains.
10. Empires can be contained by a set of administrative domains.
11. While network authentication and privacy is not a part of the security policy, it should be relatively easy to incorporate encryption methods and digital signatures that can be used for communications and for mass storage systems.
12. Perhaps the most important point of all is that this system frees users from a large amount of dependence on various administrative domains, while simultaneously freeing the various system administrations from many tedious tasks. I believe that this point will become increasingly important as distributed systems continue to multiply.

Acknowledgments

I would like to acknowledge the contributions of Ira S. Moskowitz, John McLean, Myong Kang, Bruce Montrose, Catherine Meadows, Laura Corriss, Paul F. Syverson, and all the attendees of NSPW96 that made it such a memorable and outstanding workshop.

References

[1] M. Bach. *The Design of the UNIX Operating System.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

[2] D. Bell and L. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997, The MITRE Corporation, Bedford, Massachusetts, March 1976. Available from the National Technical Information Service as report number: AD A023 588.

[3] G. Benson, I. Akyildiz, and W. Appelbe. A formal protection model of security in centralized, parallel, and distributed systems. *ACM Transactions on Computer Systems*, 8(3):183–213, August 1990.

[4] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[5] Department of Defense. Department of defense trusted computer system evaluation criteria. Technical Report DOD 5200.28-STD, Department of Defense, Washington, D.C., December 1985.

[6] D. Downs, J. Rub, K. Kung, and C. Jordan. Issues in discretionary access control. In *Proceedings of the 1985 Symposium on Security and Privacy*, pages 208–218, Oakland, California, 1985.

[7] S. Garfinkel and G. Spafford. *Practical UNIX Security.* O'Reilly and Associates, Inc., Sebastopol, California, 1991.

[8] S. Greenwald. *The Distributed Compartment Model for Resource Management and Access Control.* Ph.D. Dissertation, University of Florida, Gainesville, Florida, August 1994.

[9] S. Greenwald and R. Newman-Wolfe. The distributed compartment model for resource management and access control. Technical Report TR94-035, Department of Computer & Information Sciences, University of Florida, Gainesville, Florida 32611, October 1994. Available via anonymous ftp from site ftp.cis.ufl.edu in cis/tech-reports/tr94 as file tr94-035.ps.Z, or via the World Wide Web with "http://www.cis.ufl.edu/cis/tech-reports" as the uniform resource locator.

[10] A. Hanushevsky. Security in the IEEE mass storage system reference model. In *Twelfth IEEE Symposium on Mass Storage Systems*, pages 67–77, Monterey, California, April 1993.

[11] A. Hutt, Editor. Common facilities architecture draft 4.0. Technical report, Object Management Group, Framingham Corporation Center, 492 Old Connecticut Path, Framingham, Massachusetts 01701-4568, January 1995. Draft 4.0 of the Common Facilities Task Force Architecture document. Also available on the World Wide Web at URL: http://www.omg.org/docs/1995/95-01-02.ps.

[12] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[13] C. Landwehr and C. Heitmeyer. Military message systems: Requirements and security model. Memorandum Report 4925, Naval Research Laboratory, Washington, D.C., September 1982.

[14] C. Landwehr, C. Heitmeyer, and J. McLean. A security model for military message systems. *ACM Transactions on Computer Systems*, 2(3):198–222, August 1984.

[15] J. McLean. A comment on the "basic security theorem" of Bell and LaPadula. *Information Processing Letters*, 20(2):67–70, February 1985.

[16] S. Mullender, G. van Rossum, A. Tanenbaum, R. van Renesse, and H. van Staveren. Amoeba a distributed operating system for the 1990s. *Computer*, pages 44–53, May 1990.

[17] National Computer Security Center. A guide to understanding covert channel analysis of trusted systems. Technical Report NCSC-TG-030 Version-1, National Security Agency, Fort George G. Meade, Maryland, 1993.

[18] R. Newman-Wolfe, C. Ramirez, H. Pelimuhandiram, M. Montes, M. Webb, and D. Wilson. A brief overview of the DCS distributed conferencing system. In *Proceedings of the USENIX 1991 Summer Usenix Conference*, pages 437–452, Nashville, Tennessee, June 1991.

[19] C. Pfleeger. *Security in Computing*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[20] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control: A multi-dimensional view. In *Tenth Annual Computer Security Applications Conference*, pages 54–62, Orlando, Florida, December 1994.

[21] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38–47, February 1996.

[22] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, August 1989.

[23] J. Schiller. Secure distributed computing. *Scientific American*, pages 72–76, November 1994.

[24] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the 1988 Winter USENIX Conference*, pages 191–202, Dallas, Texas, February 1988.

[25] S. Stepney and S. Lord. Formal specification of an access control system. *Software—Practice and Experience*, 17(9):575–593, September 1987.

[26] B. Tretick, M. Cornwell, C. Landwehr, R. Jacob, and J. Tschohl. User's manual for the secure military message system M2 prototype. Memorandum Report 5757, Naval Research Laboratory, Washington, D.C., March 1986.

[27] S. Wilson, N. Goodwin, E. Bersoff, and I. Thomas N.M. Military message experiment—vol. I executive summary. Technical Report NRL Rep. 4454, Naval Research Laboratory, Washington, D.C., March 1982. Available from the National Technical Information Service as report number: AD A112 789.

[28] J. Woodward. Applications for multilevel secure operating systems. In *Proceedings of the AFIPS 1979 National Computer Conference*, volume 48, pages 319–328, Reston, Virginia, June 4–7 1979.