Authorization Services Programming Guide

Security

2009-01-06

Apple Inc.
© 2002, 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Mac, Mac OS, and Panther are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction **Introduction to Authorization Services Programming Guide 7**

Organization of This Document 7
See Also 8

Chapter 1 **Authorization Concepts 9**

Authorization 9
Authentication 11
The Security Server 12
Rights 13
The Policy Database 14
The Credentials Cache and the Authentication Dialog 15
Scenarios 16
- Simple, Self-Restricted Applications 17
- Factored Applications 17
- Installers 20

Chapter 2 **Authorization Services Tasks 23**

Authorizing in a Simple, Self-Restricted Application 23
- Creating an Authorization Reference Without Rights 23
- Requesting Authorization 24
- Releasing an Authorization Reference 28

Authorizing in a Factored Application 28
- Using Authorization Services in a Factored Application 28
- Using Authorization Services in a Helper Tool 31

Calling a Privileged Installer 34

Document Revision History 37

Glossary 39

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Figures, Tables, and Listings

Chapter 1 **Authorization Concepts 9**

Figure 1-1 An example of the System Preferences application as seen by an unauthorized user 10
Figure 1-2 An example of the System Preferences application as seen by a preauthorized user 11
Figure 1-3 An example of authentication in the System Preferences application 12
Figure 1-4 Flow chart for a simple, self-restricted application 17
Figure 1-5 Flow chart for the application part of a factored application 18
Figure 1-6 Flow chart for a helper tool 18
Figure 1-7 Flow chart for a self-repairing helper tool 20
Figure 1-8 Flow chart of an application to call a privileged installer 21
Figure 1-9 Flow chart of an installer's Authorization Services calls. 21
Table 1-1 A comparison of the steps involved in authorization and the immigration processes 12
Table 1-2 Rule attributes and descriptions 14

Chapter 2 **Authorization Services Tasks 23**

Listing 2-1 Creating an authorization reference without rights 24
Listing 2-2 Creating an authorization item array 25
Listing 2-3 Creating a set of authorization rights 25
Listing 2-4 Specifying authorization options for authorization 26
Listing 2-5 Specifying authorization options for partial authorization 26
Listing 2-6 Authorizing rights 26
Listing 2-7 Authorizing partial rights 27
Listing 2-8 Creating an authorization reference with rights 27
Listing 2-9 A one-time authorization call 27
Listing 2-10 Releasing an authorization item array 28
Listing 2-11 Releasing an authorization reference 28
Listing 2-12 Specifying authorization options for preauthorization 30
Listing 2-13 Creating an external authorization reference 30
Listing 2-14 Retrieving an authorization reference 31
Listing 2-15 Executing a helper tool with root privileges 33
Listing 2-16 Setting the setuid bit 34
Listing 2-17 Calling a privileged installer 34

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Introduction to Authorization Services Programming Guide

Note: This document was previously titled *Performing Privileged Operations With Authorization Services*.

Authorization Services defines a programming interface that facilitates fine-grain control of **privileged operations**, such as accessing restricted areas of the operating system and self-restricted parts of your Mac OS X application. This document describes how to use Authorization Services to control these privileged operations.

Performing Privileged Operations With Authorization Services explains the concepts behind authorization and provides examples of how to use Authorization Services.

Types of products that benefit from using Authorization Services include

- applications that call system-restricted tools
- software that restricts access to its own tools
- software installers that install privileged tools or require access to restricted areas of the operating system

For example, you can use Authorization Services to restart background processes or to gain access to restricted directories, such as the `/Applications` directory. Using Authorization Services properly in these situations greatly minimizes the possibility of your software inadvertently damaging restricted areas of the operating system, or allowing an unauthorized user access to these areas.

Your application can benefit from Authorization Services if it includes tools or performs operations to which you want only administrative users to have access.

Authorization Services uses the authentication mechanism in Mac OS X. If future versions of Mac OS X support additional authentication mechanisms, adopting Authorization Services now will enable your application to take advantage of these mechanisms with no change to your code.

Organization of This Document

"Authorization Concepts" (page 9) introduces you to authorization in Mac OS X and describes the difference between authorization and authentication. This chapter explores scenarios that use Authorization Services. Read this chapter to better understand whether your software could benefit from using Authorization Services.

"Authorization Services Tasks" (page 23) explains in detail how to use Authorization Services in self-restricting applications, system-restricting applications, and privileged installers.

"Glossary" (page 39) defines new terms introduced in this book.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

See Also

A companion volume to *Performing Privileged Operations With Authorization Services* is *Authorization Services Reference*, which provides a detailed explanation of every function, data type, and constant defined by Authorization Services for use by your application.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Authorization Concepts

This chapter covers concepts rather than implementation or programming details. See "Authorization Services Tasks" (page 23) for information about using specific Authorization Services functions in your application.

You should understand the basics of permissions and ownership in BSD and Mac OS X before reading this chapter. See Chapter 13, "Installation and Integration", of *Inside Mac OS X: System Overview* for a brief introduction to these concepts. For definitions of terms, see the "Glossary" (page 39).

This chapter contains the following sections:

- "Authorization" (page 9) provides a conceptual overview of the policy-based authorization used by Mac OS X.
- "Authentication" (page 11) describes how authorization uses authentication.
- "The Security Server" (page 12) describes how you use Authorization Services in your application to interact with the Security Server.
- "Rights" (page 13) describes how to name your own rights.
- "The Policy Database" (page 14) explains how the Security Server uses a policy database to make authorization decisions.
- "The Credentials Cache and the Authentication Dialog" (page 15) explains how the Security Server determines whether to display an authentication dialog.
- "Scenarios" (page 16) describes different scenarios that use Authorization Services.

Authorization

The underlying BSD portion of the Mac OS X kernel provides a user-and-owner-security model. Each file system object, such as a file or directory, has an owner and a set of **permissions**, or attributes, specifying what the owner, one group, and all others are able to do with the object.

There are cases where the BSD security model doesn't fit situations faced by Mac OS X users. For example, if you want to create a grades-and-transcripts application, you'll want teachers and school registrars to use the application, but you may want to restrict the creation of transcripts to just the registrars.

You may need to protect the user from accidentally making important changes that the underlying BSD security model allows. For example, you may want a user to authenticate as an administrator before changing application-specific preferences. Authorization Services can also be used to perform operations as **root**—also known as the superuser—such as restarting a daemon.

In these cases, a policy-based security model, used in addition to the BSD permissions, provides additional important features for your application. In a **policy-based system**, a user requests **authorization**—the act of granting a right or privilege—to perform a privileged operation. Authorization is performed through an agent so the user doesn't have to trust the application with a password. The agent is the user interface—

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

operating on behalf of the Security Server—used to obtain the user's password or other form of identification, which also ensures consistency between applications. The **Security Server**—a Core Services daemon in Mac OS X that deals with authorization and authentication—determines whether no one, everyone, or only certain users may perform a privileged operation.

Authorization offers you fine-grained control over granting users privileges to perform administrative tasks and other privileged operations. Using Authorization Services allows you to restrict parts of your application, add extra security precautions, and still satisfy the BSD security model. You should avoid bypassing the BSD security model—for example, don't run processes as root—unless you have no alternative, in which case you should limit the amount of code involved.

Note: It is your responsibility, in a policy-based system, to request authorization for your users. Your application should authorize immediately before every privileged operation.

In some circumstances it is valuable to determine if the user is authorized to perform privileged operations well before your application actually needs to perform those operations. For example, when the System Preferences application is locked, it requires a user to provide a name and password before it will allow the user to change any settings. When the user clicks on the lock button (see Figure 1-1), the System Preferences application performs **preauthorization**. Preauthorization determines a user's rights before authorization is required. By preauthorizing, System Preferences prevents users from customizing and selecting options for an operation they are not authorized to perform.

Figure 1-1 An example of the System Preferences application as seen by an unauthorized user

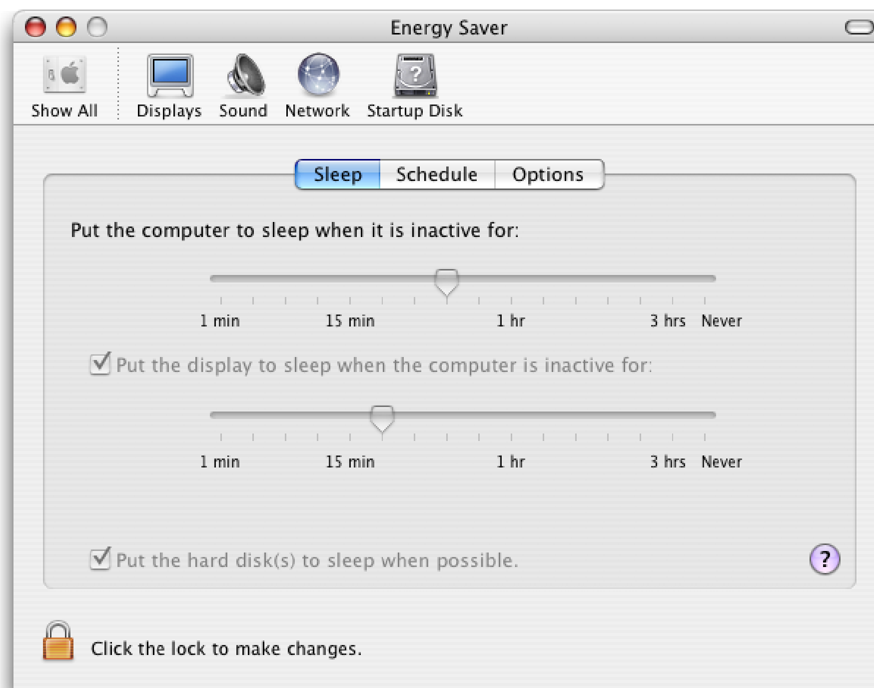
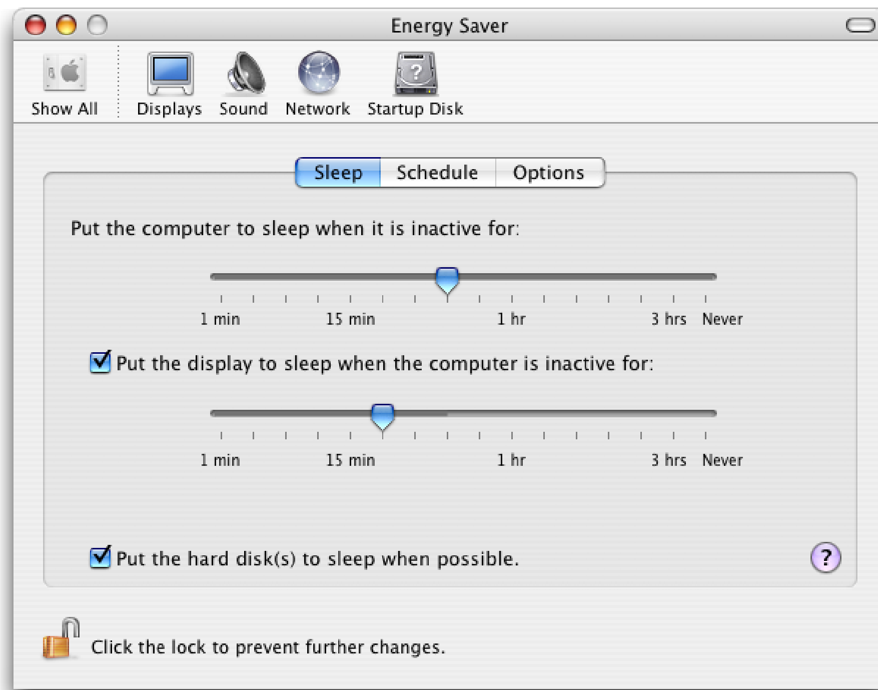


Figure 1-2 shows the window the user sees after successfully preauthorizing. However, the System Preferences application still performs authorization immediately before carrying out any privileged operation.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Figure 1-2 An example of the System Preferences application as seen by a preauthorized user



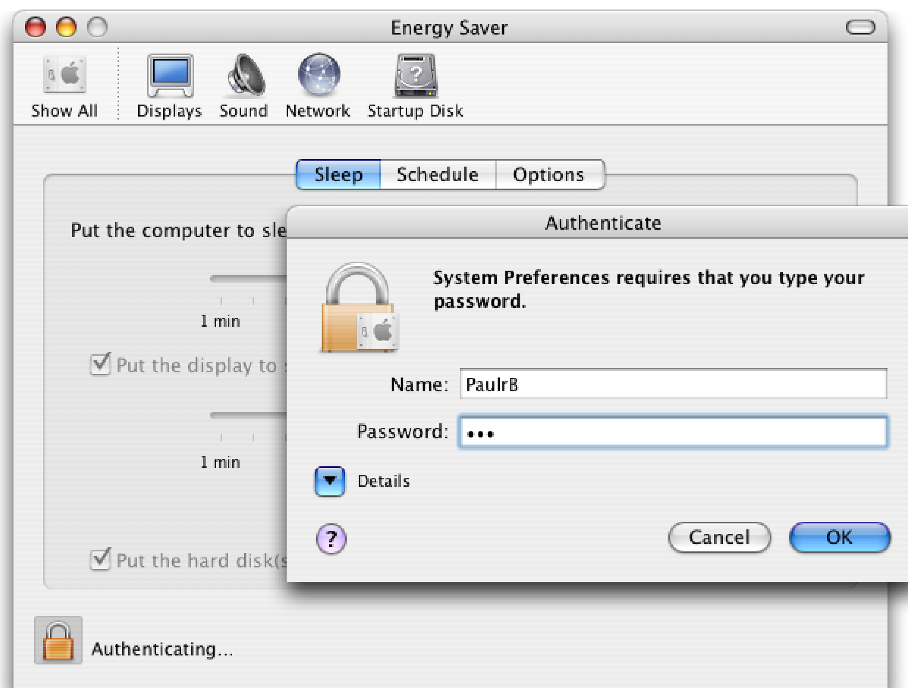
Authentication

Authentication is the act of verifying the identity of the user. A common misconception is that authorization and authentication are one and the same; however, authentication is only part of the authorization process. As discussed in "Authorization" (page 9), after the user is authenticated, the authorization process involves determining what rights or privileges that user has.

Figure 1-3 shows an example of authentication in the System Preferences application.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Figure 1-3 An example of authentication in the System Preferences application



Typically today, the user types in a user name and password to be authenticated. In future releases of Mac OS X, the user might produce a smart card, use a **biometric identifier**, such as a fingerprint or retinal scan, or use a combination of authentication methods.

When your application requests authorization of the user, you can set an option that allows the Security Server to interact with the user. Doing so tells the Security Server to request proof of identity from the user for authentication purposes, as needed.

The Security Server

The Security Server processes authorization requests in a manner analogous to how an immigration official processes visas. Table 1-1 compares the two processes.

Table 1-1 A comparison of the steps involved in authorization and the immigration processes

Immigration	Authorization
The immigrant provides a passport and visa to the immigration official.	The application provides the authorization reference, authorization rights set, and authorization options to the Security Server.
The immigration official uses the visa number to access information about the immigrant.	The Security Server uses the authorization reference to access credentials.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Immigration	Authorization
The immigration official uses the picture in the passport to validate the identity of the immigrant.	The Security Server asks the user to provide a user name and password for authentication.
The immigration official uses the privileges requested in the visa to look up the laws in the policy book.	The Security Server uses the rights in the authorization rights set to look up the rules in the policy database.
The immigration official uses the credentials to determine if the immigrant complies with the laws and should be granted the privileges requested in the visa.	The Security Server uses the credentials and authorization options to determine if the user complies with the rules and should be granted the rights requested in the authorization rights set.
The immigration official informs the immigrant whether or not he grants the privileges requested in the visa.	The Security Server returns a result granting or denying the authorization rights.

To initiate an authorization session between your application and the Security Server, you create an **authorization reference**. The Security Server uses the authorization reference to access the authorization session. You pass the authorization reference to almost every Authorization Services function.

When you request authorization, you send instructions to the Security Server in the form of **authorization options**. The authorization options tell the Security Server how to proceed with the authorization request. For example, you can specify that the call is for authorization, partial authorization, or preauthorization. You can also specify whether you want to allow the Security Server to interact with the user to perform authentication.

To authorize a user, you must pass the Security Server an **authorization rights set** that contains rights a user needs, such as the right to create a transcript or restart a daemon. A **right** is a named privilege that the application requests on behalf of a user.

A **credential** is a token representing an authenticated user that the Security Server stores as part of the authorization session. The Security Server uses these credentials as proof of authenticity. Credentials expire after a set length of time. You can also force their expiration when freeing an authorization reference.

The Security Server uses a **policy database** that contains a set of rules. A **rule** is a set of attributes that determine who should be authorized to perform a specific action. The Security Server compares the rules with a user's rights and authentication credentials to determine if the user is authorized to perform a privileged operation.

Rights that are granted are not stored in the authorization session. Instead, every time authorization is performed, the Security Server uses the credentials—or reauthenticates the user if the credentials have expired—and consults the appropriate rule in the policy database to reevaluate the authorization.

Rights

When your application requests authorization, you pass the requested rights (an authorization rights set) to the Security Server. The Security Server compares the rights you pass to the keys in the policy database. When a match is found, the Security Server uses the rules associated with the key to determine authorization. For more information about the policy database see "The Policy Database" (page 14).

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

You must create the rights your application uses. Rights use a hierarchical namespace. The right should begin with the reverse domain name of your organization. The right should then specify the name of your application and become more specific—for example, `com.myOrganization.myProduct.myRight`. Rights that are specific to Mac OS X have right names that begin with `system`.

Note: Rights are case sensitive.

Your right should represent an individual action on one or a group of targets. For example, a right might represent the individual action of restarting a daemon, such as `com.myOrganization.myProduct.inetd.restart` to restart the Internet daemon, or `com.myOrganization.myProduct.daemons.restart` to restart a group of daemons.

Because you can request multiple rights for the same user, there is no need to create rights that represent combinations of actions. For example, in a grades-and-transcripts application, if you name a right `com.myOrganization.myProduct.transcripts.create` and another right `com.myOrganization.myProduct.grades.edit`, there is no need for a separate right `com.myOrganization.myProduct.createTranscriptsAndEditGrades`.

The name you select for a right should make sense to the user. For example, `system.finder.trash.empty` is more readily understood than `system.finder.trashDirectory.deleteFiles`.

The Policy Database

The policy database contains a set of rules the Security Server uses to authorize rights for a user. Each rule consists of a set of attributes. The rules are preconfigured when Mac OS X is installed, but an application may change them at any time. Because any application can change the rights in the database, your application must take into account all possible scenarios. Table 1-2 describes the attributes defined for rules.

There are some specific rules in the policy database for Mac OS X applications. There is also a generic rule in the policy database that the Security Server uses for any right that doesn't have a specific rule.

Table 1-2 Rule attributes and descriptions

Rule attribute	Generic rule value	Description
`key`		The **key** is the name of a rule. A key uses the same naming conventions as a right. The Security Server uses a rule's key to match the rule with a right. Wildcard keys end with a '.'. The generic rule has an empty key value. Any rights that do not match a specific rule use the generic rule.
`group`	`admin`	The user must authenticate as a member of this group. This attribute can be set to any one group.
`shared`	`true`	If this is set to `true`, then the Security Server marks the credentials used to gain this right as shared. The Security Server may use any shared credentials to authorize this right. For maximum security, set sharing to `false` so credentials stored by the Security Server for one application may not be used by another application.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Rule attribute	Generic rule value	Description
timeout	300	The credential used by this rule expires in the specified number of seconds. For maximum security where the user must authenticate every time, set the timeout to 0. For minimum security, remove the timeout attribute so the user authenticates only once per session.

Your right always matches up with the generic rule unless a new rule is added to the policy database. Use the AuthorizationRightSet function to add or edit a rule in the database. Use the AuthorizationRightGet function to read the current rule. Use the AuthorizationRightRemove function to delete a rule.

To lock out all privileged operations not explicitly allowed, change the generic rule by setting the timeout attribute to 0. To allow all privileged operations once the user is authorized, remove the timeout attribute from the generic rule. To prevent applications from sharing rights, set the shared attribute to false. To require users to authenticate as a member of the staff group instead of the admin group, set the group attribute to staff.

As an example of how the Security Server matches a right with a rule in the policy database, consider a grades-and-transcripts application. The application requests the right com.myOrganization.myProduct.transcripts.create. The Security Server looks up the right in the policy database. Not finding an exact match, the Security Server looks for a rule with a wildcard key set to com.myOrganization.myProduct.transcripts., com.myOrganization.myProduct., com.myOrganization., or com.—in that order—checking for the longest match. If no wildcard key matches, then the Security Server uses the generic rule. The Security Server requests authentication from the user. The user provides a user name and password to authenticate as a member of the group admin. The Security Server creates a credential based on the user authentication and the right requested. The credential specifies that other applications may use it, and the Security Server sets the expiration to five minutes.

Three minutes later, a child process of the application starts up. The child process requests the right com.myOrganization.myProduct.transcripts.create. The Security Server finds the credential, sees that it allows sharing, and uses the right. Two and a half minutes later, the same child process requests the right com.myOrganization.myProduct.transcripts.create again, but the right has expired. The Security Server begins the process of creating a new credential by consulting the policy database and requesting user authentication.

The Credentials Cache and the Authentication Dialog

You might notice that when you call the AuthorizationCreate function or the AuthorizationCopyRights function to obtain rights for a user, sometimes an authentication dialog appears and at other times the dialog does not appear. The reason for this behavior is related to the settings in the policy database and the way in which the Security Server caches user credentials.

A credential is something that the Security Server knows about a particular user, such as the fact that a particular user has entered a valid user name and password.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

For each login session, the Security Server maintains a global credentials cache and a credentials cache per authorization instance (that is, for each time a new authorization reference is created). The rule for each right in the policy database indicates the group to which the authenticated user must belong and how long the credential is considered valid. The rule might also indicate that the credential is to be shared.

When Authorization Services needs a credential in order to grant a right to a user, the Security Server attempts to obtain the credential from a credentials cache. It first looks in the credentials cache associated with the authorization instance. If the credential isn't there and credentials are shared, it then looks in the global credentials cache. Only if the Security Server can't find the credential in a cache does it try to acquire the credential, typically by displaying an authentication dialog. (In some cases, the Security Server might be able to acquire the credential from another source, such as a smart card.)

If the Security Server successfully obtains a new credential, it stores it in the credentials cache associated with the authorization instance and—if the rule specifies that the credential should be shared—in the global credentials cache.

If the rule for the right has a timeout attribute, its value indicates how long (in seconds) a cached credential is applicable for this right. A value of `0` means that the credential can only be used once (that is, it times out immediately). If the timeout attribute is missing, the credential can be used to grant the right as long as the login session lasts, unless the credential is explicitly destroyed.

When a user who is a member of the admin group logs on to the system, for example, the user's credential (that is, the fact that they have entered a valid admin user name and password) is saved in the global credentials cache. Then when this user attempts to modify a system preference, Security Server finds the credential in the cache and does not display an authentication dialog.

On the other hand, if a user logs on with a non-admin user name and password and tries to modify one of the system preferences, Security Server cannot obtain the needed credential from a credentials cache. Therefore, it displays the authentication dialog.

The same principle applies for any application that requires a credential: if the user has been authenticated for one application and the credential has been shared, another application can use that credential.

Consequently, whether a call to `AuthorizationCopyRights` results in an authentication dialog depends on whether the Security Server has already cached the required credential.

The only way to guarantee that a credential acquired when you request a right is not shared with other authorization instances is to destroy the credential. To do so, call the `AuthorizationFree` function with the flag `kAuthorizationFlagDestroyRights`.

Scenarios

There are three main scenarios that involve Authorization Services: simple self-restricted applications, factored applications, and installers.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Simple, Self-Restricted Applications

A **self-restricted application** requires that certain features be accessible only by a specific group of users. In a simple self-restricted application, this separation of features is done within the main application. You use Authorization Services in this situation because this kind of fine-grain restriction cannot be controlled by BSD permissions.

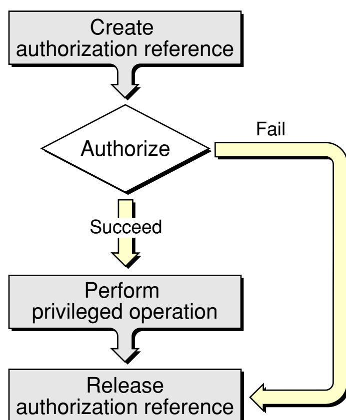
Consider a grades-and-transcripts application that allows only registrars to create transcripts, while the rest of the application is available to both teachers and registrars. When a user attempts to create a transcript, the application uses Authorization Services to decide if that user may perform the operation.

Figure 1-4 shows a flow chart of a simple, self-restricting application. The application creates an authorization reference. The authorization reference refers to the authorization session with the Security Server. Immediately before performing any privileged operation, such as creating a transcript, the application requests authorization on behalf of the user. If required, the Security Server requests authentication of the user. If authorization succeeds, the privileged operation is performed. The application releases the authorization reference when it is no longer needed.

In most cases, it is beneficial to separate the privileged operations into a separate helper tool. See "Factored Applications" (page 17) for more information on how to use Authorization Services with helper tools.

You can perform authorization without creating an authorization reference if you need to authorize just once—for example, when your application first starts up. To do so you use the result of the authorization call directly. Because in this case you did not create an authorization reference, you don't have to release it. One-time authorization is described in more detail in the section "Authorizing" (page 26).

Figure 1-4 Flow chart for a simple, self-restricted application



Factored Applications

A **factored application** is an application that delegates specific tasks to smaller, separate tools. These tools are sometimes referred to as **helper tools**. In a simple, self-restricted application, the privileged code is in the application itself, whereas in a factored application, the privileged code is in the helper tool.

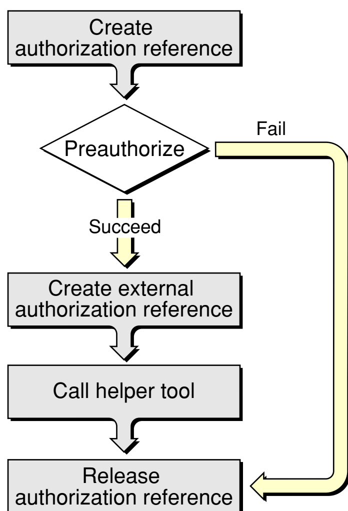
An operation that your application performs might be restricted by the BSD security model. Such an application is a **system-restricted** application. For example, an application that requires restarting the Internet daemon (`inetd`) must have root privileges, but it runs with the privileges of the user that started it.

 © 2002, 2009 Apple Inc. All Rights Reserved.

It is recommended that you factor both self-restricted applications and system-restricted applications. Factoring your application provides two benefits. The first is that it is easier to audit a factored application because the privileged operation is performed in a separate process by the helper tool. The second is that a factored application provides more security. In a nonfactored application, you not only have to trust that there are no security holes in your code but also no holes in all of the code that you link to.

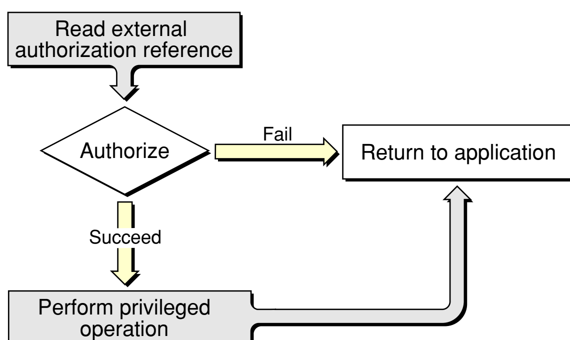
Figure 1-5 shows a flow chart for the application part of a factored application while the flow chart for the helper tool is shown in Figure 1-6.

Figure 1-5 Flow chart for the application part of a factored application



The application begins by creating an authorization reference and requesting preauthorization from the Security Server immediately before calling the helper tool. The application uses the results of preauthorization to determine if the user has the right to perform the privileged operations in the helper tool. Performing preauthorization ensures that resources and time aren't wasted invoking a helper tool that the user does not have the right to use.

Figure 1-6 Flow chart for a helper tool



The application needs to pass the authorization reference to the helper tool. Because you cannot transfer an authorization reference itself between two processes, the application uses Authorization Services to create an external, transferable form of the authorization reference to send to its helper tool.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

The helper tool uses Authorization Services to create an authorization reference from the external authorization reference. The helper tool requests authorization and uses the results to decide whether to continue with the privileged operation.

You must pass the authorization reference to your helper tool so that the authorization dialog can show your application's path rather than the path to the helper tool and to allow the system to determine whether the authorization dialog should have keyboard focus.

You must perform authorization immediately before any privileged operation, even if the user has already successfully authorized. Rights can expire, thus it is your responsibility as a developer to ensure that the user is up to date on all required rights. Any application can modify the policy database to set the length of time a right is available (see "The Policy Database" (page 14)).

Note: The ability to perform a privileged operation is governed by the BSD security model, not Authorization Services. The expiration of a right does not prevent a user with the proper BSD permissions from performing certain privileged operations—you must build that prevention into your code by authorizing immediately before privileged operations.

Some system-supplied utilities use Authorization Services. For example, the `authopen` utility uses Authorization Services to open privileged files. If you call a utility like `authopen`, then it is unnecessary to write your own helper tool.

Some privileged operations require special permissions. For example, an application that restarts the Internet daemon must have root privileges. There are three possible ways to perform this operation, all with their own problems:

- Make the application run as root by calling itself with a special Authorization Services function.
- Set the **setuid bit** of the application and change its owner to root, and then use the special Authorization Services function.
- Factor out the operation that performs the privileged operation and put it in a separate **setuid tool**—a tool that has its setuid bit set—and set the setuid tool's owner to root.

Both the first and second options are a security breach waiting to happen. When the privileged application runs, it calls a special function that Authorization Services provides—`AuthorizationExecuteWithPrivileges` (see "Calling a Helper Tool as Root" (page 32) for more details). Calling this function executes any application or tool with root privileges, regardless of the owner of the application or tool. This is very dangerous since parts of the application can be easily replaced.

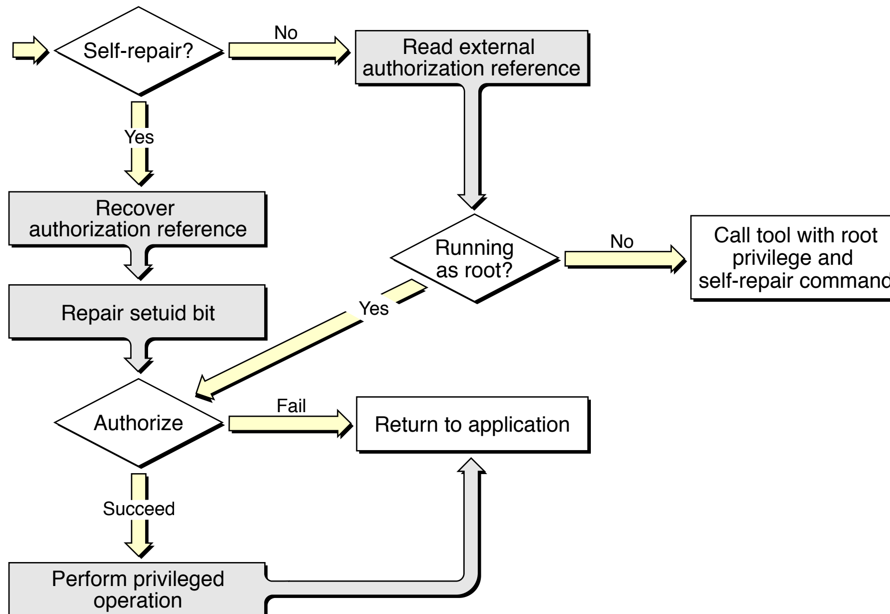
The second option is to set the setuid bit of the privileged application and change its owner to root. The setuid bit, when set, allows the process running it to masquerade as another user. Setting the setuid bit and owner of the application to a different user, such as root, makes it more difficult to replace. However, running code as root is very dangerous and should be done as seldom as possible. Setting the setuid bit on an entire application is especially dangerous because you are trusting that your entire application, and the code your application links to, is free of security holes.

The third scenario is by far the best. The application is split into an application that controls all of the graphical user interface elements and nonprivileged operations, and a helper tool that performs only the operations involved in restarting the Internet daemon. The helper tool's setuid bit is set and the owner is set to root. The proper Authorization Services functions are used as described previously. Factoring and setting the setuid bit not only minimizes the risk but also makes it easier to audit your code for security holes.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

One problem with the last scenario is that setuid bit settings are lost when a file is copied. Thus, if the user copies your setuid tool, the setuid bit is no longer set. It is possible to reset the setuid bit in the setuid tool itself. Figure 1-7 shows the flow chart of a setuid tool that repairs its own setuid bit. You may not want the user to be able to copy the application. If that is the case, you don't need to worry about repairing the setuid bit, just let the user know that they need to reinstall the application.

Figure 1-7 Flow chart for a self-repairing helper tool



When the self-repairing helper tool is run, it checks if the self-repair flag was passed. If the self-repair flag was not passed, then it reads in the external authorization reference that the application passed.

The self-repairing setuid tool then checks if it is running as root. If it is running as root, then authorization is performed and, based on the result, the privileged operation is performed. If the self-repairing setuid tool is not running as root, then it calls itself with the `AuthorizationExecuteWithPrivileges` function and passes itself the self-repair flag.

When the new self-repairing setuid tool process starts, it checks if the self-repair flag was passed. If the self-repair flag was passed, then the self-repairing setuid tool recovers the authorization reference and repairs the setuid bit. After the self-repairing setuid tool is running as root, it performs authorization and continues as a normal helper tool.

Installers

Not all installers require authorization—only those that need special privileges to copy files to restricted directories, make changes to restricted files, or set setuid bits.

An installer is a special case because unlike other applications, an installer is usually run only once. Due to the limited use, Authorization Services provides a function to invoke your installer to run with root privileges. It is up to the user to determine if the installer is from a trusted source.

 © 2002, 2009 Apple Inc. All Rights Reserved.

Figure 1-8 shows a flow chart of an application using Authorization Services to call an installer. In this case, the application creates the authorization reference and performs preauthorization. If the user successfully preauthorizes, then you call the `AuthorizationExecuteWithPrivileges` function to execute your installer with root privileges.

Figure 1-8 Flow chart of an application to call a privileged installer

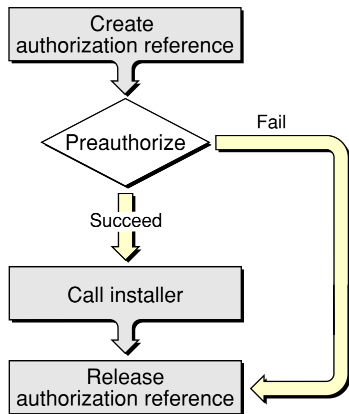
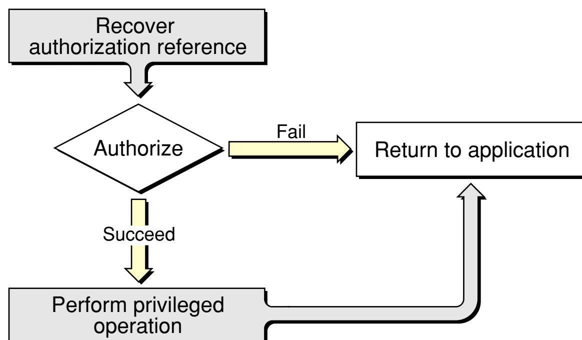


Figure 1-9 shows a flow chart for the Authorization Services calls performed in the installer itself. Authorization should be performed before any privileged operation.

Figure 1-9 Flow chart of an installer's Authorization Services calls.



2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Authorization Services Tasks

This chapter provides instructions and code samples for tasks you can accomplish with Authorization Services. You can adapt these samples in your own application to

- Restrict access to parts of your own application
- Call system utilities
- Edit privileged files
- Install your privileged tools

A simple, self-restricting application needs to restrict a user from the application's own operations with minimal security concerns—for example, a grades-and-transcripts application might only allow the registrar to create transcripts. Read "Authorizing in a Simple, Self-Restricted Application" (page 23) if you have a self-restricting application.

If you have a factored application—for example, an application that must perform an operation as root, such as restarting a daemon—you should read both "Authorizing in a Simple, Self-Restricted Application" (page 23) and "Authorizing in a Factored Application" (page 28).

If your installer must perform a privileged operation, read "Calling a Privileged Installer" (page 34) to see an example of an installer using Authorization Services.

See http://developer.apple.com/samplecode/Sample_Code/Security.htm for sample applications that perform system-restricted privileged operations.

Authorizing in a Simple, Self-Restricted Application

A simple, self-restricted application uses Authorization Services to perform the tasks described in the following sections:

- "Creating an Authorization Reference Without Rights" (page 23)
- "Requesting Authorization" (page 24)
- "Releasing an Authorization Reference" (page 28)

Creating an Authorization Reference Without Rights

The Security Server uses the authorization reference to access the state of the authorization session, which includes any stored credentials. Your application needs only one authorization reference.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

You use the AuthorizationCreate function to allocate memory for the authorization reference. The code fragment in Listing 2-1 shows a call to the AuthorizationCreate function that creates an authorization reference without rights. An authorization reference without rights is useful if the rights are not needed immediately, but the authorization reference is required so it can be used in different parts of the application. For example, in the grades-and-transcripts application, the authorization reference might be created when the application starts, but rights aren't requested until the user attempts to create transcripts.

Note: Although you can create an authorization reference and assign rights in one call, keep in mind that if authorization is denied, the reference is not created and subsequent attempts to use it will fail. Therefore, it is often preferable to create the authorization reference without rights, as shown in Listing 2-1, and then call the AuthorizationCopyRights function later to determine or change the rights, as shown in "Requesting Authorization" (page 24).

Listing 2-1 Creating an authorization reference without rights

```
AuthorizationRef myAuthorizationRef;
OSStatus myStatus;
myStatus = AuthorizationCreate (NULL, kAuthorizationEmptyEnvironment,
            kAuthorizationFlagDefaults, &myAuthorizationRef);
```

The AuthorizationCreate function takes four parameters. The first is an authorization rights set. Since NULL is passed, no rights are authorized at this time. The second parameter is the authorization environment, which is not currently implemented; pass kAuthorizationEmptyEnvironment. The third parameter is the authorization options. The constant kAuthorizationFlagDefaults is passed because the application is not requesting any rights. The fourth parameter is the address of the authorization reference you declared. On return, the authorization reference refers to the current authorization session. If the authorization reference is created successfully, the function returns errAuthorizationSuccess.

"Requesting Authorization" (page 24) describes how to use the AuthorizationCopyRights and AuthorizationCreate functions to request authorization. When your application is done with the authorization reference, use the AuthorizationFree function as described in "Releasing an Authorization Reference" (page 28).

Requesting Authorization

After you create an authorization reference, you can then request authorization. Your application should perform authorization immediately before every privileged operation. In the case of the grades-and-transcripts example, the application requests authorization immediately before creating a transcript.

When your application requests authorization, the Security Server may request the user to authenticate. Authorization Services allows you to take full advantage of the Security Server's authentication plug-in architecture to deal with authentication for you. Instead of a user name and password, the authentication may use fingerprints or smart cards, but your application code stays the same.

Figure 1-3 (page 12) shows the authentication dialog that the Security Server provides. The user enters an administrator user name and password and clicks OK. The Security Server then uses the user name and password to authenticate and authorize the user.

Authorization requires the creation of an authorization rights set and authorization options to use in a call to the functions AuthorizationCopyRights or AuthorizationCreate. In your application, request authorization by performing the tasks described in the following sections:

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

- "Creating an Authorization Rights Set" (page 25)
- "Specifying Authorization Options " (page 26)
- "Authorizing" (page 26)
- "Releasing an Authorization Item Array" (page 28)

Creating an Authorization Rights Set

To authorize a user for specific rights, you must create an authorization rights set to pass to the Security Server through the `AuthorizationCopyRights` or `AuthorizationCreate` functions. The authorization rights set consists of an authorization item array and the number of items in the authorization item array. The authorization item array contains information about the rights that your application is requesting.

Each item in the authorization item array consists of four pieces of information:

- The name of the right
- A value that contains optional data pertaining to the right
- The byte length of the `value` field
- Optional flags

Listing 2-2 shows an example of an authorization item array. In most cases, when creating an item for a right, you set the `value` field to `NULL`, and the `valuelength` and `flags` fields to `0`. You should set the `name` field to the name of the right you are requesting. For information on naming your own rights, see "Rights" (page 13).

Listing 2-2 Creating an authorization item array

```
AuthorizationItem myItems[2];

myItems[0].name = "com.myOrganization.myProduct.myRight1";
myItems[0].valueLength = 0;
myItems[0].value = NULL;
myItems[0].flags = 0;

myItems[1].name = "com.myOrganization.myProduct.myRight2";
myItems[1].valueLength = 0;
myItems[1].value = NULL;
myItems[1].flags = 0;
```

For example, a grades-and-transcripts application might request the right `com.myOrganization.myProduct.transcripts.create`. The `valueLength`, `value`, and `flags` fields would be unused and set to `0`, `NULL`, and 0, respectively.

Listing 2-3 shows an example of an authorization rights set. In the authorization rights set, the `count` field contains the number of rights in the authorization item array, while the `items` field points to the authorization item array you created.

Listing 2-3 Creating a set of authorization rights

```
AuthorizationRights myRights;
myRights.count = sizeof (myItems) / sizeof (myItems[0]);
myRights.items = myItems;
```


2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Specifying Authorization Options

You use the authorization options to instruct the Security Server how to proceed with the `AuthorizationCopyRights` and `AuthorizationCreate` functions. By setting the authorization options, you can use these functions to

- Authorize partial rights
- Authorize all rights
- Preauthorize rights

You can include with these options the option to interact with the user. The Security Server requires user interaction to perform authentication. The most common combination authorizes all rights and allows user interaction. Listing 2-4 shows an example of the authorization options for authorization.

Listing 2-4 Specifying authorization options for authorization

```
AuthorizationFlags myFlags;
myFlags = kAuthorizationFlagDefaults |
            kAuthorizationFlagInteractionAllowed |
            kAuthorizationFlagExtendRights;
```

The `kAuthorizationFlagDefaults` constant zeros the bit mask. The `kAuthorizationFlagExtendRights` constant instructs the Security Server to grant the rights. Without this flag, the `AuthorizationCopyRights` and `AuthorizationCreate` functions would return the appropriate error code, but no rights would be extended to the user.

If your application does not require all of the rights to be authorized, you can include the `kAuthorizationFlagPartialRights` constant to request partial authorization. You can then determine what to allow the user to do based on which rights the Security Server grants. Listing 2-5 shows an example of setting the authorization options for partial authorization.

Listing 2-5 Specifying authorization options for partial authorization

```
myFlags = kAuthorizationFlagDefaults |
            kAuthorizationFlagInteractionAllowed |
            kAuthorizationFlagExtendRights |
            kAuthorizationFlagPartialRights;
```

See "Requesting Preauthorization" (page 29) to learn what authorization options to set for preauthorization.

Authorizing

The code fragment in Listing 2-6 shows a call to the `AuthorizationCopyRights` function based on the authorization reference and authorization rights set you created and the authorization options you specified. In the grades-and-transcripts example, the `AuthorizationCopyRights` function is used to authorize the right to create a transcript.

Listing 2-6 Authorizing rights

```
myStatus = AuthorizationCopyRights (myAuthorizationRef, &myRights,
        kAuthorizationEmptyEnvironment, myFlags, NULL);
```


2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

The first parameter is the authorization reference created in "Creating an Authorization Reference Without Rights" (page 23). The second parameter is the authorization rights set created in "Creating an Authorization Rights Set" (page 25). The third parameter is the authorization environment. The authorization environment is not currently implemented, so pass `kAuthorizationEmptyEnvironment`. The fourth parameter is the authorization options set in "Specifying Authorization Options " (page 26).

The fifth parameter is useful when authorizing partial rights as shown in Listing 2-7. This parameter points to an empty authorized rights set you declare. On return, this consists of the rights that the Security Server actually authorizes. If you create a pointer to an authorized rights set, then you should release it as described in "Releasing an Authorization Item Array" (page 28).

Listing 2-7 Authorizing partial rights

```
AuthorizationRights *myAuthorizedRights;
myStatus = AuthorizationCopyRights (myAuthorizationRef, &myRights,
            kAuthorizationEmptyEnvironment, myFlags,
            &myAuthorizedRights);
```

The `AuthorizationCopyRights` function returns `errAuthorizationSuccess` if the Security Server grants all the rights. You can use the return status to determine whether the user may perform the privileged operation.

You can use an authorization rights set and authorization options to request authorization when you create an authorization reference.

> **Note:** Although you can create an authorization reference and assign rights in one call, as shown in Listing 2-8, keep in mind that if authorization is denied, the reference is not created and subsequent attempts to use it will fail. Therefore, it is often preferable to create the authorization reference without rights, as shown in Listing 2-1 (page 24), and then call the `AuthorizationCopyRights` function later to determine or change the rights, as shown in the preceding code samples in this section.

Listing 2-8 shows an example combining authorization with the `AuthorizationCreate` function.

Listing 2-8 Creating an authorization reference with rights

```
myStatus = AuthorizationCreate (&myRights, kAuthorizationEmptyEnvironment,
            myFlags, &myAuthorizationRef);
```

You can also use the `AuthorizationCreate` function to authorize a user for a one-time privileged operation. One-time authorization is useful if your application needs to authorize only once when it is run. Listing 2-9 shows an example of how to use an authorization rights set and authorization options with the `AuthorizationCreate` function without producing an authorization reference. Pass `NULL` instead of an authorization reference.

Listing 2-9 A one-time authorization call

```
myStatus = AuthorizationCreate (&myRights, kAuthorizationEmptyEnvironment,
            myFlags, NULL);
```


2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Releasing an Authorization Item Array

When you finish with the authorization item set in Listing 2-7 (page 27), call the `AuthorizationFreeItemSet` function, as shown in Listing 2-10, to release the memory it uses. Use this function only on authorization item arrays that the Security Server allocates, such as those used in the `AuthorizationCopyRights` and `AuthorizationCopyInfo` functions.

Listing 2-10 Releasing an authorization item array

```
myStatus = AuthorizationFreeItemSet (myAuthorizedRights);
```

Releasing an Authorization Reference

Before exiting your application, or at any time you want to end the current authorization session, call the `AuthorizationFree` function to release the authorization reference. For example, the grades-and-transcripts application would wait until the user quits the application before releasing the authorization reference. Using the same authorization reference every time the user creates a transcript allows the Security Server to reuse any shared credentials that haven't expired. In contrast, an action such as the user clicking the open-lock button in the Network preferences pane can trigger the release of the authorization reference, requiring the user to reauthorize when she clicks the closed-lock button.

The code segment in Listing 2-11 shows an example of using the `AuthorizationFree` function. You must pass the authorization reference and authorization options. For authorization options, pass the constant `kAuthorizationFlagDefaults` if you want to revoke the credentials associated with the current process, or pass the constant `kAuthorizationFlagDestroyRights` to release all shared credentials from all processes that use them.

Listing 2-11 Releasing an authorization reference

```
myStatus = AuthorizationFree (myAuthorizationRef,
            kAuthorizationFlagDestroyRights);
```

Authorizing in a Factored Application

Factored applications, whether system-restricted or self-restricted, use an application to control the graphical user interface and nonprivileged operations and use a separate helper tool to perform the privileged operations.

Read "Using Authorization Services in a Factored Application" (page 28) for a description of using Authorization Services in a factored application and "Using Authorization Services in a Helper Tool" (page 31) for a description of using Authorization Services in a helper tool.

Using Authorization Services in a Factored Application

You can use Authorization Services in your factored application to perform the tasks described in the following sections:

- "Creating an Authorization Reference" (page 29)

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

- "Requesting Preauthorization" (page 29)
- "Creating an External Authorization Reference" (page 30)
- "Calling a Helper Tool" (page 30)
- "Releasing an Authorization Reference" (page 31)

An example of a factored application is one that restarts the Internet daemon. The application performs all the nonprivileged operations while the helper tool restarts the daemon. The application creates an authorization reference and preauthorizes the right to restart the Internet daemon. The application uses the result to determine whether to start the helper tool. The application creates an external version of the authorization reference and passes it to the helper tool. When the authorization reference is no longer needed, the application releases it.

Creating an Authorization Reference

Creating an authorization reference in a factored application is the same as in a simple, self-restricting application. See "Creating an Authorization Reference Without Rights" (page 23) to learn how to create an authorization reference.

Requesting Preauthorization

You should preauthorize rights before calling a helper tool. Using the results of preauthorization, you can prevent an unauthorized user from invoking the helper tool. Doing so saves the time of starting a new process and using resources as well as saving the user from preparing to perform an operation he doesn't have privileges to perform.

Preauthorization requires the creation of an authorization rights set and authorization options to use in a call to the functions `AuthorizationCopyRights` or `AuthorizationCreate`. In your application, you preauthorize a user by performing the steps described in the following sections:

- "Creating a Preauthorization Rights Set" (page 29)
- "Specifying Authorization Options for Preauthorization" (page 29)
- "Preauthorizing" (page 30)

Creating a Preauthorization Rights Set

A preauthorization rights set is the same as an authorization rights set as described in "Creating an Authorization Rights Set" (page 25).

Specifying Authorization Options for Preauthorization

Authorization options for preauthorization are similar to the authorization options for authorization and partial authorization described in "Specifying Authorization Options " (page 26). The only difference between preauthorization and authorization is that you are not using the result to determine if a user can perform a privileged operation. Instead, you should use the result to determine if the user can be authorized at a later time.

Listing 2-12 shows an example of setting the authorization options for preauthorization. The `kAuthorizationFlagDefaults` constant zeros out the bit mask. The `kAuthorizationFlagExtendRights` constant tells the Security Server to extend any rights granted to the user. The

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

`kAuthorizationFlagInteractionAllowed` constant tells the Security Server that it may interact with the user for authentication purposes. The `kAuthorizationFlagPreAuthorize` constant tells the Security Server to preauthorize the rights requested.

Listing 2-12 Specifying authorization options for preauthorization

```
AuthorizationFlags myFlags;
myFlags = kAuthorizationFlagDefaults |
            kAuthorizationFlagExtendRights |
            kAuthorizationFlagInteractionAllowed |
            kAuthorizationFlagPreAuthorize;
```

Preauthorizing

Calling the `AuthorizationCopyRights` or `AuthorizationCreate` function is the same for preauthorization as it is for authorization. See Listing 2-6 (page 26) in the section "Authorizing" (page 26) for examples.

Creating an External Authorization Reference

After creating the authorization reference and preauthorizing rights, you need to pass the authorization reference to your helper tool. Sharing the authorization reference allows the helper tool to use any credentials that are part of the factored application's authorization session. When you pass the authorization reference to your helper tool, the authorization dialog can show your application's path rather than the path to the helper tool. It also enables the system to determine whether the authorization dialog should have keyboard focus.

One problem with the authorization reference is that it is not in a form that can be transferred from one process to another. To solve this problem, Authorization Services provides a function to translate the authorization reference into an external authorization reference that you can pass to your helper tool.

To create an external authorization reference, declare a variable of type `AuthorizationExternalForm` and pass it to the `AuthorizationMakeExternalForm` function along with the existing authorization reference. On return, `AuthorizationExternalForm` is a transferable form of the authorization reference. Listing 2-13 shows an example of creating an external authorization reference.

Listing 2-13 Creating an external authorization reference

```
AuthorizationExternalForm myExternalAuthorizationRef;
myStatus = AuthorizationMakeExternalForm (myAuthorizationRef,
            &myExternalAuthorizationRef);
```

Read "Retrieving an Authorization Reference" (page 31) to learn how to retrieve the authorization reference from the external authorization reference in your helper tool.

Calling a Helper Tool

When you are ready to call your helper tool, pass the external authorization reference to the tool using some form of interprocess communication, such as a communications pipe. For a sample application and self-repairing helper tool, see http://developer.apple.com/samplecode/Sample_Code/Security.htm. For more information on interprocess communications, see *Inside Mac OS X: System Overview*.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Releasing an Authorization Reference

Releasing the authorization reference in a factored application is the same as described in "Releasing an Authorization Reference" (page 28).

Using Authorization Services in a Helper Tool

You can use Authorization Services in your helper tool to perform the tasks described in the following sections:

- "Retrieving an Authorization Reference" (page 31)
- "Performing Authorization" (page 31)
- "Executing the Privileged Operation" (page 32)

For example, a helper tool that restarts the Internet daemon retrieves the authorization reference from the external authorization reference passed by the application. Then the helper tool requests authorization immediately before restarting the Internet daemon.

If your helper tool is actually a self-repairing helper tool, you should also read "Repairing a Helper Tool" (page 32).

Retrieving an Authorization Reference

To share the authorization session, the factored application passes an external authorization reference to the helper tool (see "Creating an External Authorization Reference" (page 30)). In the helper tool, you use the `AuthorizationCreateFromExternalForm` function to retrieve an authorization reference from the external authorization reference.

Listing 2-14 shows an example using the `AuthorizationCreateFromExternalForm` function. In this example, the external authorization reference is read in from a communications pipe between the helper tool process and the parent process. You then pass the external authorization reference to the function `AuthorizationCreateFromExternalForm`. On return, `myAuthorizationRef` is the authorization reference.

Listing 2-14 Retrieving an authorization reference

```
AuthorizationRef myAuthorizationRef;
AuthorizationExternalForm myExternalAuthorizationRef;
OSStatus myStatus;

/* *** You should read in the external authorization reference into
        myExternalAuthorizationRef here. *** */

myStatus = AuthorizationCreateFromExternalForm (&myExternalAuthorizationRef,
            &myAuthorizationRef);
```

Performing Authorization

Performing authorization in your helper tool is the same as it is for simple, self-restricted applications. See "Requesting Authorization" (page 24) for more information.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Executing the Privileged Operation

You should use the result of the authorization to determine whether the user is allowed to perform the privileged operation. There are no Authorization Services functions required for actually executing the privileged operation.

Repairing a Helper Tool

Important: For a newer and more secure alternative to the approach documented here, see the sample code in Better Authorization Sample (*BetterAuthorizationSample*), which uses the launchd daemon to launch the helper tool rather than setting the setuid bit.

If your helper tool needs to run as root to perform privileged operations, such as restarting the Internet daemon, then it should have its setuid bit set. Tools that have the setuid bit set (sometimes referred to as *setuid tools*) must be Mach-O binaries because CFM binaries don't support the setuid or setgid (set group identifier) bit. When you install your program, your installer should set the helper tool's setuid bit, and set its owner to root.

In Mac OS X v10.1 and earlier, when a user moves a setuid tool to another volume, or copies it from one place to another, the setuid bit is reset by the file system and the group and owner change to match the user moving the setuid tool. This is done purposely to reduce the security risk that a setuid tool poses by allowing any user to run the setuid tool as root. On the other hand, most users expect that when they copy an application or tool from one folder to another, it will still work. Thus, the setuid bit, group, and owner need to be reset without editing the permissions in the terminal window. This section provides code to allow your setuid tool to repair its own setuid bit when this problem occurs.

Note: In Mac OS X v10.1, setuid tools that are copied or moved lose their setuid bit. and the owner and group are changed to match the permissions of the user performing the action. In later releases, users can move setuid tools and preserve the permission set.

All setuid tools are potential security problems. This case poses a particular problem because the tool self-repairs its setuid bit even if a user tampers with the setuid tool's code. As added security, you might want to display a warning to users whenever performing this action so they can decide to continue or cancel the self-repair operation, or possibly force the user to reinstall the application from the installer.

You can repair the setuid bit on your helper tool by performing the tasks described in the following sections:

- "Calling a Helper Tool as Root" (page 32)
- "Setting the Setuid Bit" (page 33)

Calling a Helper Tool as Root

For your helper tool to set its own setuid bit, the tool must have root privileges. This is a circular problem, since you can't change permissions on your helper tool unless your helper tool is already running as root. This is where the function `AuthorizationExecuteWithPrivileges` comes into play.

The `AuthorizationExecuteWithPrivileges` function executes any application as root through a special security process. The code sample in Listing 2-15 demonstrates how a helper tool can recursively call itself with root privileges so it can repair its own setuid bit.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Note: The `AuthorizationExecuteWithPrivileges` function sets only the effective user ID (EUID) of the tool to root. The real user ID (RUID) of the tool is that of the caller.

Important: Because of the security risk, calling the `AuthorizationExecuteWithPrivileges` function is recommended only for special and infrequent use such as repairing setuid bits and installing your application. In the most secure computer systems, the right this function requests times out immediately so every time you call it, the user must authenticate.

Listing 2-15 Executing a helper tool with root privileges

```
FILE *myCommunicationsPipe = NULL;
char *myArguments[] = {"--self-repair", NULL};
char myPath[MAXPATHLEN];

/* *** You should determine the path of your tool here and put the result in
        myPath. *** */

myStatus = AuthorizationExecuteWithPrivileges (myAuthorizationRef,
            myPath, kAuthorizationFlagDefaults, myArguments,
            &myCommunicationsPipe);
```

The `AuthorizationExecuteWithPrivileges` function expects you to pass five parameters. The first parameter is the authorization reference you retrieved as shown in "Retrieving an Authorization Reference" (page 31). The authorization reference allows the helper tool to use any credentials that are part of the factored application's authorization session. The second parameter is the full POSIX pathname of the helper tool—in this case, the setuid tool—that is being called. The third parameter is the authorization options. In this function, this parameter is not implemented, so for now, set it to `kAuthorizationFlagDefaults`. The fourth parameter is a null-terminated array of arguments for the tool being called. You can use this parameter to pass any information you need from the parent process to the child process. In this case, the string `"--self-repair"` is passed to indicate to the helper tool that it should execute the code in Listing 2-16 (page 34). The fifth parameter is a communications pipe so the helper tool can pass the data it received from the factored application to itself.

Important: You may be tempted to use the function `AuthorizationExecuteWithPrivileges` to perform privileged operations rather than creating and calling your own setuid tool. Although this might seem like an easy solution, using the `AuthorizationExecuteWithPrivileges` function without the rest of the Authorization Services functions produces a severe security hole because the function indiscriminately runs any tool as the root user. Setuid tools also have security risks, but they are far less severe than using the function `AuthorizationExecuteWithPrivileges` for purposes other than those described in this document. Read "Factored Applications" (page 17) for instructions on creating your own helper tool.

Setting the Setuid Bit

In Listing 2-15 (page 33), the helper tool recursively calls itself, passing the self-repair argument, `--self-repair`. Therefore, in the same helper tool, you need to check for the self-repair argument and, if it is found, fix the setuid bit. See the More Is Better sample code (*MoreIsBetter*) for a sample self-repairing setuid tool.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Note: The purpose of the self-repair code in the helper tool discussed here is to allow the tool to execute as root after the user has moved or copied the tool, even if the file system has reset the setuid bit and changed the owner and group to match the permissions of the user performing the action. This self-repair code (that is, the call to the `AuthorizationExecuteWithPrivileges` function) works if the setuid bit has been cleared, whether the tool is owned by root or by the user. If the tool has the setuid bit set and is owned by root, the self-repair code is not called. However, if the setuid bit is set and the tool is owned by the user, the call to the `AuthorizationExecuteWithPrivileges` function fails because the setuid bit is respected in this case and the tool executes with the user's privileges rather than root privileges. If you want your self-repairing helper tool to handle this unlikely circumstance, you need to add code to clear the setuid bit before you call the self-repair code.

When you call the `AuthorizationExecuteWithPrivileges` function, you need a way to retrieve the authorization reference that is passed in the call. Listing 2-16 shows code using the `AuthorizationCopyPrivilegedReference` function to retrieve the authorization reference. The only time you use this function is to retrieve an authorization reference passed by a call to `AuthorizationExecuteWithPrivileges`.

The first parameter of the call to the `AuthorizationCopyPrivilegedReference` function is an empty authorization reference you declare. You should not call the `AuthorizationCreate` function. On return, the authorization reference points to a copy of the original authorization reference. The second parameter is not implemented, so set it to `kAuthorizationFlagDefaults`.

Listing 2-16 Setting the setuid bit

```
myStatus = AuthorizationCopyPrivilegedReference (&myAuthorizationRef,
            kAuthorizationFlagDefaults)
```

Calling a Privileged Installer

Occasionally, an installer must install files in directories that are not owned by the user running the installer. This should be a rare case and you should avoid it if at all possible. In the event that it can't be avoided, the code in Listing 2-17 shows a tool that runs the `/usr/bin/id` utility with optional flag `-un`. By replacing the utility path and including your own flags, you can use this sample code to call your installer with root privileges. Your installer will then be able to perform any privileged operations it requires.

Listing 2-17 Calling a privileged installer

```
#include <Security/Authorization.h>
#include <Security/AuthorizationTags.h>

int read (long,StringPtr,int);
int write (long,StringPtr,int);

int main() {

    OSStatus myStatus;
    AuthorizationFlags myFlags = kAuthorizationFlagDefaults;           // 1
    AuthorizationRef myAuthorizationRef;                               // 2

    myStatus = AuthorizationCreate(NULL, kAuthorizationEmptyEnvironment,   // 3
                myFlags, &myAuthorizationRef);
```


2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

```
    if (myStatus != errAuthorizationSuccess)
        return myStatus;

    do
    {
        {
            AuthorizationItem myItems = {kAuthorizationRightExecute, 0,        // 4
                    NULL, 0};
            AuthorizationRights myRights = {1, &myItems};                      // 5

            myFlags = kAuthorizationFlagDefaults |                             // 6
                    kAuthorizationFlagInteractionAllowed |
                    kAuthorizationFlagPreAuthorize |
                    kAuthorizationFlagExtendRights;
            myStatus = AuthorizationCopyRights (myAuthorizationRef,            // 7
                                        &myRights, NULL, myFlags, NULL );
        }

        if (myStatus != errAuthorizationSuccess) break;

        {
            char myToolPath[] = "/usr/bin/id";
            char *myArguments[] = { "-un", NULL };
            FILE *myCommunicationsPipe = NULL;
            char myReadBuffer[128];

            myFlags = kAuthorizationFlagDefaults;                              // 8
            myStatus = AuthorizationExecuteWithPrivileges                      // 9
                    (myAuthorizationRef, myToolPath, myFlags, myArguments,
                    &myCommunicationsPipe);

            if (myStatus == errAuthorizationSuccess)
                for(;;)
                {
                    int bytesRead = read (fileno (myCommunicationsPipe),
                            myReadBuffer, sizeof (myReadBuffer));
                    if (bytesRead < 1) break;
                write (fileno (stdout), myReadBuffer, bytesRead);
                }
        }
    } while (0);

    AuthorizationFree (myAuthorizationRef, kAuthorizationFlagDefaults);       // 10

    if (myStatus) printf("Status: %ld\n", myStatus);
    return myStatus;
}
```

Here are explanations of the numbered lines of code in Listing 2-17:

1. Declare a variable to store authorization options.

2. Declare an authorization reference.

3. Use the `AuthorizationCreate` function to initialize the authorization reference. See "Creating an Authorization Reference Without Rights" (page 23) for more information.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

4. Create an authorization item array. The user must have the right to execute to use the `AuthorizationExecuteWithPrivileges` function. To create a right to execute authorization item, set the `name` field to `kAuthorizationRightExecute`, the `value` fields to `NULL`, the `valueLength` and `flags` fields to `0`. See "Creating an Authorization Rights Set" (page 25) for more information.

5. Create an authorization rights set. Set the `count` field to the number of items in the authorization item array, and set the `items` field to point to the authorization item array. See "Creating an Authorization Rights Set" (page 25) for more information.

6. Set the authorization options to preauthorize the rights. See "Specifying Authorization Options for Preauthorization" (page 29) for more information.

7. Use the `AuthorizationCopyRights` function to preauthorize the right to execute your installer as root. In this case, there is no reason to continue if the user can't preauthorize. See "Authorizing" (page 26) for more information.

8. Set the authorization options for the `AuthorizationExecuteWithPrivileges` function to `kAuthorizationFlagDefaults`. Other authorization options, such as that specified by the `kAuthorizationFlagInteractionAllowed` constant, are not necessary because the `AuthorizationExecuteWithPrivileges` function interacts with the user whether you specify the option or not.

9. Use the `AuthorizationExecuteWithPrivileges` function to invoke your installer. Pass the authorization reference in the first parameter. Pass the installer's full POSIX pathname in the second parameter. Pass the authorization options default in the third parameter. Pass any arguments for the installer in the fourth parameter. A communications pipe to the tool may be set up through the fifth parameter. See "Calling a Helper Tool as Root" (page 32) for more information about the `AuthorizationExecuteWithPrivileges` function.

10. Release the authorization reference using the `AuthorizationFree` function. See "Releasing an Authorization Reference" (page 28) for more details.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Authorization Services Programming Guide*.

Date	Notes
2009-01-06	Corrected minor errors. Changed the title from "Performing Privileged Operations With Authorization Services."
2004-02-01	Added information about modifying the policy database file.
	Added a description of the use of the credentials caches.
2003-10-01	Updated screen shots for Panther.
	Added information about `AuthorizationExecuteWithPrivileges` and the `setuid` bit
2002-10-01	Added information about the policy database file.
2002-06-01	First version of this document.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

Glossary

administrator A user in the admin group. The user who installs Mac OS X is automatically assigned to the admin group. An administrator has fewer privileges than root, but more privileges than a normal user. An administrator cannot create, delete, or move files in the system domain.

authentication The act of verifying identity with something the user has, knows, or is. For example, a user knows information such as a name and password. The user may have something physical such as a smart card. The identity can be something the user is—a physical feature such as a fingerprint or retinal scan. Authentication may require two or more forms of identification.

authorization The act of granting a right. For example, a user asks for the right to perform an operation. The Security Server grants authorization after the user fulfills the rules specified in the policy database—such as providing a credential or authenticating.

authorization option A parameter or field that instructs the Security Server how to proceed with a request. Options include requesting preauthorization, requesting partial authorization, appending rights, and interacting with the user.

authorization reference The Security Server uses the authorization reference to access an authorization session associated with a process.

Authorization Services An API that facilitates fine-grain control of privileged operations, such as accessing restricted areas of the operating system and self-restricted parts of your Mac OS X application. The Security Server uses policy-based decisions to authorize rights for users.

biometric identifier A measurement of biological matter used for identification—for example, fingerprints, retinal scans, and face recognition.

credential Proof of user authentication. used by the Security Server. When the Security Server authenticates a user, it creates a credential as part of the authorization session.

factored application An application that uses a helper tool to perform specific tasks. Interprocess communication mechanisms are used to communicate between processes. In a factored application that uses Authorization Services, factor the code that performs privileged operations is factored into a separate helper tool.

helper tool A tool that executes some of an application's functions as a separate process. In the case of security, a helper tool performs privileged operations for the application. See also setuid tool.

key The name of a rule. The Security Server uses a rule's key to match a right with a rule.

permissions In BSD, a set of attributes governing who can read, write, and execute resources in the file system. The output of the `ls -l` command represents permissions as a nine-position code segmented into three binary three-character subcodes; the first subcode gives the permissions for the owner of the file, the second for the group that the file belongs to, and the last for everyone else. For example, `-rwsr-xr--` means that the owner of the file has read, write, execute permissions (rwx); the group has read and execute permissions (r-x); all others have only read permissions. (The left-most position is reserved for a special character that says if this is a regular file (-), a directory (d), a symbolic link (l), or a special pseudo file device.) The execute bit has a different semantic for directories, meaning they are searchable.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.

policy-based system A system that requires authorization to perform a privileged operations.

policy database A database containing the set of rules the Security Server uses to determine authorization.

preauthorization A form of authorization used before performing the actual authorization. Preauthorization is used to determine if a user has the possibility of authorizing later.

privileged operation An operation that requires special rights or permissions. For example, all operations a user performs as root are privileged.

right A named privilege. The Security Server authorizes rights for a user to perform a privileged operation.

rule A set of attributes used to set security policies for applications and for the system. See also policy database.

root (1) The user with unlimited system privileges. Also called the superuser. (2) The top directory in a BSD-style directory hierarchy. Written as a slash (/), it is the first element in every absolute pathname.

Security Server A Core Services application in Mac OS X that deals with authorization and authentication through interaction with the policy database and Pluggable Authentication Modules (PAM).

self-restricted application An application that restricts part of its features to specific users.

setuid bit The fourth bit in a resource's permissions code. When this bit is set to `s`, the system allows the process running it to masquerade as another user. For example, `-r-sr-xr-x 1 root wheel traceroute` allows the process running the `traceroute` utility to run as root.

setuid tool A tool that has its setuid bit set.

system-restricted application An application that has a portion of its features restricted to specific users because of the BSD permissions system.

2009-01-06 | © 2002, 2009 Apple Inc. All Rights Reserved.