Table of Contents

Who Should Read This Document? 1
Contents 2

Chapter 1 DTOS Introduction 1-1

DTOS Kernel 1-2
Security Enhanced Lites Server 1-3
Security Server 1-4
DTOS Prototype and FreeBSD 1-5
How DTOS Implements Security 1-6
What is Type Enforcement 1-8
Assurance Work on DTOS 1-9

Chapter 2 Installing DTOS 2-1

Chapter 3 Building Security Databases 3-1
Security Server Functions 3-3
Overview of Building a Security Database 3-5
Set of Security Permissions 3-8
Relationship of Subject/Object Pairs 3-11
MLS Implementation 3-15
Creating a New Domain 3-18
Adding/Removing Permissions for a Subject/Object Pair 3-19
Controlling Communications Between Domains 3-20
Creating a New Type 3-22
Adding a New User 3-24
Structure of the Security ID 3-25
File Services. 3-29
Network Services 3-29
Network Security Services 3-29
Security Association Database 3-29

Chapter 4 DTOS Demonstration Software 4-1
How to Install and Run the DTOS Demo 4-2
DTOS Demo Structure 4-5
Creating Trusted Applications 4-10
Creating or Deleting Permissions. 4-17
Where To Go From Here 4-22

Chapter 5 Recompiling DTOS. 5-1

Appendix A DTOS Release Materials A-1
DTOS Release Materials A-1

Appendix B Permissions B-1
DTOS Kernel Service Permissions B-2

Appendix C Getting Product Support C-1
DTOS Support Staff C-1

Bibliography

Preface

This *DTOS Prototype User's Guide* provides background concepts, procedures, and reference information needed for installing and using the DTOS prototype.

Who Should Read This Document?

The *DTOS Prototype User's Guide* is written for researchers operating in a government or academic capacity who will be using the DTOS prototype to perform secure systems research.

Contents

The *DTOS Prototype User's Guide* contains the following chapters and appendixes.

Chapter 1, DTOS Introduction

This chapter discusses the background and objectives of the DTOS prototype and gives an overview of how DTOS implements security.

For additional introductory information, please refer to the following document included with the DTOS Release Materials: *Providing Policy Control Over Object Operations in a Mach Based System* (refer to *Bibliography* item number 6).

Chapter 2, Installing DTOS

This chapter breaks down the DTOS installation process into several short procedures that start with downloading the DTOS Release Materials and end with having DTOS boot loaded and ready to use.

Chapter 3, Building Security Databases

This chapter explains the functions of the security server and provides information on the structure of the permissions set, the relationship of subject/object pairs, and the actions required for building and storing security databases.

Chapter 4, DTOS Demonstration Software

The DTOS Demonstration Software provides an example security policy implementation for a simulated hospital database. This chapter explains what the DTOS Demonstration Software is doing, how it preforms these functions, and how to run Demo. This chapter also explains concepts that you need to understand in order to build your own trusted applications. These concepts are illustrated by means of the DTOS Demo.

Chapter 5, Recompiling DTOS

This chapter describes how to prepare for recompiling DTOS, how to build the microkernel, and how to build the Security-Enhanced Lites Server.

Appendix A, DTOS Release Materials

This appendix lists all of the items included in the DTOS Release Materials, the support tools included in the release, and the sources for getting these items.

Appendix B, Permissions

This appendix lists of all permissions checked in DTOS. These permissions are fields found in the security server database files.

Appendix C, Getting Product Support

This appendix describes the product support that is available for the DTOS prototype and how to access this support.

Bibliography

The Bibliography provides a list of reference documents that will help you use the DTOS prototype.

1

DTOS Introduction

The Distributed Trusted Operating System (DTOS) program is an effort being carried out by the Secure Computing Corporation (SCC) for the Maryland Procurement Office under contract MDA904-93-C-4209. This effort is coordinated with the researchers at the Information Security Computer Science Research Division of the Department of Defense. This effort is directed toward making an experimental microkernel-based secure operating system generally available to support further research and development in a number of different aspects of secure distributed computing.

The prototype system described in this user manual consists of three primary components. A variant of CMU Mach Kernel(See the DTOS Kernel Interface Document *Bibliography* item number 4) a variant of the Lites Server and a Security Server. Throughout the rest of this document the modified Mach Kernel is referred to as the DTOS Kernel, and the modified Lites Server is referred to as the Security Enhanced Lites or just Lites. The complete prototype described in this manual is referred to as the DTOS Prototype.

The complete set of release materials includes additional software and documentation intended to help other researchers to better understand and make use of the DTOS Prototype. The list of additional material includes:

- An example security policy, using Secure Computing Corporation's Type Enforcement, which can be extended to define control over new applications,
- A simple demonstration secure database system that demonstrate the key features of fine grained control provided by the DTOS Prototype,
- Design documentation describing the modifications made to the Mach Kernel to produce the DTOS Kernel,
- Formal specifications and other documentation describing the assurance analysis that has been done on the DTOS Kernel, and
- Software tools required to build and extend the DTOS Kernel, Security Enhanced Lites Server, Security Server and the security policy enforced by the Security Server.

The remainder of this introduction provides a further overview of the major elements of the DTOS Prototype and the relationship of the DTOS Prototype to the FreeBSD environment required to make a fully operational prototype system.

DTOS Kernel

The DTOS Kernel provided in the DTOS Prototype is based on the CMU MK83a Mach Microkernel. The kernel was modified to provide policy directed control over all kernel provided services. The primary objectives that guided the work resulting in the DTOS Kernel included:

- Provide policy-based control over all Mach services,
- Minimize the impact of security enforcement on system performance,
- Maintain compatibility with the existing Mach interface,
- Support enforcement of a wide range of security policies including dynamic and adaptive policies, and

- Provide a platform on which to experiment with a range of secure applications

The changes to the kernel were of one of the following four general types,

1. Introduction of logic to perform the security checks specified in the prototype system's Formal Security Policy Model (refer to *Bibliography* item number 1),

2. Extension of the existing Mach Kernel client visible interface to make security relevant information visible to security conscious applications,

3. Addition of a new kernel interface defining the required interactions between the DTOS Kernel and the Security Server, and

4. Introduction of a cache to hold recent security policy decision information to address potential performance problems that would arise from extensive interaction between the DTOS Kernel and the Security Server.

It is important to note that though the DTOS kernel provides control over all operations on DTOS Kernel objects, it does so without knowledge of any details about the policy being enforced. More information on the nature of the DTOS Kernel is provided in the paper titled "Providing Policy Control Over Object Operations in a Mach Based System" (refer to *Bibliography* item number 6).

Security Enhanced Lites Server

The Security Enhanced Lites Server provided in the DTOS prototype was produced by the researchers at the Information Security Computer Science Research Division of the Department of Defense. The Security Enhanced version of Lites provides a UNIX(TM) operating system environment which has been extended in a number of ways.

- The file system was extended to attach security policy relevant labels to all files in the system.
- The file system was also extended to enforce policy defined control over all operations on files.
- The UNIX interface was extended to allow security conscious applications and users to specify a security policy relevant label on a process.

This makes it possible for the DTOS prototype's UNIX environment to be isolated and control operations of UNIX applications in agreement with the system's security policy as provided by the Security Server. As with the DTOS Kernel, it is important to note that the Security Enhanced Lites Server only enforces the security decisions made in the Security Server.

Security Server

The Security Server element of the DTOS Prototype embodies a specific system security policy and provides security relevant information to other system elements on demand. The DTOS Kernel interacts with the Security Server to obtain permission information between client tasks and objects not currently available in the DTOS kernel's cache of policy decision information. The Security Enhanced Lites Server interacts with the Security Server, as required, to obtain policy defined permission information relevant to the control of file access.

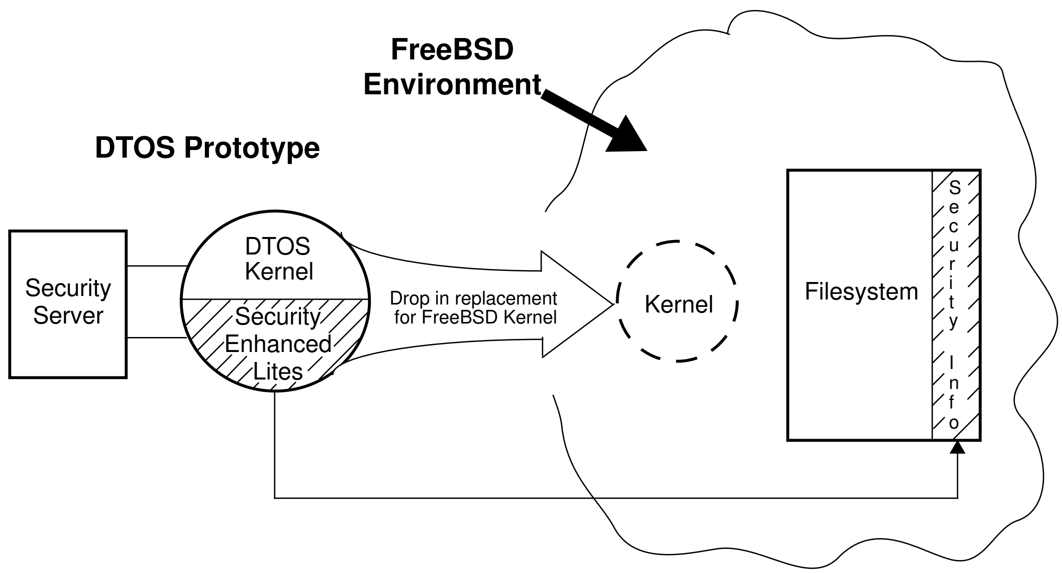
The Security Server provides services to arbitrary client tasks and system libraries to obtain policy specific information. It also provides a service to notify clients when changes are made to the policy (This feature is not yet implemented). This makes it possible for the DTOS Kernel and the Lites Server to cache permission information to address performance issue and still allow the DTOS prototype system to support dynamic changes to the policy.

DTOS Prototype and FreeBSD

In order for the DTOS Prototype to provide a full UNIX-like environment it must include the full UNIX utilities normally provided on a UNIX system. The DTOS prototype makes use of the UNIX environment provided by FreeBSD. This is the natural choice because the base Lites server used to develop the DTOS Prototype's Security Enhanced Lites Server expects a FreeBSD environment.

The following figure shows how the combined services of the DTOS Kernel and the Security Enhanced Lites Server replace the normal FreeBSD Kernel. It also shows that the Security Enhanced Lites Server makes use of the basic structures provided by the FreeBSD file system for definition and control of files. These structures are extended to support the binding of security label information to files as discussed above.

The figure also shows that the Security Server operates directly over the DTOS Kernel and is outside of the Security Enhanced Lites Server and FreeBSD UNIX environment.



Relationship of DTOS to FreeBSD

How DTOS Implements Security

The security framework that DTOS adds to the Mach microkernel is implemented by adding permission checks to all services provided by the microkernel. These permission checks consult information provided by a security server to resolve security policy decisions. That is, the higher level security policy maps down to specific permissions, and the enforcement of these permissions ensures proper system security.

The security server makes security decisions based on a set of rules (the security policy) and passes these decisions to the DTOS microkernel. It is the microkernel's job to enforce the security decisions it has received from the security server.

Moving policy decisions to a server that is external to the kernel allows the system to support more diverse security policies than is possible with a system that partially or completely implements security policy decisions within the kernel. Type Enforcement is the example security policy that is implemented in the DTOS prototype. But Type Enforcement is just one style of security policy that could be implemented using DTOS.

The DTOS prototype along with the secure filesystem that is built into Security-Enhanced Lites are a base on which you can prototype security policies and secure system components. But these elements, by themselves, are not a secure system. For example, the DTOS prototype contains security flaws and is missing some security relevant information such as authentication.

DTOS Prototype Execution Model

The DTOS prototype execution model requires that the label assigned to a file must either be a transition domain, or match the context of the domain in which it is being directed to execute. This requirement is enforced through permissions on memory regions. When Lites maps a file into memory for execution. It uses the label on the file to derive the label for the memory region. By default the only subject permitted to execute in a memory region, is the subject from which the label on memory region is derived. Thus when a subject attempts to execute a file labeled with a different domain, the subject will have no permission to the memory of the executable image. In the case that the file is labeled with a transition domain, the program will execute in the domain specified in the definition for the transition domain.

What is Type Enforcement

Type Enforcement is a style of security policy that provides a high level of security against malicious processes. A Type Enforcement policy defines the attributes of subjects and objects, and determines how subjects and objects may be related. That is, a Type Enforcement policy establishes the rules that govern whether or not the action that a subject is requesting to perform on an object should be permitted or not.

The default security policy provided by the DTOS prototype is a basic Type Enforcement model with several Trusted Computing Base (TCB) domains, a domain for the UNIX server, and a domain for all UNIX tasks. Multi-level security is also implemented.

Type Enforcement assures applications that the environment they are operating in has the following fundamental protections:

- **Compartmentalized Operations**

  Type Enforcement constrains access to data using domains and types. Users are assigned certain roles (such as database administrator, mail role, etc.), and each role is restricted to executing programs in specific execution domains. For example, a mail user is assigned a mail role and can only execute mail programs in the mail domain. Similarly, data is assigned specific types (for example, mail type) so that only programs in specific domains can access data of specific types. For example, mail programs can only access mail data.

- **Constrained Flows**

  Type Enforcement permits system designers to build assured pipelines. An assured pipeline is a series of operations that must be carried out in a specific order. Type Enforcement enforces this pipeline by constraining the flow of data from one domain to the next domain with a series of read/write restrictions on the data that is passed between domains.

- **Contained Executables**

  Contained executables refers to Type Enforcement's capability to control the execution of programs. Containment restricts how programs are installed, restricts who can execute programs, and restricts what the program can do and what resources it can access.

  Note: Restricting program execution is not implemented in this release of the DTOS prototype, since the underlying Mach kernel does not distinguish between read and execute. Code can be executed from any region that is readable regardless of whether execute access is granted or not.

For more information on Type Enforcement, refer to the Type Enforcement documents (number 9, 11, and 12) listed in the Bibliography.

Assurance Work on DTOS

Significant effort has been applied to the assurance work on DTOS. Assurance refers to using rigorous techniques to analyze the security of the system. The focus of the DTOS assurance effort is investigating techniques for assuring policy-flexible, microkernel-based systems, rather than providing complete analysis of DTOS itself. Consequently, DTOS is actually not a highly assured system.

For more information on the assurance work done on DTOS, refer to the *DTOS Formal Security Policy Model* (FSPM) and *Formal Top Level Specification* (FTLS) documents referenced in the Bibliography. Other assurance work such as specification-to-code correspondence analysis is in progress and could be made available at a later date.

2

Installing DTOS

This chapter is going to be removed. The information that was here previously can be found in the file /home/dtos/docs/INSTALLATION.

3

Building Security Databases

The DTOS security database is used by the security server to make security decisions supporting a user-specified security policy. This security policy determines what actions are allowed in the system according to the permissions set up in the security database. The security policy is also partially implemented in the security server code. So changes to the security policy may require changes to the security server code.

A security permission specifies that a subject running in a particular context can do something to an object of a certain context. For a simple example, if the subject is a UNIX process and the object is a file in the filesystem, the permission might specify that the subject is allowed to write to the file.

This chapter explains:

- The functions of the security server
- The overview of building a security database
- The structure of the permissions set
- The relationship of subject/object pairs
- How multi-level security (MLS) is implemented
- How to create a new domain and give it the permissions it needs to run as a UNIX process
- How to add or remove permissions for a subject/object pair
- How communication between domains is controlled
- How to create a new type
- How to add a user
- The structure of the security ID (SID)
- File permissions

Security Server Functions

The security server provides four services to other subjects operating on the system:

- Providing permissions for subject/object pairs
- Converting contexts (user representations of SIDs) to SIDs
- Converting SIDs to contexts
- Loading Security Databases

Permissions are computed based on subject SID (SSID)/object SID (OSID) pairs. Security IDs (SIDs) are involved in the handling of subject/object pairs. The Subject Security Identifier (SSID) is the security identifier that is tied to a subject. The object security identifier (OSID) is the security identifier that is tied to an object being acted on. Providing permissions involves computing a set of permissions based on an SSID/OSID pair that specifies which actions the subject is allowed to perform on the object. The set of permissions is also called an access vector.

Note: An object SID is also referred to as a target SID in some of the related DTOS reference documents. In this document, we use the term object SID. But please be aware that these two terms may be used interchangeably.

Security identifiers (SSIDs and OSIDs) are numeric values that represent a security context. A security context is a human-readable ASCII string that includes the names of all the security attributes associated with the subject or object. SIDs are not persistent; the SID associated with a context could be different each time the system is booted. The security-enhanced kernel calls take SIDs as parameters rather than contexts. To find the SID associated with a context, you must send a request to the security server asking for this information.

The final security server function is loading security databases. One of the key features of DTOS is that you can support multiple security policies by defining multiple security databases and easily change the current security policy by loading a new security database.

Installing a Private Security Server

The discussion of security database and security server functions in this chapter describes the security server provided as part of the DTOS prototype. But you can replace the provided security server with your own security server if you want to.

After you have created your new security server, you install it with the following copy command:

cp <*your security server binary*> /mach_servers/security_server

The copy command replaces the current security server in the /mach_servers directory with your private security server.

Files that Make Up a DTOS Security Database

The security server must read two files to perform its functions:

- **database_file**

  This file contains type enforcement information which is part of the system security database and includes the security attributes (domains, types, levels, categories, and users) that are the components of security contexts, and provides the information necessary to complete context/SID conversions. The database_file comes directly from the main.db file.

- **permissions_file**

  This file contains permissions for subject/object pairs. There are several.txt files that get concatenated and converted to machine-readable form. The set of all permissions that are defined by the concatenated .txt files is contained in the permissions_file. This set of security permissions is described in more detail in the following two topics.

Overview of Building a Security Database

In the previous topic we said that two files make up a security database: the database_file and the permissions_file. These two files are built from several user-modifiable source files. The database_file is built from the main.db file and the permissions_file is built from several .txt files. To build a security database, you must change or create the source files in your working directory and then install these changes to the database_file and the permissions_file. After you have built the security database, you have the option of loading it or saving it for future use.

The following overview procedure lists the main steps required to build a security database and load this new security policy.

1. **Modify the main.db file as needed to create (or delete) any domains, tasks, and users.**

   See the topics later in this chapter *Creating a New Domain*, *Creating a New Task*, and *Adding a User*.

2. **Modify the .txt files as needed to add (or remove) permissions for subject/object pairs.**

   See the topics later in this chapter *Set of Security Permissions*, *Relationship of Subject/Object Pairs*, *MLS Implementation*, *Adding or Removing Permissions*, and *Controlling Communications Between Domains*.

3. **Make and copy the security database files to the appropriate directory.**

   In the directory: **~dtos/build_area/src/mk/dss/security-database**

   Type the following:

   gmake
   cp main.db /mach_servers/database_file
   cp av.db /mach_servers/permissions_file

   The **gmake** command creates a new av.db file from the set of .txt files. This av.db file contains the permissions converted into bit strings (that is, into a format that is readable by the security server).

   The two copy (**cp**) commands replace the current security database files in the /mach_servers with the new, modified files. You can specify a directory other than /mach_servers, but you must use the file names database_file and permissions_file.

   Note: You can create and save as many security databases as you need, but you must store each in its own directory. You can then load the security policy of your choice (see the next step).

4. **(Optional) Load and use the new security policy.**

   From the command prompt or from a program, issue the following command: **sync;reload-policy *<directory>***

   where: <*directory*> specifies the directory that contains the base directory in which the security policy (database_file and permissions_file) reside. And **sync** causes all filesystem updates to be written to disk before the new policy is loaded.

   Note: The directory must be specified in the Mach microkernel device form. This means that you must specify the disk partition in the directory name. For example, if disk partition /dev/sd0e is mounted on /usr, and the policy is in directory /usr/policies/policy1, then the reload-policy command would be called like this:

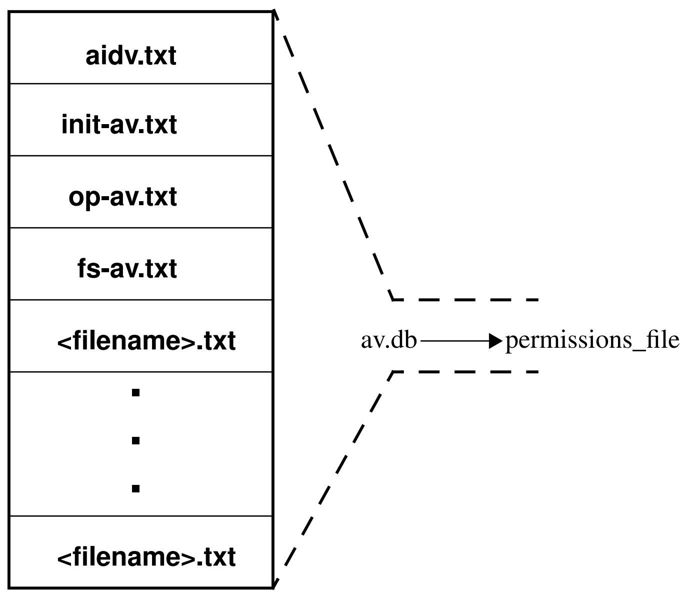
   sync;reload-policy /dev/sd0e/policies/policy1

The rest of this chapter gives you background and reference information for performing these tasks. In preparation for building security databases, you should also read the *DTOS Kernel and Security Server Software Design Document* (refer to *Bibliography* item number 3) and the *DTOS Demonstration Software Design Document* (refer to *Bibliography* item number 5).

Set of Security Permissions

Four types of .txt files contain the various security permissions that are linked together to build the permissions_file.

The following figure summarizes the set of all permissions that the security server accesses via the permissions_file.



(<filename> = user-specified file name)

Structure of the permissions_file

1. **aidv.txt**

   The aidv.txt file indicates which permissions are related to the authentication ID (AID). For each kernel managed objects, such as task_port or thread_port, a list of permissions is specified. For each of these permissions, the Security Server will verify that the authentication ID is consistent with other other security information.

2. **init-av.txt**

   The init-av.txt file represents initial permissions pre-loaded into the access vector cache (AVC) by kernel code. These permissions, or access vectors, are encountered before the system is initialized to the point that it can perform security server interactions. After the system is up and running, the kernel caches all access vectors received from the security server.

   The permissions contained in init-av.txt are listed in Appendix B.

3. **op-av.txt**

   This file contains the permissions that are needed to control normal system operations. This includes Kernel Subject, UNIX Subject, and Security Server Subject permissions.

   The permissions contained in op-av.txt are listed in Appendix B.

4. **fs-av.txt**

   This file contains the permissions that control interaction with the filesystem. The fs-av.txt file actually contains filesystem permission macros that expand into full file permissions.

   Additional information on file permissions is given later in this chapter.

5. **User-written .txt files**

   By creating additional .txt files, you can add application-specific access control rules. These user-written .txt files are concatenated into the permissions_file along with the standard permissions files (items 1-3 above). This allows the security server to know about and handle these user-specified permissions.

Note: If you need to add permissions to an SSID/OSID pair that already exists in one of the previous .txt files, then those files should be modified instead of creating a new .txt file. If the latter route is taken, the database file build process will detect duplicate SSID/OSID pairs and will not complete succesfully.

Important: If you create new .txt files, you must make two modifications to the Makefile file in your working directory to add information on these new files:

1.) Locate and change the "av_db =" line to add the names of the new files:
av_db = init-av.db op-av.db fs-av.db <filename>-av.db
where: <filename> is the user-specified filename of the source .txt file

2.) Add the following lines for each new .txt file. (The easiest way is to copy the fs-av entry and then replace **fs** with the user-specified file name **<filename>**:

```
<filename>-av.db: <filename>-av.m4 $(av_h)
<tab>calc-av.pl $(av_h) <filename>-av.m4 > <filename>-av.db

<filename>-av.m4: <filename>-av.txt database_macros.m4
<tab>m4 <filename>-av.txt > <filename>-av.m4
```

Additional information on creating and modifying .txt files is presented in sections: Adding/Removing Permissions for a Subject/Object Pair, and Setting up permissions examples.

Relationship of Subject/Object Pairs

The security permissions that are set up in the .txt files specify what subjects can do to objects. All of the specific permissions must be set up to define what a particular subject can do to a particular object.

Subjects and objects have security IDs (SIDs). An object security ID (OSID) can be derived from a subject security ID (SSID), or, in the case of root objects, an OSID can be unrelated to any subject.

The four kinds of security IDs are:

- Subject SID (SSID)
- SID related to subjects
- Root object SID
- Object SID related to root objects

(Please see the topic that follows, *An Example Subject/Object Relationship*, for more information on how the OSID is derived from the SSID.)

Each of the .txt file types contain permissions for multiple subjects. For example, init-av.txt defines permissions for the Kernel Subject, the Bootstrap Subject, Startup Subject, Unix Subject, and Security Server Subject.

The .txt file structure is further broken out by defining all of the objects that each subject is permitted to act on and what actions may be performed.

Finally, the set of permissions may be different depending on the MLS relationship between the subject and object (see the following topic, *MLS Implementation*, for more information).

An Example Subject/Object Relationship

As an example, let's look at the first subject/object relationship under the UNIX Subject in the init-av.txt file. (While going through this example, you should view the init-av.txt and main.db files from the DTOS Release Materials.)

The following lines in the file identify the subject/object relationship:

```
# Unix/unix task task_port
domain:Unix
type:unix_task_port
```

In the example above, **#Unix/unix task task_port** is a comment, **domain:Unix** indicates a subject running in the UNIX domain, and **type:unix_task_port** indicates an object (a task port) derived from a subject in the UNIX domain.

To fully understand how the OSID is derived from the SSID, it is also necessary to look at the statements that define the UNIX domain in the DOMAINS section of the main.db file:

```
LONG_NAME: Unix
SHORT_NAME: unix
ID:2
PRIVILEGE: 1
DONE
```

The **LONG_NAME: Unix** identifies the domain and corresponds to **domain:Unix** in the init-av.txt file. The **SHORT_NAME: unix** is used as the modifier for the type. (The **ID** is discussed later in this chapter under the topic *Creating a New Domain*.)

So, by looking again at the **type:unix_task_port** statement in the init-av.txt file, we see that **unix** represents the domain short name from which task_port is derived, and **task_port** represents the data type of the derived object.

This data type also determines which service permissions are defined for the particular SSID/OSID pair. In this example, the data type is task_port. Therefore, only task_port service permissions are appropriate. (Refer to *Appendix B* for additional information on service permissions grouped by data types.)

Now that the subject and object have been defined, We need to process some administrative details. The next parameter is the CACHE_FLAGS parameter. This allows the database composer to specify SSID/OSID pairs to be added or flushed from the the cache when this policy is loaded. The format is as follows:

CACHE_FLAGS:(level:cat[,cat,...][[aid]]-> level:cat[,cat,...][[aid]]=*cache_op*)

The first level:cat is the level and category of the subject sid pair, where the second is the level and category of the object. The *cache_op* is one of the following keywords **wired, load, clear, flush** and represents which cache operation to perform when loading the policy. The cache operations have the following meanings:

wired: wires the specified pair into the cache

load: adds the specified pair into the cache, but not wired

clear: removes the pair from cache, even if it wired

flush: flushes the pair from cache if it is not wired

The next parameter is the override vector. The override vector is used to specify which permissions, that are duplicated by an existing security mechanism, shall be overridden by the policy represented in this text file. For more details on how to use the override vector, consult the file ~dtos/build_area/src/lites_current/README.files, under the section on “File service override permissions”. The format of the override line, is the same as the perms line described later.

Next, the full list of permissions must be specified. These permissions are grouped under four dominance relationships, as discussed in the next topic *MLS Implementation*.

This building of subject/object pairs through the relationship of domain to type (that is SSID to OSID) is fundamental to modifying and creating .txt files in order to implement security the way you need it for your applications.

MLS Implementation

Multi-level security, or MLS, is basically implemented via four possible sets of permissions between subject/object pairs:

1. **dom_perms**

   These are the permissions granted when the subject level strictly dominates the object level. That is, the subject level is at a strictly higher security level than the object level.

2. **domby_perms**

   These are the permissions granted when the object level strictly dominates the subject level. That is, the subject level is at a strictly lower security level than the object level.

3. **eq_perms**

   These are the permissions granted when the subject level and the object level are equal. That is, the subject level is at the same security level as the object level.

4. **incomp_perms**

   These are the permissions granted when the subject level and the object level are incomparable. An example of incomparable is a subject and an object with the same security level but different compartments.

Permissions for subject/object pairs are specified in the .txt files in the following format:

```
domain:Unix
type:unix_reg_file
cache_flags:
override:

perms:av_can_send,fsv_truncate,fsv_visible,fsv_exec,fsv_write
perms:fsv_create,fsv_link,fsv_unlink,fsv_append
perms:fsv_read,fsv_chflags,fsv_chmod,fsv_chown
dom_perms:
domby_perms:-fsv_link,+fsv_visible
```

```
incomp_perms=fsv_exec,fsv_read
notify:fsv_exec,fsv_link
```

The perms lines list the permissions relevant to this domain/type pair. Each permission is assigned an MLS flow of read, write, neutral, or private as specified in the file mls_flow.txt. A read flow means that the eq_perms and dom_perms vectors can have this permission. A write flow means that the eq_perms and the domby_perms vectors can have this permission. A neutral flow means that the eq_perms, dom_perms and domby_perms vectors can have this permission. A private flow means that only the eq_perms vector can have this permission. The incomp_perms vector does not get any permission, by default. The dom_perms line is empty, indicating that it is not modifying the default MLS flow. It could have been left out in this case. The domby_perms line specifies that the permissions fsv_link must be removed and fsv_visible added to what the vector normally gets as determined by the MLS flow. The incomp_perms line indicates that this vector contains exactly the permission(s) listed, regardless of what the MLS flow says. Note that when you are completely overriding MLS, the '='is used and the permissions are not prefixed with a '+/-'.

The notify line specifies the permissions that go into the 4 notification vectors. As with permission vectors, there are 4 notification vectors, corresponding to the 4 MLS relationships. Whenever a subject causes a permission check to go off and the corresponding bit in the notification vector is set, the Security Server will be notified. This feature allows history-based security policies to keep track of which permissions have been used. By default, all 4 notification vectors will contain the permission(s) listed on the notify line. However, when given the -f option, the notification vectors will also follow the read/write flow. This is accomplished by setting the variable ntfy_option in the Makefile to -f.

The 4 permission vectors and the 4 notify vectors generated from the above example, assuming the option -f was specified, are as follows:

```
eq_perms:av_can_send,fsv_truncate,fsv_visible,fsv_exec,fsv_write
eq_perms:fsv_create,fsv_link,fsv_unlink,fsv_append
eq_perms:fsv_read,fsv_chflags,fsv_chmod,fsv_chown
dom_perms:av_can_send,fsv_visible, fsv_exec,fsv_read
domby_perms:av_can_send,fsv_create,fsv_unlink,fsv_append
```

```
domby_perms:fsv_truncate,fsv_visible,fsv_write
domby_perms:fsv_chflags,fsv_chmod
incomp_perms:fsv_stat,fsv_read
eq_notify:fsv_exec,fsv_link
dom_notify:fsv_exec
domby_notify:fsv_link
incomp_notify:
```

This example uses the following MLS flow definition

Permission	MLS Flow
av_can_send	neutral
fsv_create	write
fsv_link	write
fsv_unlink	write
fsv_append	write
fsv_truncate	write
fsv_visible	read
fsv_exec	read
fsv_write	write
fsv_read	read
fsv_chflags	write
fsv_chmod	write
fsv_chown	private

Please refer to Appendix B for a summary of the permissions defined in DTOS.

Creating a New Domain

If you want to create a new domain on the system, database modifications are required: the new domain with all of its object permissions must be created as well as any subject-to-subject permissions that you need to create (for example, for the kernel to be able to read the new domain's task port). The unix_proc macro automates creating the new domain and its required interactions with the UNIX server.

This procedure creates a new domain and gives it the permissions it needs to run as a UNIX process.

1. **Modify the main.db file to add the name and ID information for the new domain.**

   Open the main.db file using the editor of your choice. Locate the **SECTION: DOMAINS** portion of the file. You must enter a LONG_NAME, SHORT_NAME, ID and PRIVILEGE for the new domain. (Refer to the topic *Relationship of Subject/Object Pairs* for more information.) Make sure that the LONG_NAME, SHORT_NAME, and ID number you specify for the new domain are not already assigned to another domain. If this is privileged domain then, set the PRIVILEGE field to 1, otherwise it should be 0.

   For our example in this procedure, let's add the following lines to main.db:

   LONG_NAME: My_appl
   SHORT_NAME: mine
   ID: 50
   PRIVILEGE: 0

   DONE

2. **Modify the op-av.txt file to add the unix_proc macro to the file. (Optionally, you could make this change to a user-created .txt file.)**

   unix_proc (Unix,unix,My_appl,mine)

   In the unix_proc statement, **Unix,unix** is always the same and specifies the UNIX server. **My_appl,mine** specifies the new domain (in our example) that needs the permissions required to run as a UNIX process. This LONG_NAME/SHORT_NAME combination would match whatever you specified for the new domain in the main.db file.

3. **Make and copy the database files to the appropriate place**

   In the directory: **~dtos/build_area/src/mk/dss/security-database**

   Type the following:

   gmake
   cp main.db /mach_servers/policies/base/contexts.db
   cp av.db /mach_servers/policies/base/permissions.db

   The **gmake** command creates a new av.db file. The two copy (**cp**) commands replace the current security policy with the new, modified policy.

   Note: The database changes do not become effective until you reload the security policy. Refer to the topic *Overview of Building a Security Database* earlier in this chapter.

Adding/Removing Permissions for a Subject/Object Pair

When you use the unix_proc macro to create a new domain, it adds a default set of permissions for that domain. For your application security purposes, it may be necessary to modify these default permissions by adding or removing permissions.

To modify permissions, you must locate the .txt file that contains the subject/object permissions you want to change. You can then remove an existing permission or add a new permission.

After modifying subject/object permissions, be sure to run gmake to create a new av.db file and then copy av.db to the permissions_file.

Controlling Communications Between Domains

In order for domains to communicate, they must explicitly be given the privilege to talk to each other. When you create a new domain in the system, it is necessary to determine what other domain(s) the new domain needs to communicate with and create the appropriate port permissions to allow this communication.

Earlier, we discussed using the unix_proc macro to create default permissions for a new domain. When the unix_proc macro runs, it automatically defines several sets of permissions between the new domain and the UNIX server domain. However, if you want to create two new domains and have them be able to talk to each other, you must manually set up the permissions required to provide appropriate security for this inter-domain communication. First, use the procedure described under *Creating a New Domain* to create both of the new domains with the permissions required to communicate with the UNIX server domain. Then edit the appropriate .txt files to set up all of the permissions required for the two new domains to communicate with each other. The following examples will help you get started.

Setting Up Permissions Examples

For these examples, let's assume that you want Domain1 to be able to communicate with Domain2 through PortX. A port can have many senders but only one receiver. And communication through a port only flows one direction. So the way to allow a subject running in Domain1 (Subject1) to communicate with a subject running in Domain2 (Subject2) is to make Subject1 a sender to PortX and Subject2 a receiver of PortX.

In the first example, PortX is set up as a port derived from Domain2. The permissions may be set up as follows:

<u>Domain 1 .txt Setup</u>	<u>Domain 2 .txt Setup</u>
domain: Domain1	domain: Domain2
type: domain2_port	type: domain2_port
eq_perms:	eq_perms:
av_hold_send,	av_hold_receive,
av_can_send	av_can_receive

In the next example, PortX is set up as a root object that is unrelated to any subject. The permissions may be set up as follows:

<u>Domain 1 .txt Setup</u>	<u>Domain 2 .txt Setup</u>
domain: Domain1	domain: Domain2
type: any_name	type: any_name
eq_perms:	eq_perms:
av_hold_send,	av_hold_receive,
av_can_send	av_can_receive

The DTOS Demonstration System provides a good example of communication between domains. It features a Database Server managing a database of patients' records and interacting with the Security Server and several clients that are running in different domains. Refer to the *DTOS Demonstration Software Design Document* for additional information. Also, look at the Demo database for examples of how to get domains to interact.

The following directory holds the .db and .txt files for the DTOS Demonstration System:

/home/dtos/build_area/src/applications/demo/security-database

Creating a New Type

Root objects, which are not derived from a subject's domain, need to be explicitly defined in the security database. This means they must be added to the file main.db. Examples of root objects would be files, devices, or ports that you want to label with specific SIDs.

To create a new type, do the following:

1. **Modify the main.db file to add the name and ID information for the new type**

   Open the main.db file using the editor of your choice. Locate the **SECTION: TYPES** portion of the file. You must enter a LONG_NAME, SHORT_NAME, and ID for the new type. The LONG_NAME and SHORT_NAME can be any names you want. Make sure that the ID number you specify for the new type is not already assigned to another type. TRANSITION_DOMAIN is the domain that a subject will transition to as a result of executing an object of type LONG_NAME.

   As an example, you could add the following lines to main.db for a new root type:

   LONG_NAME: device_xx {must be a unique name}
   SHORT_NAME: dx {must be a unique name}
   ID: nnn {must be unique number}
   TRANSITION_DOMAIN: domain_name

   SERVICES:
   PERMISSIONS:
   DONE

   The SERVICES: and PERMISSIONS: are not currently used.

2. **Make and copy the database files to the appropriate place**

In the directory: **~dtos/build_area/src/mk/dss/security-database**

Type the following:

> **gmake**
> **cp main.db /mach_servers/policies/base/contexts.db**
> **cp av.db /mach_servers/policies/base/permissions.db**

The **gmake** command creates a new av.db file.

The two copy (**cp**) commands replace the current security database files in the /mach_servers directory with the new, modified files. You can specify a directory other than /mach_servers, but you must use the file names database_file and permissions_file.

Note: The database changes do not become effective until you reload the security policy. Refer to the topic *Overview of Building a Security Database* earlier in this chapter.

Adding a New User

To add a new user, you specify the user's name, ID, the domains that he or she can be in, and the list of security levels that he or she can operate at. To create a new user, do the following:

1. **Modify the main.db file to add the name, ID, domains, and security levels information for the new user**

   Open the main.db file using the editor of your choice. Locate the **SECTION: USERS** portion of the file. You must enter NAME, ID, DOMAINS, and LEVEL information for the new user. The user NAME and ID must match the user name and user ID set up in the password file (/etc/passwd). Also specify the domains and security levels allowed for the new user.

   As an example, you could add the following lines to main.db for a new user:

   NAME: j_smith {must match user name in password file}
   ID: 10 {must match user ID in password file}
   DOMAINS: Unix, user, security, nodomain
   LEVELS: unclassified: none; confidential: nato; secret: nato,noforn
   DONE

2. **Copy main.db file to the appropriate place**

   In the directory: **~dtos/build_area/src/mk/dss/security-database**

   Type the following:

   cp main.db /mach_servers/database_file

   You can specify a directory other than /mach_servers, but you must use the file name database_file.

   Note: The database changes do not become effective until you reload the security policy. Refer to the topic *Overview of Building a Security Database* earlier in this chapter.

Structure of the Security ID

There are four kinds of security IDs (SIDs):

1. **Subject SID (SSID)**

   These SIDs are associated with subjects.

2. **SID related to subjects**

   These SIDs are derived from subject SIDs.

3. **Root Object SID**

   These SIDs are associated with root objects which are not related to any subject.

4. **Object SID related to root objects**

   These SIDs are associated with objects which are related to stand-alone root objects.

Security Contexts

As we said earlier in this chapter, security contexts represent human readable security IDs. There is a security context format corresponding to each of the four kinds of security IDs.

Security contexts can contain the following four fields:

1. **Domain/Type**

   If this is a subject security context, then this field defines the domain associated with the subject (that is, the domain the user is running in). However, for an object security context (derived object, root object, or object related to a root object), this field contains the type associated with the object.

2. **Level**

   This is the security level associated with a subject or an object. Level is sometimes referred to as base level. Level and Categories together define the full security level.

3. **Categories**

   Categories describe the compartments of the associated security level. You can specify up to 11 categories in a security context.

4. **Classifier**

   This field is present only in security contexts of derived objects.

The following security context formats correspond to each of the four kinds of security IDs. The values that can be specified for the fields in these contexts are the values available in the main.db file (except for the classifier field for derived object contexts).

Security Context for Subjects

"Domain:Level:Category 1, Category 2, . . . , Category N"

Security Context for Objects Related to Subjects (Derived Objects)

"Domain:Level:Category 1, Category 2, . . . , Category N:Classifier"

Security Context for Root Objects

"Type:Level:Category 1, Category 2, . . . , Category N"

Security Context for Objects Related to Root Objects

"Type:Level:Category 1, Category 2, . . . , Category N:Classifier"

Note: Derived subjects/objects can have short security contexts where the Classifier field is omitted altogether.

The following points are important for the context formats shown above:

- Subject contexts have can have no Classifier field or one which indicates the kernel operating on the behalf of another subject (kern_derived_sid)
- Root object contexts have a Type field for which possible values are the TYPES listed in the main.db file.
- The valid values for domains, levels and categories are also listed in the main.db file.
- Derived object contexts have a Classifier field whose valid string values can be found in ~dtos/build_area/src/mk/dss/server-scc/ss_functions.c, in the array type_lookup_p. The corresponding numerical classifier values are defined in ~dtos/build_area/src/mk/kernel/sys/security_private.h

Obtaining a User-Specified SID

Using the context information above, you should obtain a SID from the security server. After these SIDs are obtained, you can supply them as arguments to calls such as task_create_secure or mach_port_allocate_secure.

To do this, you must know your user security context, you must create the desired context using one of the context formats shown above, and you must then convert the security context to a security ID using the function SSI_context_to_mid.

However, for some servers, such as the file system, the security context may be passed directly to the interface calls(e.g. mkdir_secure), thus bypassing the conversion stop

For additional information on the structure of the security ID, refer to Section 3.2.5, Security ID, in the *DTOS Kernel and Security Server Software Design Document*.

File Services

Please consult the file ~dtos/build_area/src/lites_current/README.files.

Network Services

Please consult the file
~dtos/build_area/src/lites_current/README.network

Network Security Services

Please consult the file ~dtos/build_area/src/apps/netsecurity/README

Security Association Database

Please consult the file ~dtos/build_area/src/apps/sadb/README

4

DTOS Demonstration Software

The Demonstration Software included with the DTOS prototype simulates a hospital database. In this database different classes of hospital staff have access to different fields within patient records. For example, a nurse and a doctor may each be able to read the medical prescriptions for a patient, but only the doctor is permitted to write a prescription. Refer to the *DTOS Demonstration Software Design Document* (*Bibliography* item number 5) for detailed demo scope and design information.

This chapter explains how to install and run the DTOS Demo, gives an overview of the DTOS Demo System structure, and provides background information useful for creating your own TCB extensions programs that take advantage of the security provided by the DTOS prototype.

How to Install and Run the DTOS Demo

When you ran the *Installing DTOS* procedures described in Chapter 2, the DTOS Demo software was installed as part of the DTOS prototype installation. However, there are some additional things you need to do before running the Demo.

Build the Demonstration Package

To build the Demonstration package, follow these steps:

1. **Change the current directory to the DTOS demo directory.**

   To change your current directory, type:

   cd ~dtos/build_area/src/applications/demo

2. **Build the DTOS Demonstration**

   Type: **gmake**

Make and Copy the Demo Database Files

Follow these steps to make the Demo security database files and copy these files to the appropriate place.

1. **Remove the old binaries.**

   Change your current directory to the demo directory, if you are not already in this directory. To change your current directory, type:

   cd ~dtos/build_area/src/applications/demo

   To remove the old binaries, type:

   gmake clean

2. **Rebuild the binaries.**

   To change to the security-database directory under /demo, type:

   cd security-database

   To create av.db, type:

   gmake

3. **Copy the security database files to the appropriate directory.**

   In the directory:

   ~dtos/build_area/src/applications/demo/security-database

   Type the following:

   cp main.db /mach_servers/database_file
   cp av.db /mach_servers/permissions_file

Reload the Demo Security Policy

From the command prompt or from a program, issue the following command: **sync;reload-policy *<directory>***

where: *<directory>* specifies the directory that contains the security policy (database_file and permissions_file) you want to load (/mach_servers).

Note: The directory must be specified in the Mach microkernel device form. This means that you must specify the disk partition in the directory name. For example, if disk partition /dev/sd0e is mounted on /usr, and the database files are in directory /usr/mach_servers, then the reload-policy command would be called like this:

sync;reload-policy /dev/sd0e/mach_servers

Label the DTOS Demo Files

The demo files need to be labeled with the appropriate contexts so that they can be run in the desired domains.

1. **Change the current directory to the DTOS demo directory, if you are not already in this directory.**

   To change your current directory, type:

   cd ~dtos/build_area/src/applications/demo

2. **Label the demo files.**

   To label the files, type: g**make label**

Run the DTOS Demo

1. **Change the current directory to the DTOS demo directory, if you are not already in this directory.**

   To change your current directory, type:

   cd ~dtos/build_area/src/applications/demo/bin
   mkdir database
   cp ../sample_database/lea_veas database

2. **Run the demo_exec program to start the DTOS Demonstration.**

   Type: **demo_exec**

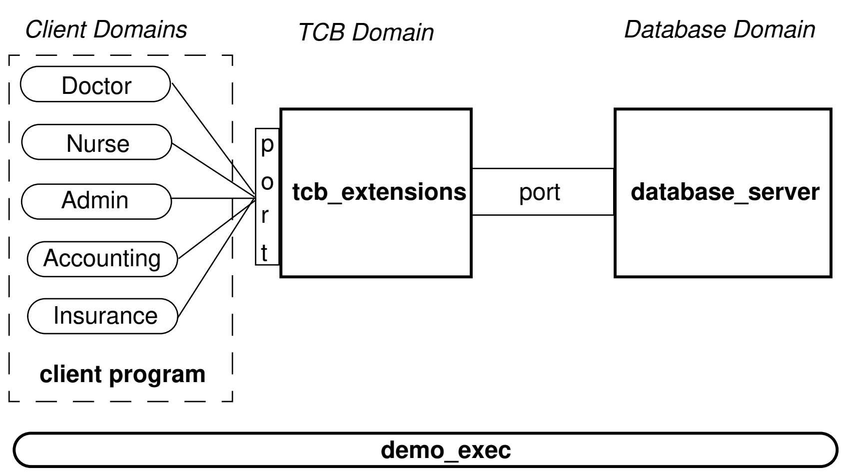
   The demo_exec program starts the database server, the tcb_extension and several copies of the client program which represent different client domains.

Refer to the file demo_howto for additional information on interacting with the Demo after it is running. The demo_howto file is located in the directory ~dtos/build_area/src/applications/demo/docs.

DTOS Demo Structure

The DTOS Demonstration Software emulates a hospital database where there are several classifications of people who need to access selected parts of patient records. Each of these database user classifications (client tasks) is assigned a SID. So each classification is a different domain and is allowed to request different data services. Whether or not access to services is allowed is determined by the relationship between the client's security context and the security context associated with the database server's service port.

The DTOS Demonstration Software consists of four programs that are depicted in the following figure and described below:



DTOS Demo Structure

1. database_server

This database_server task takes a patient's name and performs a read or a write operation of the entire database record for the patient with that name. This rudimentary database program creates a port with three services: read entry, create entry, and replace entry. No access permissions are checked within the database_server. However the only subject allowed by the security policyto send to the database_server port is the subject used by the tcb_extension. Therefore, service requests made by client tasks actually get carried out indirectly through the tcb_extension.

2. tcb_extension

This program provides the access controls to the database. The tcb_extension holds the send right to the port created by the database_server. It is the only subject that can send to the database_server port. So the tcb_extension takes requests from the client programs and performs database requests as needed.

The tcb_extension is where database related service checks take place. The port used by clients to communicate with the tcb_extension is a root object (that is, it has a defined SID assigned to it). Each client is permitted a different set of services on the port. The service checks for this port are performed within the tcb_extension according to an access vector supplied in the service request message. This access vector is based on the relationship between the client subject SID and the defined tcb_extension port SID. This access vector is checked against a mask of bits that correspond to the services provided by this port and represent each possible service that the tcb_extension can perform. These mask bits are defined by the DTOS Demonstration Software (refer to the topic *Creating TCB Extension Programs* later in this chapter).

3. client

This program is the user interface to the hospital database. The same program is used for all five classifications of users. That is, multiple instances of the same client program run in different domains. The domain in which this program runs determines which database operations are permitted. For example it can run in the doctor, nurse, admin, accounting, or insurance domains.

The client program's SID indicates the domain in which it is running. The client program requests a send right to the tcb_extension port in order to communicate with the database_server. The client program is able to send every possible request to the tcb_extension regardless of which domain it is running in. However due to service checks within the tcb_extension, only requests permitted for a given domain will succeed and be forwarded to the database_server task.

The client program may be started stand-alone or under the control of the demo_exec program.

4. demo_exec

The demo_exec program is provided to make the DTOS Demo easy to use. The demo_exec starts the other Demo programs in their proper domains and coordinates communications among these programs.

The demo_exec starts the database_server and the tcb_extension. It also starts one copy of the client program for each user classification. It then performs user interaction on behalf of each client.

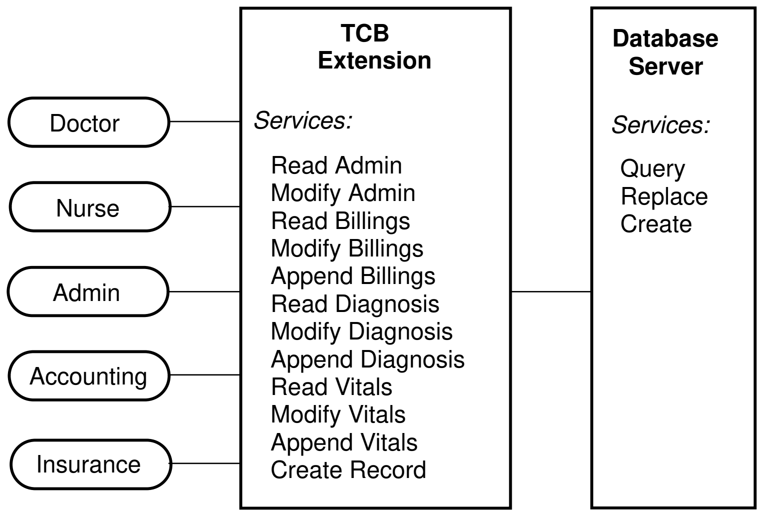
The demo_exec program runs in the default user domain.

Permitted Client Services

Different client domains are allowed different levels of access to patient database services. The following table shows the services that are allowed for each client domain:

Client Domain	Services Allowed
Doctor	Read Admin, Read Vitals, Modify Vitals, Append Vitals, Read Diagnosis, Modify Diagnosis, Append Diagnosis
Nurse	Read Admin, Read Vitals, Modify Vitals, Append Vitals, Read Diagnosis
Administrative	Create Record, Read Admin, Modify Admin, Read Billings
Accounting	Read Admin, Read Billings, Modify Billings, Append Billings
Insurance	Read Admin, Read Billings

Client requests for the services shown in the preceding table are submitted to the tcb_extension which provides the access controls to the database. The following figure shows the services available to client domains through the tcb_extension and database_server.



Services Available to Client Domains

Services in the TCB Extension are controlled by the policy, but the only controls on the services provided by the database server is access to the port.

The determination of whether of not a particular service is allowed for a given client domain is determined according to an access vector that the DTOS kernel inserts into the service request message. The following topic, *Creating TCB Extension Programs*, discusses this concept in detail.

Creating Trusted Applications

The conceptual background information presented here is intended to help you write your own trusted applications. The tcb_extension program supplied with the Demo takes advantage of the security features of the DTOS prototype to provide control over service requests in the Demo application. There are four pieces that together define the tcb_extension (located in ~dtos/build_area/src/applications/demo/db_tcb_extension):

mig/tcb_extension.defs — defines the services provided and the parameters for each service.

tcb_extensions.h — defines the format and meaning of access vectors supplied by the security server to the kernel, which in turn supplies an access vector to the tcb_extension for each service request. The access vector format must include the Interprocess Communications (IPC) part of the access vector in the proper place in the structure definition.

server.c — defines a structure that maps message IDs (service requests) to service permission bits in the access vector.

mach_message_secure_server.c — This routine performs all of the permission checks based on the information in the request message and the information in the table supplied by server.c.

We will only discuss the permission checking aspects of these pieces.

Examining the structure of the tcb_extension files that are part of the DTOS Demonstration Software will help you understand how service checks have been implemented in the Demo. This background should help you build your own secure applications. The approach described here is one way of doing service checks in an application, but it is not the only way. Please refer to the various files that are part of the Demo to supplement the discussion that follows.

Note: It will probably not be necessary for you to change the mach_message_secure_server routine, but you will probably need to add this entire routine to your application. This routine is discussed later in this chapter.

The Service Request Message

When a client requests a service, the Security Server calculates an access vector that specifies whether the requested action is permitted. The access vector computation is based on the subject SID of the requesting client and the object SID for the port to which the request is being sent. Refer to the main.db file that is part of the DTOS Demo materials for a complete list of Domains and Types (root objects). The Security Server passes the calculated access vector to the DTOS Kernel which inserts the access vector into the service request message along with the sender (subject) SID.

The following figure shows a partial service request message format.

Reply Port
Message ID
Sender ID *
Access Vector *
Patient Name
⋮

* Supplied by the DTOS Kernel

Service Request Message Format (partial)

In the partial service request message format shown here, the *Patient Name* is the field that the database keys off, the *Reply Port* is a derived object needed for a response, and the *Sender ID* is the SSID of the requesting client. But the fields of most importance to our discussion are the *Message ID* and the *Access Vector.*

<u>*Message ID*</u>

The Message ID is a number that corresponds to the service being requested. The tcb_extension checks (by means of mach_message_secure_server) to see what service is specified by the message ID and if the bit that corresponds to the requested service is set in the Services portion of the access vector. If there is a match, the service is permitted.

There is a Mach tool (mig) that generates code for messaging between clients and servers. You define services in a data file (<*name*>.defs) that is supplied to mig, and mig will then build message IDs to correspond to these services.

The contents of the <*name*>.defs file not only define available services, but also determine the numbering of message IDs (by the order that services appear in the .defs file). A portion of the file tcb_extension.defs is shown below as a sample. Please note that the first line in the file determines the base number for message IDs (in this case, 900000). Then the order in which services appear determines their message IDs. In our example, we show the first three (of twelve) services that are in the file. These three services have the following message IDs: create (900000), read_admin (900001), modify_admin (900002).

In the demo directory, you will find a subdirectory for each of the four main programs that make up the DTOS Demo. And each of these program directories has a /mig subdirectory. These /mig subdirectories contain the .defs input files that describe the client-server services as they are set up in the DTOS Demo. Please view one of these .defs files as a sample. The following example is from the file tcb_extension.defs that is located in the directory ~dtos/build_area/src/applications/demo/db_tcb_extension/mig.

```
subsystem Demo_Database_TCB_EXT 900000;

#include <mach/std_types.defs>
#include "../../include/demo.h"


type name_data_t  = array[*:MAX_DB_NAME_SIZE] of char;
type data_buffer_t   = array[*:MAX_DB_RECORD_SIZE] of char;


import "./tcb_extension.h";

userprefix TCB_;
serverprefix tcb_;

routine create(
        DBPort        : mach_port_t;
        Name          : name_data_t;
        data          : data_buffer_t;
    out     status  : int
);

routine read_admin(
        DBPort        : mach_port_t;
        Name          : name_data_t;
    out     data    : data_buffer_t
);

routine modify_admin(
        DBPort        : mach_port_t;
        Name          : name_data_t;
        old_record    : name_data_t;
        new_record    : name_data_t;
    out     status        : int
);
```

Note: For a service specified by the Message ID to be completed, the tcb_extension must also be able to hold a send right to the reply port specified in the service request message. As part of the message processing, the DTOS kernel checks if the tcb_extension can hold this send right.

Access Vector

As we stated earlier, when a client requests a service, the Security Server calculates an access vector that will control whether or not the requested action is permitted, and the DTOS Kernel inserts this access vector into the service request message.

This 64-bit access vector contains two fields: the IPC field and the Services field.

IPC	Services

Access Vector Format

The IPC, or Interprocess Communication, portion of the access vector contains the av permissions (such as av_hold_send, av_can_send, av_can_receive, etc) that control communication between the subject and the port. The DTOS Kernel performs permission checks based on the IPC portion of the access vector.

The Services portion of the access vector determines which database services are allowed for the subject domain. The tcb_extension performs permission checks based on the Services portion of the access vector. The Services field contains a bit for each of the tcb_extension services that were shown earlier under the topic *Permitted Client Services*. There is one access vector per domain/type pair which specifies all the services allowed for that domain.

This is a good time to open the file:
~dtos//build_area/src/applications/demo/security_database/demo_av.txt
and study its contents, if you have not done so already. This file specifies the IPC and Services permissions for all of the subjects (domains) that exist in the DTOS Demo.

To see where the permissions names used in the demo_av.txt file come from, view the contents of the tcb_extensions.h file (located in ~dtos/build_area/src/applications/demo/db_tcb_extension). This file contains the demo_services permissions that are available for specifying permissions in the demo_av.txt file. If you were to remove a demo_services permission from this file, that service would not longer be available to the DTOS Demo application. Conversely, you could also add new demo_services permissions to this file.

Checking Services Bits In the Access Vector

One final piece of service request processing to look at is the code that checks the service bits in the access vector against the contents of the message ID to determine if the service is permitted.

There is a standard library function, mach_message_server, that listens for messages on a port and then routes the contents of messages to appropriate routines for processing according to the message ID.

The DTOS prototype includes a secure version of this function (mach_message_secure_server) which resides in the /db_tcb_extension directory of the DTOS Demo:

~dtos/build_area/src/applications/demo/db_tcb_extensions

The mach_message_secure_server routine performs permission checking after receipt of a message according to the service permission/message ID map structure that is passed to it from server.c. A portion of the server.c file is shown below. This sample shows the data structure that provides the service mask bits which are compared to the access vector, and shows the data structure that is passed to mach_message_secure_server and used for permission checking after receipt of a message.

```
/*
 * This data structure provides a mask of what bits
 * Must be set in the access vector for this service
 * (create, read_admin, etc) to be allowed.
 * The access vector is obtained from the message where it was
 * left by the kernel after computing the callers sid to the
 * tcb_extension port sid.
 */

unsigned long long int permission_msgid_map[] = {
		0x10000ULL, /* create delete */
		0x20000ULL, /* read_admin */
		0x40000ULL, /* modify_admin */
		0x80000ULL, /* read_billings */
		0x100000ULL, /* modify_billings */
		0x200000ULL, /* append_billings */
		0x400000ULL, /* read_diagnosis */
		0x800000ULL, /* modify_diagnosis */
		0x1000000ULL, /* append diagnosis */
		0x2000000ULL, /* read_vitals */
		0x4000000ULL, /* modify_vitals */
		0x8000000ULL /* append_vitals */
};

/*
 * This data structure is a wrapper for the above structure and
 * is passed to mach_msg_secure_server to be used for permission
 * checking after receipt of a message.
 */

struct msg_id_to_av_mask_data_t msg_id_to_av_mask =
	{12, TCB_MSGID_BIAS, &permission_msgid_map[0]};
```

The service permission/message ID map structure includes a range of valid message IDs and an array of access vectors (one access vector per message ID). Each of the access vectors in the array has one or more service permission bits set. For a service request to be permitted, the bit that is set in the message ID to specify that service must have the corresponding service permission bit set in the access vector. When this is the case, mach_message_secure_server will permit the service requested by the subject.

Creating or Deleting Permissions

In *Chapter 3, Building Security Databases*, we discussed adding or removing permissions for a subject/object pair by modifying the contents of the appropriate .txt file. When you add or remove a permission for a subject/object pair, you are working with a permission that already exists. To complete the picture of working with permissions and building applications, we must also discuss how to create permissions. (To delete permissions, you follow the same process, but you remove the existing permissions that you no longer want to have available.)

There are two reasons for creating permissions:

1. You are creating a new security server.
2. You are adding an application that requires security controls.

For example, if you were to write a new file server, the permissions you created would determine if subjects can request a specific service of the file server.

To create permissions for a new security server or application, you must create/modify the header file to add the new permissions and then update the database so that these changes take effect.

Creating/Modifying the Header File

There is one header (.h) file per security server. This file defines access vectors and contains the permissions that the security server can use. You can give this .h file any name you want (<*filename*>.h). It is useful to refer to the tcb_extensions.h file in the DTOS Demo software when you are learning to create or modify a header file.

Earlier in this chapter we discussed the access vector format. We said that the access vector has an IPC portion and a Services portion. We also said that the IPC portion of the access vector contains permissions that control communication between the subject and the tcb_extension service port, and that the Services portion determines what services are allowed for the subject domain.

The following sample from the tcb_extensions.h file shows the structure that defines the services allowed in the DTOS Demo.

```
struct  demo_services
{

      unsigned char   demo_create_delete_record:1,
                      demo_read_admin:1,
                      demo_modify_admin:1,
                      demo_read_billings:1,
                      demo_modify_billings:1,
                      demo_append_billings:1,
                      demo_read_diagnosis:1,
                      demo_modify_diagnosis:1;

      unsigned char   demo_append_diagnosis:1,
                      demo_read_vitals:1,
                      demo_modify_vitals:1,
                      demo_append_vitals:1,
                      demo_pad:4;
      unsigned char   unused[4];
};
```

When you are building (or modifying) a header file, it must contain a structure similar to the sample above for the user-defined service permissions that you are creating for your application/security server.

This next sample (also from the tcb_extensions.h file) shows the structure that defines the IPC permissions in the DTOS Demo.

```
struct demo_access_vector
{
 /* IPC permissions.
    Must be the same as the IPC permissions defined in
    mk/kernel/sys/access_vector.h:struct mach_access_vector */
   unsigned char      av_can_receive: 1,
                      av_can_send: 1,
                      av_hold_receive: 1,
                      av_hold_send:1,
                      av_hold_send_once:1,
                      av_interpose: 1,
                      av_specify: 1,
                      av_transfer_ool: 1;
   unsigned char      av_transfer_receive: 1,
                      av_transfer_send: 1,
                      av_transfer_send_once: 1,
                      av_transfer_rights: 1,
                      av_unused: 4;
 /* Services permissions */
     struct  demo_services   demo_serv;
};
```

When you define the access vector permissions in the header file, this IPC portion must be included, and it must match the IPC permissions structure contained in the access_vector.h file. The access_vector.h file is located in the directory ~dtos/build_area/src/mk/kernel/sys.

Updating the Database

After you have created or modified the header file, you must make some modifications to the Makefile file in your working security_database directory in order for all required database updates to be processed when you run gmake.

The following example shows the modifications that would be necessary in the Makefile to add a new server (myserver) and a related set of new permissions defined in the file myserver_access_vector.h. These modifications are shown in boldface italic type. (You may also want to refer to the topic *Set of Security Permissions* in Chapter 3 where we previously discussed modifications to the Makefile required for creating new .txt files.)

```
#   Title: Makefile
#
#   Protection Notice :
#
#       This material may be reproduced by or for the
#       U.S. Government pursuant to the copyright license under the
#       clause at DFARS 252.227-7013 (Oct 1988).
#
#       (c) Copyright, 1995, Secure Computing Corporation. All Rights Reserved.
#
#
#   Purpose :
#
#       Controls the building of the system security database
#

mk_src = /home/dtos/build_area/src/mk
mk_db = $(mk_src)/dss/security-database
demo_src = /home/dtos/build_area/src/applications/demo

access_vector_h = $(mk_src)/kernel/sys/access_vector.h
av_to_perm_h = $(mk_src)/kernel/sys/av_to_perm.h
ss_access_vector_h = $(mk_src)/dss/server-scc/ss_access_vector.h
demo_av_h = $(demo_src)/db_tcb_extension/tcb_extensions.h
fs_av_h = $(mk_src)/dss/security-database/fs_access_vector.h
myserver_av_h = <some_path>/myserver_access_vector.h

av_db = init-av.db op-av.db fs-av.db myserver-av.db
av_h = av.h

target = av.db

all: links demo

links:
        rm -f init-av.txt ; ln -s $(mk_db)/init-av.txt .
        rm -f op-av.txt ; ln -s $(mk_db)/op-av.txt .
        rm -f fs-av.txt ; ln -s $(mk_db)/fs-av.txt .
        rm -f database_macros.m4 ; ln -s $(mk_db)/database_macros.m4 .
        rm -f header ; ln -s $(mk_db)/header .
        rm -f footer ; ln -s $(mk_db)/footer .

$(target): $(av_db) header footer
        cat header > $(target)
        cat $(av_db) >> $(target)
        cat footer >> $(target)
        dupe-finder.pl < av.db
```

```
demo: $(av_db) demo_av.db header footer
	cat header > $(target)
	cat $(av_db) >> $(target)
	cat demo_av.db >> $(target)
	cat footer >> $(target)
	dupe-finder.pl < av.db


init-av.db: init-av.m4 $(av_h)
	calc-av.pl $(av_h) init-av.m4 > init-av.db

init-av.m4: init-av.txt database_macros.m4
	m4 init-av.txt > init-av.m4


op-av.db: op-av.txt $(av_h)
	calc-av.pl $(av_h) op-av.txt > op-av.db


fs-av.db: fs-av.m4 $(av_h)
	calc-av.pl $(av_h) fs-av.m4 > fs-av.db

fs-av.m4: fs-av.txt database_macros.m4
	m4 fs-av.txt > fs-av.m4


demo_av.db: demo_av.m4 $(demo_av_h)
	calc-av.pl $(av_h) demo_av.m4 > demo_av.db

demo_av.m4: demo_av.txt database_macros.m4
	m4 demo_av.txt > demo_av.m4


myserver-av.db: myserver-av.m4 $(myserver_av_h)
	calc-av.pl $(av_h) myserver-av.m4 > myserver-av.db

myserver-av.m4: myserver-av.txt database_macros.m4
	m4 myserver-av.txt > myserver-av.m4


$(av_h): $(access_vector_h) $(ss_access_vector_h) $(demo_av_h) $(fs_av_h) $(myserver_av_h)
	cat $(access_vector_h) $(ss_access_vector_h) $(demo_av_h) $(fs_av_h) $(myserver_av_h)

clean:
	rm -f $(av_db) av.db demo_av.db init-av.m4 av.h init-av.txt op-av.txt
	rm -f *.m4 header footer fs-av.txt fs-av.m4 fs-av.db myserver-av.db
```

For additional information on defining services and determining message IDs by means of the tcb_extension.defs file, please refer to the topic *Message ID* earlier in this chapter.

Where To Go From Here

To help prepare for building your own secure applications, you should run the DTOS Demo, familiarize yourself with the programs and related source files that make up the Demo, and possibly even experiment with modifying the demo source files as a way to learn more about how these changes affect Demo operation.

If you have questions or need assistance, please contact the DTOS Support Staff (refer to *Appendix C* for instructions).

5

Recompiling DTOS

This chapter is going to be removed. The information that was here previously can be found in the file /home/dtos/docs/BUILDING.

A

Appendix A DTOS Release Materials

This appendix lists all of the items included in the DTOS Release Materials, the support tools included in the release, and the sources for getting these items.

DTOS Release Materials

The following binaries, sources, and documents are included in the DTOS prototype release. Which is available via sup from machpc.sctc.com. To retrieve the release materials perform the following steps

- ftp to machpc.sctc.com
- change directory /home/dtos/dtos-dist
- get the file README and follow the instructions..

Binaries

1. **gcc-2.7.2**

   Configured for host i386-freebsd2.1.0 and target i386-mach. Only "C" is supported. The source can be found at ftp://prep.ai.mit.edu/pub/gnu/gcc-2.7.2.tar.gz.

2. **gas-960319**

   Configured for host i386-freebsd2.1.0 and target i386-mach. This is the GAS assembler from GNU plus the binutils collection. Later snapshots will probably work also. The snapshots are available from ftp://ftp.cygnus.com/private/gas/*.

3. **gmake-3.74**

   Available from ftp://alpha.gnu.ai.mit.edu/make-<version>

4. **gawk-2.15.5**

   Available from ftp://prep.ai.mit.edu/pub/gnu/gawk-2.15.5.tar.gz

5. **perl-5.001**

   Available from ftp://ftp.cis.ufl.edu/pub/perl/src/5.0/perl5.001e.tar.gz

6. **zsh-2.5.0**

   Available from ftp://ftp.math.gatech.edu/pub/zsh/zsh-2.5.0.tar.gz

7. **i386_bnr collection from CMU.**

   This must be "supped" from CMU. The following line in a supfile will do the trick:

   ```
   mach3.release release=i386bnr host=x29.mach.cs.cmu.edu hostbase=/usr2 base=/ crypt=<???>
   ```

   Note that in the above text <???> is a crypt key obtained from CMU

8. **DTOS-related binaries**

   This includes the DTOS kernel and security server, security-enhanced Lites server and emulator, secure filesystem utilities and demo.

9. **tcsh-6.03**

   Available from ftp://tesla.ee.cornell.edu/pub/tcsh/tcsh-6.06.tar.gz

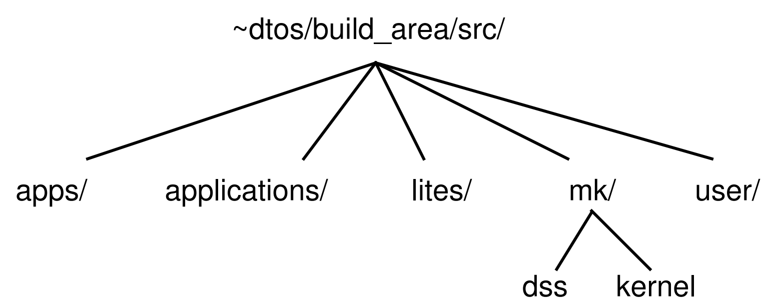
10. **m4-1.4**

    Available from ftp://prep.ai.mit.edu/pub/gnu/m4-1.4.tar.gz

Sources

1. **The DTOS microkernel, security server and utility programs/libraries**
2. **The security-enhanced Lites server/emulator and utilities**

Source Tree Diagram



Documents

1. ***DTOS Prototype User's Guide (this document)***
2. ***DTOS Kernel and Security Server Software Design Document***
3. ***DTOS Demonstration Software Design Document***
4. ***DTOS Kernel Interface Document***
5. ***DTOS Formal Security Policy Model (FSPM)***
6. ***DTOS Formal Top-Level Specification (FTLS)***
7. **1995 Usenix Unix Security Symposium Conference Paper: *Providing Policy Control Over Object Operations in a Mach Based System***

B

Appendix B Permissions

This appendix summarizes, by prefix, the permissions checked in DTOS. Permissions are fields found in the security server database files.

Refer to the *DTOS Kernel and Security Server Software Design Document* (*Bibliography* item number 3) for the actual permission names to append to the prefixes.

DTOS Kernel Service Permissions

The following table lists permission prefixes for DTOS kernel service permissions.

Service Checks Performed On	Prefix
Device Port	dsv
Host Privileged Port	hpsv
Host Port	hsv
Kernel Supplied Reply Ports	krpsv
Memory Objects Ports	mosv
Memory Control Ports	mcsv
Kernel Processor Port	psv
Kernel Processor Set Port	pssv
Kernel Task Port	tsv
Kernel Thread Port	thsv
All Message Processing (IPC Checks)	av

C

Appendix C Getting Product Support

This appendix describes the product support that is available for the DTOS prototype and how to access this support.

DTOS Support Staff

Please use the following two addresses to contact the DTOS Support Staff.

For Support Requests

Please use this mail address for DTOS prototype support requests:

dtos-request@sctc.com

Discussion Mailing List

The following mailing list is monitored by the DTOS Support Staff:

dtos-discuss@sctc.com

To subscribe to the mailing list, please send a request to:

dtos-request@sctc.com

To Submit Bug Reports/Enhancement Requests

Please use the following script to submit a bug report or enhancement request:

~dtos/tools/scripts/dtos-help

This script will prompt you for required information and will mail a correctly formatted message to dtos-support@sctc.com.

Bibliography

1. Secure Computing Corporation. *DTOS Formal Security Policy Model (FSPM)*. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, November 1994. DTOS CDRL A004. (Located in ~dtos/docs/assurance_documents/fspm-driver*.ps)

2. Secure Computing Corporation. *DTOS Formal Top-Level Specification (FTLS)*. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, December 1994. DTOS CDRL A005. (Located in ~dtos/docs/assurance_documents/ftls-27Dec94.ps)

3. Secure Computing Corporation. *DTOS Kernel and Security Server Software Design Document*. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, January 1994. DTOS CDRL A002.
(Located in ~dtos/docs/design_documents/kernel-secserver-sdd-Jul95.ps.gz)

4. Secure Computing Corporation. *DTOS Kernel Interfaces Document*. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, January 1994. DTOS CDRL A003.
(Located in ~dtos/docs/design_documents/kernel-interface-19Jul95.ps.gz)

5. Secure Computing Corporation. *DTOS Demonstration Software Design Document*. Technical report, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, January 1994.
(Located in ~dtos/docs/design_documents/demo-sdd-19Jul95.ps.gz)

6. Secure Computing Corporation. *Providing Policy Control Over Object Operations in a Mach Based System*. 1995 Usenix Unix Security Symposium Conference Paper, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113-2536, April 1995.
(Located in ~dtos/docs/papers/dtos-usenix-paper.ps.gz)

7. D.J. Thomsen. *The Security and Integrity of Type Enforcement and Set UID: Integrity Issues in Secure Systems*. Master's Thesis. May 1991.
(Located in ~dtos/docs/papers/int-issues.ps.gz)

8. D.J. Thomsen and J.T. Haigh. *A Comparison of Type Enforcement and Unix Set UID Implementation of Well Formed Transactions*. Proceedings of the 1990 Computer Security Applications Conference, pp. 304-312. 1990.
(Located in ~dtos/docs/papers/te-cw-setuid.ps.gz)

9. Todd Fine and Spencer E. Minear. *Assuring Distributed Trusted Mach*. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy. May 1993.
(Located in ~dtos/docs/papers/assuring_dtmach.ps.gz)

10. Keith Loepere. *OSF Mach Kernel Principles*. Open Software Foundation and Carnegie Mellon University, final draft edition, May 1993.
(Located in ~dtos/docs/papers/kernel-principles.ps.gz)

11. R. O'Brien and C. Rogers. *Developing Applications on LOCK*. Proceedings of the 14th National Computer Security Conference. NIST/NCSC, pp. 147-156, October 1991.
(Located in ~dtos/docs/papers/LOCK-apps.ps.gz)

12. D.J. Thomsen, Sidewinder: Enhanced Security for a Unix Firewall, ACSAC 1995 conference, June 1995.
(Located in ~dtos/docs/papers/sw-enhanced-security.ps.gz)