CA Access Control

Endpoint Administration Guide for UNIX

r12.5 SP1

Third-Party Notices

CONTAINS IBM(R) 32-bit Runtime Environment for AIX(TM), Java(TM) 2 Technology Edition, Version 1.4 Modules

CA Product References

This document references the following CA products:

- CA Access Control Premium Edition
- CA Access Control
- CA Single Sign-On (CA SSO)
- CA Top Secret®
- CA ACF2™
- CA Audit
- CA Network and Systems Management (CA NSM, formerly Unicenter NSM and Unicenter TNG)
- CA Software Delivery (formerly Unicenter Software Delivery)
- CA Service Desk Manager (formerly Unicenter Service Desk)
- CA Enterprise Log Manager
- CA Identity Manager

Documentation Conventions

The CA Access Control documentation uses the following conventions:

Format	Meaning
Mono-spaced font	Code or program output
Italic	Emphasis or a new term
Bold	Text that you must type exactly as shown
A forward slash (/)	Platform independent directory separator used to describe UNIX and Windows paths

The documentation also uses the following special conventions when explaining command syntax and user input (in a mono-spaced font):

Format	Meaning
Italic	Information that you must supply
Between square brackets ([])	Optional operands

Format	Meaning
Between braces ({})	Set of mandatory operands
Choices separated by pipe (|).	Separates alternative operands (choose one). For example, the following means *either* a user name *or* a group name: {*username*|*groupname*}
...	Indicates that the preceding item or group of items can be repeated
Underline	Default values
A backslash at end of line preceded by a space (\)	Sometimes a command does not fit on a single line in this guide. In these cases, a space followed by a backslash (\) at the end of a line indicates that the command continues on the following line. **Note:** Avoid copying the backslash character and omit the line break. These are not part of the actual command syntax.

Example: Command Notation Conventions

The following code illustrates how command conventions are used in this guide:

```
ruler className [props({all|{propertyName1[,propertyName2]...})]
```

In this example:

- The command name (ruler) is shown in regular mono-spaced font as it must be typed as shown.

- The *className* option is in italic as it is a placeholder for a class name (for example, USER).

- You can run the command without the second part enclosed in square brackets, which signifies optional operands.

- When using the optional parameter (props), you can choose the keyword *all* or, specify one or more property names separated by a comma.

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services

- Information about user communities and forums

- Product and documentation downloads

- CA Support policies and guidelines

- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short customer survey, which is also available on the CA Support website, found at http://ca.com/docs.

Documentation Changes

The following documentation updates have been made since the r12.5 release of this documentation:

- Transferring User Names (see page 217)—Updated topic corrects the name of the S50CREATE_Ldap_u.sh script.

- S50CREATE_Ldap_u (see page 217)—Updated topic corrects the name of the S50CREATE_Ldap_u.sh script.

The following chapters have been removed from this release:

- Password Synchronization with Mainframes—The information in this chapter is obsolete.

- Integrating CA Audit—The information in this chapter is obsolete. Information about how to integrate CA Access Control with CA Enterprise Log Manager is available in the *Implementation Guide*.

Contents

Chapter 8: Protecting Files and Programs   79

Chapter 9: Controlling Login Commands   99

Chapter 10: Protecting TCP/IP Services

Chapter 11: Managing Policy Models

Chapter 12: General Security Features

Chapter 15: Improving Performance   195

Chapter 16: Using UNIX Exits   209

Chapter 17: Interacting with LDAP   217

Chapter 1: Introduction

This section contains the following topics:

About this Guide

This guide describes the concepts used by CA Access Control for UNIX—a product that provides a total security solution for open systems. The guide describes UNIX endpoint management tasks and concepts.

This guide is also provided with CA Access Control Premium Edition, which offers enterprise management and reporting capabilities, and advanced policy management features.

To simplify terminology, we refer to the product as CA Access Control throughout this guide.

Who Should Use this Guide

This guide was written for security and system administrators who are implementing and maintaining a CA Access Control-protected environment.

Chapter 2: Managing Endpoints

CA Access Control is a software product that is an active, comprehensive security software solution for Open Systems, tied dynamically to the operating system. Each time a user requests a security-sensitive operation-such as opening a file, substituting a user ID, or obtaining a network service-CA Access Control can intercept the event in real time and evaluate its validity before passing control to the standard operating system (OS) functions.

This section contains the following topics:

What Is CA Access Control?

CA Access Control provides you with a powerful tool for managing security for your native platforms, making it possible to implement a security policy that can be customized entirely to an enterprise's security requirements. CA Access Control lets you provide security for users, groups, and resources beyond what is available in native operating systems. It lets you centrally manage security across the organization and integrate your Windows and UNIX security policies in a heterogeneous environment.

Why Does UNIX Need Protecting?

Many operating systems have built-in access control, using one technique or another. IBM's z/OS, a well-established and mature mainframe operating system, includes the System Authorization Facility (SAF)—a set of calls issued by the operating system itself to verify a user's authorization.

Access control software in an z/OS environment sets a return code for the SAF call and z/OS grants or denies access according to the code. The decision of what return code to set is based on the access rules and policies defined in the security database by the security administrator.

Other operating systems, such as OS/2, provide similar techniques for access control. The OS/2 access control module, called Security Enabling Services (SES), is based on the same concept as z/OS SAF.

Unfortunately, UNIX-based operating systems were not designed this way. Authorization decisions are made mainly for file accesses and are performed by the operating system itself using the nine bits (rwx-rwx-rwx) in the file's *inode* entry. Unlike SAF, no exit point for event interception is provided. Therefore, further security is necessary to perform functions that are more complex than those of mainframe-type security packages.

How Does This Work?

In addition to supplying the regular security functions-such as an access rule database, an audit log, and administration tools-CA Access Control intercepts the operating system events that are to be protected. Since CA Access Control has to work with many different operating systems, it intercepts events in memory. No changes are made to system files, and the operating system is not modified at all.

What Is Protected?

CA Access Control protects the following entities:

- **Files**

  Is a user authorized to access a particular file?

  CA Access Control restricts a user's ability to access a file. You can give a user one or more types of access, such as READ, WRITE, EXECUTE, DELETE, and RENAME. The access can be specified regarding an individual file or to a set of similarly named files.

- **Terminals**

  Is a user authorized to use a particular terminal?

  This check is done during the login process. Individual terminals and groups of terminals can be defined in the CA Access Control database, with access rules that state which users, or groups of users, are allowed to use the terminal or terminal group. Terminal protection ensures that no unauthorized terminal or station can be used to log into the accounts of powerfully authorized users.

- **Signon time**

  Is a user authorized to log on at a particular time on a particular day?

  Most users use their stations only on weekdays and only during work hours; the time-of-day and day-of-week login restrictions, as well as holiday restrictions, provide protection from hackers and from other unauthorized accessors.

- **TCP/IP**

  Is another station authorized to receive TCP/IP services from the local computer? Is another station authorized to supply TCP/IP services to the local computer? Is another station permitted to receive services from every user of the local station?

  The advantage of an open system-a system in which both the computers and the networks are open-is also a disadvantage. Once a computer is connected to the outside world, one can never be sure who enters the system and what damage an alien user may do, whether intentionally or by mistake. CA Access Control includes "firewalls" that prevent local stations and servers from providing services to unknown stations.

- **Multiple login privileges**

  Is the user permitted to log in from a second terminal?

  The term *concurrent logins* refers to a user's ability to be logged onto the system from more than one terminal. CA Access Control can prevent a user from logging in more than once. This prevents intruders from logging into the accounts of users who are already logged in.

- **User-defined entities**

  You can define and protect both regular entities (such as TCP/IP services and terminals) and functional entities (known as *abstract* objects; such as performing a transaction and accessing a record in a database).

- **Aspects of administrator authority**

  CA Access Control provides the means to both delegate superuser authorities to operators and restrict the privilege of the superuser account.

- **Substitute-user**

  Are users authorized to substitute their user IDs?

  The UNIX *setuid* system call, one of the most sensitive services provided by the operating system, is intercepted by CA Access Control to check whether the user is authorized to perform the substitution. The substitute-user authority check includes program pathing-users are permitted to substitute their user IDs only through specific programs. This is especially important in controlling who can substitute to root and thereby gain root access.

- **Substitute-group**

  Is a user authorized to issue the newgrp (substitute-group) command?

  Substitute-group protection is similar to substitute-user protection.

- **Setuid and setgid programs**

  Can a particular setuid or setgid program be trusted? Is the user authorized to invoke it?

  The security administrator can test programs that are marked as setuid or setgid executables to ensure that they do not contain any security loopholes that can be used to gain unauthorized access. Programs that pass the test and are considered safe are defined as trusted programs. The CA Access Control Self-Protection Module (also referred to as the CA Access Control *watchdog*) knows which program is in control at a particular time and checks whether the program has been modified or moved since it was classified as trusted. If a trusted program is modified or moved, the program is no longer considered trusted and CA Access Control does not allow it to run.

In addition, CA Access Control protects against various deliberate and accidental threats, including:

- **Kill attempts**

  CA Access Control can be used to protect critical servers and services or daemons against kill attempts.

- **Password Attack**

  CA Access Control protects against various types of password attacks, enforces the password-definition policies of your site, and detects break-in attempts.

- **Password Delinquency**

  CA Access Control policies delineate rules that force users to create and use passwords of sufficient quality. To ensure that users create and use acceptable passwords, CA Access Control can set maximum and minimum lifetimes for passwords, restrict certain words, prohibit repetitive characters, and enforce other restrictions. Passwords are not permitted to last too long.

- **Account Management**

  CA Access Control policies ensure that dormant accounts are dealt with appropriately.

- **Domain Management**

  CA Access Control can implement password protection and enforce security across NIS and non-NIS domains.

How Is It Protected?

CA Access Control starts immediately after the operating system finishes its initialization. CA Access Control places hooks in system services that must be protected. In this way, control is passed to CA Access Control before the service is performed. CA Access Control decides whether the service should be granted to the user.

For example, a user may attempt to access a resource protected by CA Access Control. This access request generates a system call to the kernel to open the resource. CA Access Control intercepts that system call and decides whether to grant access. If permission is granted, CA Access Control passes control to the regular system service; if CA Access Control denies permission, it returns the standard permission-denied error code to the program that activated the system call, and the system call ends.

The decision is based on access rules and policies that are defined in the database. The database describes two types of objects: accessors and resources. *Accessors* are users and groups. *Resources* are objects to be protected, such as files and services. Each record in the database describes an accessor or a resource.

Each object belongs to a class-a collection of objects of the same type. For example, TERMINAL is a class containing objects that are terminals (workstations) protected by CA Access Control.

Class Activation

CA Access Control stores information about whether a CLASS is active or inactive in the database. When CA Access Control starts, it passes a list of active classes to SEOS_syscall, so CA Access Control does not have to constantly intercept these classes. The only time CA Access Control intercepts a class is when a user changes the activity status of a class. If a class is inactive, access to the resource is not intercepted.

You can use the inactive class bypass with the following classes: FILE, HOST, TCP, CONNECT, and PROCESS.

Accessor Elements

Each user is represented by an *accessor element* (ACEE)-an in-memory reflection of the user's record in the database. CA Access Control builds the accessor element during the login process. The accessor element is associated with the user's process. Whenever the process requests a system service that is protected by CA Access Control, or issues an implicit request to access a resource, CA Access Control accesses the resource's record. It then determines whether the information in the previously created accessor element-such as the user's security level, mode, and group-lets the user access the resource.

Expanding Native Security

The following CA Access Control features expand native security.

Superuser Account Limitations

Users who administer and manage the operating systems are typically members of predefined accounts that are automatically created during system setup, such as the root account on UNIX systems, and the Administrator account on Windows systems. Each of the predefined accounts exists to perform a certain set of system functions.

Users acting as root or Administrator can perform a wide range of tasks, from creating, deleting, and modifying users to locking, reconfiguring, and shutting down servers.

One of the major security risks in these operating systems is that an unauthorized user can gain control of these accounts. If this happens, the user can cause enormous damage to the system.

CA Access Control lets you limit the rights granted to these accounts and to limit the rights of users who are members of the user groups that have these accounts as members. This reduces the vulnerability of your operating system.

CA Access Control Administrators

When you installed CA Access Control, you were asked to name one or more CA Access Control administrators. CA Access Control administrators have the authority to modify all or part of the rules database. You should have at least one full-authority administrator. This administrator can modify or create access rules freely and can designate other levels of administrators.

Once you have defined users for your system, you can assign administrative authority to other users by assigning the ADMIN attribute to them.

Note: A user with the ADMIN attribute possesses powerful authority. Consequently, the number of ADMIN users should be strictly limited. It is also a good policy to separate the roles of the native superuser and ADMIN, removing the ADMIN attribute from the superuser after you have set up one or more CA Access Control security administrators.

Because you always need at least one user with authority to manage the database, CA Access Control does not let you delete the last user that has the ADMIN attribute.

If you expect any of the CA Access Control administrators to be administering other hosts from this workstation, be sure that a rule in the database on that host gives them READ and WRITE access from this workstation.

Sub Administration

CA Access Control contains a *sub administration* feature. This lets administrators grant specific privileges that enable regular users to manage specific classes. These users are then called sub administrators.

For example, you can allow a specific user to manage users and groups only.

You can also specify a higher level of sub administration by granting access not only for specific classes, but for specified records in these classes.

Administration Rights for Regular Users

CA Access Control lets you grant ordinary users (that is, non-administrators) the necessary rights and privileges so that these users can perform administrative tasks without being members of the Administrators group. The ability to delegate tasks by granting administrative privileges in this granular way is a significant advantage of CA Access Control.

- A record in the SUDO class stores a command script to allow users to run the script with borrowed permissions.

- The data property value is the command script. This value can be modified by adding to it optional script parameter values.

- Each record in the SUDO class identifies a command for which a user can borrow permissions from another user.

- The key of the SUDO class record is the name of the SUDO record. This name is used instead of the command name when a user executes the commands in the SUDO record.

Program Pathing

Program pathing is an access rule associated with a file that requires that the file is accessed only through a specific program. Program pathing greatly increases the security of sensitive files. CA Access Control lets you use program pathing to provide additional protection for the files in your system.

B1 Security Level Certification

CA Access Control includes the following B1 "Orange Book" features: security levels, security categories, and security labels.

- Accessors and resources in the database can be assigned a *security level*. The security level is an integer between 1 and 255. An accessor can gain access to a resource only if the accessor has a security level equal to or greater than the security level assigned to the resource.

- Accessors and resources in the database can belong to one or more *security categories*. An accessor can access a resource only if the accessor belongs to all of the security categories assigned to the resource.

- A *security label* is a name that associates a particular security level with a set of zero or more security categories. Assigning a user to a security label gives the user both the security level and any security categories associated with the security label.

Note: For more information about B1 Orange Book features, see the *Implementation Guide*.

Endpoint Management

CA Access Control provides two ways to let you manage the resources in your enterprise and control who has access to them:

- **selang**—the CA Access Control command language.

  The selang command language lets you make definitions in the CA Access Control database. The selang command language is the command definition language.

  Note: For more information about using selang, see the *selang Reference Guide*.

- **CA Access Control Endpoint Management**—the endpoint administration interface.

  The web-based interface lets you administer remote endpoints through a central administration server.

  Note: For more information about installing CA Access Control Endpoint Management, see the *Implementation Guide*.

Chapter 3: Managing Users and Groups

This section contains the following topics:

Users and Groups

In CA Access Control, every action and access attempt is performed on behalf of a user, who is held responsible for submitting the request. Every process in the system is therefore associated with a certain user name. The user name identifies the user to CA Access Control.

A *user* is a person who can log on, or can be the owner of a batch or daemon program. In CA Access Control, every access attempt is performed by a user. CA Access Control can use user information from the CA Access Control database and from enterprise user stores. It stores user information in its database, in either a USER record or an XUSER record.

Note: An *enterprise user store* is a store in the operating system that stores users or groups, for example, /etc/passwd and /etc/groups on UNIX systems, or Active Directory on Windows.

A *group* is a collection of users. A group defines common access rules for users in the group. Groups can be nested (belong to other groups). CA Access Control can use group information from the CA Access Control database and from the enterprise user stores. Typically, you create groups and assign users to them, based on a role, for example, database_administrators.

The user records are the key accessor records. The main purpose for using groups in CA Access Control is to assign access authorities to all users in group at one time. Assigning access authorities at one time is easier and less error prone than assigning them separately to each user.

Where Information about Accessors Is Stored

The information that CA Access Control uses about users and groups is stored both in the CA Access Control database and in the host operating system. The host operating system information stores are called *enterprise user stores*, or just *enterprise stores*. By default, CA Access Control is configured so that it does not use the enterprise stores. You can, however configure CA Access Control so that if it cannot find a user or group defined in its database, it looks for, and uses the information from, the users and the group memberships defined in the enterprise stores.

Note: CA Access Control uses information from the enterprise stores but only writes to them if you use selang command in the native environment.

When checking for authorization, CA Access Control always checks for accessors defined in its own database before it checks the enterprise store: if you have an enterprise user with the same name as a user defined in the CA Access Control database, the enterprise user is ignored by CA Access Control.

How CA Access Control Finds a User Record

When a user logs in, CA Access Control conducts the search in the following order, until it finds a record associated with the user:

1. CA Access Control searches for a user defined in its database.

2. CA Access Control searches its cache for an enterprise user of that name.

   When the network is down, the operating system (OS) lets users log in using the OS cached credentials. The purpose of the CA Access Control cache is to let CA Access Control also use enterprise users' records in these cases.

3. CA Access Control uses the operating system to search the enterprise user stores for a user of that name.

4. If CA Access Control does not find a record associated with the user in its database or in the enterprise stores, CA Access Control assigns the user the attributes in the _undefined USER record.

Integration with the Enterprise User Stores

Typically, you configure CA Access Control to use the groups and users that are defined in the enterprise user stores.

If you do configure CA Access Control like this, by default, when an access rule that references an enterprise user or group is created, or when a user logs in to the operating system, CA Access Control creates a record in its database for that user or group, if one did not exist before. These records have the class XUSER (for enterprise users) or XGROUP (for enterprise groups). They hold the properties that CA Access Control requires to enforce access rules. You do not need to manage them, because CA Access Control creates them as required.

The only properties of an enterprise user or group that CA Access Control fetches from the enterprise user stores are the names and the group membership properties.

Guidelines for Managing Accessors in Enterprise Stores

If you decide to manage your accessors in enterprise user stores, you should consider the guidelines in the following sections.

Users and Groups that Must be Defined in the Database

CA Access Control needs some users and groups to be defined in its database, rather than in the enterprise user stores. These include:

- Predefined users (see page 35)
- Predefined groups (see page 35)
- A CA Access Control administrator
- Profile groups
- Logical users

Restrictions on the Use of Enterprise Users

CA Access Control imposes the following restrictions on the use of enterprise users:

- You cannot create, or refer to, an enterprise user in CA Access Control if it has the same name as a user defined in the database.
- You cannot create, delete or modify an enterprise user using the selang AC environment.

- You cannot use an enterprise user as a logical user.

- By default, you cannot create an enterprise user in CA Access Control unless the user is already defined in the enterprise user store. However, you can enable or disable this behavior on UNIX systems.

More information:

Enable or Disable Checking Enterprise Store before Creating XUSER Records on UNIX (see page 32)

Restrictions on the Use of Enterprise Groups

CA Access Control imposes the following restrictions on the use of enterprise groups:

- You cannot create or delete an enterprise group within the selang AC environment.

- You cannot change the membership of an enterprise group within the selang AC environment.

- You cannot use an enterprise group as a Profile Group (see page 36).

Enable or Disable the Use of Enterprise Users and Groups

By default, CA Access Control cannot use the groups and users defined in the enterprise user stores, but you can enable CA Access Control to do so. We recommend that you enable this feature unless you need compatibility with previous versions of CA Access Control.

To let CA Access Control use enterprise users and groups, set the configuration setting osuser_enabled to 1 (one). To disable this behavior, set the value of osuser_enabled to 0 (zero).

Example: Enable the Use of Enterprise Users and Groups on Windows

The following registry setting enables the use of enterprise users and groups on Windows:

- Key: HKLM\ComputerAssociates\AccessControl\OS_user

- Name: osuser_enabled

- Type: REG_DWORD

- Value: 1

Example: Enable the Use of Enterprise Users and Groups on UNIX

The following commands stop CA Access Control, enable the use of enterprise users and groups on UNIX, and restart CA Access Control:

```
secons -s
seini -s OS_User.osuser_enabled 1
seload
```

Enable or Disable the Creation of XUSER Records at Enterprise User Login

If CA Access Control is enabled to use enterprise users, by default it creates a record (in the XUSER class) for a user when that user logs in. Sometimes you do not want this, for example, if thousands of users log on at the same time each day.

To prevent CA Access Control creating XUSER records when users log in, change the value of the configuration setting create_user_in_db to 0 (zero). To re-enable this behavior set the value to 1 (one).

Example: Disable the Automatic Creation of XUSER Records on Enterprise User Login on Windows

The following registry setting disables the automatic creation of an enterprise user record in CA Access Control on Windows:

- Key: HKLM\Software\ComputerAssociates\AccessControl\OS_user

- Name: create_user_in_db

- Type: REG_DWORD

- Value: 0

Example: Disable the Automatic Creation of XUSER Records on Enterprise User Login on UNIX

The following commands stop CA Access Control, disable the automatic creation of a XUSER record on UNIX, and restart CA Access Control:

```
secons -s
seini -s OS_User.create_user_in_db 0
seload
```

Enable or Disable Checking Enterprise Store before Creating XUSER Records on UNIX

Sometimes you may want to create an enterprise user in CA Access Control when the user is not defined in the enterprise user store. On Windows you cannot create an enterprise user in CA Access Control unless the user exists in the Windows user store. On UNIX, the default behavior is the opposite to Windows. However, on UNIX, you can enable or disable this default behavior.

To disable checking (and therefore allow CA Access Control to create XUSER records when there is no enterprise user equivalent), change the value of the configuration setting verify_osuser to 0. To enforce checking, set the value to 1.

Example: Enable Creation of XUSER Records without Checking the Enterprise User Store

The following set of commands stops CA Access Control, enables the creation of XUSER records with no enterprise store equivalents, and restarts CA Access Control:

```
secons -s
seini -s OS_User.verify_osuser 0
seload
```

Recycled Enterprise Store Accounts on Windows

Recycled accounts are enterprise store users or groups that have been deleted and then recreated (using the same name). This is likely to happen when you remove a user from the user store (for example, when the user resigns) and then create a new account for a new user that has the same name as the old removed user.

Recycled accounts are a security concern because you do not necessarily want new accessors to have the same access permissions as those that were granted to the old account with the same name. To solve this problem, CA Access Control authorization is based on the SID. This means that when you create a new accessor, with the same name as a deleted accessor with existing access permissions, the new accessor does not automatically receive the old permissions of the old accessor.

Important! Recycled account accessors *do not* inherit the old access permissions. However, database access rules, which mention the accessor's name (not SID), may make it seem like these rules still apply. Use the secons -checkSID command to resolve this.

Resolve Recycled Enterprise Accounts on Windows

If an enterprise account (user or group) has associated database rules is then recycled (deleted and created with the same name), it may look like the old database rules still apply to the new account. However, as CA Access Control authorization is based on SID, these rules no longer apply and you need to create new rules for the new group. Before you can create the new rules, you have to resolve recycled accounts.

To resolve recycled enterprise accounts open a command prompt and run the following commands:

```
secons -checkSID -users
secons -checkSID -groups
```

CA Access Control works through all the enterprise user accounts it has (XUSER records) and then all the group accounts (XGROUP records) and identifies accounts with an SID that differs from the SID of the enterprise account. It renames these accounts in CA Access Control using the following naming convention: *SID* (*accountName*)

You can now create the new rules for the recycled account.

Note: Recycled user accounts are resolved in this way when the user logs in or tries to access a resource. We recommend that when you create an enterprise account, run the secons -checkSID command as a scheduled task.

Example: A Recycled Group Account

Company ABCD has a group called *interns* in its enterprise store. The group has nine members and they are working on productA. The administrator makes the group known to CA Access Control and assigns it with access permissions to the files group members need to access, as follows:

```
nxg interns owner(msmith)
auth file c:\products\productA\materials\* xgid(interns) access(all)
auth file c:\HR\interns\* xgid(interns) access(read)
```

When the interns complete their tenure with ABCD, the enterprise store administrator deletes the group. Three months later, a new group of interns with six members is created in the enterprise store, with the same name. The old rules in the CA Access Control database still exist so it seems like the new *interns* group inherited the permissions of the old group. However, these rules apply to the old interns group and the CA Access Control administrator needs to create new rules for the new group.

To do this, the administrator has to identify and resolve the recycled interns account, as follows:

```
secons -checkSID -groups interns
```

This renames the XGROUP resource, and any access rules references to it, to "*SID* (*domain*\interns)". Now, the administrator can create new rules for the new interns group that works on productB:

```
nxg interns owner(msmith)
auth file c:\products\productB\materials\* xgid(interns) access(all)
auth file c:\HR\interns\* xgid(interns) access(read)
```

Note: For more information on the secons utility, see the *Reference Guide*.

Database Accessors

Regardless of how you decide to manage your users, some accessors must be defined in the CA Access Control database, as described in the following sections.

Predefined Users

CA Access Control predefines the following users, which you cannot delete:

_dms

Installed on the advanced policy management server components' databases (DMS, DH reader, and DH writer), the _dms user is used by policyfetcher and devcalc to communicate with the DH and DMS.

_seagent

_seagent is the user name under which CA Access Control runs some internal processes, such as:

- The PMDB process, sepmdd
- (UNIX) The deviation calculation process, devcalc
- The user and group record update exit processes

The _seagent user has the SERVER attribute.

_undefined

_undefined represents all users that are undefined in CA Access Control. You can use _undefined to include undefined users in ACLs.

+devcalc

(Windows) The user name under which CA Access Control runs the deviation calculation process, devcalc.

+reportagent

The user name under which CA Access Control runs the Report Agent.

nobody

The nobody user is a user record that cannot correspond to a real user. Use this record to create rules that do not give any user the associated permissions. For example, you can set *nobody* as the owner of resources, meaning that no user will get the permissions associated with owning that record.

Predefined Groups

CA Access Control comes with predefined groups. Except for the _interactive and _network groups, you add users to these groups in the same way as you do for any other group.

_abspath

If a user is in the _abspath group when logging in, that user must use absolute path names to invoke programs.

_interactive

A user is a member of the _interactive group only for the purposes of an access attempt. Users are members of the _interactive group if they are logged into the same host as the resource they are trying to access. CA Access Control dynamically and automatically manages the membership of the _interactive group—you cannot change the membership.

_network

This is the complementary group to _interactive. A user is a member of the _network group for the purposes of access only. Users are members of the _network group if they are trying to access a resource from a different host than the resource belongs to. CA Access Control dynamically and automatically manages the membership of the _network group—you cannot change the membership.

_restricted

For users in the _restricted group, all files, and on Windows registry keys too, are protected by CA Access Control. If a file or a Windows registry key does not have an access rule explicitly defined, access permissions are covered by the _default record for that class (FILE or REGKEY).

Note: Users in the _restricted group may not have sufficient authorization to do their work. If you plan to add users to the _restricted group, consider using Warning mode initially.

_surrogate

When a user uses a member of the _surrogate group as a surrogate, CA Access Control writes a full trace in the audit trail of the surrogate's actions, tagged with the original user's name.

Example: Adding a User to the _restricted Group Using selang

The following selang command adds the enterprise user john_smith to the _restricted group:

```
joinx john_smith group(_restricted)
```

Profile Groups

A *profile group* is a group defined in the CA Access Control database that contains default values for user properties. When you assign a user to a profile group, the profile group provides those values to the user unless they have already been set for the user.

You can specify a profile group for a user when you create the user, or you can assign the user to the profile group afterwards.

Profile groups let administrators efficiently create a standard setup with specific permissions for any new user assigned to that group. This setup can specify such things as the home directory of the user, the audit properties, the PMDB that defines the access authorities, and various password rules affecting a user who is associated with a profile group.

How CA Access Control Uses Profile Groups to Determine User Properties

The following process describes how CA Access Control uses profile groups to determine user properties:

1. CA Access Control checks if the user's record in the USER or XUSER class has a value for the property.

   If the user's record has a value for the property, CA Access Control uses that value.

2. CA Access Control checks if the user is assigned to a profile group.

   If the user is assigned to a profile group, the process continues. If the user is not assigned to a profile group, CA Access Control assigns the default property value to the user.

3. CA Access Control checks if the profile group has a value for that property.

   If the profile group has a value for the property, CA Access Control assigns that value to the user. If the profile group does not have a value for the property, CA Access Control assigns the default property value to the user.

   Note: If the audit property of a user or profile group is not set, the audit property of a group can affect the audit property of a user.

More information:

How CA Access Control Determines the Audit Mode for a User (see page 167)

Accessor Management

You can create, modify, and delete database or enterprise user or group records by using CA Access Control Endpoint Management or by using selang.

Manage Users or Groups

If you want to view or modify the properties of a particular accessor, or if you want to delete an accessor, you must first find that accessor.

To manage users or groups

1. In CA Access Control Endpoint Management, do as follows:

   a. Click Users.

   b. Click either the Users *or* Groups subtab.

   Depending on your selection, the Users or the Groups page appears.

2. Complete the following fields in the Search section:

   User/Group Name

   Defines a mask for the accessors you want to find. You can enter the full name of the accessor you are after or you can use a mask. For example, use *admin* to list accessors whose name contains "admin".

   Use an * (asterisk) to list all accessors and a ? (question mark) to replace a single character.

   User/Group Repository

   Specifies the source from which you want to fetch a list of accessors. The source can be either:

   – **Internal Accounts**—accessors defined in the CA Access Control database.

   – **Enterprise Accounts**—accessors defined in specific enterprise user stores.

   Show only AC accounts/profiles

   Specifies whether to list only those accounts that have records in the CA Access Control database as follows:

   – If you chose Internal Accounts, the application lists only those accounts that exist in the CA Access Control database (no native accounts).

   – If you chose Enterprise Accounts, the application lists only those accounts that have a CA Access Control enterprise profile (XUSER or XGROUP records).

   Click Go.

   A list of accessors that exist in the repository you chose appears.

3. Do *one* of the following:

   ▪ Click 🔍 in the View column to view the properties of the accessor.

   ▪ Click 🗑 in the Delete column to delete the accessor.

   ▪ Click the name of the accessor to modify the properties of the accessor.

   ▪ Select the accessors you want to delete and click Delete.

   ▪ Click Create User or Create Group to create a user or group record in the CA Access Control database.

Example: Search for Enterprise Users in a Repository

The following graphic shows you the result of looking for all users in the ABC-DM1 enterprise user store.

User Management Using selang

Use the following selang commands for records of enterprise users:

- **newxusr** and **editxusr—**define a new enterprise user record
- **chxusr** and **editxusr—**change the CA Access Control properties of an enterprise user
- **find xuser**—list enterprise users that have a CA Access Control record
- **rmxusr—**delete a user
- **show xuser—**display the CA Access Control properties of an enterprise user

Use the following selang commands for CA Access Control database user records:

- **newusr** and **editusr—**define a new user record
- **chusr** and **editusr—**change the properties of a user
- **rmusr—**delete a user
- **find user—**list database users
- **show user—**display the properties of a user

Example: Define a User in the Database Using selang

The following selang command defines a new user in the CA Access Control database with security level 100:

newusr internalUser level(100)

Example: Change a Property of an Enterprise User Using selang

The following selang command gives the AUDITOR property to an enterprise user Terry:

chxusr Terry auditor

Group Management Using selang

You can change any property of any group, except that you cannot change the name or the membership of enterprise groups (from within CA Access Control).

To change group properties or to assign access rights associated with groups, you can use CA Access Control Endpoint Management or the following selang commands:

- **join[-]** and **joinx[-]**

  Change the membership of an internal group

  Use join to add internal accessors to the group. Use joinx to add enterprise groups and users to an internal group. Use the - (minus) form of the commands to remove accessors.

- **editgrp**, **newgrp**, **chgrp**

  Change the non-membership properties of an internal group

- **editxgrp**, **newxgrp**, **chxgrp**

  Change the non-membership properties of an enterprise group

- **rmgrp, rmxgrp**

  Remove a user group

Example: Define a Group in the Database Using selang

The following selang command defines a new group "sales" in the database. The full name of the group is "Sales Department":

newgrp sales name('Sales Department')

Example: Change a Property of a Group Defined in the Database Using selang

The following selang command makes CA Access Control audit all events for members of the group AC_admins:

chgrp AC_admins audit(all)

Example: Add an Enterprise Group to an ACL Using selang

The following selang command adds the enterprise group mygroup to the ACL of the myfile:

Authorize FILE (myfile) xgid(mygroup)

Example: Add an Enterprise User to a Group Defined in the Database Using selang

The following selang command adds the enterprise user mydomain\administrator to the group AC_admins which is defined in the database:

joinx mydomain\administrator group(AC_admins)

Example: Add an Enterprise Group to a Group Defined in the Database Using selang

The following selang command adds the enterprise group Guests to the _restricted group:

joinx Guests group(_restricted)

Chapter 4: Managing Resources

This section contains the following topics:

Resources

A *resource* is an entity that can be accessed by an accessor and protected by an access rule, or the CA Access Control database record that corresponds to that entity. Examples of resources are files, programs, hosts, and terminals.

The main purpose of creating resource records in CA Access Control is to define access permissions for the resource that corresponds to the resource record. The access permissions that are required to access a resource are specified in the resource record's access control lists.

Resource Groups

A *resource group* is a resource that contains a list of other resources. A resource group is a member of one of the following classes: CONTAINER, GFILE, GSUDO, GTERMINAL, or GHOST.

Because a resource group is itself a resource, it has the same properties as its member resources. Therefore the advantage of using resources groups is that it simplifies administration. You can change the properties of all the member resources by changing the properties of the resource group.

Classes

In CA Access Control, the *class* of a record defines the properties that the record can have. All records in a class have the same properties, though different values for these properties.

Examples of classes are:

- TERMINAL class. This contains records for terminals, such as tty1, tty.

- FILE class. This contains records for files.

- PROGRAM class. This contains records of programs.

Each record contains values for the properties appropriate to the record class. For example, a record in the XUSER class includes such properties as the enterprise user's location and working hours, while a record in the HOSTNET class includes such properties as net services and IP address data.

CA Access Control includes predefined classes. You can also define new classes, called user-defined classes.

Default Record for Class

Most classes can include a default record (_default) specifying access types for resources of that class that are not defined in database records of their own.

Like other resource records, the _default record can include an ACL and a defaccess field. You can create a _default record for all classes except USER, GROUP, CATEGORY, SECLABEL, and SEOS.

UACC Class (Deprecated)

The UACC class is no longer recommended. To specify the default values for records in a class, use the _default record.

Some earlier versions of CA Access Control used a separate class, called UACC, for records resembling the _default records of other classes. The UACC class is no longer recommended, and if you use a _default record, the equivalent record in the UACC class is not checked. In future versions, the UACC class may no longer be supported.

For example, suppose user Henderson tries to kill process store_log. CA Access Control checks for authorization in the following order. The primary question is this: Is the process store_log defined in the database? CA Access Control searches the database for a record named store_log in the PROCESS class.

■ If no such record can be found, the process is not defined to CA Access Control. In that case, CA Access Control therefore uses either the _default record of class PROCESS, or the PROCESS record in the UACC class, to determine whether Henderson is allowed to kill store_log.

  – If user Henderson appears in the _default record's ACL, the authority specified in it is applied.

  – If Henderson does *not* appear in the _default record's ACL, the authority specified in the defaccess property of the _default record is applied. This authority is applied to all users who do not appear explicitly in the _default ACL.

- If process store_log is defined in the database, then the question is whether user Henderson appears in the ACL for process store_log in the database.

  – If user Henderson appears in the ACL for process store_log, the authority specified there is applied.

  – If Henderson does *not* appear in the ACL, CA Access Control applies the authority specified in the default access property of the store_log resource. This authority is called the resource's default access.

Note: If the default access (defaccess) of _default is set to NONE, or if _default is not specified and the default of the corresponding resource in the UACC class is NONE, then any accessor attempting to access a resource not defined in the class is denied access to the resource.

If the default access of _default (or UACC) is set to the highest authority (ALL, or in some cases READ or EXECUTE), then any resource that is not explicitly protected is accessible to everyone.

Predefined Classes

The predefined classes can be categorized into the following types:

Class Type	Purpose
Accessor	Defines objects that access resources, such as users and groups
Definition	Defines objects that define security entities, such as security labels and categories
Installation	Defines objects that control the behavior of CA Access Control
Resource	Defines objects that are protected by access rules

The following table contains a list of all predefined classes.

Class	Class Type	Description
ADMIN	Definition	Lets you delegate administrative responsibilities to users who do not have the ADMIN attribute. You give these users global authorization attributes and limit their administration authority scope.
AGENT	Resource	Not applicable to CA Access Control
AGENT_TYPE	Resource	Not applicable to CA Access Control
APPL	Resource	Not applicable to CA Access Control

Class	Class Type	Description
AUTHHOST	Accessor	Not applicable to CA Access Control
CALENDAR	Resource	Lets you define a Unicenter TNG calendar object for user, group, and resource enforced time restrictions.
CATEGORY	Definition	Lets you define a security category.
CONNECT	Resource	Lets you protect outgoing connections. The records in this class define which users can access which Internet hosts. Before you activate the CONNECT class, be sure that the streams module is active.
CONTAINER	Resource	Lets you define a group of objects from other resource classes, thus simplifying the job of defining access rules when a rule applies to several different classes of objects.
FILE	Resource	Lets you protect a file, a directory, or a file name mask.
GAPPL	Resource	Not applicable to CA Access Control
GAUTHHOST	Definition	Not applicable to CA Access Control
GFILE	Resource	Each record in this class defines a group of files or directories. Grouping is accomplished by explicitly connecting files or directories (resources of the FILE class) to the GFILE resource in the same way users are connected to groups.
GHOST	Resource	Each record in this class defines a group of hosts. Grouping is accomplished by explicitly connecting hosts (resources of the HOST class) to the GHOST resource in the same way users are connected to groups.
GROUP	Accessor	Each record in this class defines an internal group.
GSUDO	Resource	Each record in this class defines a group of commands that one user can execute as if another user were executing it. The sesudo command uses this class.
GTERMINAL	Resource	Each record in this class defines a group of terminals.
HNODE	Definition	The HNODE class contains information about the organization's CA Access Control hosts. Each record in the class represents a node in the enterprise.
HOLIDAY	Definition	Each record in this class defines one or more periods when users need extra permission to log in.

Class	Class Type	Description
HOST	Resource	Each record in this class defines a host. The host is identified by either its name or its IP address. The object contains access rules that determine whether the local host can receive services from this host. Before you activate the HOST class, be sure that the streams module is active.
HOSTNET	Resource	Each record in this class is identified by an IP address mask and contains access rules.
HOSTNP	Resource	Each record in this class defines a group of hosts, where the hosts belonging to the group all have the same name pattern. Each HOSTNP object's name contains a wildcard.
LOGINAPPL	Definition	Each record in the LOGINAPPL class defines a login application, identifies who can use the program to log in, and controls the way the login program is used.
MFTERMINAL	Definition	Each record in the MFTERMINAL class defines a Mainframe CA Access Control administration computer.
POLICY	Resource	Each record in the POLICY class defines the information required to deploy and remove a policy. It includes a link to the RULESET objects that contain a list of the selang commands for deploying and removing the policy.
PROCESS	Resource	Each record in this class defines an executable file.
PROGRAM	Resource	Each record in this class defines a trusted program that can be used with conditional access rules. Trusted programs are setuid/setgid programs that are monitored by the Watchdog to ensure they are not tampered with.
PWPOLICY	Definition	Each record in the PWPOLICY class defines a password policy.
RESOURCE_DESC	Definition	Not applicable to CA Access Control
RESPONSE_TAB	Definition	Not applicable to CA Access Control
RULESET	Resource	Each record in the RULESET class represents a set of rules which define a policy.
SECFILE	Definition	Each record in this class defines a file that must not be altered.
SECLABEL	Definition	Each record in this class defines a security label.
SEOS	Installation	The one record in this class specifies your active classes and password rules.

Class	Class Type	Description
SPECIALPGM	Installation	Each record in the SPECIALPGM class registers backup, DCM, PBF and PBN functions in Windows or xdm, backup, mail, DCM, PBF, and PBN programs in UNIX or associates an application that needs special authorization protection with a logical user ID. This allows you to set access permissions according to what is being done rather than who is doing it.
SUDO	Resource	This class, used by the sesudo command, defines commands that one user (such as a regular user) can execute as if another user (such as root) were executing them.
SURROGATE	Resource	Each record in this class contains access rules for an accessor that define who can use that accessor as a surrogate.
TCP	Resource	Each record in this class defines a TCP/IP service, for example, mail or http or ftp.
TERMINAL	Resource	Each record in this class defines a terminal-a device from which a user can log in.
UACC	Resource	Defines default access rules for each resource class.
USER	Accessor	Each record in this class defines an internal user.
USER_ATTR	Definition	Not applicable to CA Access Control
USER_DIR	Resource	Not applicable to CA Access Control
XGROUP	Resource	Each record in this class defines an enterprise group to CA Access Control.
XUSER	Resource	Each record in this class defines an enterprise user to CA Access Control.

Note: For more information about CA Access Control classes, see the *selang Reference Guide*.

User-Defined Classes

CA Access Control enables you to define new classes, so that you can protect abstract objects by creating appropriate records for them.

Example: User-Defined Class for a Database View

A site may use a database to store and display proprietary data.

You can define a user-defined class DATABASE_VIEWS, and define each database view to be a resource member of that class. Give the resource an ACL that defines the access authority required to create that database view. When a user attempts to create a database view, CA Access Control checks the access authority of the user, and permits or disallows the creation based on the ACL.

Wildcards in User-defined Classes Resources

By using wildcards in the name of a resource in a user-defined class, you can create a resource record that corresponds to multiple physical resources: any physical resource with a name that matches the wildcard pattern is protected by the access authorities associated with the resource record.

The wildcards you can use are:

- * for any number of any characters
- ? for any one character

If a physical resource name matches more than one resource record name, the longest non-wildcard match is used for that resource.

CA Access Control does *not* accept the following wildcard patterns as resource names:

- *
- /*
- /tmp/*
- /etc/*

User-Defined Class—Example

Suppose that your system serves a bank and you want to protect transfers of large amounts between accounts. You can use the following outline to set up this security.

1. Define a class to contain the records that describe transfers, called, for example, TRANSFERS.

2. For each monetary level transfer that you might want to protect, define a record in the TRANSFERS class.

    For example, you might define records named Upto.$1K, Upto.$1M, Upto.$10M, and Over.$10M.

    Define any other resources that you need to control transfers as members of the TRANSFERS class.

3. To give different users permission to perform different maximum transfers, grant or deny them access to the various records in the TRANSFERS class.

4. In addition, to handle programmatic transfers, insert in the bank's money-transfer program a call to the CA Access Control API, so that it checks the user's permission before it allows a transfer to proceed.

Chapter 5: Managing Authorization

This section contains the following topics:

Access Authorities

The main purpose of CA Access Control is to assign and enforce access authorities, also known as access rights.

An access authority always has the following components:

■   The resource that the access applies to, for example, a file, host, or terminal

■   The type of access, for example read, write, delete, log in, run

■   The accessor, which is either a user or a group

A user has the authority to access a resource in a certain way because one or more of the following are true:

■   The user has the access authority, as granted by the resource ACL

■   The user is a member of a group that has access authority.

■   The user is running a program that has the access authority. For example the user has the authority to run a program in the SPECIALPGM class, or to run a command in the SUDO class.

Note: For more information about access authority by class, see the *selang Reference Guide*.

Setting Access Authority - Examples

Example: Give an internal User Read Access

The following selang command adds the internal user internal_user to the ACL of terminal tty30, to give read access to the terminal:

```
authorize TERMINAL tty30 access(READ) uid(internal_user)
```

Example: Give an Enterprise User Read Access

The following selang command adds the enterprise user Terry to the ACL of terminal tty30, to give read access to the terminal:

authorize TERMINAL tty30 access(READ) xuid(Terry)

Example: Change an Access Authority of an Enterprise User to a Resource

The following selang command sets Terry's access to terminal tty30 to none, and so denies Terry access:

authorize TERMINAL tty30 access(NONE) xuid(Terry)

Example: Remove the Access Authority of an Enterprise User from a Resource

The following selang command removes Terry from the ACL in the terminal tty30:

authorize- TERMINAL tty30 xuid(Terry) access-

Terry now has the default access to the terminal.

Example: Give an Enterprise User Sub-administrator Access

The following selang commands set up the enterprise user Terry as a sub-administrator with the authority to manage users and files:

authorize ADMIN USER xuid(Terry)
authorize ADMIN FILE xuid(Terry)

Access Control Lists

The access authorities to a resource are specified in an access control list. Every resource record has at least two access control lists:

ACL

Specifies the accessors that are granted access to the resource, together with the type of access that they are granted.

NACL

Specifies the accessors that are denied authorization to the resource, together with the type of access that they are denied.

The access authority can also depend on the circumstances around the access, such as whether the user is logged in locally or not.

Conditional Access Control Lists

Conditional Access Control Lists (CACLs) provide an extension to ACLs. When an accessor attempts to access a resource, if the resource's ACL and NACL do not define an access authority for the user, CA Access Control examines the conditional access control lists.

The conditional access control lists specify access to resource where the access is by a particular method, for example by using a specified program.

For example you can use a conditional access control list to define a program pathing rule.

CA Access Control allows the following conditional access control lists:

- Program Access Control Lists (PACLs)

- TCP class access control lists

- CALENDAR class access control lists

To define an entry in a conditional access control list entry, you can use the via option of the selang authorize command.

In common with other access control lists, each entry in a conditional access control list specifies the accessors that are granted access to the resource, together with the type of access that they are granted. In addition, an entry in a conditional access control list specifies the condition under which the authority is assigned. For a PACL, the condition is the name of a program which the accessor needs to run to have the access.

Example: Using a PACL

To allow the enterprise user sysadm1 to become superuser only by running the program secured_su, you can specify the corresponding conditional access rule using the following selang command:

authorize SURROGATE user.root xuid(sysadm1) via(pgm(secured_su))

defaccess—The Default Access Field

The record for a resource can include a default access field, defaccess. The value of the defaccess field specifies the access authority that is allowed to accessors who are not covered by any of the resource access control lists.

How Access Authority to a Resource Is Determined

When an accessor attempts to access a resource, CA Access Control checks the access authority by running through one or more checks in a pre-determined order, until it gets a result. If any check produces an access result (deny or allow access), CA Access Control does not check any further, but instead returns the result.

The order in which it runs through these checks is important. For each resource, CA Access Control checks the access records in the following order by default:

1. The resource's time based restrictions
2. The resource's ownership (owners are allowed access)
3. B1 checks
4. The resource's NACL
5. The resource's ACL
6. The resource's PACL
7. The resource's defaccess field

The order of the last two checks is determined by the setting of the accpacl option. You can disable the use of resource PACL by using the selang command setoptions setpacl-.

One access control list can contain more than one entry that affects a user. For example, it can contain an entry that mentions a user explicitly, and also entries for each of the groups to which the user belongs. CA Access Control checks all the possible entries at each level before it goes to the next level. For more information about how it resolves conflicting rules at each level, see Interaction Between User and Group Access Authorities (see page 55).

Example: The Resultant Permission on a File

For the following table, assume that an accessor named user1 attempts to read the resource file1.

In the following table CA Access Control is following the default setting of the accpacl option to use the PACL.

Entry in NACL for user1	Entry in ACL for user1	Entry in PACL for user1	Entry in defaccess	Resulting Permission
Read	*(Any)*	*(Any)*	*(Any)*	Read denied
(Not defined)	None	*(Any)*	*(Any)*	Read denied

Entry in NACL for user1	Entry in ACL for user1	Entry in PACL for user1	Entry in defaccess	Resulting Permission
(Not defined)	Read	*(Any)*	*(Any)*	Read granted
(Not defined)	*(Not defined)*	via pgm securereader	*(Any)*	Read allowed through the securereader program
(Not defined)	*(Not defined)*	*(Not defined)*	Read	Read granted

Where an entry is shown as *(Not defined)*, this means that no entry for user1 exists in that access control list.

Where an entry is shown as *(Any)*, this means that the entry in that access control list does not matter, because CA Access Control does not check it.

The order that CA Access Control checks is from left to right. Notice that for all rows, the cells to the right of a cell with a defined access have the value *(any)*. Conversely all the cells to the left of a cell that contains a defined access have the value *(not defined)*.

Interaction Between User and Group Access Authorities

You can explicitly grant or deny access authorities to a user, and also to groups to which the user belongs. Sometimes these can conflict. The following example shows what results if conflicting access authorities are assigned to the same resource when a user is a member of two groups (Group 1 and Group 2).

It assumes that the accumulative group rights (see page 56) option is set (the default setting).

Access Authority for User	Access Authority for Group 1	Access Authority for Group 2	Resulting Access Authority
Access denied	*(Any)*	*(Any)*	Access denied
Access granted	*(Any)*	*(Any)*	Access granted
(Not defined)	Access granted	*(Not defined)*	Access granted
(Not defined)	*(Not defined)*	Access granted	Access granted
(Not defined)	Access granted	Access granted	Access granted
(Not defined)	Access denied	*(Any)*	Access denied

Access Authority for User	Access Authority for Group 1	Access Authority for Group 2	Resulting Access Authority
(Not defined)	*(Any)*	Access denied	Access denied

Where an entry is shown as *(Not defined)*, this means that no entry for the user or group is defined.

Where an entry is shown as *(Any)*, this means that the access authority does not matter, because CA Access Control does not check it.

Accumulative Group Rights (ACCGRR)

The *accumulative group rights* option (ACCGRR) affects how CA Access Control checks a resource's ACL. If ACCGRR is enabled, CA Access Control checks the ACL for the authorities granted from all the groups to which the user belongs. If ACCGRR is disabled, CA Access Control checks the ACL to see if any of the applicable entries contain the value none. If so, access is denied. Otherwise CA Access Control ignores all group entries except the first applicable one in the access control list. By default the option is enabled.

To enable the ACCGRR option, you can use the following selang command:

setoptions accgrr

To disable the ACCGRR option, you can use the following selang command:

setoptions accgrr-

Security Levels, Categories, and Labels

Security levels and security categories provide additional ways to restrict access to a resource, complementary to the use of access control lists.

Security labels are a means to bundle security levels and categories together, to manage them more easily.

Security Levels

A *security level* is an integer between 0 and 255 that you can assign to accessors and resources. An accessor cannot access a resource if the accessor has a security level less than the security level assigned to the resource, even if the user is granted access authority in the resource's access control list. If a resource has a zero security level, security level checking is not checked for that resource.

An accessor with a security level of zero cannot access any resource that has a non-zero security level.

Security Categories

A *security category* is the name of record in the CATEGORY class. You can assign a security category to accessors and to resources. An accessor can access a resource only if the accessor is assigned to all of the security categories assigned to the resource.

Security Labels

A *security label* is the name of a record in the SECLABEL class. A security label bundles together a security level and a set of security categories. Assigning a security label to an accessor or a resource gives the accessor or resource the combined security level and security categories associated with the security label. A security label overrides any specific security level and category assignments in an accessor or resource.

Example: Use of a Security Label High_Security

Assume High_Security is a security label that contains a security level 255 and the security categories MANAGEMENT and CONFIDENTIAL.

if you assign a user user1 to the security label High_Security, user1 has a security level of 255 and also has the security categories MANAGEMENT and CONFIDENTIAL.

Chapter 6: Protecting Accounts

This section contains the following topics:

Why Protect Accounts?

User accounts are often the object of password attacks. Root account protection involves monitoring substitute user (su) requests and using the Surrogate DO (SUDO) facility, which solves the dilemma of superuser privileges. CA Access Control provides a two-level password protection system: serevu (revoke user daemon) and PAM (Pluggable Authentication Module). You can also protect accounts by specifying automatic lockouts after a period of user inactivity.

Safe User Substitution

The UNIX su command lets a user switch to another user using the target user's password. A user who wants to switch a user ID must memorize the target user's password, write it down, or ask the target user to use a trivial password. This violates several password policies. Also, the su command does not record who invoked the command so a user pretending to be the owner of an account is indistinguishable from the actual owner.

CA Access Control includes the sesu utility, which is an enhanced version of the UNIX su command. You can configure sesu to prompt the user for their password as a means of authentication, rather than prompting for the target user's password. The authorization process is based on the access rules defined in the SURROGATE class and, optionally, on the password of the user executing the command.

Unlike permission to su, permission to sesu does not depend on knowing the target user's password. Instead, it depends on permissions specified in the database; users remain accountable for their actions because their login identities are remembered.

If a user is a surrogate to one of the users in the _surrogate group, CA Access Control sends a full trace of the user's actions as the new user to the audit trail.

To protect against inadvertent use, sesu is marked in the file system so that no one can run it. The security administrator must mark the program as executable and setuid to root before you can use it.

Important! Before you use the sesu utility, define all users to the CA Access Control database and set sesu prerequisites. This prevents you from opening up the entire system to users who are not defined to CA Access Control.

Set User ID Substitution Rules

To prevent or let users substitute other users you need to set user ID substitution rules. These rules are governed through SURROGATE class resources. To define any user substitution rules you need to create SURROGATE records.

To set user ID substitution rules

1. In CA Access Control Endpoint Management click the Users tab, then click the Authorization and Delegation subtab.

    The Authorization and Delegation menu options appear on the left.

2. Click Users ID Substitution.

    The Users ID Substitution page appears.

3. Click Create User ID Substitution.

    The Create User ID Substitution page appears.

4. Complete the fields in the tabbed pages, then click Save.

    Note: For more information on SURROGATE class properties, see the *selang Reference Guide*.

How to Set Up sesu for User Substitution

By default, the sesu utility is marked in the file system so that no one can run it. Before you make sesu available to your users, you must set database rules to ensure it is used safely. You then need to lock the system's su utility so that users are forced to use the CA Access Control sesu utility instead.

To set up sesu, do the following:

1. Set basic user substitution rules (see page 61).

2. Replace the system's su utility with the CA Access Control sesu utility (see page 61).

3. Prevent users from running the system's su utility (see page 64).

Note: After you complete this setup, when CA Access Control is running the system's su utility will not execute and users will be forced to use the secured sesu utility. When CA Access Control is not running, the system's su utility will work.

Set Basic User Substitution Rules

Before you start using the sesu utility, you should set up some common user substitution rules in the database. These rules prevent unknown users undesirably substituting privileged user accounts, but permit specific users and processes to perform necessary user substitution activities.

To set basic user substitution rules

1. Create a surrogate resource for the root user (USER.root) with the following attributes:

   ■ *nobody* as owner

   ■ Default access *none*

   ■ All administrators should have full control

   This prevents all users from substituting root, unless explicitly authorized. All administrators are explicitly authorized to substitute root.

   Note: You can authorize individual administrators separately or authorize all administrators using the administrator's group.

2. Create a surrogate resource for root's group (GROUP.other) with the following attributes:

   ■ *nobody* as owner

   ■ default access of *none*

   ■ All administrators should have full control

   This prevents all users from substituting root's group, unless explicitly authorized. All administrators are explicitly authorized to substitute root's group.

   Note: On most UNIX systems root's group is either *other* or *sys*.

3. Change the user substitution rules for USER._default as follows:

- *nobody* as owner

- Default access *none*

- Authorize root to substitute to any undefined user

- Authorize the administrators' group to substitute to any undefined user

This prevents all users from substituting any group, unless explicitly authorized, and authorizes root and root's group to substitute any user, unless explicitly denied.

Note: You need to specifically authorize root to permit programs such as dtlogin to switch session ownership from root, the default X window owner (uid=0), to anyone else. If you do not do this, login attempts will fail because CA Access Control is blocking any user substitution activity that has not been explicitly authorized.

4. Change the group substitution rules for GROUP._default as follows:

- *nobody* as owner

- Default access *none*

- Authorize root to substitute any undefined groups

- Authorize the administrators' group to substitute to any undefined group

This prevent all users from substituting any group, unless explicitly authorized, and authorizes root and root's group to substitute any group, unless explicitly denied.

Example: Set Basic User Substitution Rules in selang

Use the following selang commands to set basic user substitution rules in your environment:

```
nr surrogate USER.root defacc(n) own(nobody)
auth surrogate USER.root gid(sys_admin_GID) acc(a)
nr surrogate GROUP.other defacc(n) own(nobody)
auth surrogate GROUP.other gid(sys_admin_GID) acc(a)
cr surrogate USER._default defacc(n) own(nobody)
cr surrogate GROUP._default defacc(n) own(nobody)
auth surrogate USER._default uid(root) acc(a)
auth surrogate GROUP._default uid(root) acc(a)
auth surrogate USER._default gid(sys_admin_GID) acc(a)
auth surrogate GROUP._default gid(sys_admin_GID) acc(a)
```

Replace the System's su Utility with the CA Access Control sesu Utility

By default, the sesu utility is marked in the file system so that no one can run it. To let users substitute other users by using the sesu utility, you must enable sesu and replace the system su with this utility.

To replace the system's su utility with the CA Access Control sesu utility

Note: You need to be root or another authorized user to perform the following steps.

1. Permit users to run the sesu utility using the following command:

   chmod +s /opt/CA/AccessControl/bin/sesu

2. Find out the location of the system's su utility using the following command:

   which su

3. Rename the system's su utility using the following command:

   mv *su_dir*/su *su_dir*/su.ORIG

   where *su_dir* is the directory where su resides.

4. Link the sesu utility to the su command:

   ln -s /opt/CA/AccessControl/bin/sesu *su_dir*/su

   This lets users continue to use the su command, although it now runs the sesu utility.

5. Stop CA Access Control using the following command:

   secons -s

6. Modify CA Access Control configuration settings using the following commands:

   seini -s sesu.SystemSu *su_dir*/su.ORIG
   seini -s sesu.UseInvokerPassword yes

   The token SystemSu is set so that sesu can call the original system su utility if CA Access Control is not running.

   The token UseInvokerPassword is set to tell CA Access Control to prompt the user for their original password instead of root's password or another user's password. The user needs to re-authenticate before the user substitution is permitted.

7. Reload CA Access Control using the following command:
   seload

Prevent Users from Running the System's su Utility

Although the sesu utility is configured, anyone can run su.ORIG (the renamed system su utility), as before, with root's or a user's password. To prevent this, use the PROGRAM class to explicitly prevent su.ORIG execution when CA Access Control is running.

Note: If you used seuidpgm during CA Access Control installation and configuration, you do not need to follow this procedure. su will not run as it has been modified (renamed to su.ORIG).

To prevent users from running the system's su utility

1. In selang, set CA Access Control to monitor the renamed su utility, using the following command:

    nr program *su_dir*/su.ORIG defacc(x) own(nobody)

2. Logged in as root, change file access and modification time, using the following command:

    touch *su_dir*/su.ORIG

    CA Access Control is watching su.ORIG and, because the file has been *touched*, will prevent it from being executed.

Setting Up the Surrogate DO Facility

Operators, production personnel, and end users often need to perform tasks that only the superuser can perform. These tasks include the following:

- Mounting a CD-ROM

- Using backup scripts

- Setting up a printer

The traditional solution is to supply all these users with the superuser's password, which compromises the security of the site. The secure alternative-keeping the password secret-results in the system administrator being overloaded with legitimate requests from users to perform routine tasks.

The Surrogate DO (sesudo) utility solves this dilemma. It allows users to perform actions that are defined in the SUDO class, where each record contains a script, specifies which users and groups can run the script, and lends them the necessary permissions for the purpose.

For example, to define a SUDO resource that mounts a CD-ROM as if the user were root, enter the following command:

newres SUDO MountCd data('mount /usr/dev/cdrom /cdr') targuid(root)

This newres command defines MountCd as a protected action that some users may receive root authority to perform. This example uses the targuid(root) parameter to show that root is the ID of the target user-the user whose permissions are borrowed. In practice, the parameter would be unnecessary for this example because root is the default target ID for a SUDO record

Important! In the data property, use a full absolute path name. A relative path name could accidentally execute a Trojan horse program planted in an unprotected directory.

In addition, users can be authorized to perform the MountCd action by using the authorize command. For example, to allow the user *operator1* to mount the CD-ROM, enter the following command:

authorize SUDO MountCd uid(operator1)

You can also explicitly prevent a user from performing the protected action by using the authorize command. For example, to prevent the user *operator2* from mounting the CD-ROM, enter the command:

authorize SUDO MountCd uid(operator2) access(None)

Executing the sesudo utility performs the protected action. For example, the user *operator1* would mount the CD-ROM using the following command:

sesudo MountCd

The sesudo utility first checks whether the user is authorized to perform the SUDO action and then, provided the user is authorized to the resource, executes the command script defined in the resource. In the case of our example, sesudo checks whether operator1 is authorized to perform the MountCd action and then invokes the command mount /usr/dev/cdrom /cdr.

If you would like sesudo to request the user's password before executing, define or modify the SUDO record with a command that includes the PASSWORD parameter. If you do not use that parameter, the user's ability to execute the command is based solely on the access rules for the SUDO object.

Note: For more information about the sesudo utility and managing SUDO records (editres command), see the *Reference Guide*.

Defining SUDO Records

A record in the SUDO class stores a command script so that users can run the script with borrowed permissions. The ability to borrow permissions is tightly controlled by the SUDO record, as well as by the sesudo command that executes the scripts.

In a SUDO record, the comment property is used for a special purpose, and often it is known by its alternate name: the data property.

The data property's value is the command script, with the optional addition of one or more script parameter values that are to be prohibited or permitted. The entire data property value must be enclosed in single quotes, and executables should be referenced by their complete path names to prevent Trojan horses from taking their place.

This is the format for the data property:

data('*cmd*[;[*prohibited-values*][;*permitted-values*]]')

Because the lists of prohibited and permitted values are optional, a simple data property value could be the following:

newres SUDO MountCd data('mount /dev/cdrom /cdr')

The simple value in the command means that the command sesudo MountCd executes the script mount /dev/cdrom /cdr. No particular script parameter values are prohibited; all are permitted.

Wildcards and powerful variables give you flexibility in specifying prohibited and permitted parameters. The wildcards you can use are the standard UNIX wildcards. The variables are these:

Variable	Description
$A	Alphabetic value
$G	Existing CA Access Control group name
$H	Home path pattern of the user
$N	Numeric value
$O	Executor's user name
$U	Existing CA Access Control user name
$e	SUDO commands with no parameters
$f	Existing file name
$g	Existing UNIX group name

Variable	Description
$h	Existing host name
$r	Existing UNIX file name with UNIX read permission
$u	Existing UNIX user name
$w	Existing UNIX file name with UNIX write permission
$x	Existing UNIX file name with UNIX exec permission

If you append a list of *prohibited* parameter values to the script:

■   Separate the script from the prohibited parameter values with a semicolon, but keep them all inside the single quotes. For example, if you want to prevent the user from using -9 but you permit the user to use all other parameters, enter the following command:

newres SUDO *scriptname* data('*cmd*;-9')

where *cmd* represents your script.

Alternatively, if you do not allow any parameter values, but rather want all parameters defaulted, define the SUDO record as follows:

newres SUDO *scriptname* data('*cmd*;*')

■   If a script parameter has more than one prohibited value, use the space character as a separator. For example, if you want to prevent the user from using -9 and -HUP but you permit the user to use all other parameters, enter the following command:

newres SUDO *scriptname* data('*cmd*;-9 -HUP')

■   If more than one script parameter has prohibited values, use the pipe character (|) as a separator between sets of prohibited values. For example, if you want to prevent the user from using -9 and -HUP for the script's first parameter and from using any existing UNIX user name for the second parameter (see the previous list of variables), enter the following command:

newres SUDO *scriptname* data('*cmd*;-9 -HUP | $u')

If the script has more parameters than you list, then your last set of prohibited parameters applies to all the remaining parameters.

If you append a list of *permitted* parameter values to the script:

- The sesudo utility enforces two checks: Not only must the parameter values not match any of the corresponding prohibited values; they must also match at least one of the corresponding permitted values.

- Separate the list of *permitted* values from the list of *prohibited* values with a semicolon, but keep them all inside the single quotes. Even if you have no list of prohibited values, you still need the semicolon; otherwise what you intend to permit is prohibited. For example, if you want to allow only the value NAME as a parameter value for the script, enter the following command:

  newres SUDO scriptname data('cmd;;NAME')

- Just as in the other list:

  - If a script parameter has more than one permitted value, use the space character as a separator.

  - If more than one script parameter has permitted values, use the pipe character (|) as a separator between sets of permitted values.

  For example, if you have two parameters, and the first must be numeric but must not be a UNIX user name, and the second must be alphabetic but must not be a UNIX group name, enter the following command:

  newres SUDO *scriptname* data('*cmd*; $u | $g ; $N | $A')

  If the script has more parameters than you list, then your last set of permitted parameters applies to all the remaining parameters.

Thus, the overall format for the data property is this: first the script; then the prohibited values, parameter by parameter; then the permitted values, parameter by parameter:

```
data('cmd;
param1_prohib1 param1_prohib2 ... param1_prohibN | \
param2_prohib1 param2_prohib2 ... param2_prohibN | \
...
paramN_prohib1 paramN_prohib2 ... paramN_prohibN ; \
param1_permit1 param1_permit2 ... param1_permitN | \
param2_permit1 param2_permit2 ... param2_permitN |
...
paramN_permit1 paramN_permit2 ... paramN_permitN')
```

Preventing Password Attacks

The most common type of unauthorized access is that of hackers who guess passwords. CA Access Control provides two tools that detect and protect against password attacks: serevu and pam_seos.

Another method of protecting against password attacks is controlling passwords used in your environment by setting password policy rules.

serevu

The serevu daemon locks the accounts of users who performed more than a specified number of login attempts. This prevents potential password attacks by rejecting further attempts to enter the account; it also prevents "dictionary attacks".

Normally, the danger in using the user lockout utility is that it opens the system to denial of service denial attacks. One common type of denial of service attack is an attempt to break into the system administrator's account. After a few attempts, the system administrator account is revoked and the system administrator can no longer log in. If similar attacks are performed on all critical user accounts, the system may be rendered unusable, with no way of recovering. To prevent this, the serevu daemon provides the following two modes of operation:

- The account is revoked for a specified period of time, after which it is automatically restored.
- The account is permanently revoked.

serevu never revokes root, so the system is never locked out.

Note: For more information about the serevu daemon, see the *Reference Guide*.

Note: Take special care regarding the root user's password to prevent successful dictionary attacks on root.

pam_seos

pam_seos is a Pluggable Authentication Module (PAM) that CA Access Control uses for advanced account management functions. CA Access Control calls pam_seos during the login procedure of any login program. The module is a shared object that can be dynamically loaded to provide the necessary functionality upon demand.

You can configure pam_seos to perform three actions:

- Detect login failures

  The Account Management Component detects any failed login attempt and logs it to both the audit file and a special failed logins file. This module detects UNIX failures, not cases in which CA Access Control denies access.

  CA Access Control writes the failed login attempts to a special file. The serevu utility reads this file and uses the information to determine if and when user access should be revoked.

- Provides debug mode

  When CA Access Control denies a login, it usually does not show the reason for denial during the login session. If the pam_seos module's debug mode is set, CA Access Control gives a short description of the reason for login denial. For example, "grace logins" means that the user has no remaining logins.

- Checks for expired passwords and grace logins

  The Password Management Component invokes the segrace utility, which checks for a user's password expiration and the number of grace logins. If a user's password expires, and the user has no grace logins left, segrace invokes the sepass utility to allow the user to change the password.

Note: CA Access Control invokes segrace only when a password change is needed.

The installation program adds the relevant lines to the pam.conf configuration file, and stores the old configuration file as /etc/pam.conf.bak.

Configuration of the pam_seos modules is performed through the seos.ini file. Set the following tokens, located in the [pam_seos] section, according to the required functionality:

To use the Password Expiration and Grace Logins check, set the following token in the seos.ini file:

call_segrace = Yes

To use Login Debug Mode, set the following token in the seos.ini file:

debug_mode_for_user = Yes

To make serevu use pam_seos login failure detection, set the following token in the seos.ini file:

serevu_use_pam_seos = Yes

Restrictions and Limitations

The protection techniques described in this section have the following restrictions and limitations:

- On Sun Solaris, after five failed login attempts, serevu is notified.

- The pam_seos module is only implemented in the versions of Sun Solaris, HP-UX, and Linux that support PAM.

Checking User Inactivity

The inactivity feature protects the system from unauthorized access through accounts whose owners are away or no longer employed by the organization. An inactive day is a day in which the user does not log in. You can specify the number of inactive days that must pass before the user account is suspended and cannot log in. Once an account is suspended, you must manually reactivate it.

Note: Password changes count as activities, in terms of inactivity checks. If a user's password changes, that user cannot become suspended due to inactivity.

You can set the number of inactive days with the inactive property of a USER class record or a GROUP class record. The latter affects only users that have that group as a profile group. You can also set inactivity for all users systemwide with the INACT property of the SEOS class.

In selang, use the following command to specify inactivity globally:

setoptions inactive (*numdays*)

To set the number of days for a group (which overrides the systemwide inactive setting for that group), use the following command:

editgrp *groupName* inactive (*numdays*)

To set the number of days for a user (which overrides group and systemwide settings for that user), use the following command:

editusr *userName* inactive (*numdays*)

To reactivate a suspended user account, use the following command:

editusr *userName* resume

To reactivate a suspended profile group, use the following command:

editgrp *userName* resume

To disable inactive login checking at the systemwide level, use the following command:

setoptions inactive-

To disable inactive login checking for a group, use the following command:

editgrp *groupName* inactive-

To disable inactive login checking for a user, use the following command:

editusr *userName* inactive-

Chapter 7: Managing User Passwords

This section contains the following topics:

Password Control

Passwords are the most popular device for authentication, but password protection has well-known problems:

- Trivial passwords are easy to guess.

- Passwords that last for years and cyclic passwords are eventually broken.

- Listeners can trap passwords that are sent in clear text over the network.

Defining Password Policies

The most important password rule is that users must not give out their passwords explicitly or indirectly (by using trivial passwords). The only way to achieve acceptable password security is by training and education. CA Access Control cannot replace education, but it can enforce rules and policies that force users to use passwords of a minimum quality. The rules that you can specify include the following:

- The new password cannot match previous passwords.

- The new password cannot contain the user name.

- The new password cannot contain the password that it is replacing.

- The new password cannot be contained by the password that it is replacing.

- The new password cannot match the password that it is replacing, regardless of case sensitivity.

- The new password must have at least the minimum number of alphanumeric characters, special characters, digits, lowercase characters, and uppercase characters.

- The new password must not have more repetitive characters.

- The new password cannot be one of the restricted words in the dictionary to which the Dictionary token in the seos.ini file points.

■ Each password must have a maximum lifetime; that is, it must expire, forcing the user to choose a new password after a certain interval.

■ Each password must have a minimum lifetime. (By specifying a minimum lifetime, you can prevent users from quickly and repeatedly changing passwords. By quickly changing passwords, they could overflow the password history list and then re-use a previous password.)

Important! Password rules only affect sepass and not native password tools. Make sure you replace passwd with a link to sepass.

Configure Password Quality Checking

To configure password quality checking

1. In CA Access Control Endpoint Management click the Configuration tab.

   The configuration menu options appear on the left.

2. Click Class Activation in the Miscellaneous section options.

   The Class Activation page appears.

3. Select PASSWORD in the User Identity Control section, and click Save.

   This activates password quality checking.

4. Click User Password Policy in the Policies section options.

   The User Password Policy page appears.

5. Define the rules to be used for the password checks, and click Save.

   The rules you define for password checks are now enforced when passwords are changed.

6. (UNIX only) Update the new passwords by using the sepass utility.

   Note: For more information about sepass utility, see the *Reference Guide*.

Example: Define Password Checking Rules

The following selang commands activate password quality checking and define password rules that enforce a minimum of:

■ Six alphanumeric characters

■ Three lowercase characters

■ Two numeric characters

```
setoptions class+ (PASSWORD)
setoptions password(rules(alpha("6") lowercase("3") numeric("2")))
```

Note: For more information about the format of the setoptions command, see the *Reference Guide*.

Changing Passwords

CA Access Control includes the executable *ACInstallDir*/bin/sepass (where *ACInstallDir* is the installation directory for CA Access Control, by default /opt/CA/AccessControl), with which most users should change their passwords (instead of with /bin/passwd).

- Only sepass ensures that the new password matches CA Access Control password policies. And only sepass updates the database with the new password and the date on which the password was changed. In addition, sepass performs the same functions as /bin/passwd.

- The original /bin/passwd executable should not be used unless you choose to *discard* the password quality checks performed by CA Access Control. In this case, you can continue to use the original /bin/passwd, and CA Access Control accepts the system's password without performing any quality checks on passwords.

You can also change passwords using selang. Enter the following command to assign a password to a user:

chusr *userName* password(*string*)

Note: If you change another user's password (as an administrator) and password checking is enabled, the user must change the password at the next login.

Password Expiration and Grace Logins

The interval parameter sets the maximum number of days a password can be used. When the specified number of days passes, CA Access Control informs the user that the current password has expired. The user can then renew the password immediately, or continue using the old password until the number of grace logins is reached. In the latter case, the user cannot access the system and must contact the system administrator to select a new password.

Specify the Password Interval

At the systemwide level, you use the setoptions command to specify the interval before the system prompts all users for a new password. If the segrace utility is part of the user's login script or if you configure PAM to call segrace (if your native operating system supports PAM), CA Access Control informs the users that the current password has expired when the specified number of days is reached. The users can then immediately renew the password, or continue using the old password until the number of grace logins is reached. After reaching the number of grace logins, the users are denied access to the system and must contact the system administrator to select a new password.

To set or cancel the password interval at the systemwide level, use the following command:

setoptions password({interval(*NumDays*)|interval-})

The value of *NumDays* must be zero or a positive integer. An interval of zero disables password interval checking for users. Set the interval to zero if you do not want passwords to expire. An interval of zero should only be used for users with low security requirements.

The interval- parameter cancels the password interval setting. If the user has a profile group with a value for this parameter, that value is used. Otherwise, the default set by the setoptions command is used. Only use this parameter with the chusr or editusr command.

Set Individual User or Group Password Intervals

You can also set the interval for specific users or profile groups. These settings override the systemwide interval for those users or groups. When the specified number of days is reached, CA Access Control informs the users that the current password has expired. The users can then immediately renew the password, or continue using the old password until the number of grace logins is reached. After reaching the number of grace logins, the users are denied access to the system and must contact the system administrator to select a new password.

To set or cancel the password interval for a user:

editusr {interval(*NumDays*) | interval-}

To set or cancel the password interval for a group:

editgrp password{(interval(*NumDays*)) | (interval-)}

The value of *NumDays* must be zero or a positive integer. An interval of zero disables password interval checking. Set the interval to zero if you do not want a password to expire. An interval of zero should only be used for users with low security requirements.

The interval- parameter cancels the password interval setting. If it is canceled and a value for interval is set in the user record, the value in the user record is used. Otherwise, the default set by the setoptions command is used. Use this parameter with the setoptions, chgrp, or editgrp commands only.

Grace Logins

With password checking enabled, CA Access Control checks whether the user's password has expired each time a user attempts to log in. After the password expires, the user can be "graced" with the opportunity to log in a few more times, after which the user can no longer log in.

The grace login option sets the maximum number of logins that are permitted after password expiration before the user is suspended. The number of grace logins must be between 0 and 255. After the number of grace logins is reached, the user is denied access to the system and must contact the system administrator to select a new password. If grace is set to zero, the user cannot log in. The default number of grace logins is five.

You can use this method to force a user to change their password. Reset the user's password and give them one grace login whence they can change their password.

Track Grace Logins

To allow the end user to keep track of grace logins after the expiration, insert a call to the segrace utility in the user's .login, .profile, or .cshrc file. The segrace utility then displays a message to the user stating the number of remaining grace logins. You can also check whether a user's password has expired graphically with the segracex utility.

Note: For more information about the segrace and segracex utilities, see the *Reference Guide*.

To set the systemwide default value for the number of grace logins, enter the following command:

setoptions password(rules(grace(*nLogins*)))

To set or cancel grace logins for a specific user, enter the following command:

chusr *userName* {grace(*nLogins*) | grace-}

To set or cancel grace logins for a profile group, enter the following command:

chgrp *groupName* {grace(*nLogins*) | grace-}

The value set by the chusr or chgrp command overrides the system value for the users specified in that command.

Note: The grace property for a GROUP class and also the global grace login setting set the number of grace logins for a user **after** the user's password expires. However, the grace property in the USER class sets the password to expire immediately; the grace logins are automatically set up (using the GROUP record or the system default) after the user's password expires. You cannot set password expirations for a group, only for users.

Chapter 8: Protecting Files and Programs

This section contains the following topics:

Restricting Access to Files and Directories

CA Access Control leaves the UNIX system of permissions intact but adds a layer of enhanced access control to it.

CA Access Control intercepts each of the following file access operations and verifies that the user has authorization for the specific operation before returning control to UNIX. The access type is in parentheses.

- File create (create)

- File open for read (read)

  Note: If you want *read* privileges to control whether users can perform operations that obtain information about the file (such as ls -l), set the STAT_intercept configuration setting to 1. For more information, see the *Reference Guide*.

- File open for write (write)

- File execute (execute)

- File delete (delete)

- File rename (delete, rename)

- Change permission bits (chmod)

- Change owner (chown)

- Change timestamp—for example, as a result of executing the touch command (utime)

- Edit native ACL—using the acledit command—for systems that support ACLs (sec)

- Change directory (chdir)

CA Access Control access checking differs from the native UNIX authorization in the following ways:

- CA Access Control bases its authorization checks on the original user ID of the user who logged in, not on the effective user ID (euid). For example, if *userA* invokes the su command to surrogate to another user, *userA* still only has access to those files to which *userA* is permitted. Surrogating to another user does *not* automatically give the original user access to the target user's files as it does in UNIX.

- CA Access Control does not give the superuser (root) automatic access to every file on the system. The superuser is subject to authorization checking like all other users of the system.

- Authorization checking is based on the CA Access Control normal and conditional access lists, day and time restrictions, security levels, security categories, and security labels.

- If you do not specifically authorize a user to access a file, CA Access Control checks whether that user belongs to any group authorized to access the file.

- Each file access is audited trough the normal CA Access Control audit procedures.

- When deleting a file, CA Access Control requires the user to have DELETE access authority to the specified file, whereas UNIX requires the user to have WRITE authority for the parent directory.

- To rename a file, the user must have DELETE access authority to the source file and RENAME access authority to the target file. UNIX also requires that they user have WRITE access authority for the parent directory.

- All users are given permanent READ access (as a minimum) to the files /etc/passwd and/etc/group, regardless of the default setting of these files. This prevents the possible hanging of the system.

- The owner of a FILE object in the CA Access Control database always has full access to the file protected by the object.

- The chdir access type controls the chdir command specifically, and does not execute, as UNIX does.

The following are the limits of the File Protection System:

- With respect to users who are not members of the special _restricted group, CA Access Control protects only those files and directories that:

  - Are defined by their individual names in the database

  - Match a name pattern (for example, /etc/*) that is defined in the database

  For users that belong to the group _restricted, all system files are protected by CA Access Control. For files that are not defined in the database, authorization is based on the _default record of the FILE class.

- CA Access Control maintains a table of all file names and directory names (including patterns using wildcards) that indicate resources that need protection. The amount of memory available for this table is limited. Normally, the maximum number of files and directories you can define by individual names in the database is 4096, and the maximum number of name patterns is 512.

- Some files receive protection even if no explicit access rules exist for them. These include the CA Access Control database files, audit logs, and configuration files.

  Note: For more information, see the FILE class in the *Reference Guide*.

CA Access Control supports the following access types for files.

- ALL

- CHDIR

- CHMOD

- CHOWN

- CONTROL

- CREATE

- DELETE

- EXECUTE

- NONE

- READ

- RENAME

- SEC

- UPDATE

- UTIME

- WRITE

The File Protection System is useful for protecting selected sets of files that contain sensitive data. For example, you can use CA Access Control to protect the following files:

- /etc/passwd

- /etc/group

- /etc/hosts

- /etc/shadow

You should use CA Access Control to protect databases (access should be granted only to the server daemon) and all other sensitive files at your site.

Some files that always need access control are governed by rules even without you specifying them.

How File Protection Works

When the seosd daemon starts, it performs the UNIX stat command for each discrete file object defined in the database. It then builds a table in memory that contains an entry for each file object. In addition, for each discrete file, the table contains the file's inode and device; with this information, CA Access Control can also protect the hard links to the files because the protection is according to device and inode. The database does not keep information about a file's inode and device.

When creating a new file rule through CA Access Control:

- If the file exists in UNIX, CA Access Control first performs a stat command for the file and then adds a new entry to the file table with the file's inode and device information.

- If the file does not exist in UNIX, CA Access Control adds a new entry of the file's name to the file table (without inode and device information). This entry is the same as the entry for a generic file object. At the same time, the kernel keeps an indication in its internal tables that this file must be checked during creation for inode and device information. When the file is subsequently created, the kernel intercepts its creation and informs seosd of the file's inode and device information so that seosd can update the file's entry in the file table.

When you delete a file, CA Access Control deletes its entry in the seosd file table, but the entry remains in the CA Access Control database in case you create it again.

Protect Files

To define a protected file in selang, enter the following command:

newres FILE *filename*

For example, to register a file named/tmp/binary.bkup, enter the following command:

newres FILE /tmp/binary.bkup

Note: When you define a file rule without specifying its access type, the default access of NONE is assigned. In this case, the file's owner is the only one who can access the file.

Most protected files should be protected from access by the superuser. Otherwise, any user who knows the superuser's password gains automatic access to the files. At the same time, you can prevent all other users except the file's owner from accessing the file.

To protect several similarly named files, use a file name pattern that includes a wildcard. The wildcards are ***** (which indicates zero or more characters) and **?** (which indicates any one character, other than **/**).

The pattern that you specify is matched against the file's full path name so that the pattern /tmp/x* matches files named /tmp/x1, /tmp/xxx, and even /tmp/xdir/a.

Patterns that CA Access Control does *not* let you specify are: **/***, **/tmp/***, and **/etc/***.

Important! Because file name patterns are such a powerful tool, you should not experiment freely with them.

For example, the following command defines as protected every file in the /tmp directory that has a name starting with a, and ending with b (this would include a file like /tmp/axyz/axyzb):

newres FILE /tmp/a*b

Wildcards in FILE Resource Names

By using wildcards in a file resource name, you can create a file record that corresponds to multiple files: any file with a name that matches the wildcard pattern is protected by the access authorities associated with the record.

The wildcards you can use are:

- * for any number of any characters

- ? for any one character

If a physical resource name matches more than one resource record name, the longest non-wildcard match is used for that resource.

CA Access Control does *not* accept the following patterns in names of FILE resources:

- *

- /*

- /tmp/*

- /etc/*

Example: Use of Wildcards in a FILE Resource

The FILE resource /usr/lpp/bin/* protects all files and sub-directories under /usr/lpp/bin (however deeply nested).

Restricting File Access

To restrict a file from access by the superuser in selang, use a longer version of the newres command. For example, to prevent the file /tmp/binary.bkup from being accessed by the superuser, as well as any other user except the user myuser, you can use the following selang command:

```
newres FILE /tmp/binary.bkup owner(myuser) defaccess(N)
```

This command does the following:

1. Defines /tmp/binary.bkup as a protected file.

2. Sets the user myuser as the owner of the file, granting myuser access to the file.

3. Sets the default access of the file to NONE, preventing any other user from accessing the file. To permit other users access to the file, you must explicitly define access rules for that file.

Important! If you invoke the selang command under root authority and then define FILE records without explicitly specifying another user as their owner, root becomes the owner of those files. As the owner, root (or any user who logs in as root) has complete and free access to the files.

Note: You can set the token use_unix_file_owner in the seos.ini file to yes. This permits regular UNIX users to define access rules for the files they own.

Preventing File Access

Sometimes it is convenient to define a FILE record that has no owner. To define a FILE record that does not have an owner in selang, use the special owner "nobody."

For example, to define the file /tmp/binary.bkup as a protected file and prevent all users from accessing the file, enter the following selang command:

```
newres FILE /tmp/binary.bkup owner(nobody) defaccess(N)
```

This newres command ensures that even the user who defined the command, whether root or otherwise, cannot access the file. After preventing all users from accessing a file, you must usually grant one or more users access to that file explicitly.

To explicitly permit a user access to a protected file, use the authorize command. For example, to grant the user "*userJo*" update access to all files in the /tmp directory beginning with Jo, enter the selang command:

```
authorize FILE /tmp/Jo* uid(userJo) acc(Update)
```

Note: CA Access Control protects only those files defined in its database.

Restrict Users from Getting File Information

If you do not provide users with *read* access permissions to a file or directory, by default, they can still use the stat function to get information about the file. For example, a user without *read* access permissions to file /tmp/abc can perform the following operation:

ls -l /tmp/abc

To prevent users who do not have *read* access permissions from getting file information, set the STAT_intercept configuration setting to 1.

Note: For more information about the STAT_intercept configuration setting, see the *Reference Guide*.

Viewing Default Access Authority

To view the default access of users in the _restricted group (when no matching records are found), use the selang showres command with the _default record of the class.

For example, to view the default access that users in the _restricted group have for files that are not in the CA Access Control database, use the showres selang command to display the _default resource of FILE class:

showres FILE _default

Note: All other users have the access defined by specific CA Access Control database rules.

Using Conditional Access Control Lists

You can make access to a file conditional on the program used to access the file. To make file access conditional in this way is called program pathing.

Note: If the program specified to access the file is a shell script, the shell script must have #!/bin/sh as its first line.

The following code is an example, allowing any process to update the file /etc/passwd under the control of the password change program /bin/passwd. All access attempts to the /etc/passwd file that do not originate from /bin/passwd are blocked.

newres FILE /etc/passwd owner(nobody) defaccess(R)
authorize FILE /etc/passwd gid(users) access(U) via(pgm(/bin/passwd))

The newres command defines the file /etc/passwd to CA Access Control and allows any user, including the file's owner, to read the file. The authorize command allows all users to access the file when the access is made under the program /bin/passwd. Once the password file is protected in this manner, any Trojan horse that inserts entries into the /etc/passwd file or any update to the password file by a user of the group "users" is blocked if the user is not using the /bin/passwd program.

Conditional access lists are also useful for controlling access to the files of a database management system (DBMS). Usually, you should permit users to access such files only through the programs and utilities supplied by the database vendor. Consider the following commands:

authorize FILE /usr/dbms/xyz uid(*) via(pgm(/usr/dbms/bin/pgm1)) access(U)
authorize FILE /usr/dbms/xyz uid(*) via(pgm(/usr/dbms/bin/pgm2)) access(U)

This set of authorize commands allows all CA Access Control users to access the file xyz of the DBMS system provided the access is made by either program pgm1 or program pgm2, which belong to the DBMS binaries directory. Note the use of the asterisk in the user operand. The asterisk specifies all users who are defined to CA Access Control. The use of the asterisk is similar in concept to the default access, except that default access also applies to users who are not defined to CA Access Control. Note that you can use the _undefined group for users not defined in the CA Access Control database.

You can also use the Unicenter TNG calendar ACL property to permit or deny access to specific users and groups for the current resource according to the Unicenter TNG calendar status. There are two types of ACL properties for Unicenter TNG calendars: regular and restrictive.

For example, the following command adds a user named george to a conditional access control list for a regular calendar named basecalendar:

auth file file1 uid(george) calendar(basecalendar) access(rw)

And the following command removes a user named george from the Unicenter TNG calendar:

auth- file file2 uid(george) calendar(basecalendar)

Using Negative Access Control Lists

You can deny a user or group specific access types using a Negative Access Control List (NACL).

With the CA Access Control language (selang), use the following command to deny access:

auth className resourceName [gid(group-name...)] \
[uid({user-name...|*})] [deniedaccess(accessvalue)]

Blocking Trojan Horses with the _abspath Group

Any relative path names in the $PATH variable, but particularly the dot (.) path name meaning "current directory," is a security weakness. Consider the following scenario:

- At the top of the PATH variable for root is the current (.) directory.

- A malicious user creates a destructive program-a Trojan horse-and stores it as /tmp/ls.

- In time, as the malicious user expects, root issues the ls command in the /tmp directory. Instead of running the usual ls command, root actually runs-with full administrative privileges-the Trojan horse that had been stored in the /tmp directory.

To eliminate this security weakness, CA Access Control provides a user group named _abspath. All members of the _abspath group are forbidden to use relative path names in invoking programs.

You can add a user to the _abspath group just as you add one to any other group. Effective at the next login, the user is forbidden to use relative path names when accessing programs.

Synchronization with Native UNIX Security

Although CA Access Control permissions are more complex than native UNIX permissions, you can synchronize your native UNIX permissions to your CA Access Control permissions. That is, you can make the permissions coincide. However, the synchronization is subject to some limitations:

- Synchronization is not retroactive. Once it is in effect, it can govern all newly issued CA Access Control authorization commands, but it does not govern pre-existing access rules.

- Permissions that you grant in CA Access Control can be passed to UNIX, but permissions granted in UNIX are not passed to CA Access Control.

- Because of limitations in its own system of permissions, UNIX may be unable to adopt more than a simplified form of the CA Access Control permissions. Even UNIX versions that feature access control lists (ACLs) may be unable to reflect all the complexity of the CA Access Control ACLs.

UNIX platforms with ACLs that can be synchronized to CA Access Control are Sun Solaris, HP-UX, and Tru64.

Without such ACLs, you can still synchronize the traditional UNIX rwx permissions to the CA Access Control permissions, to the extent possible.

Synchronization is controlled by the combination of the authorize command's UNIX option and the seos.ini file's SyncUnixFilePerms token:

■ By including the UNIX option, the authorize command calls for implementation in UNIX as well as in CA Access Control. The command can even grant UNIX permission where permission did not exist before.

(When the UNIX option is *not* used, selang commands have no effect on UNIX security. Moreover, where UNIX retains a prohibition, a CA Access Control permission is not effective. So the only way that selang can overcome a UNIX prohibition is with the UNIX option of the authorize command.)

■ In the authorize command, the UNIX option works only when the SyncUnixFilePerms token is appropriately set in the [seos] section of the seos.ini file. The token has several permitted values:

– **no** specifies not to synchronize ACL permissions. This is the default value.

– **warn** specifies not to synchronize ACL permissions, but to issue a warning if the CA Access Control and native UNIX permissions conflict.

– **traditional** specifies to adjust the rwx permissions for the group according to the CA Access Control ACL (and permissions for individual users are not copied to UNIX).

– **acl** specifies to adjust the UNIX ACL according to the CA Access Control ACL.

– **force** specifies to adjust the UNIX world access attribute according to the CA Access Control defaccess permissions.

Any change in the SyncUnixFilePerms token value takes effect only after you restart the seosd daemon.

Example: Synchronization

The following example involves a file named /var/temp/newdata and a user named fowler, and assumes that a record in the FILE class already represents the file.

1. Shut down the seosd daemon, so you can edit the seos.ini file:

   # secons -s

2. Logged in as a user with permission to edit the seos.ini file, edit the seos.ini file to make the SyncUnixFilePerms line, in the [seos] section, look like this:

   SyncUnixFilePerms = acl

   Remember, acl means that the UNIX option adjusts the UNIX ACL according to the CA Access Control ACL. The UNIX option will have this function as long as the token remains set to acl.

3. Restart the seosd daemon:

   # seosd

4. Invoke selang, then issue the following selang command:

   authorize FILE /var/tmp/newdata uid(fowler) access(r w) unix

   The command gives fowler Read and Write access to the new data file and, by specifying the UNIX option, it grants the corresponding native UNIX permissions.

HP-UX Limitations

The ACL of HP-UX is limited in how it can reflect the ACL of CA Access Control.

- In HP-UX, the ACL assigns access per user and group *combination*. That is, the assigned access applies to the specified user only when the user's primary group is also specified.

  CA Access Control, on the other hand, assigns access per user or per group, but not per combination.

  Accordingly, CA Access Control permissions are mapped to HP-UX user/group combinations in which either the user or the group is set to the equivalent of "*" or "any."

- HP-UX does not support ACLs on file-systems that are under control of the volume manager (LVM). Thus, some important HP-UX machines are likely to allow ACL synchronization only on the "root" file-system.

- The ACL of HP-UX is limited to 16 entries. CA Access Control synchronization uses the available entries as efficiently as possible, but 16 entries may not be enough to reflect every CA Access Control ACL completely.

Sun Solaris Limitations

Under Sun Solaris, native UNIX ACLs are not implemented in the /tmp directory.

Monitoring Sensitive Files

The Watchdog can protect the binaries of your setuid/setgid programs, as well as any other files you specify. The seoswd utility (the Watchdog daemon) continually checks two issues:

- Whether the seosd daemon is alive and responding. (If necessary, the watchdog daemon restarts the seosd daemon.)

- Whether a user has modified any trusted programs or files. (If so, seoswd prevents these files from executing.)

When the seosd daemon forks, it automatically executes the seoswd program to start the Watchdog.

Note: For more information about seoswd, see the *Reference Guide*.

The seos.ini file contains several tokens that control the scanning and time-out values of the watchdog. It also contains the most up-to-date documentation on these values.

Note: For a description of the seos.ini file, see the *Reference Guide*.

You can use the Watchdog to perform the same background checks as those made for the setuid and setgid programs on ordinary files, including generating audit records when these files are altered.

For example, consider a configuration where only the security administrator is allowed to modify the file /etc/inittab. To make CA Access Control monitor the file and generate an alert in any case of modification, use the following command in selang:

newres SECFILE /etc/inittab

The file /etc/inittab is now constantly monitored for modifications.

Protecting setuid and setgid Programs

Set user ID (setuid) programs are among the most frequently used programs at a UNIX site. A process that invokes a setuid program automatically acquires the identity of the owner of the setuid program. If the owner of a setuid program is root, then any regular user automatically becomes superuser by invoking the setuid program. When the setuid program starts, the process can do anything a superuser can do, so it is extremely important to make sure that setuid programs do exactly what they are supposed to do and nothing else. Back doors or shells within a setuid program grant the user access to everything on the system.

CA Access Control uses the PROGRAM class to protect setuid and setgid programs. Upon installation, CA Access Control permits any program execution by default. After defining trusted programs in the database, you can change the behavior of CA Access Control so that execution of a setuid or setgid program is prohibited unless the program is defined as a trusted program. For example, to allow /bin/ps (the process status program) to run as a setgid program (as it is supposed to), use the following selang command:

newres PROGRAM /bin/ps defaccess(EXEC)

CA Access Control registers the program /bin/ps as a trusted program. It then calculates and stores its CRC, inode number, size, device number, owner, group, permission bits, last modification time, and, optionally, other digital signatures in a record in the PROGRAM class of the database.

The Watchdog periodically checks the program's CRC, size, inode, and the rest of the characteristics. If any of these values have changed, the Watchdog automatically asks seosd to remove the program from the trusted programs list and deny access to it. This ensures that no one can misuse the program by modifying or moving setuid programs. Note that the permission in the example newres command allows all users, including those not defined in the database, to run the /bin/ps command.

Untrusted setuid programs are possibly the most dangerous security loophole of UNIX-based operating systems. By using trusted programs' access rules, the security administrator can restrict the use of setuid to certain trusted programs that were tested and checked to ensure their integrity. However, any user cannot automatically start a trusted executable; the access rule must specify explicit users and groups that are granted access to that setuid program. For example, the following set of selang commands grants the execution of /bin/su only to the System Department users (group sysdept):

```
newres PROGRAM /bin/su defaccess(NONE)
authorize PROGRAM /bin/su gid(sysdept) access (EXEC)
```

Use an asterisk (*) to specify all users who are defined in the database. For example, to permit all users who are defined to CA Access Control to perform the su command, enter the following command:

```
authorize PROGRAM /bin/su uid(*) access(EXEC)
```

This description is also true for setgid executables.

You can use the nr and er commands to register the setuid and setgid programs in the PROGRAM class. Important non setuid and setgid programs can be registered in the PROGRAM class similarly. Define a FILE rule for these programs to prevent unauthorized users from upgrading them. If you want to allow the program execution when it is untrusted (after upgrade, the program is executed without being retrusted), set the blockrun property to no.

- If the blockrun property is set to yes, the program is not executed until it is re-trusted and is not allowed to access any file that the relevant PACL would allow. The PACL is effectively disabled until the program is re-trusted.

- If the blockrun property is set to no, the program is executed. However, the program cannot access any resources the relevant PACL would allow.

To set the value of the blockrun property to yes, use the following editres/newres command:

```
er program /bin/p blockrun
```

To set the value of the blockrun property to no, use the following editres/newres command:

```
er program /bin/p blockrun-
```

By default, for all the programs registered in the PROGRAM class, the blockrun property is set to yes. You can change this using the SetBlockRun token in the seos.ini file. Refer to the seos.ini file description for details.

Define setuid/setgid Programs Automatically

CA Access Control provides a way to define all your setuid and setgid programs automatically. Use the utility program /bin/seuidpgm to build the set of commands to define all the setuid programs and their permissions.

For example, to scan the entire file system for setuid and setgid programs and write the generated selang commands to the file /tmp/pgm_script, enter the following selang command:

```
# seuidpgm -qln / -x /home > /tmp/pgm_script
```

You can edit and modify the output file generated by seuidpgm according to your needs before submission.

Note: For more information about the seuidpgm utility, see the *Reference Guide*. To learn how to give similar protection to programs that are neither setuid nor setgid programs, see the SECFILE class in the *Reference Guide*.

Conditional Access

Another sophisticated permissions technique is the conditional access rule. For example, suppose you have a very secure version of the su command called securedSU that uses a fingerprint reader to verify the user's identity before allowing the user to become a superuser.

One way to ensure that *UserX* can become superuser only under that program is to set a conditional access rule as follows: (Before setting the rule, you must also set defaccess(none) for USER.root.)

```
authorize SURROGATE USER.root uid(UserX) via(pgm(securedSU))
```

Protecting the Login Command

We strongly recommend that you limit the use of /bin/login to the superuser only. Otherwise, any user who knows another user's password can log in as another user and supply the other user's password to bypass all surrogate and terminal restrictions.

To change the /bin/login permissions in selang, use the following command:

chres LOGINAPPL /bin/login defaccess(N) owner(root)

Protecting Regular Programs

CA Access Control can also protect regular programs in the same way it protects setuid and setgid programs. To do this, set the blockrun property in the PROGRAM class to the value you choose.

Note: For more information about possible options, see the *Reference Guide*.

Kernel Modules Load and Unload Protection

A *kernel module* is a component of the UNIX operating system that you can load to extend the running kernel, and unload when no longer required. This adds flexibility, letting you load functionality as required, without wasting memory resources that would otherwise be required to cover all possible expected functionality in the base kernel.

You can disable and enable kernel module protection in CA Access Control. If you enable kernel module protection, CA Access Control intercepts the system calls that load and unload a kernel module, and then checks the requested access against the associated record in the database, which is a record of class KMODULE. When access is requested for a kernel module record, CA Access Control, the requested access is either "load" or "unload".

On all non-Linux systems, the name of the KMODULE record must match the name of the kernel module file (not the full path). This is because the name of the module is the same as the name of the file. On Linux, the name of KMODULE record needs to match only the name of the kernel module, which, may be different from the actual file name. Changing the file name on Linux does not change the module name which Linux uses and the KMODULE record remains valid.

If you enable file path checking on kernel module loads and the requested access is load, CA Access Control performs the following additional checks:

- The filepath property in the KMODULE record holds only valid absolute file paths.

- The files in the path name filepath have modules that match the KMODULE record name.

- The kernel module matches the KMODULE properties (filepath for non-Linux systems, signature for Linux systems).

  Note: CA Access Control produces a unique signature for kernel module file on Linux systems, and inserts this as the value of the signature property in the kernel module record. CA Access Control checks the signature on each access. You do not need to enter the signature yourself, because CA Access Control calculates and inserts it automatically. However you can do so using the seretrust utility.

More information:

Enable and Disable File Path Checking on Kernel Module Loads (see page 96)

Protect a Kernel Module

You can protect the loading and unloading of kernel modules, and so help protect the operating system.

To protect a kernel module

1. Ensure you have enabled kernel module protection.

2. Create a KMODULE record in CA Access Control.

   To create a kernel module, you need to define:

   - The name of the kernel module

     On all non-Linux systems, the name of the KMODULE record must match the name of the kernel module file (not the full path). This is because the name of the module is the same as the name of the file. On Linux, the name of KMODULE record needs to match only the name of the kernel module, which, may be different from the actual file name.

   - The owner of the record (defaults to the user creating the module)

   - (Optional) The absolute file path to the kernel module file, or a list of file paths if there is more than one version of the module.

   Note: On HP and Solaris systems, you can define the special kernel module _ALL_MODULES to protect the unloading of all kernel modules.

3. Define the users or groups that are authorized to load and unload the module.

Example: Protect a Kernel Module Using selang Commands

The following selang commands define and authorize a kernel module serial.o to CA Access Control and authorizes the enterprise user kadmin to load and unload it:

```
newres kmodule serial.o owner(kadmin) defaccess(none) \
filepath(/lib/modules/2.2.19/serial.o:/lib/modules/2.2.20/serial.o)
authorize kmodule serial.o access(load, unload) xuid(kadmin)
```

Enable and Disable Kernel Module Protection

When kernel module protection is enabled, CA Access Control checks the loading and unloading of the kernel modules that are defined in the CA Access Control database.

By default, CA Access Control enables protection of kernel modules.

To enable or disable kernel mode protection, enable or disable the KMODULE class, for example by using the setoptions command.

Example: Enable Kernel Mode Protection Using selang

The following selang command enables kernel mode protection:

```
setoptions class+(kmodule)
```

Example: Disable Kernel Mode Protection Using selang

The following selang command disables kernel mode protection:

```
setoptions class-(KMODULE)
```

Enable and Disable File Path Checking on Kernel Module Loads

If kernel module protection is enabled, you can also enable file path checking on kernel module loading. When this is enabled, CA Access Control checks that the kernel module to be loaded matches the filepath property of the KMODULE record (for non-Linux systems), or matches the signature of the KMODULE record (for Linux systems).

To enable file path checking, in the seosd section of the configuration file seos.in, set the special_check token to yes (the default is no).

CA Access Control does file path checking only if file path checking and kernel mode protection are both enabled.

To enable file path checking for kernel module loads, you can use the seini and secons utilities as follows:

```
seini -s seosd.special_check yes
secons -rl
```

Protecting Binary Files from the kill Command

You must protect mission-critical processes, such as database servers or application daemons, against denial of service attacks. The native UNIX security system bases its process protection on the process user ID. This implies that under native UNIX, root can do anything to any process. CA Access Control adds to UNIX process protection by defining rules based on the executable file running in the process. CA Access Control process protection is *independent* of the user ID of the process. A record in the PROCESS class must define every process that CA Access Control will protect.

For example, to protect the ASCII viewer /bin/more from being killed, follow this procedure:

1. Start selang.

2. Enter the following selang command:

   ```
   newres PROCESS /bin/more defaccess(N) owner(nobody)
   ```

   This command defines /bin/more as a process to be protected from kill attempts; therefore the default access is *none* (N). The **owner(nobody)** setting ensures that even the user who defined this rule cannot kill the /bin/more process.

3. Exit selang.

4. Test the rules that Step 2 defined:

   a. Enter the command:

      ```
      /bin/more /tmp/seosd.trace
      ```

   b. Assuming the file /tmp/seosd.trace is large enough to keep /bin/more from exiting immediately, press Ctrl+Z to suspend the /bin/more process.

   c. Try to kill the suspended job by entering the command:

      ```
      kill %1
      ```

      Your attempt should fail, with CA Access Control displaying the "Permission denied" message.

To make an exception that permits a specific user to kill the /bin/more processes, enter the selang command:

authorize PROCESS /bin/more uid(*username*)

Note: Use the same procedure to protect other binary executables on your system from being killed.

CA Access Control protects regular kill signals (SIGTERM) and the kill signals that an application cannot mask (SIGKILL and SIGSTOP). It passes other signals, such as SIGHUP or SIGUSR1, to the process to determine whether to ignore or react to the kill signal.

Chapter 9: Controlling Login Commands

This section contains the following topics:

Controlling the Login Process

CA Access Control provides two types of login protection: by terminal, and by application. Using the TERMINAL class, you can establish which users can log in from which terminals or hosts.

Note: For more information about the TERMINAL class, see the *Reference Guide*.

You can also control which user or group can log in using a certain login application (such as telnet, ftp, and rlogin) with the LOGINAPPL class. By establishing the access rules of the class, you define specific rules for each login application. For instance, you can define rules that permit all users to ftp to your host, a limited number of users to telnet to your system, and no one to rlogin to the system. Each record in the LOGINAPPL class defines access rules for a specific login application.

Examples: LOGINAPPL

For example, to permit only an anonymous user to use the ftp application, use the following procedure:

1. Change the ftp default access to none with the following selang command:

   cr LOGINAPPL FTP defaccess(NONE) owner(nobody)

2. Permit the user anonymous to use ftp with the following selang command:

   auth LOGINAPPL FTP uid(anonymous) access(X)

To restrict users from the group named account to use only telnet:

1. Block the use of rlogin and rsh with the following selang command:

   auth LOGINAPPL(RLOGIN RSH) gid(account) access(N)

2. Permit the group named account to use telnet with the following selang command:

   auth LOGINAPPL TELNET gid(account) acc(X)

Note: The previous example shows RLOGIN and RSH restrictions, but other login programs should be included as well.

Whenever you add or use a new login program, you must add a new LOGINAPPL record.

The login interception sequence always starts with setgid or setgroup events, which are called *triggers*. The sequence ends with a setuid event that changes the user's identity to the real user who logged in.

Login applications issue a variety of system calls, which CA Access Control uses to monitor login activity. These login sequences are preset for standard login applications. You can see them by studying the CA Access Control trace file.

Note: For more information about the LOGINAPPL class and setting a sequence, see the *selang Reference Guide*.

Controlling Generic Login Applications

CA Access Control can also control and protect generic login applications; this means that you can protect groups of login applications that match a certain rule with a generic pattern. To define a generic login application, use the LOGINAPPL class.

Defining a Generic Login Application

To define a generic login application with selang, use the same commands as setting regular login restrictions, except for the LOGINPATH parameter, which should include a generic path composed of a regular expression using one or more of the following characters: [,], *, ?. For example, to define a generic telnet application, issue the following command:

er LOGINAPPL GENERIC_TELNET loginpath(/usr/sbin/in.tel*)

Generic Login Program Interception

With regular login restrictions, the activated rules are obvious; if a LOGINAPPL object that has the intercepted login program specified for the loginpath property exists in the database, the rules for that object would apply.

However, for generic LOGINAPPL objects, CA Access Control does the following:

1. seosd searches for an exact match for the intercepted login application. (A matching login path for the LOGINAPPL object.) If found, the rules for that object apply.

2. If not found, the search continues for a LOGINAPPL object with a generic login path that matches.

3. If there is more than one match, the rules for the object with the more specific match apply.

Defining User Authority to Use Terminals

One of the most effective ways to block intruders from accessing the system is by terminal protection, that is, the source of the login. The source can be the host or the terminal (such as an X terminal or a console) from which the user logs in.

In today's modern architecture, a terminal is no longer the teletype machine UNIX was developed for. On most sites, a "pseudo terminal" is allocated through the pseudo terminal server (PTS) or by the X window manager, and the terminal's name is meaningless symbol for the security system. CA Access Control protects what we understand as a terminal. CA Access Control implements terminal protection during the login stage, when CA Access Control defines a terminal in one of three ways:

■ When the user logs in from an X terminal using the XDM login window, CA Access Control takes the IP address of the X terminal translated to host name (from /etc/hosts, NIS, or DNS) to be the terminal used for the login request. CA Access Control can also protect using the IP addresses if the translation to the host name fails or if you prefer to use IP addresses.

■ When the user logs in from a dumb terminal, the TTY name identifies the terminal.

■ When the user logs in from the network (through telnet, rlogin, rsh, and so on), the requesting IP address translated to the host name (through /etc/hosts, NIS, or DNS) is taken to be the terminal name.

You can define login rules for a specific host by defining this host in the TERMINAL class and adding the appropriate users and groups to the object's access list. For each login source, you can also limit the days and hours in which login from this host or terminal is allowed by setting the day and time restrictions for the TERMINAL object. You can also use wildcards in the TERMINAL class to define hosts that match a pattern (host name or IP address).

In most cases, highly authorized users such as the superuser or system administrators must be restricted to terminals that are located in secure places. Intruders and hackers who wish to enter the system as superuser are not able to do it from their own remote stations; they have to work from one of the authorized terminals, which should be in a secured location.

When logging in from the network, you cannot be certain that the user is indeed sitting in front of the host console. The user could be sitting in front of any terminal attached to that host or communicating from any other node in the network authorized to receive services from the requesting host. Permitting a user to log in from another host implies that we permit login to that user not only from that specific station but also from any other terminal authorized by that station. To ensure isolation between departments, define terminal groups and allow users of each department to work only from the terminal group of their department.

Unlike other resources, in terminal authorizations the more the user is authorized to access information, the lower the user's terminal authorization should be. The superuser must be the most restricted user in terminal access to ensure that nobody can log in as root from remote unsafe terminals.

When defining terminals, CA Access Control requires you to explicitly specify the owner of the terminal definition. The reason is that if root, as the security administrator, becomes the owner of the terminal by default, it makes the terminal eligible for superuser login. In most cases, this is not wanted. To guard you from making such mistakes that may unintentionally cause loopholes, CA Access Control makes you define an owner when defining the terminal.

To define the terminal tty34, use the following command:

```
newres TERMINAL tty34 defaccess(none) owner(userA)
```

This command creates a record for the terminal tty34, sets its default access to NONE, and defines userA as its owner. Note that userA, as the owner of the terminal, is automatically allowed to enter the system through terminal tty34.

To prevent all users from logging in from the terminal tty34, specify "nobody" as the owner:

```
newres TERMINAL tty34 defaccess(none) owner(nobody)
```

To permit a user to log in from a particular terminal, enter the following command:

authorize TERMINAL tty34 uid(USR1)

This command permits USR1 to log in from terminal tty34.

Permission to use a terminal can also be granted to a group. For example, the following command permits members of the group DEPT1 to use the terminal tty34:

authorize TERMINAL tty34 gid(DEPT1)

To define a group of terminals (known as a terminal group), enter the following command:

newres GTERMINAL TERM.DEPT1 owner(ADM1)

To add member terminals to terminal group TERM.DEPT1, enter the following command:

chres GTERMINAL TERM.DEPT1 mem(tty34, tty35)

To authorize USR1 to use this terminal group, enter the following command:

authorize GTERMINAL TERM.DEPT1 uid(USR1)

This grants USR1 the authority to use both tty34 and tty35.

Restricting Terminals for Root Users

Another issue to consider is the default rule of the TERMINAL class. At the initial implementation stages, the default is set to permit anything that is not defined. In the case of a TERMINAL, this could be a shortcoming.

Consider the following situation: A site has a few hundred terminals, and you want most users to be able to log in from any terminal, but you want root to be able to log in only from two predefined terminals.

First we consider that setting the default of the TERMINAL class to READ enables anyone-including root-to log in from any terminal that does not have a specific TERMINAL record in the database. You do not want the superuser to be able to log in from any terminal. But, we also consider that setting the default of the TERMINAL class to NONE forces you to define each terminal in the database, which may be impractical.

To solve this problem, CA Access Control supports the definition of an access control list within the _default record of the TERMINAL class. The following commands show you how to restrict root to two terminals with minimum effort:

```
newres TERMINAL term1 defaccess(N) owner(root)
newres TERMINAL term2 defaccess(N) owner(root)
newres TERMINAL _default defaccess(R)
authorize TERMINAL _default uid(root) access(N)
```

The first two commands define term1 and term2 as terminals owned by root, so they are eligible for superuser login. The newres TERMINAL _default and chres commands set the default access to READ, so that any terminal not defined in the database is accessible to anyone. The authorize command explicitly denies access of the superuser to undefined terminals.

Note: The UACC class still exists; you can use it to specify the default access of a resource. However, using _default records to specify the default access of a resource is much easier.

Recommended Restrictions

You should restrict the use of the loopback terminals, local host terminals, and station host names if the default access for the TERMINAL class is READ. Allowing users to use these terminals permits all other users to substitute their own user IDs if they know the target user's password. For example, consider the following scenario:

- User U is allowed to work from terminal T.

- Terminal T is not allowed for superuser login.

- User U is not authorized to substitute user ID to root.

- User U managed to get the superuser password.

- All users are permitted to log in from terminal loopback.

User U can bypass this set of access rules by simply performing the command telnet loopback, specifying the user ID root, and supplying the password. Now a superuser session has started from terminal T, which is not supposed to allow superuser login. A user can similarly bypass access rules by exploiting the local host or the station's host name.

To restrict these three vulnerabilities, use the following definitions:

```
newres TERMINAL loopback defaccess(N) owner(nobody)
newres TERMINAL localhost defaccess(N) owner(nobody)
chres TERMINAL hostname defacc(N) owner(nobody)
```

An alternative approach to preventing this security breach is to limit the TCP requests for telnet, ftp, and so forth from local host.

Yet another option is to set default access for the TERMINAL group to NONE, then specify TERMINAL and GTERMINAL rules.

Password Checking and Login Restrictions

CA Access Control does not replace the /bin/login executable. Even when CA Access Control is running, passwords continue to be checked against /etc/passwd, the shadow password file, or the NIS passwd map. But CA Access Control also performs additional checks, described in the following section.

Logon Checks

After the login process passes the authentication stage, CA Access Control intercepts the process and checks the following points:

- Has the password expired?

  If it has, the user receives a number of grace logins accompanied by warnings before being denied access. Following access denial, the security administrator must reassign the user's password. The number of grace logins is determined by the user password policy, which you can specify either globally with the setoptions command, or for a profile group with the chgrp command.

  Note: For more information about the setoptions command, see the *Reference Guide*.

  You can use the segrace utility to view the number of grace logins left for a user, the number of days remaining until the user's existing password expires, or the date and time the user last logged on and from which terminal.

  Note: For more information about the segrace command, see the *Reference Guide*.

- Is the user logging on from an authorized terminal?

  If so, login proceeds normally to the next check; if not, the user cannot log in.

- Do the current time-of-day and day-of-week allow login (per the predefined restrictions)?

  If they do, login proceeds normally to the next check; otherwise, the user cannot log in.

- Was this user name unused for more than a predefined number of days?

  If it was, access is denied. (The default is 90 days; use the setoptions command to change it.)

Defining Time and Day Login Rules

Information security is most vulnerable in times of low activity. Late hours of the night and weekends are ideal times for breaking in, because fewer people are available to monitor the audit records. Setting up appropriate terminal authority rules forces an intruder to use a terminal that is in a protected location. Setting up days-of-week (DOW) and time-of-day (TOD) access rules forces the intruder to make break-in attempts during work hours when offices are open and active. This combination severely restricts alien break-ins.

Limiting the days and hours in which a user can log in is done on a user-by-user basis. To define the DOW and TOD login restrictions for a user, use the following command:

chusr USR1 restrictions(days(Mon,Tue,Wed)time(800:1700))

This command permits user USR1 to log in only between 8:00 and 17:00 on Mondays, Tuesdays, and Wednesdays. USR1 cannot log in outside the specified time on the specified days, or on days other than those specified.

The days parameter also accepts the values ANYDAY (allow logins on all seven days of the week) and WEEKDAYS (allow logins Monday through Friday). The time parameter also accepts the value ANYTIME (allow logins at any time of the day).

Note: You can apply the DOW and TOD restrictions to *many* resources defined in the database. This feature is particularly useful for giving terminals and terminal groups limited periods of usability.

Disabling Concurrent Logins

Most UNIX-based operating systems allow concurrent logins. But if a user is permitted to log in from more than one terminal, there is a danger that while the user is logged in, other users can log in from elsewhere and masquerade as that user.

After you log in, CA Access Control allows you to disable your own concurrent login permission so that no one else can log in as you from another terminal. However, you can still log in repeatedly from the particular terminal that you are using. Use the secons command with the following switches:

```
# secons -d-          (disables concurrent login)
# secons -d+          (enables concurrent login)
```

Any user can issue the -d option. (All other options are only allowed for users with the ADMIN or OPERATOR attribute). Users who want to disable concurrent logins can use this command in their initial scripts. Although they are then able to open as many windows as they want, they cannot log in from a second terminal.

Note: If you use the secons -d- command to prevent concurrent logins, you must remember to use secons -d+ before logging out, to avoid being locked out of the system. If you forget to reinstate concurrent logins and try to log in again, CA Access Control allows you to log in provided no process with the same user ID is running.

Limiting Concurrent Logins for a User

CA Access Control can control the number of concurrent logins in two ways:

Administrator Level

Set a systemwide definition in the database of the number of concurrent sessions a user can have. You can set this value globally, for a profile group, or for individual users.

User Level

Users individually control the number of concurrent logins allowed for them. This way, when logging in, users can block the option of more login sessions with their names, thus protecting themselves.

Note: The number of concurrent logins is independent of the number of sessions the user is running on a particular terminal. Multiple sessions on one terminal are considered as a single login. The concurrent-logins limit restricts the number of *terminals* a user can concurrently log in from, not the number of logins from each terminal.

Limiting Concurrent Logins Globally

In selang, enter the following command:

setoptions maxlogins(*NumLogins*)

Limiting Concurrent Logins Individually

In selang, enter the following command:

chusr *username* maxlogins(*NumLogins*)

The concurrent logins limit set for a user overrides the systemwide limit. To prevent CA Access Control from enforcing the concurrent logins limit for a specific user, set the user's concurrent logins limit to zero. (Note that you cannot use selang if you set the maximum number of concurrent logins to one.)

Recognizing a Login Event

CA Access Control does not treat all attempts to change the user ID of a process as login events. Usually a program attempts to change its user ID with a setuid system call. The SURROGATE class controls these events, which are not necessarily considered login events, and do not necessarily change the user identity from the point of view of CA Access Control.

CA Access Control always preserves the original user identity-the identity with which the user logged in initially. Ordinary setuid system calls do not cause CA Access Control to register a change in user identity.

For CA Access Control to recognize the identity change, it must recognize this event as a login event. It recognizes login events using the following rules:

- The program that attempts to change the identity is defined as a *login program*. All programs in the LOGINAPPL class are login programs.
- The program executes a series of system calls corresponding to its definition in the LOGINAPPL class.

When you begin an administration session (in selang or CA Access Control Endpoint Management), CA Access Control performs a dummy login event. This is not a true login; rather, CA Access Control performs certain internal checks, which are similar to login checks.

Note: For more information, see the SEQUENCE property for the LOGINAPPL class in the *selang Reference Guide*.

At the start of an administration session, the user name is checked in the machine to be administered. You get access to this machine for administration only if you have WRITE access for the terminal from which you perform the session.

For example, if you are logged in to host Minerva and would like to administer CA Access Control on host Artemis, two conditions are necessary:

■   A TERMINAL object called Minerva (or the relevant fully qualified name) is in the database record for Artemis.

■   You are listed in the ACL of this object with WRITE permission.

These conditions are checked prior to any other user authority check. Note that you also need administrative authority in the database.

Chapter 10: Protecting TCP/IP Services

Protecting TCP/IP services is most important for file servers that contain sensitive data. These servers must provide certain services only to trusted stations, and not to intruders or computers that are unknown to the host.

This section contains the following topics:

Restricting TCP/IP Services

In an open network, any station can request services from other computers on the network. The TCP/IP protocol can be used to supply many services. Some of these services, such as rlogin, rcp, rsh, ftp, telnet, and rexec, are common to all UNIX-based operating systems. Others are provided by in-house and third-party software.

CA Access Control intercepts the accept processes of TCP/IP at the host computer and determines whether the accept program should continue normally or be overridden. CA Access Control bases its decision on access rules governing hosts and services that you define. You can create TCP/IP access rules in the database to specify the computers and networks that are allowed to receive services such as file transfers, remote login, and remote shell from a specific computer.

The following examples show how TCP/IP access rules can be defined and set to efficiently block unwanted outsiders. If you have not yet had time to develop a complete database, you may want to let any station that is not defined in the database receive any service. If so, set the HOST record in the UACC class as follows:

```
chres UACC HOST defaccess(READ)
```

A station that is to have access rules for TCP/IP services from the local host is defined in a record in the database under the HOST class. For each of these stations, the services allowed are listed in the record. For example, the following command sequence defines a record for station ws5 and denies it from receiving any TCP/IP service from the local host:

```
newres HOST ws5
authorize HOST ws5 service(*) access(NONE)
```

The following command allows ws5 to perform telnet to the local computer:

authorize HOST ws5 service(telnet)

These settings allow users to telnet to the local computer, which means that the remote user must specify a user name and password before using the local system. To allow a station to receive all TCP/IP services from the local computer, you can use an asterisk in the service keyword. For example, the following command allows ws5 to invoke any TCP/IP service from the local computer:

authorize HOST ws5 service(*)

The service can be specified in several ways, some of which involve the *port number*. The port number is an identification number for a service. All services have port numbers, and the port numbers are mapped to the services in the file /etc/services. You can specify a service in the following ways:

- By its name as defined in the file /etc/services

- By its port number

- As a range of port numbers

- As an RPC port that is listed in the /etc/rpc system file

For example, the following command permits ws5 to receive any TCP/IP service whose port number falls between 7045 and 7050:

authorize HOST ws5 service(7045-7050)

In many cases, it is more economical to define a group of hosts and set its permissions once, instead of making permissions for each individual computer. CA Access Control provides the GHOST class, where each GHOST record defines a group of hosts. To define a GHOST record and add hosts to its member list, enter the following commands:

newres GHOST gh1 mem(ws2, ws3, ws5)
authorize GHOST gh1 service(ftp)

The newres command defines a group of hosts called gh1 that contain the members ws2, ws3, and ws5. The authorize command allows all three stations to receive ftp (file transfer) services.

Managing host groups is easier than managing individual stations, but to supply more flexibility, CA Access Control also supports the definition of network access rules. Networks are defined in the HOSTNET class. For example, consider the following set of commands:

```
newres HOSTNET hn1 mask(255.555.0.0) match(192.168.0.0)
authorize HOSTNET hn1 service(*) access(NONE)
authorize HOSTNET hn1 service(ftp)
```

- In the first line, the newres command, defines a network called hn1. With its mask and match values, it specifies that any computer with an IP address whose first two qualifiers are 192.168 is considered as coming from the hn1 network.

- The combination of the second and third lines permits any station from the hn1 network to perform ftp, but not any other service, in the host computer.

Another method CA Access Control provides for defining TCP/IP access rules is name-pattern access rules. CA Access Control supports the definition of generic records in the HOSTNP class (host name pattern) with wildcards.

Note: For information on how CA Access Control performs string matching, see the *selang Reference Guide*.

For example, the following command sequence permits all hosts whose names start with the characters "lin" and end with the characters ".org.com" to receive all TCP/IP services on the local host:

```
newres HOSTNP lin*.org.com
authorize HOSTNP lin*.org.com service(*).
```

Note: Hosts that are managed by NIS must be identified by their official names that appear in a NIS map and not by their aliases. The chart in the following section summarizes the TCP/IP check flow.

Using the TCP Class

Alternatively, you can specify protection by service instead of by host, by using the TCP class.

Note: For more information about the TCP class, see the *Reference* Guide.

Use the TCP class to control incoming *and* outgoing services.

For example, the following commands create a record for the ftp service, with READ (meaning the service can be used) as default access type, but prevent hosts that match the name pattern PUBLIC* from receiving the service.

```
newres TCP ftp defaccess(READ)
authorize- TCP ftp hostnp(PUBLIC*) access(N)
```

You can also specify that a particular user or group be only permitted to receive a particular service. For example, to allow all users to ftp to a host called hermes, but to specify that only members of the group called acctng can access hermes with telnet, enter the following commands:

```
newres HOST hermes
newres TCP ftp owner(nobody) defaccess(read)
newres TCP telnet owner(nobody) defaccess(read)
authorize TCP ftp uid(*) host(hermes) access(write)
authorize TCP telnet gid(acctng) host(hermes) access(write)
```

Note: defaccess(read) disables outgoing services. defaccess(write) disables incoming services.

If the HOST class is active (that is, if it is used as a criterion for access), then the TCP class cannot effectively be active. You can use the command setoptions class- HOST to deactivate the HOST class; then use the command setoptions class+ TCP (if necessary) to activate the TCP class. Deactivating the HOST class automatically deactivates GHOST, HOSTNET, and HOSTNP as well.

Also, if the TCP class is active, use the setoptions command class- CONNECT to deactivate the CONNECT class.

Streams Module for Network Interception

By default, the TCP class is not active. Before you activate the TCP class, the CONNECT class, or the HOST class, be sure that the streams module is enabled.

To load the CA Access Control streams module on Solaris, complete the following steps:

1. Stop CA Access Control. Enter the following command:

   secons -s

2. Enter the following command:

   SEOS_load -s

3. Start CA Access Control. Enter the following command:

   seload

Note: If you attempt to activate the TCP class when the streams module is not loaded, an error appears:

ERROR: *className* class cannot be activated when streams are not loaded.

Please use SEOS_load -s to load the streams.

The algorithm for incoming authorizations is:

```
                    ┌─────────────┐
                   (   Diagram     )
                   (      A        )
                    └─────────────┘
                           │
                           ▼

        ┌────────┐                      ┌────────┐
       ╱  Does    ╲        yes         ╱   Use    ╲
      ╱ service TCP╲──────────────────╱  service   ╲
      ╲   record   ╱                  ╲   record    ╱
       ╲  exist?  ╱                    ╲    ACL     ╱
        └────────┘                      └────────┘
            │                                │
           no                               │
            │                               │
            ▼                               │
        ┌────────┐                          │
       ╱   Use    ╲                         │
      ╱ _default   ╲                        │
      ╲  record    ╱                        │
       ╲   CACL   ╱                         │
        └────────┘─────────────────────────┘
            │
            ▼
        ┌────────┐
       ╱   Does   ╲
      ╱   HOST     ╲
      ╲  record    ╱
       ╲  exist?  ╱
        └────────┘
            │
            ▼
        ┌────────┐
       ╱  See the ╲
      ╱  previous  ╲
      ╲  diagram   ╱
       ╲          ╱
        └────────┘
```
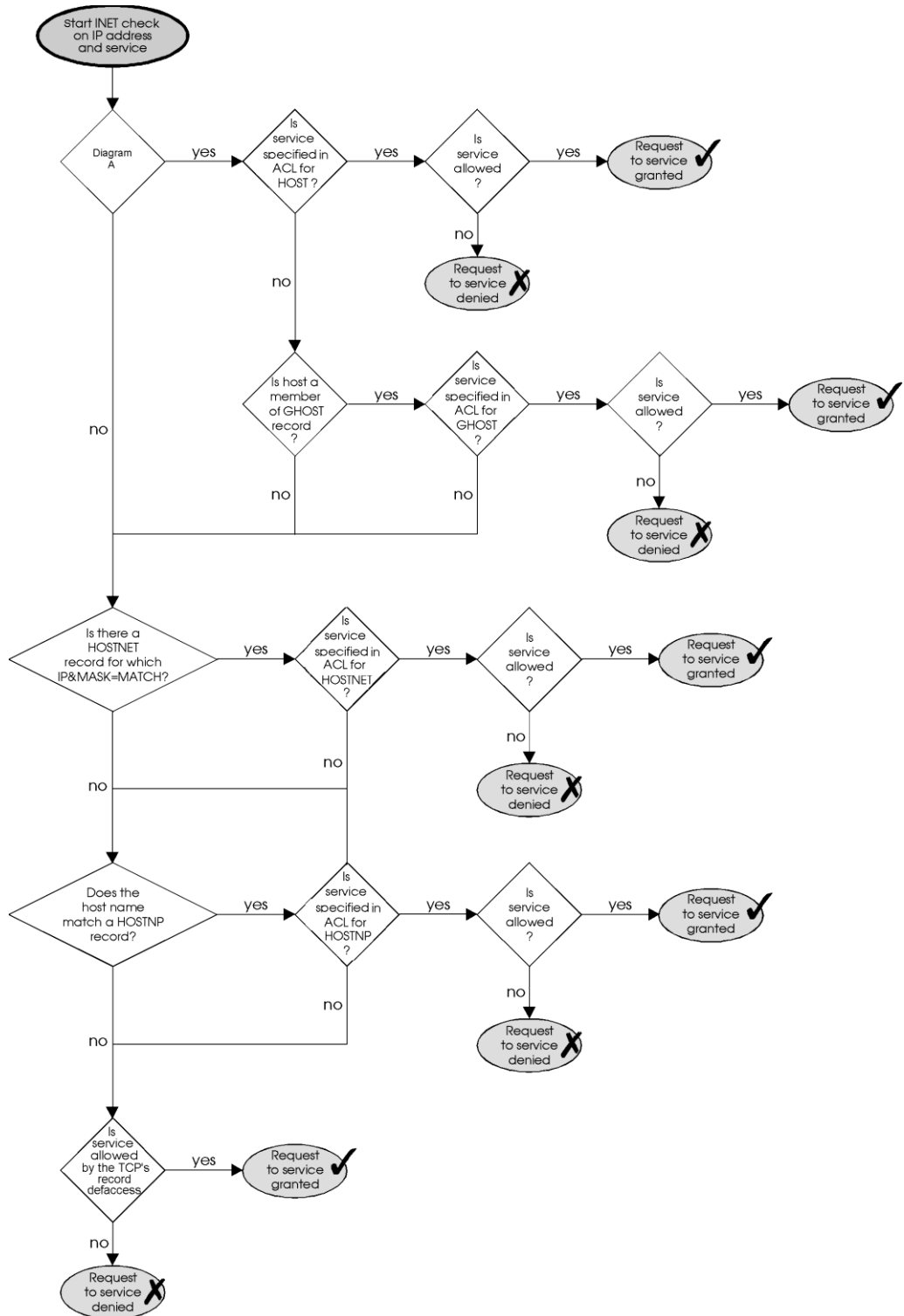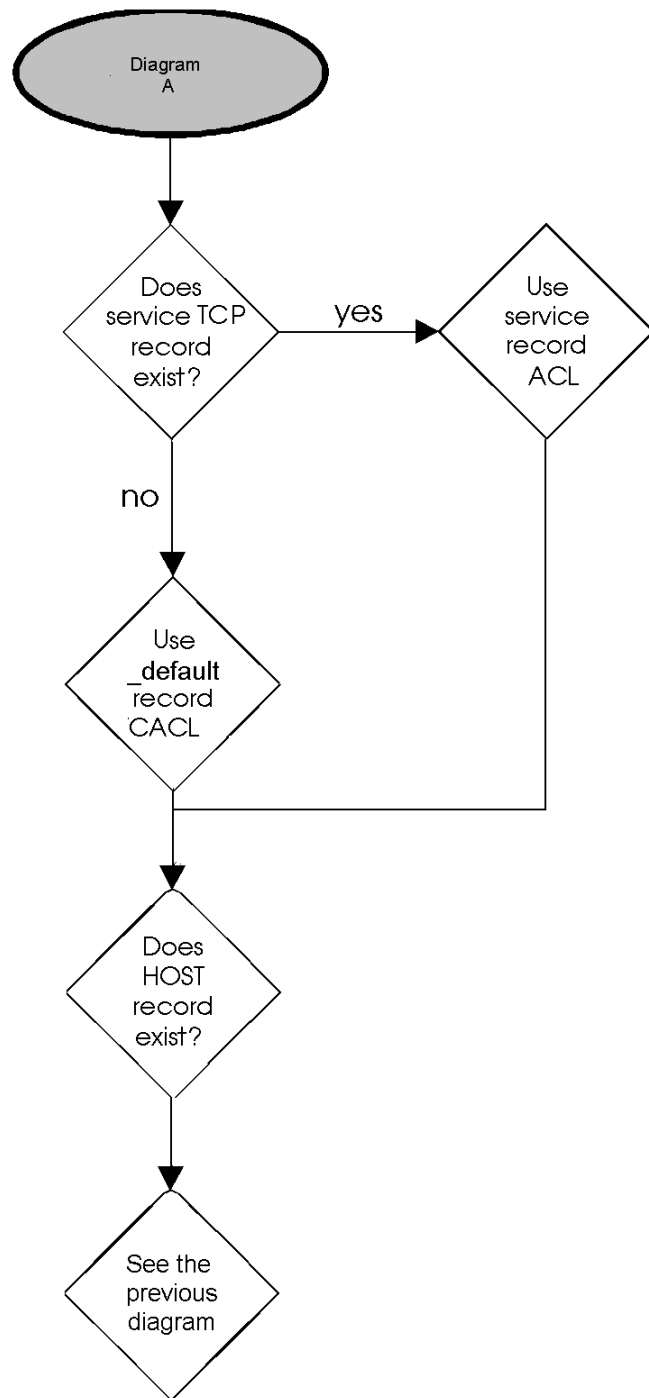
The algorithm for outgoing authorizations is:

start INET check
on IP address
and service

Does
service TCP
record
exist ?
— yes → Use
service
record
ACL

no

Use
_default
record
CACL

Is user
defined
in SeOS?
— yes → Does user
appear
in CACL?
— yes → Use diagram B for
CACLs in which
user appears

no   no   no

Do any of the
groups which the user
belongs to appear
in the CACL?
— yes → Use diagram B for
CACLs in which
groups appear

no

Does user*
appear
in CACL?
— yes → Use diagram B for
CACLs in which
user* appears

no   no

Is
service
allowed
by the TCP's
record
defaccess
?
— yes → Request
to service
granted

no

Request
to service
denied

Diagram B

Does HOST record exist?
— yes → **Is service specified in CACL for HOST ?**
— yes → **Is service allowed ?**
— yes → **Request to service granted** ✔
— no → **Request to service denied** ✘

Is service specified in CACL for HOST ? — no →

Is host a member of GHOST record ?
— yes → **Is service specified in CACL for GHOST ?**
— yes → **Is service allowed ?**
— yes → **Request to service granted** ✔
— no → **Request to service denied** ✘

Is host a member of GHOST record ? — no
Is service specified in CACL for GHOST ? — no

Does HOST record exist? — no →

Is there a HOSTNET record for which IP&MASK=MATCH?
— yes → **Is service specified in CACL for HOSTNET ?**
— yes → **Is service allowed ?**
— yes → **Request to service granted** ✔
— no → **Request to service denied** ✘

Is service specified in CACL for HOSTNET ? — no
Is there a HOSTNET record for which IP&MASK=MATCH? — no →

Does the host name match a HOSTNP record?
— yes → **Is service specified in CACL for HOSTNP ?**
— yes → **Is service allowed ?**
— yes → **Request to service granted** ✔
— no → **Request to service denied** ✘

Is service specified in CACL for HOSTNP ? — no
Does the host name match a HOSTNP record? — no →

See the previous diagram

Chapter 11: Managing Policy Models

This section contains the following topics:

The Policy Model Database

Managing tens or hundreds of databases individually is not practical. CA Access Control supplies the Policy Model service, a component that lets you manage many databases from one central database. Using the Policy Model service is optional, but it greatly simplifies administration at large sites.

The Policy Model (PMD) service uses a Policy Model database (PMDB). Like other CA Access Control databases, the PMDB contains users, groups, protected resources, and rules governing access to the resources. In addition, the PMDB contains a list of *subscriber* databases. Each subscriber is a CA Access Control database that resides on a separate computer, or another PMDB that resides on the same or another computer. A PMDB that updates a subscriber is the subscriber's *parent*.

The PMDB is a useful tool for managing many databases that have similar authority restrictions and access rules.

Note: For information about administrating a PMDB (sepmd utility), see the *Reference Guide*. For information about managing PMDBs remotely using selang, see the *selang Reference Guide*.

PMDB Location on Disk

All PMDBs reside in a common directory (one per computer). The name of the directory is specified by the _pmd_directory_ token in the [pmd] section of the seos.ini file. The default value of _pmd_directory_ is *ACInstallDir*/policies, where *ACInstallDir* is the installation directory for CA Access Control (by default /opt/CA/AccessControl).

Each PMDB occupies a subdirectory in the common directory. The name of the subdirectory is the name of the Policy Model. The files in the subdirectory contain all the data required to define the Policy Model including the pmd.ini file.

Managing Local PMDBs

CA Access Control offers several utilities for administrating local PMDBs:

sepmd

A PMDB administration utility that lets you:

- Administer subscribers
- Truncate the update file
- Administer Dual Control
- Manage the Policy Model log file
- Perform other administrative tasks

sepmdadm

Creates PMDBs and configures them with the necessary settings for setting up your hierarchy.

Note: For a comprehensive discussion of the Policy Model utilities, see the *Reference Guide*.

Managing Remote PMDBs

CA Access Control also offers you a range of selang commands that you can use in the pmd environment. These commands let you manage PMDBs remotely:

backuppmd

Backs up a PMDB.

createpmd

Creates a PMDB.

deletepmd

Deletes a PMDB.

findpmd

Displays the names of all PMDBs on the computer.

listpmd

Lists the following information about a PMDB:

- Subscribers and their status
- PMDB description and its status
- Commands in the update file and their offsets
- Contents of the error log

pmd

A PMDB administration command that lets you:

- Remove a subscriber from the list of unavailable subscribers

- Clear the Policy Model error log

- Lock and unlock the Policy Model

- Start and stop the Policy Model daemon

- Truncate the update file

- Reload the initialization files

restorepmd

Restores a PMDB from its backup files.

subs

A PMDB subscription command that lets you:

- Add an existing subscriber to a parent PMDB

- Add a new subscriber to a parent PMDB

- Assign a parent PMDB to a database (CA Access Control or another PMDB)

subspmd

Assigns a parent PMDB to the local database.

unsubs

Removes a subscriber from the PMDB.

Note: For a comprehensive discussion of selang commands you can use in the pmd environment, see the *selang Reference Guide*.

Architecture Dependency

When deploying CA Access Control, you should consider the hierarchy of your environment. At many sites, the network includes a variety of architectures. Some policy rules, such as the list of trusted programs, are architecture-dependent. On the other hand, most rules are independent of the system's architecture.

You can cover both kinds of rules by using a hierarchy. You can define a global database for architecture-independent rules, and give it subscriber PMDBs that define architecture-dependent rules.

Note: The root PMDB and all of its subscribers can reside on the same computer or on separate computers, depending on the physical needs of your environment.

Example: A Two-tiered Deployment Hierarchy

The following UNIX example also applies to a Windows architecture with small modifications.

In the example, the site consists of IBM AIX and Sun Solaris systems. Since the list of trusted programs on IBM AIX differs from the one on Sun Solaris, the PMDBs need to consider architecture dependency.

To set up a multiple-architecture PMDB, set up your PMDBs as follows:

1. Define a PMDB named whole_world, to contain the users, groups, and all other architecture independent policies.

2. Define a PMDB named pm_aix, to contain all the IBM AIX specific rules.

3. Define the PMDB pm_sol, to contain all the Sun Solaris specific rules.

    The PMDBs pm_aix and pm_solaris are subscribers of the PMDB whole_world. All IBM AIX computers at the site are subscribers of pm_aix. All Sun Solaris computers at the site are subscribers of pm_sol. The concept is illustrated in the following chart.
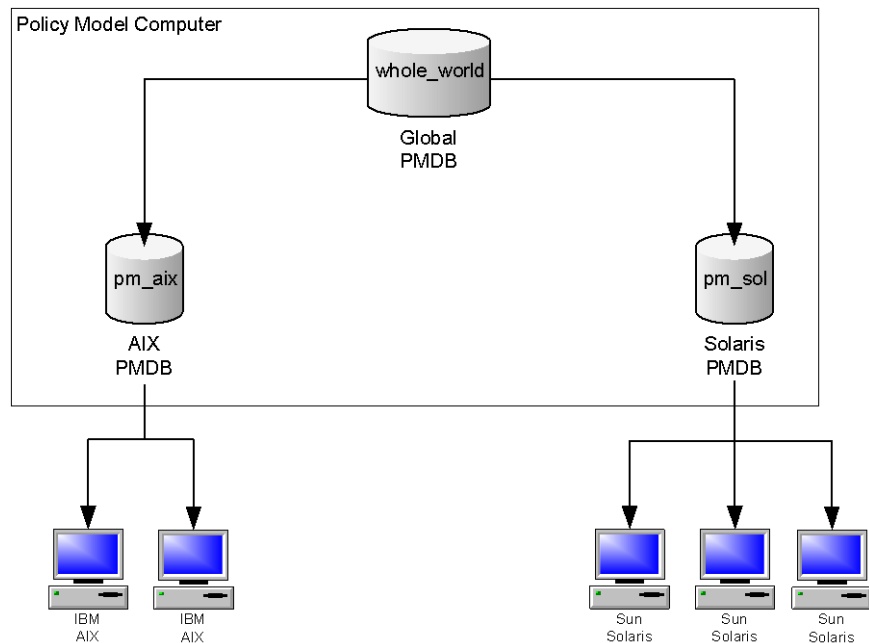
4. When you enter platform-independent commands in whole_world, such as adding a user or setting a SURROGATE rule, all databases at the site are automatically updated.

5. When you add a trusted program to pm_aix, only IBM AIX computers are updated, without affecting the Sun Solaris systems.

Methods for Centrally Managing Policies

CA Access Control lets you manage several databases from a single computer in the following ways:

■ **Automatic rule-based policy updates**â€"Regular rules you define in a central database (PMDB) are automatically propagated to databases in a configured hierarchy.

  Note: Dual control (see page 147) is only available with this method and on UNIX only. Information about dual control for automatic rule-based policy updates is found in the *Endpoint Administration Guide for UNIX*. Information about automatic rule-based policy updates can also be found in the *Endpoint Administration Guide for Windows*.

■ **Advanced policy management**â€"Policies (groups of rules) you deploy are propagated to all databases based on host or host group assignment. You can also undeploy (remove) policies and view deployment status and deployment deviation. You need to install and configure additional components to use this functionality.

  Note: Information about advanced policy management is found in the *Enterprise Administration Guide*.

Automatic Rule-based Policy Updates

Single-rule policy updates (regular selang rules) you make in a central database are automatically propagated to the subscriber databases. By subscribing several computers to the same database, and by subscribing one database to another, you can create a hierarchy. You configure your environment for automatic rule-based policy updates after installation.

Note: This method of managing policies is limited to letting you make single-rule policy updates across your hierarchy. Other functionality is only available through implementing advanced policy management and reporting.

How Automatic Rule-based Policy Updates Work

When you configure your environment for automatic rule-based policy updates, each rule you define in the central database is automatically propagated to all of its subscribers in the following way:

1. A rule is defined for any PMDB with at least one subscriber.

2. The PMDB sends the command to all subscriber databases.

3. The subscriber database applies the propagated command.

    a. If the subscriber database does not respond, the PMDB sends the command at a regular interval (by default, every 30 minutes) until the subscriber database has been updated.

    Alternatively, you can update subscriber databases as soon as they become available, by setting the pull_option token to yes in the [pmd] section of the seos.ini file on the subscriber computer.

    b. If a subscriber database is responding, but refuses to apply the command, the PMDB places the command in the Policy Model error log (see page 141).

4. If the subscriber database is a parent to other subscribers, it then sends the command to its subscribers.

Example: Removing a user from all computers in a hierarchy

If a user is deleted from a PMDB using the rmusr command, the same rmusr command is sent to all the subscriber databases. In this way, a single rmusr command can remove a user from many databases on a variety of computers.

How You Use a PMDB to Propagate Configuration Settings

When you edit a Policy Model's configuration, the new configuration values are propagated to the Policy Model's subscribers.

The following process describes how configuration updates are propagated to a Policy Model's subscribers:

1. You edit one or more of the Policy Model's configuration values.

2. The Policy Model writes the new configuration values to the virtual configuration file.

    Note: The virtual configuration file does not contain values for the audit.cfg file. The Policy Model does not write any changes you make to this file to the virtual configuration file.

3. The Policy Model sends the new configuration values to its subscribers.

4. selang commands update each subscriber with the new configuration values.

Virtual Configuration File

Each Policy Model has a virtual configuration file that contains the configuration values for its subscribers. The virtual configuration file is located in the PMD directory, and is named cfg_*configname*, where *configname* is the name of the Policy Model configuration.

The virtual configuration file does not contain the configuration values held in the audit.cfg file.

How New Subscribers Are Configured

The Policy Model configures each new subscriber with the existing configuration values. The existing configuration values are stored in the virtual configuration file.

Note: The virtual configuration file does not store configuration values from the audit.cfg file. Any changes you make to the audit.cfg file prior to creating a new subscriber are not propagated to the new subscriber.

The following process describes how a Policy Model configures new subscribers:

1. You create a new subscriber to the Policy Model.

2. The Policy Model reads the values in its virtual configuration file.

3. The Policy Model adds the configuration values from its virtual configuration file to the updates.dat file. The updates.dat file also contains the access rules for the Policy.

4. The Policy Model sends the updates.dat file to the new subscriber.

5. selang commands configure the new subscriber with the values in the updates.dat file.

How You Can Set Up a Hierarchy

CA Access Control uses the Policy Model service to propagate rule-based policy updates across the configured hierarchy. By subscribing several CA Access Control computers to the same PMDB, and by subscribing one PMDB to another, you create a hierarchy.

To enable automatic rule-based policy updates, do the following:

1. Create and configure the master PMDB (see page 128).

2. (Optional) Create and configure subscriber PMDBs (see page 130).

3. Define parent PMDBs for the subscribing computers (see page 132), called *endpoints*.

Note: The following sections show how you set up a PMDB hierarchy. There are other ways of creating PMDBs and then setting their hierarchy. For a comprehensive discussion of the Policy Model utilities, see the *Reference Guide*.

Create and Configure the Master PMDB

To let you manage policies from a central location, you first need to create and configure a master PMDB. To do this on a local host, you can use the sepmdadm command.

Note: The following procedure shows the interactive form of the sepmdadm command. For information about using the command-line parameters for all input, see the *Reference Guide*.

To create and configure the master PMDB

1. In a command line, enter the following command:

   sepmdadm -i

   CA Access Control starts the Policy Model database administration script (sepmdadm) and displays a menu with options for you to choose from.

2. Enter 1, to select the first option (create a master PMDB and define its subscribers).

   The script is configured to ask you the relevant questions.

3. Press Enter to continue.

   The script continues to ask you the first question.

   Note: If CA Access Control is not running, the script issues a warning and lets you start CA Access Control before the script is rerun.

4. Enter a name for the Policy Model you want to create.

   The script registers the Policy Model name and continues.

5. Enter the name of the first subscriber computer you want to specify.

   The script registers the name of the first subscriber and then asks you to enter the name of the next subscriber.

6. Continue to enter subscriber names as necessary, then press Enter without entering a subscriber name.

    The script registers all subscriber names and continues.

    Note: You still must point each subscriber computer to its parent PMDB.

7. If you are running NIS, NIS+, or DNS, choose whether you want to update the NIS/DNS tables with PMDB changes.

    Updates are made to users and groups in the PMDB. The tables provide information on users and their characteristics. If you choose yes, a UNIX user or UNIX group updated through the Policy Model is also updated in the NIS passwd and group files.

    a. Enter **y** if you want to update the NIS/DNS tables.

        The script now asks you for the location of the NIS passwd and group files.

        a. Enter the full path of the NIS password file.

            The script registers the full path and continues.

        b. Enter the full path of the NIS group file.

            The script registers the full path and continues.

    b. Enter **n** or press Enter if you want to update the NIS/DNS tables.

        The script registers your answer and continues.

8. Enter the users you want to give special attributes for the PMDB:

    a. Enter CA Access Control administrator names as necessary, then press Enter without entering an administrator's name.

        Administrators are authorized to change the properties of the PMDB.

        Note: At least one administrator must be defined in a PMDB (*root* is the default).

    b. Enter enterprise user administrator names as necessary, then press Enter without entering an administrator's name.

    c. Enter CA Access Control auditor names as necessary, then press Enter without entering an auditor's name.

        Auditors are authorized to view the PMDB's audit log files.

    d. Enter enterprise user auditor names as necessary, then press Enter without entering an auditor's name.

e. Enter CA Access Control password manager names as necessary, then press Enter without entering a password manager's name.

f. Enter enterprise user password manager names as necessary, then press Enter without entering a password manager's name.

Password managers are authorized to change passwords in the PMDB.

The script registers your answer and continues.

9. Enter administration terminals as necessary, then press Enter without entering an administration terminal.

The script registers all administration terminals and then reports the selections you have made and asks you to confirm them.

10. Press Enter to confirm the selections you have made, or enter **n** to rerun the script with new inputs.

If you confirm your selections, a new PMDB is created using the answers you supplied.

More information:

Create and Configure Subscriber PMDBs (see page 130)
Define Parent PMDBs for Subscribing Computers (see page 132)

Create and Configure Subscriber PMDBs

Once you have a master PMDB configured, if you want to extend your hierarchy, you need to create and configure subscriber PMDBs. To do this on a local host, you can use the sepmdadm command.

Note: The following procedure shows the interactive form of the sepmdadm command. For information about using the command-line parameters for all input, see the *Reference Guide*.

To create and configure subscriber PMDBs

1. In a command line, enter the following command:

   sepmdadm -i

   CA Access Control starts the Policy Model database administration script (sepmdadm) and displays a menu with options for you to choose from.

2. Enter 2, to select the second option (create a subsidiary PMDB and define its subscribers and parent.).

   The script is configured to ask you the relevant questions.

3. Press Enter to continue.

   The script continues to ask you the first question.

   Note: If CA Access Control is not running, the script issues a warning and lets you start CA Access Control before the script is rerun.

4. Enter a name for the Policy Model you want to create.

   The script registers the Policy Model name and continues.

5. Enter the name of the first subscriber computer you want to specify.

   The script registers the name of the first subscriber and then asks you to enter the name of the next subscriber.

6. Continue to enter subscriber names as necessary, then press Enter without entering a subscriber name.

   The script registers all subscriber names and continues.

   Note: You still must point each subscriber computer to its parent PMDB (see page 132).

7. Enter the name of the parent PMDB.

   The script registers the parent PMDB name and continues.

   Note: sepmdadm only lets you enter one parent for each subscribing database. You can, however, define multiple parents for each database. To do this, modify the parent_pmd token of the pmd.ini configuration file. For more information about using this token, see the *Reference Guide*.

8. If you are running NIS, NIS+, or DNS, choose whether you want to update the NIS/DNS tables with PMDB changes.

   Updates are made to users and groups in the PMDB. The tables provide information on users and their characteristics. If you choose yes, a UNIX user or UNIX group updated through the Policy Model is also updated in the NIS passwd and group files.

   a. Enter **y** if you want to update the NIS/DNS tables.

      The script now asks you for the location of the NIS passwd and group files.

      a. Enter the full path of the NIS password file.

         The script registers the full path and continues.

      b. Enter the full path of the NIS group file.

         The script registers the full path and continues.

   b. Enter **n** or press Enter if you want to update the NIS/DNS tables.

      The script registers your answer and continues.

9. Enter the users you want to give special attributes for the PMDB:

   a. Enter CA Access Control administrator names as necessary, then press Enter without entering an administrator's name.

      Administrators are authorized to change the properties of the PMDB.

      Note: At least one administrator must be defined in a PMDB (*root* is the default).

   b. Enter enterprise administrator names as necessary, then press Enter without entering an administrator's name.

   c. Enter CA Access Control auditor names as necessary, then press Enter without entering an auditor's name.

      Auditors are authorized to view the PMDB's audit log files.

   d. Enter enterprise user auditor names as necessary, then press Enter without entering an auditor's name.

   e. Enter CA Access Control password manager names as necessary, then press Enter without entering a password manager's name.

      Password managers are authorized to change passwords in the PMDB.

   f. Enter enterprise user password manager names as necessary, then press Enter without entering a password manager's name.

   The script registers your answer and continues.

10. Enter administration terminals as necessary, then press Enter without entering an administration terminal.

    The script registers all administration terminals and then reports the selections you have made and asks you to confirm them.

11. Press Enter to confirm the selections you have made, or enter **n** to rerun the script with new inputs.

    If you confirm your selections, a new PMDB is created using the answers you supplied.

Define Parent PMDBs for Subscribing Computers

To establish an endpoint computer as a subscriber to a PMDB, you must do more than register the subscriber's name in the PMDB. You also need to complete a procedure at the subscriber computer.

To define parent PMDBs for subscribing computers

1. In a command line on the subscriber computer, start sepmdadm in interactive mode:

   sepmdadm -i

   CA Access Control starts the Policy Model database administration script (sepmdadm) and displays a menu with options for you to choose from.

2. Enter 3, to select the third option (define the parent and password PMDBs of the local host).

   The script is configured to ask you the relevant questions.

3. Press Enter to continue.

   The interactive script continues to ask you the first question.

   Note: If CA Access Control is running, the script issues a warning and lets you stop CA Access Control before the script is rerun.

4. Enter the name of the parent PMDB.

   The script registers the name of the parent PMDB name and continues.

5. Enter the name of the parent password PMDB.

   The script registers the name of the parent password PMDB name and then reports the selections you have made and asks you to confirm them.

6. Press Enter to confirm the selections you have made, or enter **n** to rerun the script with new inputs.

   If you confirm your selections, the subscriber computer is set up with these inputs.

Note: sepmdadm only lets you enter one parent for each subscribing database. You can, however, define multiple parents for each database. To do this, modify the parent_pmd token of the seos.ini configuration file. For more information about using this token, see the *Reference Guide*.

UID/GID Synchronization

As an administrator, you may receive messages that refer to users by UID and to groups by GID. You need to make sure that the UIDs and GIDs have the same meaning everywhere.

By default, the PMDB attempts to use the same UIDs and GIDs for new users and groups everywhere, but you can help by providing the necessary conditions from the start. Start with identical passwd files and identical group files, making sure that the synch_uid token in the pmd.ini file is set to yes. If your local database is a subscriber to your PMDB, and the PMDB is the only source of new users and new groups for your subscriber databases, then you can depend on compatibility between the UIDs and between the GIDs of your local database, your PMDB, and your PMDB subscribers.

If you create a new user with a UID that is already in use in the PMDB or in some other subscriber computer, the subscriber's individual update fails; but in all other subscriber computers where no such conflict exists, the update succeeds.

An alternative to synchronizing your passwd and group files is to explicitly specify the UID of each new user and the GID of each new group.

Synchronize Users and Groups

To ensure the lists of users and groups in your various databases correspond correctly at all times, you need an initial set of identical lists. Because the password and group files are so important, synchronize them before they begin accumulating local user and group information.

To synchronize users and groups

1. Copy your /etc/passwd file and /etc/group file to your Policy Model directory.

    This is a one-time procedure that destroys any previous passwd and group files in your Policy Model directory (see page 121).

    Note: If you are using a shadow file and want to synchronize passwords, we recommend using the secrepsw utility. For more information, see the *Reference Guide*.

2. Copy the /etc/passwd file and /etc/group file to each subscriber computer so that they are identical to the ones on your own computer.

3. On the computer where the PMDB resides, ensure that the synch_uid token in your pmd.ini file is set to yes.

    By default, the value of the token synch_uid is yes. If you ever want a subscriber database to have independent default UIDs and default GIDs (that is, not necessarily attempting to match those of the PMDB), you can set synch_uid to no.

Specify UIDs Explicitly

Another way to send an identical UID or GID to the PMDB and to all its subscribers is to explicitly set it when you create a new user.

To specify UIDs explicitly use the userid or groupid parameter with each newusr command.

Example: Create a new user with a specified UID

If you want to establish 1234 explicitly as the UID of new user terry_jones (and assuming that no one else in the database has that UID yet), enter the command:

newusr terry_jones unix (userid(1234))

If the specified UID is already being used in the PMDB, then the PMDB will not itself be updated, but the command will still propagate to the other subscriber databases. Among the other databases, wherever the particular UID is already in use, the subscriber's individual update will fail; but where no such conflict exists, the update succeeds.

How the Policy Model Updates Subscribers

When updating subscribers, the Policy Model performs the following actions:

1. The Policy Model tries to fully qualify subscriber names as they are added or deleted from the Policy Model.

2. The PMDB daemon, sepmdd, attempts to update a subscriber database for the amount of time defined by the token _QD_timeout_.

3. If the maximum time elapses and the daemon does not succeed in updating a subscriber, it skips that particular subscriber and tries to update the remainder of the subscribers on its list.

4. After it completes its first scan of the subscriber list, sepmdd then performs a second scan, in which it tries to update the subscribers that it did not succeed in updating during its first scan. During the second scan, it tries to update a subscriber until the connect system call times out (approximately 90 seconds).

Note: The token _QD_timeout_ may be found in both the seos.ini and pmd.ini files. If the token exists in both files, sepmdd uses the value in the pmd.ini file.

Note: Whenever a PMDB encounters an error while propagating updates to subscribers, the sepmdd daemon creates an entry in the Policy Model error log file (see page 141). This file, ERROR_LOG by default, is located in the PMDB directory (see page 121).

Update a Policy Model Database

Working at the computer where the PMDB resides does not automatically update the PMDB itself. To update a PMDB, you need to specify it as your target database.

To specify a target database, use the hosts command in the selang command shell:

hosts *pmd_name@pmd_host*

All selang commands now update the policy model database specified. The commands then automatically propagate to the active databases on this computer and of all subscriber computers.

Example: Specify a target PMDB

To set the target database to policy1 on myPMD_host, use the following command:

hosts policy1@myPMD_host

If you now enter the newusr command, the new user is added to the policy1 database as well as the active databases on this computer and of all subscriber computers.

Clean Up the Update File

The sepmd utility automatically writes each update it receives in the updates.dat file. To prevent the file from growing too large, we recommend that you delete processed updates from the file periodically.

To clean up the update file, use the following command:

sepmd -t *pmdbName* auto

sepmd calculates the offset of the first update entry that has not been propagated and deletes all the update entries before it.

Note: For more information about sepmd utility, see the *Reference Guide*.

Encrypt the Update File

After you create a PMDB, but *before* you start sepmdd, you can specify that information saved to the updates.dat file be encrypted.

To encrypt the update file, set the UseEncryption token to yes in the [pmd] section of the pmd.ini file.

To decrypt the updates.dat file, use the sepmd utility with the -de switch.

Note: For more information about sepmd, see the *Reference Guide*.

Exclude Subscribers

You can skip subscribers so that they do not receive updates from parent PMDBs.

To exclude the local host, set the token exclude_localhost to yes in the pmd.ini file.

To add additional subscribers to the excluded list, set the token exclude_file (*name-of-file*).

To make a subscriber receive updates, remove the subscriber from the excluded list.

Propagate Passwords

When a user changes a password using the sepass utility, the new password is normally sent to the computer's parent PMDB. The parent PMDB is defined in the parent_pmd or the passwd_pmd token in the [seos] section of the seos.ini file or in both. However, if the user changes the password with the utility sepass, you can also specify that the user's new password should be sent to and propagated by a separate PMDB.

To send a new user's password to a separate PMDB, use the pmdb parameter with the newusr, chusr, or editusr command.

Example: Specifying a separate PMDB for password propagation

To specify that the new passwords created with sepass for the user Tony should be sent to and propagated by a separate PMDB pw_pmdb@name1.yourorg.com, enter the following command:

```
editusr tony pmdb(pw_pmdb@name1.yourorg.com)
```

Remove a Subscriber

If you no longer want to propagate updates to a particular subscriber, you should remove it. Alternatively, you can exclude a subscriber from receiving updates (see page 137).

To remove a subscriber

1. Remove the computer from the subscription list:

    sepmd -u *PMDB_name computer_name*

    The computer is removed from the Policy Model subscription list.

2. Shut down seosd on the computer that you removed from the subscription list:

    secons -s

    The daemon seosd is shut down.

3. Delete the value of the parent_pmd token in the [seos] section of the seos.ini file on the computer you removed from the subscription list.

    The computer will stop accepting updates from the parent PMDB.

4. Restart seosd.

    The active database on the computer that you removed from the subscription list is no longer a subscriber of the specified PMDB.

Note: Once the database is unsubscribed from the PMDB, the PMDB no longer sends commands.

Filter Updates

If you want your PMDB to update different subsets of data at different subscriber databases, you need to define which records are sent to subscriber databases.

To filter updates

1. Configure PMDBs to serve as parents to subsets of subscribers (see page 132).

2. Modify the *filter* token in the pmd.ini file of the parent PMDB, to point to a filter file you set up on the same computer.

    Updates to the subscriber databases are then limited to the records that pass the filter.

Note: When you execute a join or join- selang command in the native UNIX environment, CA Access Control changes the command to change group (cg). To filter join or join- commands in the native UNIX environment, use the following line in the filter file:

MODIFY UNIX GROUP *GroupName* USERS NOPASS

You cannot filter join or join- commands by user name in the native UNIX environment. This rule does not apply to join or join- commands in any other environment.

Policy Model Filter File

A filter file consists of lines, each with six fields. The fields contain information on:

■   The form of access permitted or denied.

    For example, READ or MODIFY

■   The environment affected:

    For example, AC or native

■   The class of the record.

    For example, USER or TERMINAL

■   The objects, within the class, that the rule covers.

    For example, User1, AuditGroup, or TTY1

■ The properties that the record grants or cancels.

For example, OWNER and FULL_NAME in the filter line means that any command having those properties is filtered. You must enter each property exactly as it appears in the *Reference Guide*.

■ Whether such records should be forwarded to the subscriber database or not:

PASS or NOPASS

The following rules apply to each line in the filter file:

■ You can use an asterisk (*) to denote all possible values in any field.

■ If more than one line covers the same records, the *first* applicable line is used.

■ Spaces separate the fields.

■ In fields with more than one value, semicolons separate the values.

■ Lines beginning with **#** are considered a comment line.

■ Empty lines are not allowed.

Example: Filter file

The following example describes a line from a filter file:

CREATE	AC	USER	*	FULL_NAME;OBJ_TYPE	NOPASS
form of access	environment	class	record name (* =all)	properties	treatment

In this example, if we name the file with this line TTY1_FILTER and edit the pmd.ini file for PMDB TTY1 so that filter=/opt/CA/AccessControl/TTY1_FILTER, then PMDB TTY1 will not propagate to its subscribers any records that create new users with the FULL_NAME and OBJ_TYPE property.

Policy Model Error Log File

The Policy Model error log, which is organized chronologically, looks similar to this:

Error Text	Error Category

20 Nov 03 11:56:07 (pmdb1): fargo nu u5 0 Retry
   ERROR: Login procedure failed (10068)
   ERROR: Cannot accept update from a non-parent PMDB
 (pmdb1@name.company.com) (10104)
	Configuration Errors

20 Nov 03 19:53:17 (pmdb1): fargo nu u5 0 Retry
   ERROR: Connection failed (10071)
   Host is unreachable (12296)
	Connection Errors

20 Nov 03 11:57:06 (pmdb1): fargo nu u5 560 Cont
   ERROR: Failed to create USER u5 (10028)
   Already exists (-9)
 	
 20 Nov 03 11:57:06 (pmdb1): fargo nu u5 1120 Cont
   ERROR: Failed to create USER u5 (10028)
   Already exists (-9)
	Database Update Errors

The Policy Model error log is in binary format; you can view it only by entering the following command:

ACInstallDir/bin sepmd -e *pmdname*

Note: Do not manually delete an error log (for example, with the UNIX rm command). To delete the log, only use the following command:

ACInstallDir/bin sepmd -c *pmdname*

Important! The error log in CA Access Control r5.1 and later versions has a format that is not compatible with the format of earlier versions. sepmd cannot handle error logs from these earlier versions. When you upgrade to a version that has this format, the old error log is copied to ERROR_LOG.bak; a new log file is created when you start sepmdd.

Example: PMDB Update Error Message

The following example shows a typical error message:

```
date       time       pmdb name          subscriber   command   offset   flag

20 Nov 03 19:53:17 (pmdb1): fargo   nu u5   0   Retry
ERROR: Connection failed (10071) ◄_____ major level (type of error)
Host is unreachable (12296) ◄_____ minor level (cause of error)

                return code
```

- The top line always consists of the date, time, and subscriber. The command that generated the error appears next, followed by the offset (in decimal format), which indicates the location of the failed update inside the updates file. Lastly, the flag indicates whether the PMDB retries the update automatically or continues without it.

- The second line shows an example of a major level message (what type of error occurred) and its return code.

- The third line displays an example of a minor level message (why the error occurred), and its return code.

Example: Error Message

A command may generate and display more than one error. Also, an error may consist of a major level message, a minor level message, or both.

The following error has only one message level:

Fri Dec 29 10:30:43 2003 CIMV_PROD:Release failed. Return code = 9241

This message occurs when sepmd pull attempts to release a subscriber that is already available.

Policy Model Backup

When you back up a PMDB, you copy the data in the Policy Model database to another directory. This includes:

- policy information

- the list of the Policy Model's subscribers

- configuration settings

- registry entries

- the updates.dat file

You cannot restore a PMDB from backup files that use another platform, operating system, or version of CA Access Control. Ensure you back up the Policy Model to a host running the same platform, operating system, and version of CA Access Control.

Back Up a PMDB Using sepmd

When you back up the PMDB, you copy the data from the Policy Model database to a specified directory. You should store the backed up PMDB files in a secure location, preferably protected by CA Access Control access rules.

You can use the sepmd utility to back up a PMDB on a local host. You can also use selang commands to back up a PMDB on a remote host.

Note: You can back up a PMDB recursively. A recursive backup backs up all the PMDBs in a hierarchy to the host you specify, and modifies the PMDB subscribers so that the subscription still works when the backup is moved to the host. You can only use a recursive backup if the master and child PMDBs are deployed on the same host.

To back up a PMDB using sepmd

1. Lock the PMDB using the following command:

   sepmd -bl *pmdb_name*

   The PMDB is locked and cannot send commands to its subscribers.

2. Do one of the following:

   ■ Back up the PMDB using the following command:

   sepmd -bh *pmdb_name* [*destination_directory*]

   ■ Back up the PMDB recursively using the following command:

   sepmd -bh *pmdb_name* [*destination_directory*] [*backup_host_name*]

   Note: If you do not specify a destination directory, the backup is saved to the following directory:

   ACInstallDir/data/policies_backup/pmdb_name

3. Unlock the PMDB using the following command:

   sepmd -ul *pmdb_name*

   The PMDB is unlocked and can send commands to its subscribers.

Back Up a PMDB Using selang

When you back up the PMDB, you copy the data from the Policy Model database to a specified directory. You should store the backed up PMDB files in a secure location, preferably protected by CA Access Control access rules.

You can use selang commands to back up a PMDB on a local or remote host. You can also use the sepmd utility to back up a PMDB on a local host.

Note: You can back up a PMDB recursively. A recursive backup backs up all the PMDBs in a hierarchy to the host you specify, and modifies the PMDB subscribers so that the subscription still works when the backup is moved to the host. You can only use a recursive backup if the master and child PMDBs are deployed on the same host.

To back up a PMDB using selang

1. (Optional) If you are using selang to connect to the PMDB from a remote host, connect to the PMDB host using the following command:

    host *pmdb_host_name*

2. Move to the PMD environment using the following command:

    env pmd

3. Lock the DMS using the following command:

    pmd *pmdb_name* lock

    The PMDB is locked and cannot send commands to its subscribers.

4. Back up the DMS database using the following command:

    backuppmd *pmdb_name* [destination(*destination_directory*)] [hir_host(*host_name*)]

    Note: If you do not specify a destination directory, the backup is saved to the following directory:

    ACInstallDir/data/policies_backup/pmdbName

5. Unlock the PMDB using the following command:

    pmd *pmdb_name* unlock

    The PMDB is unlocked and can send commands to its subscribers.

Policy Model Restoration

When a Policy Model is restored, CA Access Control copies the backup PMDB files into the specified directory. Everything that is in the original PMDB files is copied to the new PMDB directory, including:

- policy information

- the list of the Policy Model's subscribers

- configuration settings

- registry entries

- the updates.dat file

If there is an existing PMBD in the destination directory, CA Access Control deletes the existing files before copying the restoration files into that directory.

You cannot restore a PMDB from backup files that use another platform, operating system, or version of CA Access Control. Ensure you back up the Policy Model to a host running the same platform, operating system, and version of CA Access Control.

Restore a PMDB

When you restore a PMDB, CA Access Control copies the data from the PMDB backup files into the directory you specify. CA Access Control must be running on the terminal you do the restoration on.

Note: If you back up and restore the PMDB on different terminals, the PMDB does not automatically update the terminal resource in the restored PMDB database. You must add the new terminal resource to the restored PMDB. To add the new terminal resource, stop the restored PMDB, run the *selang -p pmdb* command, then start the restored PMDB.

To restore a PMDB, run *one* of the following on the terminal that you want to restore the PMDB on:

- sepmd -restore utility

- selang restore pmd command

Note: For more information about the sepmd utility, see the *Reference Guide*. For more information about selang commands, see the *selang Reference Guide*.

Dual Control

Dual Control is a way of operation that divides the process of updating the PMDB into two stages:

- Creating a transaction which consists of one or more commands.

  The *maker* - any user with the ADMIN attribute - enters one or more commands that update the PMDB. The transaction is given a unique ID number and placed in a file, where it waits to be processed before execution.

- Authorizing the transaction for execution.

  The *checker* - not the same user, but any *other* user with the ADMIN attribute - locks the commands in the transaction, checks the commands, and authorizes or rejects them. If the transaction is authorized, then the commands are executed in the PMDB. If the transaction is rejected, then the transaction is deleted and the PMDB is not updated. The checker cannot authorize some of the commands in a transaction and reject others; the transaction must be processed as a whole.

  Note: Only the find and show commands do not need the authorization of a checker.

Using the parameters in the sepmd utility, makers can list, retrieve and edit, or delete unprocessed transactions; checkers can lock transactions in order to authorize or reject them, and they can unlock transactions for processing at a later time or by a different checker.

When the sepmdd daemon receives the start_transaction command, it sends the child process a unique number. The child process tags any further commands with this identifying number, and the number is added to the new transaction and kept in the memory of the sepmdd daemon. When sepmdd receives the end_transaction command, the authorization algorithm is invoked. The authorization algorithm checks that none of the commands in the transaction pertain to the maker of the transaction, and none of the objects in the commands are already locked by another transaction that is waiting to be processed prior to execution.

You cannot use the same objects in different transactions before they are processed. If the check passes, then the relevant objects are locked, the transaction is assigned a unique sequential number, and the data is saved in a file. Each transaction is saved in a different file.

Note: For more information about the sepmd utility or the sepmdd daemon, see the *Reference Guide*.

Activate Dual Control

Dual Control lets you divide the duty of updating PMDBs between two people: a maker and a checker.

To activate Dual Control, set the is_maker_checker token, in the pmd.ini file *and* in the [pmd] section of the seos.ini file, to yes:

is_maker_checker=yes

Note: Create the Policy Model maker *before* setting these token values.

Create or Edit Transactions

When Dual Control is activated, the maker needs to create transactions before these are processed by a checker.

To create a transaction

1. Make sure the following is true:

   ■ You (as a maker) have the ADMIN authority.

   ■ None of the commands pertain to you. (You cannot enter commands that change yourself.)

   ■ None of the objects in the commands are already part of another transaction that has not been processed by a checker yet.

   ■ All the objects in the commands exist.

   ■ You are not editing an existing transaction that another maker invoked. (You can only edit your own transactions.)

2. Connect to the maker PMDB:

   hosts maker@

   The hosts command connects you to the PMDB (maker). When Dual Control is activated, the name of the PMDB is always "maker." After you enter the hosts command, a message reports whether the connection to the host is successful or not.

3. Start the transaction:

   start_transaction *transactionName*

   Use start_transaction command as the first step when entering or updating a transaction. You can describe the transaction or give it any name you want, up to 256 alphanumeric characters.

4. Enter your transaction.

    This is a list of commands. For example:

    ```
    newusr mary owner(bob) audit(failure,loginfailure)
    chres TERMINAL tty30 defaccess(read) \
     restrictions(days(weekdays)time(0800:1800))
    ```

5. End the transaction:
    ```
    end_transaction
    ```

    The transaction is complete; you are presented with the unique ID number assigned to your transaction. The commands are placed in a file, where you can still access and change them until a checker, in preparation for processing, locks them.

    Note: Make sure you record the transaction ID number if you want to be able to edit the transaction later.

To edit a transaction

▪   When you enter the end_transaction command, an ID number displays. This is a unique number that identifies the transaction. If you want to overwrite your transaction later, then the process is the same as creating a new transaction, except that you add to the file the transaction's ID number after the name. You can enter to the file any changes you want to make. For example:

    ```
    hosts maker@
    start_transaction transactionName transactionId
    ```

    You can then enter the appropriate commands to update the transaction:

    ```
    chusr mary category (FINANCIAL)
    end_transaction
    ```

▪   View specific unprocessed transactions with the following parameters.

    Make sure you are in the *ACInstallDir*/bin path (where *ACInstallDir* is the installation directory for CA Access Control, by default /opt/CA/AccessControl).

Command with Parameter	Description
sepmd -m l	Lists the unprocessed transactions of the user who invoked the parameter.
sepmd -m la	Lists all the transactions of all the makers that are waiting to be processed.
sepmd -m lo	Lists the transactions of all the makers except those of the user who invoked the parameter Each transaction in the list includes the name of the maker, the ID number of the transaction, and

Command with Parameter	Description
	a description of the transaction, if the maker entered one.

■ Retrieve a specific transaction to the standard output with the following command:

sepmd -m r transactionId

■ Delete a specific transaction with this command:

sepmd -m d transactionId

Checking and Processing Transactions

When Dual Control is activated, the checker needs process transactions created by a maker.

To check a transaction

1. Make sure the following is true:

   ■ You (as the checker) have ADMIN authority.

   ■ Another Checker does not lock the transaction.

   ■ None of the commands pertain to you. (You cannot process commands that involve yourself.)

2. Navigate to the *ACInstallDir*/bin path

   where *ACInstallDir* is the installation directory for CA Access Control, by default /opt/CA/AccessControl

3. View the transactions that are waiting to be processed before execution:

   sepmd -m la

   Or, view all the transactions except the transactions that you yourself created:
   sepmd -m lo

   Each transaction includes the name of the maker, the ID number of the transaction, and the name or description of the transaction.

4. Lock the transactions before processing them:

   sepmd -m r *transactionId*

   Note: A locked transaction cannot be changed.

5. Process the transaction:

sepmd -m p *transactionId code*

code

Can be *one* of the following:

- **0**—The transaction is rejected.

    In this case, all the commands in the transaction are deleted and no changes are implemented in the PMDB.

- **1**—The transaction is authorized.

    The commands in the transaction are immediately implemented in the PMDB.

- **2**—The transaction is unlocked.

    The transaction returns to the queue of waiting transactions and can be processed later, perhaps by a different checker.

A message appears stating which commands were successful and which failed.

Note: For more information on makers and checkers, see the sepmd utility in the *Reference Guide* and the start_transaction command in the *selang Reference Guide*.

Using the seagent and sepmdd Daemons

The seagent daemon is responsible for accepting requests from remote computers and applying them to PMDBs; the seagent daemon also sends requests to seosd. The sepmdd daemon is the PMDB daemon. This section describes how these daemons work together in the PMDB environment.

The seagent Daemon

The seagent daemon waits for connections on the seoslang and seoslang2 TCP services (whose default values are 8890 and 8891, respectively). When a connection request arrives, seagent forks a child process to handle the communication on the connection and then continues waiting for new connections.

When a user enters the *hosts* command in selang, seagent forks a child process on the machine that the user is connected to. The child process then receives commands from the command language interface and passes them on to the sepmdd daemon.

The sepmdd Daemon

The sepmdd daemon performs the following functions:

- Administers the PMDB

- Administers the subscriber databases

- Propagates changes from the PMDB to the subscriber databases

The sepmdd daemon is automatically started by seagent when seagent has to access the PMDB. Normally you do not need to run sepmdd explicitly.

Note: sepmdd runs under the logical user _seagent (*not* under root) in the AC environment. To permit or restrict access to resources by sepmdd (for example, to restrict access to the PMDB directory), create the relevant rules for _seagent.

Using a Shadow File

Usually, sepmdd does not use a shadow file when updating a native environment. You can, however, set up a shadow file. To do this, set the UseShadow token in the [pmd] section of the pmd.ini file to yes.

If the UseShadow token is set to yes, sepmdd uses a default shadow file in the same directory as the PMDB. If you want to change the location of the shadow file, specify the new location with the YpServerSecure token in the [pmd] section of the pmd.ini.

If you change the location of the shadow file (with the YpServerSecure), to the local host's shadow file (for example, /etc/shadow), sepmdd sets a token, UseSystemFiles, to yes.

Important! Do not change the UseSystemFiles token yourself. The sepmdd or seagent daemons change it automatically.

Note: For more information about the seagent or sepmdd daemons, see the seagent and sepmdd utilities in the *Reference Guide*.

Mainframe Password Synchronization

CA Access Control supports password synchronization among mainframes running CA Top Secret, CA ACF2, or RACF security products (and CA Common Services CAICCI package) and Windows or UNIX computers running CA Access Control. Synchronization is accomplished using the standard CA Access Control password Policy Model method.

Any password change a mainframe user makes is propagated to all the machines in the password Policy Model hierarchy.

Chapter 12: General Security Features

This section contains the following topics:

Protection of Idle Stations

Information is extremely vulnerable when terminals are left open and active. An intruder who happens upon such a terminal (for example, during a lunch break) need not try to break passwords or have complicated equipment to sniff the network lines, since all terminals at the site are already logged in and ready for work. Although screen savers that prompt for the password before restoring the desktop are useful, the security administrator cannot make sure that all users are using secured screen savers.

CA Access Control provides selock, a screen-locking utility that guards all terminals and stations by locking them whenever they are idle for more than a specified period of time. When returning to work, the user is prompted to specify the password. If the correct password is not specified within one minute, the terminal remains locked. The selock utility can find the password of users who can unlock a screen even if those users change their passwords while selock is active.

Note: For more information about the screen lock utility selock, see the *Reference Guide*.

You should choose to use selock options that suit your requirements:

- Less security, more convenience

  Use the -timeout option to set the timeout to a large value, such as 10 minutes, and the -lock-timeout option to set the lock timeout to an even larger value, such as 60 minutes. This prevents selock from excessively interrupting your work by switching to the *saver* mode. Also, this setting locks your screen only in cases when your terminal is left inactive for extended periods.

- More security, less convenience

  Use the -timeout option to set the timeout to a small value, such as 1 minute, and -lock-timeout option to set the lock timeout to a small value, between 0 and 2 minutes. This always hides your work soon after you stop accessing your terminal, and requires a password for restoring access. To ensure that selock always requires password-entry to reactivate your terminal after the saver mode starts, use the -lock-timeout option to set the lock timeout to zero.

- The selock command can be part of the X startup shell, so that it starts automatically every time the user logs in to the system. The script must be run under the user ID, not under the root ID. The way you integrate the selock command into the startup script depends on the specific environment of the site.

  Note: For more information on startup scripts, see the documentation for your UNIX system.

Protection Modes

selock offers three modes of operation:

Monitor Mode

The monitor mode is the initial mode of selock. In this mode, selock monitors keyboard and mouse activity. If selock detects no keyboard or mouse activity during the time-out period-and the transparent parameter is off-selock automatically switches to the saver mode. No password entry is required for the transition from the monitor mode to the saver mode.

Saver Mode

In the saver mode selock blanks the entire screen and displays a system icon that shifts position. The blank screen and shifting icon provide two operational advantages:

■ Reduced risk of screen viewing by unauthorized people

■ Reduced screen burn-in

You can manipulate the appearance and repositioning of the icon using selock options. When selock detects any keyboard or mouse activity, it immediately returns from the saver mode to the monitor mode, restoring the screen display to what it was before it switched to saver mode. No password entry is required for the transition from the monitor mode to the saver mode.

If selock remains in the saver mode for the period specified by the lock-timeout parameter, it automatically switches to the lock mode. selock does not give any visual indication of the transition from the saver mode to the lock mode.

Lock Mode

In lock mode with the default settings, selock continues to display a moving icon on a black background. When selock detects any keyboard or mouse activity, a dialog containing a prompt for the user's password appears.

When the user enters the correct password, selock switches back to monitor mode. If the user enters an incorrect password, the password-entry dialog closes and selock remains in the lock mode.

If you set the -transparent option to *on*, selock locks the screen but displays the contents and updates the on-going processes. The background of the screen changes to indicate that the screen is locked. When you use the lock mode, saver mode is never invoked.

Set Stations to Lock when Idle

The selock utility lets you lock idle stations to prevent unauthorized access to these stations when they are left idle.

To set stations to lock when idle

1. (Optional) Set the DISPLAY environment variable.

   For the selock command to work, you must set the DISPLAY environment variable. However, you can specify the target display directly in the selock command instead.

2. Place the selock command in the user's login script (the .login file).

   Alternatively, you can place the selock command in the /etc/login or /etc/cshrc file.

Note: Two users can always unlock a locked screen. By default, these users are the current user and root. However, you can replace root with any other user if you specify the other user's name in the unlocking_user token, located in the [selock] section of the seos.ini file. You can replace the current user with any other user by using the -user option when executing selock.

Example: Idle station lockup command in a startup file

The following is a typical startup command, suitable to be placed in X startup files:

selock -display $DISPLAY -timeout 5

This command activates selock after five minutes of terminal inactivity.

We recommend that you place the following line in the global xstartup script. The xstartup script usually resides in the directory /usr/lib/X11/xdm/Xstartup.

selock -display $DISPLAY -user $USER -timeout 3 &

This statement enforces use of the terminal locking program for all users who are using X terminals.

Change the Screen Lock Icon

The default system icon that selock uses is the CA Access Control logo and is located in the file ACInstallDir/data/admin/Selogo.xpm

To select an icon of your own choice, replace this file.

Note: The icon file must be in XPM version 3.3 format.

Protecting Resources Using APIs

If you have defined resources that are not part of CA Access Control (that is, in-house resources), you can protect them by using CA Access Control APIs. Each API has two layers:

The function library

Enables programmers to use the CA Access Control authorization engine.

The user exits

Enable the system administrator to tailor CA Access Control behavior to the requirements of the site.

Note: For more information about CA Access Control APIs, see the *SDK Guide*.

Protecting Against Stack Overflow: STOP

Stack overflow enables hackers to execute arbitrary commands on remote or local systems, many times as the root user (the superuser). They do this by exploiting bugs in the operating system or other programs. These bugs allow users to overwrite the program stack, changing the next command to be executed.

Stack overflow is not simply a bug; it is possible to create a block that overwrites the return address with a meaningful address, resulting in transferred control to unauthorized code (usually in the same block).

Stack Overflow Protection (STOP) is a feature that prevents hackers from creating and exploiting stack overflow to break into systems.

Starting and Stopping STOP

When STOP is first installed, stack overflow protection is activated by default. To deactivate it, you must change a token in the [seos_syscall] section of the seos.ini file and restart CA Access Control. To do this, use the seini command as follows:

seini -s SEOS_syscall.STOP_enabled 0

You could manually change the seos.ini file instead.

To re-enable STOP, change the value of the token to 1 and restart CA Access Control.

Note: When STOP is active on Sun Solaris 7 systems, the dbx program cannot work properly. If you need to use dbx on a system that is protected by STOP, you must first disable STOP.

Defining Day and Time Access Rules for Resources

You can use CA Access Control to specify day-of-week and time-of-day restrictions for resource access. This feature can be exploited for TERMINAL access, SURROGATE requests, and user-defined resources. For example, the following rule completely disables the terminal ws3 on weekends and outside the 08:00-19:00 time period every day:

chres TERMINAL ws3 restrictions(days(weekdays) time(0800:1900))

No login request from that station is accepted outside these periods.

You can use CA Access Control to protect against substitution requests to highly authorized users outside work hours. Suppose user AcctMgr is the Accounting Manager, who is allowed to perform financial transactions, and you have restricted AcctMgr login to work hours and weekdays only. Intruders or unauthorized personnel may try to access the account of AcctMgr by invoking the command **su** AcctMgr. Use the following command to make it impossible to substitute the user name to AcctMgr outside the specified period:

chres SURROGATE USER.AcctMgr restrictions(days(weekdays) time(0800:1900))

The same technique can be implemented for any protected resource, including user-defined abstract classes that are used for implementing in-house applications.

B1 Security Level Certification

CA Access Control includes the following B1 "Orange Book" features:

- Security categories
- Security labels
- Security levels

Security Levels

When security level checking is enabled, CA Access Control performs security level checking in addition to its other authorization checking. A security level is a positive integer between 1 and 255 that can be assigned to users and resources. When a user requests access to a resource that has a security level assigned to it, CA Access Control compares the security level of the resource with the security level of the user. If the user's security level is equal to or greater than the security level of the resource, CA Access Control continues with other authorization checking; otherwise, the user is denied access to the resource.

If the SECLABEL class is active, CA Access Control uses the security level associated with the security labels of the resource and user; the security level that is explicitly set in the resource and user records is ignored.

To protect a resource with security level checking, assign a security level to the resource's record. The level parameter of the newres or chres command assigns a security level to a resource.

To allow a user access to resources protected by security level checking, assign a security level to the user's record. The level parameter of the newusr or chusr command assigns a security level to a user.

Enabling Security Level Checking

The following setoptions command enables security level checking:

setoptions class+ (SECLEVEL)

Disabling Security Level Checking

The following setoptions command disables security level checking:

setoptions class- (SECLEVEL)

Security Categories

When security category checking is enabled, CA Access Control performs security category checking in addition to its other authorization checking. When a user requests access to a resource that has one or more security categories assigned to it, CA Access Control compares the list of security categories in the resource record with the category list in the user record. If every category assigned to the resource appears in the user's category list, CA Access Control continues with other authorization checking; otherwise, the user is denied access to the resource.

If the SECLABEL class is active, CA Access Control uses the list of security categories associated with the security labels of the resource and user; the lists of categories in the user and resource records are ignored.

To protect a resource by security category checking, assign one or more security categories to the resource's record. The category parameter of the newres or chres command assigns security categories to a resource.

To allow a user access to resources protected by security category checking, assign one or more security categories to the user's record. The category parameter of the newusr or chusr command assigns security categories to a user.

Enabling Security Category Checking

The following setoptions command enables security category checking:

setoptions class+ (CATEGORY)

Disabling Security Category Checking

The following setoptions command disables security category checking:

setoptions class-(CATEGORY)

Defining a Security Category

Define a security category by defining a resource in the CATEGORY class. The following newres command defines a security category:

newres CATEGORY *name*

where *name* is the name of the security category.

To define the security category "*Sales*," enter the following command:

newres CATEGORY Sales

To define the security categories "*Sales*" and "*Accounts*," enter the following command:

newres CATEGORY (Sales,Accounts)

Listing Security Categories

To display a list of all the security categories that are defined in the database, use the show command as follows:

find CATEGORY

The list of security categories displays on the screen.

Deleting a Security Category

Delete a security category by removing its record from the CATEGORY class. The following rmres command removes a security category:

rmres CATEGORY *name*

where *name* is the name of the security category.

To remove the security category "*Sales*," enter the following command:

rmres CATEGORY Sales

Security Labels

A security label represents an association between a particular security level and zero or more security categories.

When security label checking is enabled, CA Access Control performs security label checking in addition to other authorization checks. When a user requests access to a resource that has a security label assigned to it, CA Access Control compares the list of security categories specified in the resource record's security label with the list of security categories specified in the user record's security label. If every category assigned to the resource's security label appears in the user's security label, CA Access Control continues with the security level check; otherwise, the user is denied access to the resource. CA Access Control compares the security level specified in the resource record's security label with the security level specified in the user record's security label. If the security level assigned in the user's security label is equal to or greater than the security level assigned in the resource's security label, CA Access Control continues with other authorization checking; otherwise, the user is denied access to the resource.

When security label checking is enabled, the security categories and security level specified in the user and resource records are ignored; only the security level and categories specified in the security label definitions are used.

To protect a resource by security label checking, assign a security label to the resource's record. The label parameter of the newres or chres command assigns a security label to a resource.

To allow a user access to resources protected by security label checking, assign a security label to the user's record. The label parameter of the newusr or chusr command assigns security labels to a user.

Enabling Security Label Checking

The following setoptions command enables security label checking:

setoptions class+(SECLABEL)

Disabling Security Label Checking

The following setoptions command disables security label checking:

setoptions class-(SECLABEL)

Defining a Security Label

Define a security label by defining a resource in the SECLABEL class. The following newres command defines a security label:

newres SECLABEL *name* category(*securityCategories*) level(*securityLevel*)

where:

name

Specifies the name of the security label.

securityCategories

Specifies the list of security categories. To specify more than one, separate the security category names with a space or a comma.

securityLevel

Specifies the security level. Use an integer between 1 and 255.

To define the security label Managers to contain the security categories Sales and Accounts and a security level of 95, enter the following command:

newres SECLABEL Manager category(Sales,Accounts) level(95)

Listing the Security Labels

To display a list of all the security labels that are defined in the database, use the show command as follows:

find SECLABEL

The list of security labels appears on the screen.

Deleting a Security Label

A security label is deleted by removing its record from the SECLABEL class. The following rmres command removes a security label:

rmres SECLABEL *name*

where *name* is the name of the security label.

To remove the security category *"Manager"* enter the following command:

rmres SECLABEL Manager

Chapter 13: Auditing Events

This section contains the following topics:

Setting Audit Rules

For security auditing, CA Access Control keeps audit records for events of access denial or access grants according to the audit rules defined in the database.

Every accessor and resource has an AUDIT property that can be set to one or more of the following values:

FAIL

Logs access failures by the accessor to the resource.

SUCCESS

Logs successful accesses by the accessor to the resource.

LOGINFAIL

Logs every logon failure by the accessor. (This value does not apply to resources.)

LOGINSUCCESS

Logs every successful logon by the accessor. (This value does not apply to resources.)

ALL

Logs the same information as FAIL, SUCCESS, LOGINFAIL, and LOGINSUCCESS for accessors or FAIL and SUCCESS for resources.

NONE

Logs nothing concerning the accessor or resource.

TRACE

Logs the same information as ALL and all system events. (This value does not apply to resources.)

Whenever you create or update an accessor or resource record in the database, you can specify the AUDIT property. You can also specify whether email notification of logged events should be sent and to whom.

The records in the audit log accumulate according to these audit rules. The decision whether to log an event is based on the following:

- If the resource or accessor has AUDIT(ALL), all login events for the accessor and all events concerning resources protected by CA Access Control are logged, regardless of whether access failed or succeeded.

- If access to a resource protected by CA Access Control is successful and the accessor or resource has AUDIT(SUCCESS), the event is logged.

- If access to a resource protected by CA Access Control fails and the accessor or resource has AUDIT(FAIL), the event is logged.

In addition, if you set a user to be traceable, each time a trace record is written for that user, a corresponding audit record is written to the audit log.

Define the Audit Events That CA Access Control Writes to the Audit Log

CA Access Control writes access success and failures to the audit log. To define which access events CA Access Control writes to the audit log, change the value of the AUDIT property for the resource or accessor that you want to audit. You can also use this method to specify that CA Access Control logs every trace event to the audit log.

You use the AUDIT property to specify the audit events that CA Access Control writes to the audit log. Use selang or CA Access Control Endpoint Management to set the AUDIT property for resources and accessors as follows:

Value of AUDIT	What CA Access Control Logs	Applicable Objects
FAIL	Access failures	Users and resources
SUCCESS	Access successes	Users and resources
LOGINFAIL	Login failures	Users
LOGINSUCCESS	Login successes	Users
ALL	Equivalent to FAIL, SUCCESS, LOGINFAIL and LOGINSUCCESS	Users and resources
TRACE	Equivalent to ALL plus all system events	Users

Value of AUDIT	What CA Access Control Logs	Applicable Objects
NONE	No logging	Users and resources

Note: If the audit property of a user is not set, the AUDIT value of a group or profile group can affect the audit mode CA Access Control uses for the user.

How CA Access Control Determines the Audit Mode for a User

The audit mode for a user specifies which audit events CA Access Control sends to the audit log for that user. The following process describes how CA Access Control determines the audit mode for a user:

1. CA Access Control checks if the user's record in the USER or XUSER class has a value for the AUDIT property.

   If the user's record has a value for the AUDIT property, CA Access Control uses that value as the audit mode for the user.

2. CA Access Control checks if the user is assigned to a profile group. If the user is assigned to a profile group, CA Access Control checks if the profile group's record in the GROUP class has a value for the AUDIT property.

   If the user is assigned to a profile group and the profile group's record has a value for the AUDIT property, CA Access Control uses that value as the audit mode for the user.

3. CA Access Control checks if the user is a member of a group. If the user is a group member, CA Access Control checks if the group's record in the GROUP or XGROUP class has a value for the AUDIT property.

   If the user is group member and the group's record has a value for the AUDIT property, CA Access Control uses that value as the audit mode for the user. If the user is not a member of a group, or if the group's record does not have a value for the AUDIT property, CA Access Control assigns the systemwide audit mode to the user.

   Note: The user's audit mode accumulates if a user is a member of more than one group and the groups have different audit modes. The audit mode for the user is the sum of all the audit modes for the groups of which they are members.

Note: If CA Access Control uses the value of a group's AUDIT property to determine the audit mode for a user, and you change the group's audit mode while the user is logged in, the audit mode for the logged-in user also changes. The user does not have to log off for the change in group audit mode to take effect.

The following diagram shows how CA Access Control determines the audit mode for a user:

```
                          ┌─────────┐
                          │  Start  │
                          └────┬────┘
                               │
                               ▼
                         ╱Does the user have an╲
          ◄─────Yes──────    audit mode?
                         ╲                    ╱
                               │
                               No
                               │
                               ▼
                         ╱Is the user assigned╲        ╱Does the profile╲
                         ╲ to a profile group? ──Yes──► group have an audit
                                                ╲        mode?          ╱
                               │                      │          │
                               No                     No         Yes
                               │
                               ▼
                    ┌──────────────────┐
                    │ Accumulate the audit │
                    │ mode from the groups of │◄──────────────┘
                    │ which the user is a  │
                    │     member       │
                    └──────────┬───────┘
                               │
                               ▼
┌──────────┐  ┌──────────┐  ╱Do one or more╲  ┌──────────┐  ┌──────────┐
│ The audit │  │ The audit │  ╲ groups have an ──No──► The audit │  │ The audit │
│ mode for   │  │ mode for  ◄Yes   audit       │  │ mode for  │  │ mode for  │
│ the user is│  │ the user is│  ╲  mode?     ╱  │ the user is│  │ the user is│
│ the user's │  │ the       │                  │ the       │  │ the       │
│ audit mode │  │ accumulated│                  │ systemwide │  │ profile   │
└────┬─────┘  │ group audit│                  │ audit mode│  │ audit mode│
              └────┬─────┘                  └────┬─────┘  └────┬─────┘
                               │
                               ▼
                          ┌─────────┐
                          │   End   │
                          └─────────┘
```

Example: Audit by Groups

User Jan is a member of Group A and Group B. Group A has an audit mode of FAIL and Group B has an audit mode of SUCCESS. Because Jan is a member of both groups, Jan has the accumulated audit mode of FAIL and SUCCESS.

More information:

How CA Access Control Uses Profile Groups to Determine User Properties (see page 37)

Default Audit Modes for Users and Enterprise Users

When you create a user (USER object), CA Access Control assigns the default AUDIT_MODE to the object. The default value of the AUDIT_MODE property is Failure, SuccessLogin, SuccessFailure.

When you create an enterprise user (XUSER object), by default CA Access Control does not assign a default AUDIT_MODE value to the object.

Note: (UNIX) To change the default value of the AUDIT_MODE property for USER objects, edit the value of DefaultAudit in the [newusr] section of the lang.ini file.

Warning Mode

Warning Mode is a property that you can apply to a resource, and an option that you can apply to a class. If Warning mode is applied to a resource or a class and an access violates an access rule, CA Access Control writes an audit log entry with the return code W, but permits the access to the resource. If a class is in Warning mode, all the resources in that class are in Warning mode.

Warning Mode only has an effect if CA Access Control is in Full Enforcement mode.

Note: Full Enforcement mode is the only mode CA Access Control for UNIX supports. CA Access Control for Windows also supports Audit Only mode.

You can use Warning mode when you introduce or modify an access policy. If you do this, you can examine the audit log to preview the results of your intended policy before you put that policy into effect. You can display the audit log by using the seaudit command.

If a class has the property *warning,* you can put the class into Warning mode. If a resource group or class is in Warning mode, when an access rule is violated, CA Access Control allows the access and writes an entry in the audit log that references the resource (not the resource group or class).

The Warning mode settings on a resource and on a class are independent: if you put a resource into Warning mode, it remains in Warning mode, even if it belongs to a class and you remove Warning mode from that class.

Note: You can only put resources or classes into Warning mode if they have the property *warning*; not all resources or classes have this property.

Put a Resource into Warning Mode

You put a resource into Warning mode to monitor the effects of access rules, without needing to enforce these rules.

Note: As well as putting individual resources into Warning mode, you can put a class into Warning mode (see page 171).

To put a resource into Warning mode

1. In CA Access Control Endpoint Management edit the resource you want to put into Warning mode.

   The appropriate Modify page appears.

2. Click the Audit tab.

   The Audit Modes page for the resource appears.

3. Select Warning Mode, and click Save.

   The resource you modified is now in Warning mode.

Note: In Warning mode, CA Access Control always writes warning records to the audit log when access is permitted but access rules are violated: you do not need to set the audit property on the resource for this to happen.

Use the sereport utility (report number 6) to see all resources in Warning mode.

Example: Put a File into Warning Mode

The following selang example puts the file c:\myfile into Warning mode:

```
chres FIlE c:\myfile warning
```

Example: Clear Warning Mode from a File

The following selang example takes the file c:\myfile out of Warning mode:

```
chres FIlE c:\myfile warning-
```

Warning mode is now not active for the myfile, so CA Access Control enforces the access rules for myfile.

Example: Put a Terminal into Warning Mode

The following selang example puts the terminal myterminal into warning mode:

```
chres terminal myterminal warning
```

CA Access Control permits access by any authorized user from the terminal myterminal, but logs an audit record for any user that normally would be denied access from that terminal.

Put a Class into Warning Mode

Rather than putting individual records into Warning mode, you can put all records in a class into Warning mode. You might use Warning mode to monitor the effects of access rules, without needing to enforce these rules.

To put a class into Warning mode

1. In CA Access Control Endpoint Management, do as follows:

    a. Click Configuration.

    b. Click Class Activation.

    The Class Activation page appears.

2. Select the check box in the Warning column for the class you want to put into Warning mode.

3. Click Save.

    A confirmation message appears, letting you know that CA Access Control options have been successfully updated.

Find Out Which Resources Are in Warning Mode

You should use Warning mode as a temporary measure when implementing CA Access Control. Once you are comfortable that users have the required access to the resources they require, you should turn off Warning mode and CA Access Control will start enforcing the associated rules.

To find out which resources are in Warning mode, you can create a report that shows all resources with Warning mode.

To create a report, enter the following command:

sereport -r 6

CA Access Control creates the report.

Note: For more information about the sereport utility, see the *Reference Guide*.

Find Out Which Classes Are in Warning Mode

You should use Warning mode as a temporary measure when implementing CA Access Control. Once you are comfortable that users have the required access to the resources they require, you should turn off Warning mode and CA Access Control will start enforcing the associated rules.

To find out which classes are in Warning mode, you can get CA Access Control to display this data.

To display this data, enter the following selang command:

setoptions cwarnlist

CA Access Control displays a table showing the classes that are in Warning mode.

Note: For more information about setoptions, see the *selang Reference Guide*.

Audit Logs

The audit records are stored in a file called the audit log. The location for the audit log is specified in the seos.ini file. The seaudit utility or CA Access Control Endpoint Management can be used to list recorded events in the audit log, filter events by time restrictions or event type, and so on.

Note: For more information about seaudit, see the *Reference* Guide.

The audit logs are stored locally, but you can use CA Access Control to distribute the auditing information by using the log routing facility. Consider archiving old audit logs to tape, to allow you to scan the events later.

By default, the authorization daemon seosd creates the audit logs with root ownership, since the seosd program is executed by the user root. For the same reason, the audit logs are created with read/write permissions granted only to root.

To enable other users to read the audit logs without having to su (substitute user) to root, CA Access Control includes two entries in the seos.ini file that specify which group ownership is assigned to the log files.

- One entry is for the audit log.

  Suppose the auditors at your site are all members of a group named auditforce. You want these users to be able to browse through the local audit log files. Edit the seos.ini file so that the audit_group token in the [logmgr] section is set to auditforce. CA Access Control then gives the auditforce group read permission to your local audit logs. From this point, any local audit logs created at your station have the auditforce group as their owner.

  The log routing daemons consult the same token to see who should have access rights to the audit logs that the daemons produce and collect. Note that the audit logs are subject to access control like any other files, and CA Access Control rules can keep users from accessing them.

- The other entry is for the error log, and it is used in the same way to specify group ownership for that file.

The System Auditor

A system auditor is a user to whom the AUDITOR attribute is assigned. Users defined as system auditors are permitted to perform auditing tasks such as changing the auditing attribute that is assigned to users and resources.

Auditing tasks can be carried out from central locations. To collect auditing information from the various stations on the network in a single host, the auditor can use the log routing facility.

Set Up the Log Routing Facility

To set up the log routing

1. Create a log routing configuration file.

   Unless you specify otherwise with the RouteFile token in the seos.ini file, CA Access Control expects your log routing configuration file to be named *ACInstallDir*/log/selogrd.cfg

   where *ACInstallDir* is the installation directory for CA Access Control, by default /opt/CA/AccessControl.

   You can find sample log routing configuration files in the directory *ACInstallDir*/samples/selogrd.init. Alternatively, as a very simple log routing configuration file, you can create a file consisting of the following three lines:

   ```
   Rule
   host destination
   .
   ```

   For *destination*, enter the name of the host that should receive the audit records. All classes, resources, accessors, and results are logged.

   Note: For more information about the syntax of the configuration file, see the selogrd utility in the *Utilities Guide*.

2. Start the emitter daemon (selogrd) on all hosts that are to route auditing information, and execute the collector daemon (selogrcd) on all hosts that are to collect auditing information.

   Note: For more information about using these daemons, see the *Reference Guide*.

File Notifications

Besides compiling the log, the log routing facility can also send notifications to the host's display screen, to an email address, or to other destinations. You can base notifications on information from your station's own audit log or from logs that the collector daemon has brought to your station.

To set up such notifications, you need to use the log routing configuration file *and* a selang command. For example, suppose you want to notify the user John whenever a setuid request to user root is successfully made.

1. Issue the following selang command:

   chres SURROGATE USER.root notify(John)

   This chres command specifies that each time someone surrogates user to root, a special audit log record is created, and the seosd daemon is to notify the user named John. The daemon also creates a special kind of audit record called a *notification record*.

2. Once you have specified notification for one or more resources, you can add the following three lines to the log routing configuration file.

   Rule2
   notify default

   .

   This line causes the log routing emitter to create a mail message for the notification audit record.

   Note: For more information about the configuration file format and setting up the log routing daemons, see the *Reference Guide*.

Log Routing

CA Access Control uses the log routing daemon, selogrd, to distribute selected local audit log records to specific hosts; reformat audit log records into email messages, ASCII files, or user windows; and transmit notification messages based on audited events.

To determine audit record routing, selogrd uses a configuration file, selogrd.cfg. This file is a list of which audit log records to route-or not to route-and to where. For a complete description of this file, see the *Reference Guide*.

Log Routing Configuration

To start selogrd or selogrcd automatically when seosd starts, set the seos.ini tokens selogrd or selogrcd in the [daemons] sections to yes. Then when you run seload, seload starts the daemons for you.

For example, the appropriate tokens in the [daemons] section of the soes.ini should look as follows:

selogrd = yes
selogrcd = yes

Since the log-routing facility uses RPC to route audit records, placing a log audit collector behind a firewall does not allow simple blocking of UDP ports because there is no way to know which port the portmapper assigns to the server daemon. To solve this problem, you can use the token ServicePort to assign a predefined port to the server daemon.

If the firewall allows port 111 from outside the network (portmapper port), you should only change the seos.ini file in the server. If the firewall does not allow communication to portmapper in the protected network, both clients and server must agree on a specific port.

You can ensure this by setting the same value in the ServicePort token in the seos.ini files of both clients and the server. You can specify a number-which means that the daemons bind to the specified port-or a service name. If you specify a service name, both clients and the server must have the same service resolution. For example, if you specify the service name seoslogr, then add the following to the /etc/services file of the clients and the server:

seoslogr 2022/udp # Audit log-routing

If the clients or the server are using NIS to resolve services, you must update the NIS services map.

Audit Log Route Encryption

You can encrypt audit log records. When you use encryption, the selogrd daemon encrypts audit log record before sending it to the collector (selogrcd or audit log router). The collector in turn decrypts the received records.

CA Access Control provides two encryption styles for selogrd: CA Access Control standard encryption, and audit log encryption through adcipher. For encryption, selogrd uses functions from shared library objects, as specified in the [selogrd] section of the seos.ini file.

Standard encryption uses the shared library libcrypt; Audit encryption uses functions from a file specified by the CipherName token. By default, the file name is adcipher, which is a symbolic link to the desired shared library. The CA Access Control installation process places four shared libraries in the CA Access Control/lib directory: lib1des, lib3des, libIDEA, and libblowfish.

CA Access Control maintains the standard encryption key in the shared library, while the audit encryption uses a separate file as specified by the KeyFile token (default value: adcipher.bin).

Use the UseEncryption token to determine the type of encryption:

- To use CA Access Control standard encryption, specify UseEncryption=native

- To use audit log encryption through adcipher, specify UseEncryption=eTrust, and enter the appropriate values for the CipherName and KeyFile tokens.

- To disable selogrd encryption, specify UseEncryption=no.

Use the RefuseUnencrypted token to accept or deny unencrypted audit. It is used in conjunction with the UseEncryption token and is redundant if the UseEncryption is set to no:

- To refuse unencrypted audit, specify RefuseUnencrypted=yes

- To accept both encrypted and unencrypted audit, specify RefuseUnencrypted=no

Note: The selogrcd daemon uses the same tokens in the seos.ini file.

To change the encryption key, use the sechkey utility, described in this chapter.

Important! If you send records to the audit collector, be sure that both selogrd and the collector use the same shared encryption file and encryption key.

Send Audit Log Records via Email

selogrd can send records to email targets directly. You can direct email messages through a mailer utility (the old method), or directly to the mail exchange server using SMTP.

To send audit log records directly to the mail exchange server, set the UseSmtpMail token in the [selogrd] section of the seos.ini file.

You can also specify the following:

- A time-out in case the mail server does not answer, using the SmtpTimeLimit token

- The "From:" mail header field, using the SmtpMailFrom token

- The mail server host address, using the SmtpMailServer token

Note: This method does not use UNIX mail utility; rather, it establishes a direct connection with mail server, and uses SMTP protocol to send mail.

Configure SNMP Traps

For systems that use the Internet network management protocol SNMP (Simple Network Management Protocol), you can configure selogrd to create SNMP traps using CA Access Control audit records.

To implement the SNMP traps, first locate the SNMP shared objects provided in the CA Access Control libraries, and then configure selogrd correctly using these shared objects.

The shared objects-usually found in the directory *ACInstallDir*/lib- are called snmp.*xx* and libsnmp.*xx*, where the *xx* extension varies according to the platform. The possible extensions are:

- **.o**—AIX platform

- **.sl**—HP platform

- **.so**—All other platforms

If you want to use the SNMP extension of selogrd, and CA Access Control is not installed in the default location, you must set the following environment variables before running selogrd:

- In AIX, set LIBPATH to *ACInstallDir*/lib

- In Solaris, set LD_LIBRARY_PATH to *ACInstallDir*/lib

- In Linux, set LD_LIBRARY_PATH to *ACInstallDir*/lib

- In HP, set SHLIB_PATH to *ACInstallDir*/lib

where *ACInstallDir* is the directory where you installed CA Access Control.

To configure selogrd to use the shared objects

1. Create a file called *ACInstallDir*/etc/selogrd.ext.

2. Define where the SNMP shared objects are by adding a single line to the file *ACInstallDir*/etc/selogrd.ext with the appropriate path for the snmp.so. (It is enough to specify this shared object for the other to automatically be linked.) For example:

   snmp /opt/CA/AccessControl/lib/snmp.so

3. Finally, you must configure the selogrd.cfg file to specify what type of action should trigger SNMP traps, and which location should be notified when SNMP traps are triggered. Configuration is very similar to that for other auditing notification, with the delivery system specified as snmp.

   For example, suppose you want to have SNMP traps activated when CA Access Control starts and shuts down, and have notification of these SNMP traps sent to AuditPC. You can do this by adding the following section to the selogrd.cfg configuration file:

   ```
   snmpRule
   snmp AuditPC
   include Class(START).
   include Class(SHUTDOWN).
   .
   ```

Similarly, you can activate the SNMP traps by other actions or types of access, or have them sent to other locations.

Migrate User Trace Filters

If you set a user to be traceable, each time a trace record is written for that user, a matching audit record is written to the seos.audit file. In previous releases of CA Access Control, these audit records were filtered by the trcfilter.init file. In CA Access Control r12.0 SP1 and later, the audit records generated by user trace records are filtered by the audit.cfg file, which filters all other audit records.

You must manually migrate the audit record filters from trcfilter.init to audit.cfg. If you do not migrate the filters, the audit records generated by user traces will not be filtered.

Note: Trace records are still filtered by trcfilter.init. Do not migrate trace filters from trcfilter.init to audit.cfg.

To migrate the user trace filter

1. In trcfilter.init, find the user trace filter that you need to migrate.

   The trace_filter setting in the seosd section of the seos.ini file determines the location of this file.

2. In audit.cfg, type the following, where *usertracefilter* is the user trace filter from trcfilter.init:

   TRACE;*;*;*;*;*usertracefilter*

3. (Optional) Repeat Steps 1-2 for each user trace filter that you need to migrate.

Example: Migrate User Trace Filter

In this example, the following user trace filter is in the trcfilter.init file:

*ExampleFilter

To migrate this user trace filter, type the following on a new line in the audit.cfg file:

TRACE;*;*;*;*;*ExampleFilter

Chapter 14: Scope of Administration Authority

This section contains the following topics:

Global Authorization Attributes

Global authorization attributes are set in the user record. Each global authorization attribute permits the user to perform certain types of functions. This section describes the functions and the limits of each global authorization attribute.

ADMIN Attribute

The ADMIN attribute lets a user execute almost all commands in CA Access Control. Users who are defined in the database with the ADMIN attribute can define and update users, groups, and resources in the database. This is the most powerful attribute in CA Access Control, but it does have limitations:

- If only one user in the database has the ADMIN attribute, that user cannot be deleted, and the ADMIN attribute cannot be removed from the record.

- Users with the ADMIN attribute but without the AUDITOR attribute cannot change the type of auditing that is done on a user, group, or resource (audit mode). If you have the ADMIN attribute and need to change the auditing characteristics of a user, group, or resource, assign yourself the AUDITOR attribute.

- Users with the ADMIN attribute cannot delete superuser (the root account on UNIX or the Administrator account on Windows), but they can set root to be a non-ADMIN user.

AUDITOR Attribute

Users with the AUDITOR attribute can monitor system usage. Explicit privileges of a user with the AUDITOR attribute include the following:

- Users can display information in the database.

  Auditors can execute the selang commands *showusr*, *showgrp*, *showres*, and *showfile*.

- Users can set the audit mode for existing records.

  Auditors can execute the selang commands *chusr*, *chgrp*, *chres*, and *chfile*.

OPERATOR Attribute

Users with the OPERATOR attribute have READ access to all files. With this access, they can list everything in the database, and they can run backup jobs. To list database records, operators use the showusr, showgrp, showres, showfile, and find commands. The OPERATOR attribute also lets a user use the secons utility.

Note: For more information about the secons utility, see the *Reference Guide*.

PWMANAGER Attribute

The PWMANAGER attribute gives a regular user the authority to use the chusr or sepass command to change the passwords of other users.

Note: To let the PWMANAGER change the ADMIN user's password, set the *cng_adminpwd* option of the setoptions command. For more information, see the *selang Reference Guide*.

The PWMANAGER attribute does not include authority to change the number of grace logins, the password interval of another user, or general password rules.

The PWMANAGER's authority also includes use of the showusr and find commands.

Note: If a user has the nochngpass property set to yes, a PWMANAGER cannot change the password for that user.

SERVER Attribute

CA Access Control, like many other security models, does not permit a regular user to ask: "Can user A access resource X?" The only question a regular user can ask is: "Can I access resource X?" However, a process that supplies services to many users, such as a database server service or an in-house application, should be permitted to ask for authorization on behalf of other users.

The SERVER attribute allows a process to ask for authorization for users. Users with the SERVER attribute set can issue the SEOSROUTE_VerifyCreate API.

Note: For more information about the server attribute and CA Access Control APIs, see the *SDK Guide*.

IGN_HOL Attribute

The IGN_HOL attribute allows users to log in during any period defined in a holiday record. Each record in the HOLIDAY class defines one or more periods when users need extra permission to log in. With the IGN_HOL attribute, users can log in at any time, regardless of the periods defined in holiday records.

Note: For more information about the HOLIDAY class, see the *Reference Guide*.

Group Authorization

It is necessary to understand the concept of parentage before discussing group authorization attributes.

Parentage

The concept of subordinate and superior groups, also known as parentage, is important when discussing group administration privileges. One group can be the parent-superior-of one or more groups. A *child* or subordinate group can have only one parent. Assigning a parent to a group is optional. Consider the following diagram:
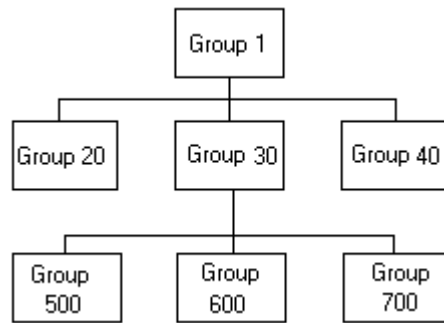
Group 1 is the parent of the three Groups 20, 30, and 40. Group 30 is also the parent of three groups-500, 600, and 700. Group 600 has only one parent-Group 30. Group 1 has no parent.

Group Authorization Attributes

All records, including resource records and accessor records alike, have owners. Owning a record means having authorization to view, edit, and remove it.

A group can own its own records. However, within a group that owns records, only certain privileged users can manage the records. These special users have a group authorization attribute set in their own user records. The group authorization attributes are the following:

- GROUP-ADMIN

- GROUP-AUDITOR

- GROUP-OPERATOR

- GROUP-PWMANAGER

The join command-which only a properly authorized user can issue-sets these attributes. The join command serves the purpose of both putting a user into a group, and specifying the user's group authorization attribute (if any).

The privileged members of the group may or may not be authorized to manage the user records that define the members of the group, depending on who owns those records.

More information:

Ownership (see page 186)

GROUP-ADMIN Attribute

Users with a group administration authorization attribute can create a certain set of records. In order to create a record, the group administrator has to specify the owner of the record.

The owner of the records must be the group in which the user has a group authorization attribute. If that group is the parent of other groups, the owner can also be from one of the sub groups. The whole set of records is called the group scope. The authorization examples provided illustrate the concept of group scope.

Users with the GROUP-ADMIN attribute have the following access authority for the records within their group scope:

Access	Description	Commands
Read	Show the properties of the record.	showusr, showgrp, showres, showfile
Create	Create new records in the database. You must specify the owner.	newusr, newgrp, newres, newfile
Modify	Change the properties of the record.	chusr, chgrp, chres, chfile
Delete	Remove records from the database.	rmusr, rmgrp, rmres, rmfile
Connect	Join a user to a group or separate a user from a group.	join, join-

The GROUP-ADMIN attribute also has limits:

- GROUP-ADMIN users cannot make resources inaccessible to themselves, so:
  - GROUP-ADMIN users cannot assign a security level that is higher than their own security level.
  - GROUP-ADMIN users cannot assign a security category or security label that they do not have.
- GROUP-ADMIN users cannot delete the user superuser (the root account on UNIX or the Administrator account on Windows) from the database.

- Several limitations concern the global authorization attributes described in Global Authorization Attributes in this chapter:

  – A GROUP-ADMIN user cannot delete the only ADMIN user record in the database.

  – A GROUP-ADMIN user cannot remove the ADMIN attribute from the record of the last ADMIN user in the database.

  – GROUP-ADMIN users without the AUDITOR attribute cannot update the audit mode. Only a GROUP-ADMIN user with the AUDITOR attribute can update the audit mode.

  – GROUP-ADMIN users cannot set the global authorization attributes-ADMIN, AUDITOR, OPERATOR, PWMANAGER, and SERVER-for any user.

GROUP-AUDITOR Attribute

A user with the GROUP-AUDITOR attribute can list the properties of any record within the group scope. The group auditor can also set the audit mode for any record within the group scope.

GROUP-OPERATOR Attribute

A user with the GROUP-OPERATOR attribute can list the properties of any record within the group scope.

GROUP-PWMANAGER Attribute

A user with the GROUP-PWMANAGER attribute can change the password of any user whose record is within the group scope.

Ownership

Every record in the database-including both accessor records and resource records-has an owner. When you add a record to the database, you can either explicitly assign its owner by using the owner parameter or let CA Access Control assign the user who defines the record as the owner of the record.

Accessors own a record if *any* of the following are true:

- They are defined as the owner of the record.

- They are members of a group that is defined as the owner of the record *and* they have joined the group with the GROUP-ADMIN property.

- They are owners of a resource group record that the resource is a member of.

If you remove a user or group that owns records from the database, the records no longer have an owner.

Users who own records have the following access authority for the records they own:

Access	Description	Commands
Read	Show the properties of the record.	showusr, showgrp, showres, showfile
Modify	Change the properties of the record.	chusr, chgrp, chres, chfile
Delete	Remove the record from the database.	rmusr, rmgrp, rmres, rmfile
Connect	Join a user to a group or separate a user from a group.	join, join-

If you do not want a user or group to have ownership authority over a particular record, assign the owner *nobody* to the record and to any resource group record that the record is a member of.

The limits of the ownership privileges are as follows:

- The owner of the last ADMIN user in the database cannot delete that user record.

- Owners who do not have the AUDITOR attribute cannot update the audit mode. Only an owner with the AUDITOR attribute can update the audit mode.

- The owner of a superuser (the root account on UNIX or the Administrator account on Windows) cannot delete root from the database.

- Owners cannot set the global authorization attributes-ADMIN, AUDITOR, OPERATOR, and PWMANAGER-for the users they own.

- Owners cannot make resources inaccessible to themselves, so:

  – Owners cannot assign a security level that is higher than their own security level.

  – Owners cannot assign a security category or security label that they do not have.

File Ownership

CA Access Control allows the owner of a file to protect the file by defining a record in the FILE class. The owner of the file has full authority over the record of that file, so the owner can use the newfile, chfile, showfile, authorize, and authorize- commands with all parameters for the record that protect the file.

On UNIX, when a user creates a file, UNIX assigns the user as the owner of the file. CA Access Control allows UNIX file owners to define FILE records, unless this feature is explicitly disabled. If you do not want file owners to define FILE records, make sure that the use_unix_file_owner token in the [seos] section of the seos.ini file to no. (This is the default setting.)

Authorization Examples

Following are diagrams that illustrate the concepts of group authorization attributes, parentage, ownership, membership, and group scope. These diagrams only contain users and groups, but the concept of ownership also applies to resource and file records.

Single Group Authorization

In the following diagram, four users are members of Group 1: MU1, MU2, MU3, and MU4. Group 1 also owns three users-OU5, OU6, and OU7. The member MU4 has the GROUP-ADMIN attribute.
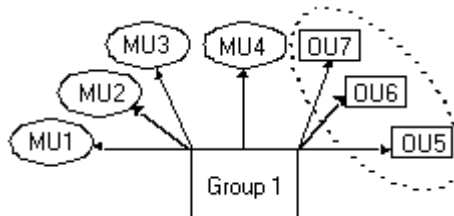
The ellipse indicates the group scope of the commands executed by user MU4. It includes all the users owned by Group 1-OU5, OU6, and OU7.

Parent and Child Groups

In the following diagram, four users are members of Group 1: MU1, MU2, MU3, and MU4. Group 1 also owns three users-OU5, OU6, and OU7. The member MU4 has the GROUP-ADMIN attribute set in its record.
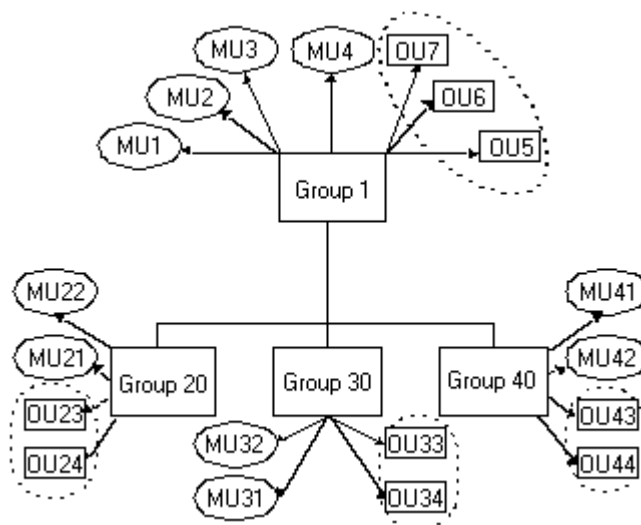
Group 1 is also the parent of three groups-20, 30, and 40. Each of these subordinate groups has two users who are members of the group and two users who are owned by the group.

The four ellipses indicate the group scope of the commands executed by user MU4. It includes all the users owned by Group 1, as well as the users owned by the groups subordinate to Group 1. The users in the group scope of MU4 are OU5, OU6, OU7, OU23, OU24, OU33, OU34, OU43, and OU44.

If there were groups subordinate to Groups 20, 30, or 40 that owned users, groups, or resources, the records owned by these groups would also be in the group scope of commands executed by user MU4.

Sub Administration

Security administrators (users with the ADMIN attribute) can grant specific administrative privileges to regular users. These regular users are then called sub administrators. Sub-administrators have privileges to manage only specified CA Access Control classes or objects. For example, a sub administrator can be authorized to manage only user and group objects. You can set a higher level of sub administration by authorizing the sub admin user the administrative privileges for specific objects in a class.

Sub administrators of users, groups and resources can use selang to perform administrative tasks related to these resources.

How to Grant Specific Administrative Privileges to Regular Users

Because administrators—users with the ADMIN attribute—can execute almost all actions in CA Access Control, you may want to delegate specific administrative tasks to sub administrators. To do this, you need to grant those users with privileges to classes in the CA Access Control database that control the specific administrative tasks the user needs to perform as follows:

1. Identify one or more classes that control the tasks you want to delegate.

   For example, CA Access Control uses the USER and GROUP classes to create accessor resources. If you want to delegate accessor management, you then need to use the USER and GROUP records of the ADMIN class.

2. Authorize one or more sub administrator to the applicable resource of the ADMIN class.

   For example, to let a sub administrator view and modify user records, grant the user with *read* and *modify* access to the USER record of the ADMIN class.

The ADMIN Class

Sub administrators—users listed in the access control list (ACL) of records in the class ADMIN—have privileges similar to users with the ADMIN attribute. However, the privileges of users in the ACL for records in the class ADMIN are limited to the particular class represented by the record. For example, the SURROGATE record in the ADMIN class determines which users can administer records of the SURROGATE class.

Note: For more information about CA Access Control classes, see the *Reference Guide*.

A user in the ACL for a particular record in class ADMIN can execute the following commands:

Access	Description	Commands
Read	Show the properties of the record in the class.	showusr, showgrp, showres, showfile, find
Create	Create new database records in the class.	newusr, newgrp, newres, newfile
Modify	Change properties in the class.	chusr, chgrp, chres, chfile
Delete	Remove existing class records from the database.	rmusr, rmgrp, rmres, rmfile
Connect	Add users to and remove users from groups. This access is valid only in the ACL of the GROUP record.	join, join-
Password	Control the password of all users within the database, and their password attributes. This access grants the same authority as the access permitted a user with the PWMANAGER attribute. This is valid only in the ACL for record USER.	chusr

Users with ADMIN class privileges have the following limitations:

■ Users defined in the ACL of the USER record in class ADMIN cannot delete the last ADMIN user in the database.

■ ADMIN class users cannot set the global authorization attributes-ADMIN, AUDITOR, OPERATOR, and PWMANAGER-for the users they own.

■ ADMIN class users cannot necessarily update the audit mode. Only an ADMIN class user with the AUDITOR attribute can update the audit mode.

■ ADMIN class users cannot delete superuser (the root account on UNIX or the Administrator account on Windows), but they can set root to be NOADMIN.

■ ADMIN class users cannot make resources inaccessible to themselves, so:

– ADMIN class users cannot assign a security level to a resource that is higher than their own security level.

– ADMIN class users cannot assign a security category or security label that they do not have.

These limitations are part of the B1 security level certification.

Environmental Considerations

One of the factors governing whether you can update information in your database is the position you occupy in the environment.

Remote Administration Restrictions

You may access a remote station over a network and update the database on the remote station. To update the database on the remote station, both you and your terminal need permission.

■ You must be explicitly defined as a user in the database of the remote station. For whatever commands you want to execute, the appropriate attribute must be set in your user record in the database of the remote station.

■ You must explicitly mention your local terminal's needs in a rule granting it WRITE permission for accessing the remote station; otherwise, you cannot perform CA Access Control administration there.

With WRITE permission through a default access field (_default), or through the UACC class, you can enter the selang command shell at the remote station. However, you *cannot* execute any selang commands or otherwise access to the remote database. With READ permission, you can log in to the remote station but you cannot perform CA Access Control administration there.

Here is an example of this distinction between WRITE and READ permission:

1. To specify a new terminal with READ as default access, where administrators can log in from the terminal but cannot manipulate the database from it, issue the following command:

   newres TERMINAL tty13 defacc(read)

2. To grant user ADMIN1 permission to manipulate the database from the new terminal (that is, grant WRITE permission as well as READ permission), issue the following command:

   authorize TERMINAL tty13 uid(ADMIN1) access(r,w)

UNIX Environment

For managing users and groups in UNIX, users in CA Access Control with global or group authorization attributes have the same privileges and limits for UNIX as they do for CA Access Control.

If you use selang while the seosd daemon is *not* running (for example, at installation time), you must follow these rules:

- You must include the -l option in the selang command.

- The user of selang must be root. (This exclusive root privilege complies with regular UNIX restrictions.)

Windows Environment

For managing users and groups in Windows, users in CA Access Control with global or group authorization attributes have the same privileges and limits for Windows as they do for CA Access Control.

If you use selang while the seosd daemon is *not* running (for example, at installation time), you must follow these rules:

- You must include the -l option in the selang command.

- The user of selang must be Administrator or belong to the Administrators group.

Chapter 15: Improving Performance

This section contains the following topics:

Using Global Access Check

The Global Access Check feature (GAC) lets you access protected, frequently opened files-whose access rules are unlikely to change-much faster than otherwise possible.

GAC allows a CA Access Control administrator to cache rules for read, write, chown, chmod, rename, unlink, utimes, chattr, link, chdir, create, and all, so that appropriate access to files is granted without passing control to seosd. The default is all. Execute requests, however, are not eligible for GAC because they could pose a security loophole.

Without GAC, CA Access Control runs thorough security checks whenever a user or program attempts to access protected files. Frequently accessed files need repeated in-depth checks to confirm access permissions.

GAC allows an administrator for CA Access Control to take for granted that certain frequently accessed protected files require shorter security checks. An administrator for CA Access Control can select files suitable for a shorter check. Before CA Access Control allows a shorter security check, the file must first undergo a full security check based on the set rule. The rule itself consists of a generic file name and a list of accesses. Rules are cached according to users.

Selecting certain files for a shorter check is reliable because, with the GAC feature in place, if a change is actually made to rules regarding the protected files, the shorter security check table is flushed, and an initial full security check is instituted.

Note: GAC restrictions mean that this feature works for every user except root.

How Does GAC Work?

CA Access Control monitors access to specified files and builds a table of permitted accesses during execution time. These are the files you specify in advance in order to set up GAC rules.

Whenever CA Access Control concludes that a user should be granted a certain level of access to a certain file, it checks whether the following two additional conditions are met:

- The granted access is unconditional (that is, not dependent on time, day, program from which executed, or other like conditions).
- The file matches one of its preselected sets of file masks.

Note: File rules define permissions for access to files.

If these conditions are met, CA Access Control generates a UID-file rule-access triplet and stores it in a table composed of such triplets. This table is examined before any database access rule interpretation takes place. Whenever a user attempts to access a file, this table is consulted as a filtering mechanism.

The table is best described as a do-not-call-me table because it contains a list of file masks that, once recognized, no longer need to undergo access permission checks. It is also described as an always-grant table because access is always granted to files specified within its list of file masks.

Whenever a user attempts to access a file, the table is consulted. If the file matches one of the triplets found in the table, the appropriate access is granted without passing control to seosd. This bypasses the access rules analysis. Subsequently, all access to files that match this pattern is granted, based on the triplet stored in the table, without consulting the access rule database.

Whenever a new access rule is added to the database, the entire table is flushed, and the learning process starts from the beginning.

Implementing GAC

To set up GAC, you must choose masks for sets of files that are accessed often, set up a GAC file containing these file masks, and then start the caching process.

Setting Up GAC Rules

Note: File rules in the database are created using the class FILE parameter and file masks. Rules apply to all files matching the file masks. FILE access types include: all, chdir, control, create, delete, execute, none, read, rename, sec, update, utime, write.

From the file rules defined in the database, choose the file masks that you want to cache. Enter a list of file masks into the *ACInstallDir*/etc/GAC.init file (where *ACInstallDir* is the installation directory for CA Access Control, by default /opt/CA/AccessControl), in exactly the same form as they appear in the database.

Each such mask should be specified on a separate line. For example, if the database contains a file mask for /tmp/mydir/* and you want it to be cached, add the following line to the *ACInstallDir*/etc/GAC.init file:

/tmp/mydir/*

Note: Specific file names cannot be specified in the GAC.init file. Only file masks are used.

Starting GAC

To turn your current version of CA Access Control into a GAC compatible version, prepare the file *ACInstallDir*/etc/GAC.init with the file masks that are eligible for caching. Only file masks can be used.

An example is a file named GAC.init in *ACInstallDir*/etc/ with only one line:

/IBBS/REL63/*

GAC Restrictions

GAC implementation has proved to be very efficient, especially in cases where there are hundreds of file accesses in a second, but it has the following restrictions:

■ By default, GAC rules are not applicable for the root user (usually ADMIN). To make the rules applicable to root, set the following token in the [SEOS_syscall] section of the seos.ini file:

GAC_root=1

The default value of the token is 0. To restore the default, set the token to 0, or remove the token.

■ You must not include a file rule that is protected conditionally (for example with day or time restrictions, program pathing, and so on) in the table. If you do specify such a file rule in the GAC.init file, the day or time restrictions and other restrictions no longer apply.

■ A file rule that has audit(ALL) or audit(success) attributes must not be included in the GAC.init file. If such file rule is specified in the GAC.init file, audits of successful accesses are not recorded.

■ The filtering process uses the real (current) UID (that is, the UID that is associated with the process at the time of execution). This provides a loophole to the CA Access Control tracking of the original UID (the one with which the user has originally logged in) and not the current UID. (CA Access Control implements tracking of UID usage to provide the security of more accountability.)

Let us examine an example of how someone might try to take advantage of this loophole. User Tony is not authorized to access the file Accounts/tmp. So Tony surrogates (through /bin/su) to user Sandra, who *is* authorized to access Accounts/tmp. If Sandra has already accessed the Accounts/tmp file, the file appears in the do-not-call-me table with her UID. Tony, using Sandra's UID, is then permitted to access the file. This is because the kernel code does not maintain the history of UIDs.

However, if Sandra has not previously accessed the file, the access permissions are checked in the regular manner using seosd, and Tony is denied access to the file. To close this loophole, the ADMIN user must protect the SURROGATE objects in the database. For this example, the ADMIN could add the following rule to the database:

newres SURROGATE USER.Sandra default(N) owner(nobody)

This command ensures that Tony cannot use the su command to gain Sandra's access privileges.

■ The caching system does not have any impact if the accessor is root. The reason is that no access is granted to root without consulting the database.

Troubleshooting GAC

You can test GAC as follows to see if it is working:

1. Enable the trace (secons -t+).

2. Access a file that corresponds to one of the file masks specified in GAC.init. The first access should be reported in the trace.

3. Try to access the file again. The second file access should not be recorded in the trace.

   If it is, GAC is not working. Check the GAC.init to see that it contains the correct format.

Using the Resource Cache

Another performance improvement tool that CA Access Control offers is resource caching (file cache).

The cache "remembers" the previous answer to an authorization request (permit or deny) for resources in the FILE class. The result is saved with the file name, user name, and authorization response (access mode, program name, and result). When an identical authorization is requested, the request is answered with the last response that was stored in the cache memory tables. This saves time because CA Access Control does not have to reevaluate the request; CA Access Control can return the answer immediately. When rules are changed, the cache is automatically and immediately synchronized.

The cache is a runtime table. An administrator can configure it in two ways:

- Set initialization parameters in the seos.ini file.

- Switch caching to ON or OFF and change parameters at runtime.

The security administrator can define table size, intervals between cleaning tables, and other internal table parameters with tokens in the seos.ini file.

A user with administrative privileges can switch cache tables ON or OFF, change cache parameters, and write cache tables to standard output.

Note: For more information about the secons utility or the [seosd] section of the seos.ini initialization file, see the *Reference Guide*.

Tuning Recommendations

Use these recommendations to improve performance even more:

- If one of the three tables (pools) has the maximum number of records and another table does not, expand the size of the full table.

  Note: The three tables are: file, user, and authorization.

  If a pool has low settings, increase them to expand the pool.

- Do not set the maximum size tokens unless you must. Larger tables take more time when scanning for records.

Using the Network Cache

The network or IP caching feature stores accepted, incoming TCP requests, so they are not sent to the database; instead, they are permitted automatically with the syscall function. This feature improves performance for hosts, which launch many incoming TCP connections.

To activate the IP caching feature, change the following tokens in the [seosd] section of the seos.ini file and restart CA Access Control:

network_cache_timeout

Defines how often to clean the cache table. This token is important if you want to set time limits for the accept requests.

UseNetworkCache

Set this token to yes to activate IP caching.

When caching is enabled, all accepted TCP connections are saved in the kernel table. The records consist of a peer IP address, peer port, and local port. Every new connection is searched in this cache. If a matching set of data for IP address, IP port, and local port is located, the connection is immediately permitted. The time to establish connection is reduced.

Using the Real Path Cache

File name resolution is a long process because CA Access Control uses information from file system. The kernel of CA Access Control translates node numbers to full file names when it intercepts appropriate events. Real path caching saves file names within an internal table.

To enable this feature, set the token cache_enabled to 1 in the [SEOS_syscall] section of the seos.ini file. File names are cached in the table with a data pair: inode number and device number.

Note: For more information about the seos.ini initialization file, see the *Reference Guide*.

Using Fork Synchronization

The fork synchronization token (synchronize_fork) in the [SEOS_syscall] section of the seos.ini file manages fork event behavior when new processes are created. Lowering the value of this token improves performance because fork events are frequent.

Note: For more information about seos.ini initialization file, see the *Reference Guide*.

Using High Priority

CA Access Control contains an option to set a real-time priority for the seosd daemon on some platforms. To activate this feature, set the rt_priority token in the [seosd] section of the seos.ini file to yes. Running in real time improves system performance.

Note: For more information about the seos.ini initialization file, see the *Reference Guide*.

Bypassing the Process File System

To reduce system load, you can specify whether CA Access Control should check file access when the file belongs to a process file system (/proc).

To activate this feature, use the proc_bypass token in the [SEOS_syscall] section of the seos.ini file. The token stores access information to be bypassed whenever CA Access Control must access the process file system.

Note: For more information about seos.ini file tokens, see the *Reference Guide*.

You can set this token as a *sum* of accesses. Access values are as follows:

- **1**-read
- **2**-write
- **4**-chown
- **8**-chmod
- **16**-rename
- **32**-unlink
- **64**-utimes
- **128**-chattr
- **256**-link
- **512**-chdir
- **1024**-create

For example, proc_bypass=513 specifies that all read(1) and chdir (512) access attempts should not be verified (1+512=513).

Bypassing Real Paths

Searching for files with absolute file paths (instead of relative paths) creates heavier system loads; bypassing this search accelerates file events.

To activate this bypass, set the bypass_realpath token to 1 in the [SEOS_syscall] section of the seos.ini file. If you enable this token, CA Access Control does not obtain real file names, which, for example, could be a symbolic link.

Note: For more information about seos.ini file tokens, see the *Reference Guide*.

Important! This feature should be used with extreme care because it impacts security-generic rules do not work when files are accessed with a relative path.

Bypassing Trusted Process Authorization

CA Access Control allows you to define programs as trusted. CA Access Control stores the trusted programs and their children programs in a table. All events (inbound *and* outbound) related to trusted processes (and their corresponding ports) are permitted without authorization as part of a full network bypass.

To specify these programs, use the SPECIALPGM class:

■ To bypass file and network events for the specified program, use the property PGMTYPE with values pbf and pbn.

■ To bypass setuid and setgid events for a specified program, use the property PGMTYPE with the value surrogate.

■ To propagate bypasses to all programs that are called from the specified program, use the property PGMTYPE with the value propagate.

    Note: Security privilege propagation works with PBF, PBN, DCM, and SURROGATE privileges only.

Bypass Ports for Network Activity

To specify that all connection events (inbound *and* outbound) related to specific TCP/IP ports can be established without CA Access Control authorization, you can define a bypass for these ports. Bypassing these ports reduces system load and speeds event processing. Bypassed connection events are not logged in the audit and trace files.

Note: CA Access Control lets you bypass the network connection event only; not any subsequent events that use the network connection (for example, opening a file).

Trusted inbound connections are specified separately from outbound connections:

■ To bypass *incoming* connections, modify the *bypass_TCPIP* configuration setting in the [seosd] section of the seos.ini file.

■ To bypass *outgoing* connections, modify the *bypass_outgoing_TCPIP* configuration setting in the [seosd] section of the seos.ini file.

Note: For more information about the seos.ini initialization file, updating tokens, and affecting changes, see the *Reference Guide*.

Example: Bypass incoming Telnet events

If you set the bypass_TCPIP configuration setting to 23 (the Telnet port), the audit and trace files no longer log the network event when you Telnet *to* that workstation. Events related to other services, such as ssh, login, and FTP, and subsequent events that use the network connection (for example, opening a file), will still be logged.

Example: Bypass outgoing FTP events

If you set the bypass_outgoing_TCPIP configuration setting to 21 (the FTP port), the audit and trace files no longer log the network event when you FTP *from* that workstation. Events related to other services, such as ssh, login, and Telnet, and subsequent events that use the network connection (for example, opening a file), will still be logged.

Reducing Audit and Trace Loads

CA Access Control uses a file system to keep audit data and trace data. Most processes in the system could be blocked while CA Access Control writes to this file system. To reduce access time to the file system, do the following:

- Set the audit mode only for resources and accesses you need.

- Open the trace only when you need to.

- Store audit file, trace file, and CA Access Control database files on the fastest available file system.

- Store the lookaside database directory on a fast file system.

Reducing Database Loads

How you define rules to the database affects system performance:

- Generic rules for commonly used directories produce many verifications, resulting in a greater system load.

  For example, protecting /usr/lib/* causes every action in the system to be checked by CA Access Control. To improve performance, avoid using generic rules for frequently used files.

- Deep hierarchies of users and resources require system loads to obtain and check all dependencies. To improve performance, avoid deep hierarchies in the database.

Improving PMDB Updates

Policy Models send commands to their subscribers one by one in a loop. To control the maximum number of commands that the Policy Models sends to each subscriber during each loop, use the updates_in_chunk token, which is described in the [pmd] section of the appendix "The pmd.ini File."

If you increase the value of this token, the Policy Model uses fewer cycles to send commands. After each loop, the Policy Model checks for new requests. If the token is set higher, the Policy Model does not check for new requests as often.

For example, when you add a new subscriber to the Policy Model (using the sepmd -n option), increase the token value because other subscribers have already received the commands that the Policy Model is sending. The Policy Model spends less time sending commands to the other subscribers and spends more time sending commands to the new subscriber, shortening the time it takes to add the subscriber.

Note: Do not set this token value to more than 100.

Improving Watchdog Performance

To reduce system load, set the Watchdog daemon (seoswd) to periodically scan secured files instead of constantly scanning. You can specify the Watchdog to scan at times when the system is less loaded.

To activate this feature, use the IgnoreScanInterval token in the [seoswd] section of the seos.ini file, and set additional tokens for intervals and start times.

Note: For more information about these tokens, see the seos.ini initialization file in the *Reference Guide*.

Improving Class Parameters

Use the class activation and class authorization features for CA Access Control to improve performance further.

Class Activation

CA Access Control stores information about whether a CLASS is active or inactive in the database. When CA Access Control starts, it passes a list of active classes to SEOS_syscall, so CA Access Control does not have to constantly intercept these classes. The only time CA Access Control intercepts a class is when a user changes the activity status of a class. If a class is inactive, access to the resource is not intercepted.

You can use the inactive class bypass with the following classes: FILE, HOST, TCP, CONNECT, and PROCESS.

Class Authorization

The resource class SEOS controls the behavior of the CA Access Control authorization system. The SEOS class has modifiable properties that specify whether a class is active. You can disable unused classes (using the setoptions command) to reduce authorization time.

Resolving Names

Several tokens in the [seosd] section of the seos.ini file (including GroupidResolution, HostResolution, ServiceResolution, and UseridResolution) control how CA Access Control performs name resolution. Setting these tokens appropriately improves performance.

Alternatively, you can create a lookaside database (instead of using system name resolution). To improve performance, select the lookaside database option. Tokens for this feature include the lookaside_path and use_lookaside.

Note: For more information about these tokens, see the seos.ini initialization file in the *Reference Guide*.

Whenever CA Access Control must perform UID to username, GID to groupname, ipaddr to host name, and port to service translations, it may impact CA Access Control performance. How CA Access Control performs these translations depends on the value of certain tokens in the seos.ini file-in particular, the under_NIS_server, use_lookaside, GroupidResolution, HostResolution, ServiceResolution, UseridResolution, and resolve_timeout tokens.

When native operating system mechanisms perform the resolution, the impact on system performance is relatively small. When translating ipaddr to host name, an external mechanism such as DNS must perform the translation. This may result in significant degradation of system performance. This degradation occurs because, while seosd is waiting to receive the host name, all other processes that CA Access Control has intercepted must also wait until seosd completes its processing.

- If you set the value of the under_NIS_server token to no, seosd allows UNIX to translate UID, GID, IP addresses, and port numbers by taking data from the following sources:

Type of Station	Source
Stand-alone	Seosd uses the following files for translations; ■ /etc/passwd for UID to user name ■ /etc/group for GID to group name ■ /etc/hosts for IP address to host name

Type of Station	Source
	■ /etc/services for service ports to service names
	The source of the information varies, depending on the operating system and its version number. The information is usually taken from /etc files and the NIS server. However, in some systems, the /etc files are not the source and the order in which translation is made is changed during system configuration. For instance, in the Solaris 2.x system the file /etc/nsswitch.conf determines the translation order.
DNS client	Translation for users, groups, and services is performed using /etc files. Host names are translated by calls to the DNS server and, on some systems, the /etc/hosts file is also read.
NIS and DNS clients	The ipaddr to host name translation is performed by DNS. For user, group, and service translations, the translations are performed in the same way as NIS client translations.

■   If you set the value of the under_NIS_server token to yes, seosd performs its own translations. If seosd caches data for its translations, the sources of its data are as follows:

Type of Station	Source
NIS server	The server machine usually behaves as both server and client, and consults the NIS server daemon for any type of translation. The files which contain the sources of the NIS resolution maps are usually located in /var/yp, but the location may vary, depending on the site configuration, and the type and version of the operating system.
DNS server	The source of the information used for translation depends on the configuration of the site. DNS does not have an option to scan its resolution database; therefore, CA Access Control cannot use caching, and must use a lookaside database. You must configure the lookaside database so that the utility sebuildla uses a host list file. For more information, see sebuildla in this chapter.
all others	Same as DNS server.

In versions 2 and higher of CA Access Control, seosd can also use the tokens GroupidResolution, HostResolution, ServiceResolution, UseridResolution, and resolve_timeout to control the translation process. For more information on these tokens, see the *Reference Guide*.

Chapter 16: Using UNIX Exits

This section contains the following topics:

UNIX Exits

A UNIX exit is a specified program-a shell script or an executable-that runs automatically as a result of another defined CA Access Control activity taking place. CA Access Control supports UNIX exits when loading or unloading the CA Access Control kernel module, or when issuing specific selang commands. For example, you can run an initialization process for each new user that you add.

A UNIX exit can run on one or more of the following occasions:

- As a pre-update exit, *before* each selang command that updates a *user* or *group* record

- As a post-update exit, *after* each selang command that updates a *user* or *group* record

- As a pre-load exit, *before SEOS_load* loads the CA Access Control kernel

- As a post-load exit, *after SEOS_load* loads the CA Access Control kernel

- As a pre-unload exit, *before SEOS_load -u* unloads the CA Access Control kernel

- As a post-unload exit, *after SEOS_load -u* unloads the CA Access Control kernel

User or Group Record Update Exits

UNIX exits are called whenever a selang command that updates user or group records is executed in the UNIX environment, regardless of whether the tool is a command-line interface (selang) or a GUI (such as CA Access Control Endpoint Management).

The term *update* refers to creating, modifying, or deleting a user or group record. Querying a user or a group does not cause any UNIX exit to run. These are the commands that can cause a UNIX exit to run:

- newusr

- newgrp

- chusr

- chgrp

- editusr

- editgrp

- rmusr

- rmgrp

From the UNIX point of view, each exit processes runs as a root process, but from the CA Access Control point of view, it runs under the agent identity _seagent.

How the Provided selang Exit Script Works

CA Access Control provides a script that you can use as a master script to call other programs according to the nature and status of the current selang command. The exit script that is supplied as part of CA Access Control is *ACInstallDir*/exits/lang_exit.sh (where *ACInstallDir* is the CA Access Control installation directory.) Here is how it works:

1. CA Access Control automatically gives values to three parameters of the script.

Parameter	Possible Values
CLASS	USER	GROUP
ACTION	CREATE	MODIFY	DELETE
STAGE	PRE	POST

The parameters indicate whether CA Access Control is dealing with a user or a group; whether the user or group is being created, deleted, or modified; and whether the selang command is about to be executed (PRE) or has just been executed (POST).

The script can pass the parameter values to programs that it calls.

Parameter	Possible Values
EXEC_RV	Receives the return value of a UNIX command that you use to determine whether the exit command succeeded or failed.
	For PRE commands, the value is always zero. For POST commands, you can use the value to decide whether to run or skip an exit.
	For an example of how to use this parameter, locate *ACInstallDir*/samples/exits_src

2. Using the CLASS and STAGE parameters, CA Access Control looks for programs in the appropriate directory:

ACInstallDir/exits/USER_PRE/
ACInstallDir/exits/USER_POST/
ACInstallDir/exits/GROUP_PRE/
ACInstallDir/exits/GROUP_POST/

3. In the appropriate directory, CA Access Control selects all the programs that have file names that begin with a capital S, refer to the appropriate action, and have the following format:

Snnaction_string

Where *nn* is a two-digit decimal number defining the order of the program in the execution sequence, *action* is one of CREATE, MODIFY, or DELETE, and *string* is a descriptive string.

4. CA Access Control runs all the appropriate programs according to the numerical order of the second and third characters of their names.

Example: UNIX Exit Script

You are going to delete a user, and the directory *ACInstallDir*/exits/USER_PRE/ includes the following files:

- S10CREATE_precustom.sh

- S10DELETE_precustom.sh

- S99DELETE_prermusrdir.sh

When you issue the command to delete the user, the first program is not run because you are deleting and not creating a user. The second and then the third programs are run in that order based on the two digits after the initial S.

Arguments You Can Pass to selang Exits

When writing exits you can take advantage of the three parameters mentioned previously (CLASS, ACTION, and STAGE), and all the standard CA Access Control data such as names and permissions. You can also designate extra user or group data especially for use by the exit scripts. To store such additional data for a user or group, define it within single quotes as the value of the user's or group's UNIX APPL property in a newusr, chusr, newgrp, or chgrp command. For example:

```
chusr JONESY unix APPL('HIRED=MAY93,CLEARANCE=2')
```

Your exit program must be able to handle whatever is between the single quotes.

Specify selang Exit Programs to Run

To tell CA Access Control which exit programs to run, modify the [lang] section of the seos.ini file. CA Access Control provides the lang_exit.sh script for pre-user, post-user, pre-group, and post-group exits. You can also specify no exit or create your own exit.

To specify your own selang exits set any or all of the settings in the [lang] section of seos.ini as required.

Note: An exit is called only if its full pathname appears as the value of an exit token.

Example: Specify selang Exits

In the following example, the seos.ini file tokens are set so that the program groupcheck runs before group operations, the program flag_exceptions runs after group operations, the program lang_exit.sh runs after user operations, and no exit program runs before user operations. The seos.ini file tokens are set as follows:

```
[lang]
pre_group_exit = /opt/CA/AccessControl/exits/groupcheck
post_group_exit = /opt/CA/AccessControl/exits/flag_exceptions
post_user_exit = /opt/CA/AccessControl/exits/lang_exit.sh
```

Time Out and Other Failures

Exit execution times out after 15 seconds, unless the exit_timeout variable in the seos.ini file specifies otherwise. A nonzero return value indicates failure.

- If a *pre*-update exit times out or returns a return code of greater than or equal to 16, then CA Access Control kills the exit process, displays an error message, and aborts execution of the CA Access Control update command. Any other positive return code does not abort the execution of the command.

- If a *post*-update exit times out or returns a nonzero value, then CA Access Control kills the exit process and displays an error message. Having already been executed, the CA Access Control update command remains in force.

selang Exit Samples

By examining the scripts in the following directories, you can familiarize yourself with recommended scriptwriting techniques.

ACInstallDir/samples/exits-src

ACInstallDir/samples/sample_exits

CA Access Control Kernel Loader Exits

UNIX exits are called whenever the CA Access Control kernel is being loaded or unloaded (SEOS_load). This lets you define how you want to handle operating system and third-party programs when loading or unloading the CA Access Control kernel. For example, you can use kernel-unloading UNIX exits to automatically stop, and later restart, processes that prevent CA Access Control from unloading when running *SEOS_load -u*.

For some operating systems, CA Access Control comes with some kernel load exits, kernel unload exits, or both out of the box.

Note: For more information about identifying processes that prevent CA Access Control kernel from unloading, see the secons utility in the *Reference Guide*.

How the Kernel Loading Exits Work

To let you control operating system and third-party processes, CA Access Control lets you automatically make calls to UNIX exits when loading the CA Access Control kernel extension.

When you run **SEOS_load**, CA Access Control performs the following actions:

1. Looks for programs in the following directory:

   ACInstallDir/exits/LOAD

2. Selects all the programs that have file names of the following format:

   SEOS_load_*string*.always

   Where *string* can be any descriptive strings.

3. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

   SEOS_load_*string*.always -pre

   Each file is executed with the *-pre* parameter so that you can write your exits to detect the parameter and perform the actions required before the kernel is loaded.

   Note: If the exit returns a nonzero value, CA Access Control kills the exit process, displays an error message, and aborts the kernel loading.

4. Loads the kernel (SEOS_syscall).

5. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

   SEOS_load_*string*.always -post

   Each file is executed with the *-post* parameter so that you can write your exits to detect the parameter and perform the actions required after the kernel is loaded.

   Note: If the exit returns a nonzero value, CA Access Control kills the exit process and displays an error message. Having already been loaded, the CA Access Control kernel remains loaded.

How the Kernel Unloading Exits Work

To let you control operating system and third-party processes, CA Access Control lets you automatically make calls to UNIX exits when unloading the CA Access Control kernel extension.

When you run **SEOS_load -u**, CA Access Control performs the following actions:

1. Looks for programs in the following directory:

   ACInstallDir/exits/LOAD

2. Selects all the programs that have file names of the following format:

   SEOS_unload_*string*.always

   Where *string* can be any descriptive strings.

3. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

   SEOS_load_*string*.always -pre

   Each file is executed with the *-pre* parameter so that you can write your exits to detect the parameter and perform the actions required before the kernel is unloaded.

   Note: If the exit returns a nonzero value, CA Access Control kills the exit process, displays an error message, and aborts the kernel unloading.

4. Tries to unload the kernel.

   If the kernel *does not* unload:

   a. Selects all the programs that have file names of the following format:

      SEOS_unload_*string*.opt

   b. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

      SEOS_unload_*string*.opt -pre

      Each file is executed with the *-pre* parameter so that you can write your conditional exits to detect the parameter and perform the additional optional actions required before the kernel is unloaded.

      Note: If the exit returns a nonzero value, CA Access Control kills the exit process, displays an error message, and aborts the kernel unloading.

c. Unloads the kernel.

d. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

SEOS_unload_*string*.opt -post

Each file is executed with the *-post* parameter so that you can write your conditional exits to detect the parameter and perform the additional optional actions required before the kernel is unloaded.

Note: If the exit returns a nonzero value, CA Access Control kills the exit process and displays an error message. Having already been unloaded, the CA Access Control kernel remains unloaded.

5. Executes, in lexicographical order, each file it found in the directory *ACInstallDir*/exits/LOAD:

SEOS_unload_*string*.always -post

Each file is executed with the *-post* parameter so that you can write your exits to detect the parameter and perform the actions required after the kernel is loaded.

Note: If the exit returns a nonzero value, CA Access Control kills the exit process and displays an error message. Having already been unloaded, the CA Access Control kernel remains not loaded.

Chapter 17: Interacting with LDAP

This section contains the following topics:

Transferring User Names

If you are using both CA Access Control and LDAP, you can transfer user names between them using scripts of your own design; three sample scripts are provided.

Important! To set up sebuildla and the required LDAP configuration settings you must to be familiar with LDAP and be able to execute the ldapsearch command. We recommend that you read the man pages for ldap(1), ldapsearch(1) and the information about setting up in the documentation for your LDAP client.

Two of the provided scripts-ldap2seos and seos2ldap-export whole sets of users from CA Access Control to an LDAP server and imports them from an LDAP server to CA Access Control.

A third sample script, S50CREATE_Ldap_u.sh, automatically transfers new UNIX user names from CA Access Control to LDAP as they are created.

The sample scripts require access to a TCL shell environment; they use the Language Client API (LCA) library extension, tcllca.so.

Note: For more information about LCA and the TCL extension, see the Language Client API and the appendix the LCA Extension respectively in the *SDK Guide*.

If you do not have TCL, consult the FAQ posted monthly to comp.lang.t_c_l by Larry Virden, which is available on the MIT web site and the Terafirm website.

You can also refer to the Sun web site for TCL news, documentation, and resources.

S50CREATE_Ldap_u

S50CREATE_Ldap_u.sh uploads new UNIX users to LDAP as they are created.

CA Access Control supplies a sample shell script to import new UNIX users automatically to an LDAP server. The script you need can vary from the sample.

To employ the sample shell script, assuming that you are already using the provided exit script, do the following:

1. Copy the S50CREATE_Ldap_u.sh file to the directory *ACInstallDir*/exits/USER_POST. In this directory, the script becomes a post-user exit.

2. In the seos.ini file in the [ldap], set the base_entry token to the LDAP base entry.

   For example, for an organization named ServerWorld, located in Canada, the base entry might be: o=ServerWorld, c=CA.

3. In the same section, set the host name to the host name of the LDAP server. Set the path to the LDAP base directory. (The sample script looks for the line command utilities in the bin directory under that directory.)

Common Names (cn) are derived from the user's full name. If the CA Access Control database contains, for example, only the user name and surname, these will comprise the Common Name. You are essentially locked into the Common Name, so we recommend that you do not base it on a user name.

Each user subsequently added to UNIX with selang is automatically uploaded to the LDAP server. If the user already exists in LDAP, an error message results.

When you add users with this script, the relevant LDAP replies and warnings, if any exist, are collected in the /tmp/add_User2Ldap.tcl.log file. You can examine this file, using vi or any other standard UNIX editor, to check for errors. The file is overwritten with the new set of replies and warnings each time you add new users.

Chapter 18: Unicenter Security Migration and Integration

This section contains the following topics:

Unicenter Security Integration Tools

CA Access Control is fully integrated into the Unicenter Enterprise Management environment.

Important! To have Unicenter TNG integration with CA Access Control, you must have Unicenter TNG installed on the same machine as CA Access Control.

Note: For complete installation instructions, see the *Implementation Guide*.

Unicenter Security Integration Features

The following sections describe how CA Access Control integrates with Unicenter TNG.

SSF/EMSec API Support

The security APIs on UNIX all channel into a message queue. A new utility now processes the API requests sent through the message queue and routes these reformatted and rerouted requests to CA Access Control. The utility then translates the return codes of CA Access Control to Unicenter TNG equivalents. This approach protects the integrity of existing applications that are currently using the EMSec API.

In UNIX the utility is called *sessfgate*. The gateway is active after the Unicenter Integration setup procedure completes. In fact, if Unicenter NSM Event Notification Facility (ENF) is running, the gateway is automatically started or stopped whenever the CA Access Control services are started (using seload) or stopped. If ENF is not running, the CA Access Control seload program will not start the sessfgate daemon. The Unicenter Integration setup installs the sessfgate program in the *ACInstallDir*/tng/bin directory. After Unicenter Security is shut down and CA Access Control is started, sessfgate can accept API requests instead of SSF.

Run sessfgate as follows:

sessfgate [-i|-s|-l] -t

-I

Specifies to start the gateway.

-s

Specifies to stop the gateway.

-l

Specifies the status.

-t

Toggles the tracing file (log file = *ACInstallDir*/log/sessftrace.log).

Note: If you run seload before running Unicenter TNG, you must start sessfagte manually with the following command:

ACInstallDir/tng/bin/sessfagte -l

where *ACInstallDir* is the directory in which you installed CA Access Control.

CA Access Control to Unicenter Security Synchronization Utility

Unicenter Security and CA Access Control manage the administration of your enterprise IT environment before total migration occurs.

To reduce the complexity of using different product tools to perform administrative tasks, CA Access Control provides a synchronization daemon. The daemon is called seostngd. CA Access Control sends Policy Model database (PMDB) updates through CA Common Communication Interface (CAICCI) to seostngd. The daemon listens for updates on CAICCI and then translates the messages into equivalent cautil commands to update the Unicenter Security database with this global data.

Current Unicenter TNG processing can still update other Unicenter TNG client installations. You must run seostngd on the same machine as the Unicenter Security database (normally referred to as a Unicenter TNG master machine.) CA Access Control should also be running on the same machine.

Note: The daemon should only be used during the migration process. Once the migration process has been completed, the daemon should not be used. This daemon is not a long-term solution for your IT environment.

The major task of seostngd is to take changes you make on CA Access Control (using either a graphical interface or selang commands) and apply them to the Unicenter Security database. If the changes can be applied to the Unicenter Security database successfully, you should see the same behaviors as you did on CA Access Control.

For example, when you create a USER object with CA Access Control, you should see the same USER object being created in Unicenter TNG (as long as the required fields are present).

Starting and Stopping the Subscription Daemon

To start the subscriber daemon, enter:

seostngd

To stop the daemon, enter one of the following:

seostngd -shut

or

seostngd -stop

Important! Do not use selang -c during migration if you are listing more than one command. Instead, use selang -f input_file_name.

Notes:

- In order to start the daemon, you must be user root in the Unicenter TNG SSF_AUTH user list.

- The subscriber daemon should be manipulated manually. For example, if you reboot the machine, restart this utility with the seostngd command because this is not controlled by the CA Access Control startup program.

- You cannot use this before performing the Unicenter TNG integration installation procedure. During the setup procedure, a configuration file, *ACInstallDir*/data/tng/assettypes.txt, is generated. This file is required to run this utility.

- Be sure that $CAIGLBL0000/bin is in the PATH environment, so you can run the Unicenter command line utility, cautil. To do this, run the script file (from ksh or sh #) $CAIGLBL0000/scripts/envset.

- The Unicenter Security daemon (sdm) must be running, otherwise, the Unicenter TNG subscriber daemon cannot apply changes to the Unicenter Security database.

SEOSTNGD Limitations

Different maximum field lengths between CA Access Control and Unicenter Security can cause truncation of data values. The following table lists the significant differences in supported field lengths.

	Unicenter TNG	CA Access Control
User ID	20 characters	256 characters
Password	8 characters	14 characters
User group ID	8 characters	254 characters
Asset group ID	8 characters	255 characters

- Unicenter TNG does not support renaming a user or asset object, so the seostngd daemon ignores CA Access Control *rename* commands for users and assets.

- Unicenter TNG user groups and asset groups must have at least one member. If no member is specified in the CA Access Control command for creating user groups or asset groups, the corresponding Unicenter TNG user group or asset group is not created until at least one member is added.

- If the last member is removed from a CA Access Control User group or asset group, that user group or asset group is removed from Unicenter TNG.

- Unicenter TNG assets require at least one defined accessor, so a new Unicenter TNG asset cannot be created until at least one CA Access Control "authorize" command is executed for the asset.

- CA Access Control removes any associated rules for an object when it is deleted. However, Unicenter Security does not.

- CA Access Control users have an ADMIN attribute that has a similar meaning as when a Unicenter TNG user is a member of the SSF_AUTH user list in the Security Options. However, Unicenter TNG does not provide any automatic way for manipulating remote Security Options, so manual modifications to SSF_AUTH user list are required.

- In order to force the creation of a new Unicenter TNG user, the new CA Access Control user must be created in the Native environment with a value supplied for the CA Access Control password field. Unicenter TNG has default password restrictions that require a minimum length of six characters-two alphabetic and one numeric.

- The CA Access Control *authorize* command supports the asterisk (*) as an accessor ID, but this is not supported in Unicenter TNG. The seostngd daemon ignores CA Access Control *authorize* commands like this.

- The CA Access Control *authorize* command supports conditional access rules using the *via* parameter, but this is not supported in Unicenter TNG. The seostngd daemon ignores CA Access Control *authorize* commands like this.

- If the *access* parameter is not specified on a CA Access Control *authorize* command, the seostngd daemon grants READ permission for any UNIX-FILE or Unicenter TNG asset group, and grants all permissions for any other Unicenter TNG predefined asset type.

Unicenter Security Data Migration Features

The following sections describe how to migrate Unicenter Security data to CA Access Control.

Unicenter Security Options Migration

CA Access Control features a program called migopts that extracts selected Unicenter Security options and customizes the targeted CA Access Control database according to these options. To activate this feature, you must run the Unicenter Integration with Unicenter Security Data Migration setup procedure. This setup procedure automatically runs migopts.

Important! This feature is part of Unicenter Security Data Migration and is intended for users who use Unicenter Security only for its integration with the cautil command processor, Event Management, and Workload Management. Because of differences between Unicenter Security and CA Access Control architecture, Unicenter Security data migration is not intended for people who use Unicenter Security to protect their file systems.

Note: The following Unicenter Security options can be migrated *completely* into the CA Access Control environment.

- CREDAUTHEXIT

- DEFSESID

- PASSWORD_ALPHA

- PSWDVALEXIT

- PWDQUEUESIZE

- SSF_MAXPWDVIO

- SSF_MINPWDLEN

- SSF_NUMSUBP

- SSF_SECPWEXCL

- SSO_APPLNAME

- USER_PWDCHANGE

- USER_PWDCHGMAXDAYS

- USER_PWDCHGMINDAYS

Additionally, exporttngdb migrates Unicenter TNG users who are members of the **SSF_AUTH** Unicenter Security Option into the CA Access Control environment by setting the Users' "admin" attribute before adding them to CA Access Control.

Note: The migopts utility is run by the migration scripts *ACInstallDir*/tng/bin/uni_migrate_master.sh and *ACInstallDir*/tng/bin/uni_migrate_node.sh. For more information, see the *Reference Guide*.

Unicenter Security Database Migration

CA Access Control features a program called exporttngdb that extracts data from the Unicenter Security database and translates it into CA Access Control commands to populate the CA Access Control database. exporttngdb migrates the following:

- Unicenter Security Users

- Unicenter Security User Groups

- Unicenter Security Rules

Notes:

1. We do not recommend running Unicenter TNG login intercepts after running the Unicenter Integration and Migration Installation. Once the Unicenter Integration and Migration Installation is successfully completed, you should verify that Unicenter TNG login intercepts are disabled.

2. Unicenter TNG Data Scoping and Keyword Scoping rules (rules that target Unicenter TNG asset types with a -DT or -KW suffix) are not supported by the CA Access Control Migration process. Rules of this type are ignored during the migration process.

3. Unicenter Security rules that have been implemented against any of the following Unicenter Security asset types are obsolete because Unicenter Security is no longer used: CA-USER, CA-ACCESS, CA-USERGROUP, CA-ASSETGROUP, CA-ASSETTYPE, and CA-UPSNODE. Rules that target any of these asset types, or any of their derivatives, are ignored during the migration process.

4. The exporttngdb utility is run by the migration scripts *ACInstallDir*/tng/bin/uni_migrate_master.sh and *ACInstallDir*/tng/bin/uni_migrate_node.sh.

   Note: For more information about the exporttngdb utility, see the *Utilities Guide*.

To activate exporttngdb, you must run the Unicenter Integration with Unicenter Security Data Migration setup procedure. This setup procedure automatically performs the Unicenter Security Data Migration process.

Important! This feature is part of Unicenter Security Data Migration and is intended for users who use Unicenter Security only for its integration with the cautil command processor, Event Management, and Workload Management. Due to the dramatic differences between Unicenter Security and CA Access Control architecture, Unicenter Security Data Migration is not intended for people who use Unicenter Security to protect their file systems.

Note: Creation and modification statistics of all Unicenter TNG objects are lost in the migration process.

Due to Unicenter TNG and CA Access Control product differences, the following attributes of Unicenter Security **users** cannot be migrated to CA Access Control:

Statistics

The following User statistics are not supported by CA Access Control:

■ Last login statistics (date and time, node of last login)

■ Password change statistics (date and time, node, user who changed last password, and expiration date of the password)

■ Password violation statistics (date and time, node of last unsuccessful login, and number of unsuccessful logins since last successful login)

■ Access violation statistics (date and time, node of last access violation, and number of access violations)

■ Suspension statistics (date and time of suspension)

PWDCHANGE VALUE (RANDOM)

Random password generation

UPSSTATGROUP

UPS station group. It is not supported by CA Access Control.

VIOLMODE

Violation mode (FAIL, MONITOR, WARN, QUIET). CA Access Control supports FAIL mode only.

VIOLACTION

Violation action (CANUSER, CANU&LOG, CANU&LOG&SUS). CA Access Control supports CANUSER action only.

Due to Unicenter TNG and CA Access Control product differences, the following attributes of Unicenter Security **Rules** cannot be migrated to CA Access Control:

EXPIRES

Rule expiration date is not supported by CA Access Control.

Unicenter TNG User Exit Support

To help current Unicenter TNG users who are migrating to CA Access Control, CA Access Control lets you run existing Unicenter Security user exits unchanged in the CA Access Control environment. You do not have to rewrite all user exits as part of the migration.

Using only the existing user exit interfaces in Unicenter Security and CA Access Control, each installed component is registered as a standard CA Access Control user exit, which then brings up the corresponding Unicenter Security exit.

In order to initiate this feature, you must run the Unicenter Integration with Unicenter Security Data Migration setup procedure. Once the setup procedure is completed, this functionality is active.

Important! This feature is part of Unicenter Security Data Migration and is intended for users who use Unicenter Security only for its integration with the cautil command processor, Event Management, and Workload Management. Due to the dramatic differences between Unicenter Security and CA Access Control architecture, Unicenter Security Data Migration is not intended for people who use Unicenter Security to protect their file systems.

Note: Because Unicenter TNG and CA Access Control use different architectures, only the exit points and data items that are comparable between Unicenter Security and CA Access Control are supported. The following Unicenter Security exit points are supported:

EmSec_CredExit()

The input to the Unicenter Credential Authentication exit, EmSec_CredExit(), is mapped by EMSECSIGNON. With CA Access Control, only the user and node members within this structure have meaningful data. The user member is set to the user name being authenticated, and the node member is set to the current local node name. All other members of the EMSECSIGNON structure are set to binary zeros. The other parameters, the detailed return code, and the message passed back from the Unicenter Resource Check Exit are ignored.

EmSec_PwExit()

The Password Validation exit, EmSec_PwExit(), is fully supported.

Note: The exits are contained in libemsec2.xx which is located at: /usr/local/Calib/ or ACInstallDir/Calib/ (where ACInstallDir is the installation directory for CA Access Control, by default /opt/CA/AccessControl).

Once the Unicenter Integration setup completes, this functionality is active.

Use a PMDB to Secure Unicenter Security Objects

CA Access Control PMDBs can be used with Unicenter TNG objects to create rules that secure Unicenter TNG objects from being manipulated by the various Unicenter TNG components (such as the cautil command processor, Event Management, and Workload Management). You must perform the integration manually.

To use a PMDB for Unicenter TNG objects:

1. Create the PMDB.

2. Migrate Unicenter Security options into the PMDB with the following command:

   migopts -d *pmdb-name*

   where *pmdb-name* is the name of your PMDB.

   Important! This step is required only if you used Unicenter Security and ran the Unicenter Integration installation script for Security Data Migration (uni_migrate_master.sh and uni_migrate_node.sh). If you did not use Unicenter Security, then you never established any security options and there is nothing to migrate into your PMDB.

   Note: Migration and integration are two separate procedures.

3. Create classes for any user-defined Unicenter TNG asset types with the following command:

   defclass.sh.*pmdb-name*

   where *pmdb-name* is the name of your PMDB

   Important! This step is only required if you used Unicenter Security and created user-defined asset types. Unicenter TNG asset types are automatically defined in every new PMDB if you selected Unicenter Integration during the CA Access Control installation.

Unicenter TNG Calendar

Unicenter TNG provides a calendar facility, with which you can set time restrictions for users, groups, and resources. The calendar contains time intervals of 15 minutes that you can set to ON or OFF. A calendar time interval set to OFF prevents access to resource; a calendar time interval set to ON continues resource authorization.

In UNIX, an administrator can set calendar usage before security startup only.

To use Unicenter TNG calendars in CA Access Control, complete the following steps:

Note: Unicenter TNG must be installed on the local machine. CA Access Control uses local Unicenter TNG services to retrieve calendar settings.

1. Stop CA Access Control security. Enter:

   secons -s

2. Set the TNG_calendars token in the [seauxd] section of seos.ini to yes.

   TNG_calendars=yes

3. Start CA Access Control security. Enter:

   seosd

4. Check that the auxiliary daemon seauxd is running. Enter:

   issec

You can also modify the following tokens in the seos.ini file:

TNG_refresh_interval

Specifies the time interval in minutes to refresh CA Access Control calendars. The default is 10 minutes.

TNG_lib_path

Specifies the full path to Unicenter TNG libraries. The default is /usr/local/Calib.

TNG_cal_lib

Specifies the name of the Unicenter TNG calendar library. The default is libcalendar.

To link a CA Access Control resource with the calendar, you must issue the following database commands:

Note: The issued calendar name must be identical to the case-sensitive Unicenter TNG calendar name.

nr CALENDAR calendar_name
nr file /tmp/test calendar (calendar_name) defaccess (a)

The Unicenter TNG calendar Access Control List (ACL) is an additional security constraint feature. The regular Unicenter TNG calendar property restricts the current resource according to the appropriate Unicenter TNG calendar status. The Unicenter TNG calendar ACL property restricts access for (or gives access to) specific users and groups for the current resource according to the Unicenter TNG calendar status.

Two types of ACL Unicenter TNG calendar properties are regular and restrictive:

- The regular calendar ACL property permits user or group access to the resource accordingly to ACL access.

- The restrictive (denied) calendar ACL property denies user or group access to the resource accordingly to ACL.

To add a user or group to the regular calendar ACL (CALACL), enter the following command in selang:

```
auth resource_class_name object_name \
uid_or_gid_name calendar(calendar_name) access(access_value)
```

For example:

```
auth file file1 uid(george) calendar(basecalendar) access(rw)
```

To add a user or group to the denied calendar ACL, enter the following command in selang:

```
auth resource_class_name object_name uid_or_gid_name \
calendar(Unicenter_calendar_name) deniedaccess(access_value)
```

For example:

```
auth file file2 uid(george) calendar(holidays) deniedaccess(rw)
```

You can use both regular and restrictive properties for the same resource (such as calendar and uid). The following command adds a user named George with read access to the denied calendar ACL for file1.

```
auth file file1 uid(george) calendar(holidays) deniedaccess(r)
```

To remove a user or group from a Unicenter TNG calendar ACL property, use auth- :

```
auth- file file2 uid(george) calendar(holidays)
```

Use the Show Resource (sr) command to see all Unicenter TNG calendar ACLs assigned to a specific resource:

```
sr file file1
```

Certification with Unicenter TNG and Unicenter NSM

The following features comply with Unicenter TNG 2.2 SP1, Unicenter TNG 2.4, and Unicenter NSM 3.0:

- Sending "events"

- Synchronizing mainframe passwords

- Using the Unicenter TNG calendar

Audit Events Integration

Integration with Unicenter TNG is set up at installation.

You can choose to send audit data to Unicenter TNG. Audit events that are passed to Unicenter TNG appear in the Console logs in the Unicenter Enterprise Management\Enterprise Managers\Windows NT\Event window.

Audit Event	Display Color	Severity
Success	Blue	S
Denied	Orange	F
Fail	Orange	F
Warning	Blue	W
CA Access Control stopped (audit down)	Blue	I
CA Access Control started (audit start)	Blue	I

The second option permits launching CA Access Control from the Unicenter WorldView menu by pointing to the icon representing the TCP/IP Network in the Managed Objects window and selecting CA Access Control from the right-click menu.

CA Access Control also sends following information about events:

- Product name (CA Access Control + version number)

- User name

- Terminal name

- Class name

- Resource name

- Process name

- Event's time

- Full audit message in the format of CA Access Control auditing

The fields User name, Terminal name, Class name, Resource name, and Process name are not always sent, depending on event type.

Chapter 19: Configuring Settings

CA Access Control lets you manage CA Access Control endpoint configuration settings remotely. To do this you can use CA Access Control Endpoint Management or the selang config environment.

This section contains the following topics:

Configuration Settings

CA Access Control stores endpoint and Policy Model configuration settings it uses in:

- The Windows registry on Windows computers

- Initialization files (.ini) on UNIX computers

Note: For information about the configuration settings you can make and what they mean, see the *Reference Guide*.

Change Configuration Settings

To affect how CA Access Control and any Policy Models work, you need to make changes to the configuration settings.

To change configuration settings

1. In CA Access Control Endpoint Management, do as follows:

   a. Click Configuration.

   b. Click Remote Configuration.

   The Remote Configuration page appears.

2. In the Remote Configuration Sections pane on the left, expand the configuration tree as required to reveal the section that contains the configuration setting you want modify, then click that section.

   The Section: *sectionName* System Tokens page appears, displaying all the configuration settings in it.

3. Locate and edit the configuration settings as required, then click Save Tokens.

   The changed configuration setting is saved.

Change Audit Configuration Settings

To affect how CA Access Control generates and stores audit records, you need to make changes to the settings in the audit configuration files. You use selang commands to change the settings in the audit configuration files.

To change audit configuration settings

1. (Optional) If you are using selang to connect to a remote host, connect to the host using the following command:

   host *host_name*

2. Move to the config environment using the following command:

   env config

3. Use the editres config command to modify the configuration settings as required.

   The audit configuration settings are changed.

Example: Modify Audit Configuration File

The following example adds a line to the audit configuration file:

```
er CONFIG audit.cfg line+("FILE;*;Administrator;*;R;P")
```

Appendix A: NIS Configuration

This section contains the following topics:

Installation Notes

Note: This section supplements material covered by the installation script. This appendix assumes you are familiar with Network Information Systems (NIS), Domain Name Services (DNS), and UNIX name resolution concepts.

During installations of CA Access Control, you can use one of two options to resolve user ID to user name, group ID to group name, host IP address to host name, and service port to service name:

■ Use the system functions, which define a bypass for the net cashing daemon on your system.

– If you use Digital DEC UNIX and it is *not* an NIS server, the default uses the system functions for name resolution.

– If you use Digital DEC UNIX and it *is* an NIS server, the installation prompts you to choose one of two options: use a lookaside database or use system functions, which define a bypass for the net caching daemon.

■ Use a lookaside database, which is created by the sebuildla utility.

– If you are using CA Access Control configured to run on an *NIS server*, use the lookaside database.

– The installation default uses the lookaside database on the following platforms: HP-UX 11.0 and higher, Sun Solaris 2.6 and higher, IBM AIX 5.1L and higher, and all supported Linux platforms.

Note: On IBM AIX platforms, you must use the lookaside database; there is no option to use the system functions.

Name Resolution

CA Access Control intercepts requests to access system resources and decides whether to permit or deny these requests. The decision is based on access rules and policies that are defined in the database. The interception of requests to access system resources takes place at the kernel level.

To control hosts, groups, users, and services, the kernel and the relevant system calls use codes or numbers (that is, IP addresses, group IDs, user IDs, and service numbers) instead of names. CA Access Control defines access rules based on names. CA Access Control translates names into codes recognizable by the kernel. This process is called name resolution.

On stand-alone stations, except for stations running Sun Solaris 2.5 or higher, name resolution is completed directly through the local user, group, and host files (/etc/passwd, /etc/group, and /etc/hosts). When CA Access Control needs to resolve a name, it simply calls a system function that in turn reads the relevant file.

On larger networks, however, this information is seldom stored locally. When you use NIS, DNS, or both, there are no local files that you can consult during name resolution. The information is requested and received from a server over the network.

Name Resolution on an NIS/DNS Client

CA Access Control performs name resolution on a client-only NIS or DNS station (which is not its own server) as follows:

1. CA Access Control generates a network request to connect to the relevant server.

2. The CA Access Control kernel extension intercepts the request.

3. The CA Access Control kernel extension permits the request because it knows that the request was made internally by the CA Access Control process.

4. A connection to the NIS or the DNS server is established and the information necessary for name resolution is retrieved.

5. Once the name is resolved, CA Access Control continues the process of deciding whether to permit or deny the original access request.

A standard CA Access Control configuration is sufficient for CA Access Control to easily handle name resolution on a client server.

Name Resolution on a Server: Deadlock

CA Access Control performs name resolution on a server that includes itself as a client as follows:

1. CA Access Control generates a network request to connect to the relevant server.

2. The kernel extension intercepts this request.

3. The kernel extension permits the request because it knows that the request was made internally by the CA Access Control process.

4. The NIS or DNS server (which is located on the same station) generates a request to accept the network connection.

5. The kernel extension intercepts this request.

6. The kernel extension knows that a CA Access Control process did not make this request. It places this request on the queue of requests awaiting seosd decision.

7. The seosd daemon is now caught in a deadlock. It is waiting for the reply necessary to complete name resolution, but the process that should provide this reply cannot proceed until seosd gives it permission to accept the network connection. The first request generates the second, and creates a deadlock.

Name Resolution on Sun Solaris: Deadlock

Name resolution on Sun Solaris entails accessing the *nscd* cache. The nscd is a process that provides a cache for the most common name service requests. nscd furnishes caching for the passwd, group, and hosts databases.

The cache is not permanent. It becomes invalid as changes are made to the passwd, group, and hosts databases, or as the time-to-live stamp expires.

The Sun Solaris setup can create a deadlock like the one described in the previous section. Here, the interaction between CA Access Control and the nscd process causes the deadlock.

1. During name resolution, CA Access Control accesses the nscd cache.

2. The nscd process can decide that the cache is too old. In this case, it attempts to refresh the information by accessing the passwd, group, and hosts databases (locally or on a server).

3. The request to access these databases is intercepted by the kernel extension. Since a CA Access Control process is not making the request, it is placed on a queue awaiting seosd decision. But no such decision is possible because seosd is still engaged in the previous request. The first request generates the second, and creates a deadlock.

Avoiding Deadlocks: The Lookaside Database

The setting of the under_NIS_server token in the seos.ini configuration file has a default setting of yes to avoid deadlocks. The token tells CA Access Control to use its own internal name resolution tables instead of NIS, DNS, or the nscd cache. Unless otherwise specified, these tables reside in memory.

CA Access Control internal name resolution is much faster than NIS name resolution and even faster than using files; using CA Access Control internal name resolution improves performance even in an environment where there is no danger of deadlocks.

Note: There is no cache for the internal name resolution tables in the lookaside database. CA Access Control uses an open file handle to read data from the tables.

Storing Resolution Tables on Disk

CA Access Control name resolution tables are generated while CA Access Control is starting up. The tables should be maintained on disk, not in memory because storage in memory can lead to memory overload. Also, when the information is read into memory, it is static. Because of this, CA Access Control would not know of any changes made to user, group, or host information. The only way to update the tables in memory is to restart CA Access Control.

To keep data current, CA Access Control provides a lookaside database that makes sure internal name resolution tables are stored on disk.

Note: To implement the lookaside database you need to use seos.ini configuration settings. For more information about seos.ini configuration settings, see the *Reference Guide*.

Setting Up the Lookaside Database

The four tables in the lookaside database are userdb.la, groupdb.la, hostdb.la, and servdb.la. These four tables handle user, group, host, and service name resolution requests. The tables are located in the directory specified by the lookaside_path token in the seos.ini file, which by default is /opt/CA/AccessControl/ladb.

Lookaside Database with Four Tables

To set up the lookaside database with the four tables, do one of the following:

- If you are installing CA Access Control, answer yes when asked if you want to create the lookaside database.

- If you already installed CA Access Control:

  a. In the [seosd] section of seos.ini change the following tokens to **yes**:

     - under_NIS_server

     - use_lookaside

  b. Run sebuildla -a to create all four tables.

Lookaside Database with Less Than Four Tables

You can also create one, two, or three tables. For example, if you want to use the lookaside database to resolve hosts only, complete the following steps:

1. After you install CA Access Control, change the following tokens in the [seosd] section of the seos.ini file:

   - Set under_NIS_server to blank.

   - Set HostResolution to ladb.

2. Run sebuildla -h to create a table of all hosts, including local and DNS hosts.

   or

   Run sebuildla -e to create a table of local hosts only (defined in /etc/hosts).

To create a lookaside database with other tables, use the appropriate tokens in the seos.ini file and then run the appropriate option with sebuildla.

Note: For descriptions of these tokens, see the seos.ini initialization file in the *Reference Guide*. For more information about sebuildla, see the *Utilities Guide*.

Important! Run sebuildla whenever you add a host.

How the Lookaside Database Works

The four tables in the lookaside database (groupdb.la, hostdb.la, servdb.la, and userdb.la) contain resolution information for groups, hosts, services, and host names. The tables are located in the directory specified by the lookaside_path token in the seos.ini file, which by default is /opt/CA/AccessControl/ ladb.

CA Access Control internal name resolution is much faster than NIS name resolution and even faster than looking up th files.

Implementing the Lookaside Database

Note: The problems and solutions outlined here are for informational purposes only. Actual settings are correct upon installation and most users need not take any action.

Here is a broad overview of how CA Access Control implements the lookaside database:

- The relevant tokens in the seos.ini file are set.

- The relevant symbolic links in the /opt/CA/AccessControl/exits directory are defined.

- The command /opt/CA/AccessControl/bin/sebuildla -a was issued to build the lookaside database.

The sebuildla utility taps into the native resolution mechanisms such as th files and NIS to build the lookaside database.

No security-sensitive information (such as password, location of the home directory, or gecos) is kept in the lookaside tables. The lookaside database tables contain only a numeric ID number and a name.

Once the lookaside database is created, update it using the sebuildla utility. You do *not* need to restart CA Access Control.

Updating the Hosts Lookaside Table

You must update the hosts lookaside table. To do so, execute sebuildla -h at regular intervals (site-specific). Use cron jobs to do this.

Every time you change the UNIX user or group databases utilizing selang, you must run the sebuildla utility. CA Access Control provides exit scripts for this purpose, which runs sebuildla with the appropriate parameters.