RSA ACE/Server 5.2 External Authorization API Guide

Contact Information

See our Web sites for regional Customer Support telephone and fax numbers.

RSA Security Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie

Trademarks

License agreement

Note on encryption technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Distribution

Limit distribution of this document to trusted personnel.

RSA Security Notice

First printing: October 2003

Contents

Preface

This guide is intended only for programmers, security administrators and other trusted personnel. Do not make it available to the general user population.

This guide explains how to use a generic Application Program Interface (API) for External Authorization to create and set up customized External Authorization criteria. These customized criteria work in addition to, not in place of, RSA ACE/Server authentication. When External Authorization is enabled, users attempting to gain access to your system must meet an extra set of criteria that you create based on the needs of your organization.

It is assumed that users of this guide are programmers who are experienced with C programming on Windows or UNIX systems, and are familiar with RSA ACE/Server authentication procedures. Because this guide is for experienced programmers, it gives detailed procedures only where necessary. After reading this guide, you should be able to set up and test External Authorization on an RSA ACE/Server system with the customized External Authorization criteria. Once you have finished these setup tasks, an RSA ACE/Server administrator must enable External Authorization as explained in the *RSA ACE/Server 5.2 Administrator's Guide* (**ace_admin.pdf** in the ***ACEDOC*** directory).

Supported Platforms and Operating Systems

Refer to the *RSA ACE/Server 5.2 for UNIX Installation Guide* or the *RSA ACE/Server 5.2 for Windows Installation Guide* (**ace_install_unix.pdf** or **ace_install_windows.pdf** in the ***ACEDOC*** directory) for information on supported platforms and operating systems.

Getting Support and Service

RSA SecurCare® Online	**https://knowledge.rsasecurity.com**
Customer Support Information	**www.rsasecurity.com/support**

Before You Call for Technical Support

Note: Technical support is not provided during the warranty period unless a valid Software Service Contract is in force.

Make sure you have direct access to the computer running the RSA ACE/Server software.

Please have the following information available when you call:

- Your RSA Security Customer/License ID. You can find this number on the license distribution medium or by running the Configuration Management application on the Windows NT or Windows 2000 platforms, or by typing ‘sdinfo’ on any UNIX platform.
- RSA ACE/Server software version number.
- The make and model of the machine on which the problem occurs.
- The name and version of the operating system under which the problem occurs.

RSA ACE/Server 5.2 External Authorization API

About External Authorization

External Authorization lets you use a generic Application Program Interface to create customized criteria for authorizing users within your organization. This customized authorization is an addition to the RSA ACE/Server authentication, not a replacement for it. If the External Authorization option is enabled, users attempting access must meet this extra set of criteria before they are permitted access to your system. If the External Authorization option for authorization of remote login requests is also enabled, users are permitted access from a remote realm.

The RSA ACE/Server first applies the default authentication criteria to determine a user's identity. If the RSA ACE/Server authenticates the user's identity, a call is made to External Authorization. This call initiates a process that applies your customized criteria to determine whether or not a user is permitted access.

If External Authorization fails, a message is written to the event log (Windows) or the system log (UNIX) indicating that, during authentication, a particular user did not meet the customized authorization criteria.

When the RSA ACE/Server is stopped, a call is made to terminate your External Authorization routines.

External Authorization Components

RSA Security provides the dynamic linked library file **libsdxauthr.dll** for Windows-based systems, and these shared library files for UNIX systems: **libsdxauthr.so** (Solaris), **libsdxauthr.sl** (HP), **libsdxauthr.so** (AIX).

The libraries are loaded on your Server when you install RSA ACE/Server 5.2 software. They contain the following four API routines:

iSDExtAuthorInit(). Performs any required initialization for External Authorization.

iSDExtAuthorCheck(). Determines whether or not the user is authorized to log in to the Agent Host.

iSDExtAuthorGetHomeData(). Gets authorization data from the home realm as part of the enterprise authentication.

iSDExtAuthorShutdown(). Shuts down External Authorization.

RSA Security also provides the following stubbed routines on the RSA ACE/Server 5.2 CD that you use to create your own External Authorization routines.

sdxauthr.c. The template source file used to build the default dynamic linked library on Windows, and the default shared libraries on UNIX (**acesupp\xauth_sdk\src**).

sdxauthr.h. The header file containing definitions to use when you create your customized External Authorization routines (**acesupp\xauth_sdk\inc**).

RSA Security does *not* provide specific External Authorization code. Instead, you must create your own External Authorization routines based on the template source file. Your customized routines must

- Conform to this External Authorization API.
- Define a customized authorization scheme that integrates tightly with the RSA ACE/Server user-authentication process.

A Task Overview

To set up External Authorization, you must perform the following tasks:

1. Review the information in the following section, "Before You Begin," and in the "Sample Calling Sequences" on page 9.
2. Ask your RSA ACE/Server administrator to verify the authorization service port settings.
3. Create customized External Authorization routines that conform to this API.
4. Substitute your customized External Authorization routines for the stubbed routines supplied by RSA Security.
5. Optionally, create error messages for your customized External Authorization routines.
6. Compile your customized routines and link them, as a dynamic linked library for Windows or as a shared library for UNIX.
7. Test your customized routines, using the RSA ACE/Server in a test environment.

Once you have finished these tasks, an RSA ACE/Server administrator must enable External Authorization as explained in the *RSA ACE/Server 5.2 Administrator's Guide* (**ace_admin.pdf** in the ***ACEDOC*** directory).

Before You Begin

Review the information in this section before you begin setting up External Authorization. This review will help ensure that the setup goes smoothly and produces the results you want.

- Copy the **xauth_sdk** folder from the RSA ACE/Server 5.2 CD (located in the **ace\supp** directory) to the machine on which you are creating your external authorization routines. RSA Security recommends that, for compiling and linking purposes, you do not re-name this folder.
- Note that the RSA ACE/Server 5.2 is multithreaded, and can accommodate simultaneous executions of your customized External Authorization routines. You should verify that your routines are reentrant, and that they are capable of executing in a multithreaded environment.
- You *must* include all four routines in the dynamic linked library, or shared library. Do *not* exclude a stubbed routine if you do not plan to use it. Instead, stub the routine in the library to return 0 status. Maximum lengths for null-terminated strings include the byte for a trailing zero.

- Note the location of the source library files that shipped with the RSA ACE/Server 5.2 software. You must substitute your customized versions of these files in the *same location* as the source files.
- You must provide your RSA ACE/Server system administrator with a list explaining any error messages that you create for your customized External Authorization routines. For messages with return values, you must explain the meaning you assigned to these values.
- After completing the tasks in this guide, ask your RSA ACE/Server system administrator to enable the External Authorization options that the administrator wants. Then, restart your RSA ACE/Server so that your options take effect.

Sample Calling Sequences

Review the following sample sequences before you decide how to set up External Authorization. These sample sequences illustrate how an RSA ACE/Server with External Authorization enabled processes a user's request for access. The processing sequence depends on both your network configuration and which External Authorization options are enabled.

Single Server Configuration

The following sample sequence shows External Authorization of a user on an RSA ACE/Server.

External Authorization enabled
Receive an authentication request from the Agent Host. ↓
At the first request to authenticate a user, call **iSDExtAuthorInit()**. ↓
If RSA ACE/Server authentication is successful, call **iSDExtAuthorCheck()**. Log a message on return. ↓
Send an authentication response to the Agent Host.

Multi-Realm Configuration with External Remote Authorization Enabled on the Home Server

The following sample sequence shows External Authorization of a user from a remote realm. The enterprise authentication request goes to the user's home realm, which accepts the remote request.

Home Server: External Authorization enabled; External Remote Authorization enabled		**Remote Server: External Authorization enabled**
		At the first request from the client to authenticate a user, call **iSDExtAuthorInit()**. ↓
At the first request to authenticate a user, call **iSDExtAuthorInit()**. ↓	←	Send an enterprise authentication request with the External Authorization flag set.
Process the enterprise authentication request. ↓		
If the enterprise authentication is successful, call **iSDExtAuthorGetHomeData()** to check authorization on the user's home realm and return data for authorization.	→	If authentication and home authorization are successful, call **iSDExtAuthorCheck ()** with data from the home realm. ↓
		Send authentication response to the client.

Multi-Realm Configuration with External Remote Authorization Disabled on the Home Server

The following sample sequence shows External Authorization of a user from a remote realm. The enterprise authentication request goes to the user's home realm, which has remote authorization disabled.

Home Server: External Authorization enabled; External Remote Authorization disabled		**Remote Server: External Authorization enabled**
		At the first request to authenticate a user, call **iSDExtAuthorInit()**. Process the enterprise authentication request. ↓
Process the enterprise authentication request. ↓	←	Send an enterprise authentication request with the External Authorization flag set.
Return disabled status indicating that external remote authorization is not enabled on the home realm.	→	Send an authentication response to the client.

Multi-Realm Configuration with External Authorization Disabled on the Remote Server

The following sample sequence shows authentication of a user from a remote realm that has disabled external remote authorization. The enterprise authentication request goes only to the user's home realm for authentication.

Home Server: External Authorization enabled; External Remote Authorization disabled		**Remote Server: External Authorization disabled**
		Receive an authentication request from the client. ↓
Process the enterprise authentication request only, *not* external remote authorization. ↓	←	Send an enterprise authentication request with *no* External Authorization flag set.
Send an authentication response to the client.		

Multi-Realm Configuration with External Authorization Disabled on the Home Server

The following sample sequence shows External Authorization of a user from a remote realm. The enterprise authentication request goes to the user's home realm, which has disabled External Authorization.

Home Server: External Authorization disabled		**Remote Server: External Authorization enabled**
		At the first request to authenticate a user, call **iSDExtAuthorInit()**. ↓
		Authentication request received from the client. ↓
Process the enterprise authentication request. ↓	←	Send an enterprise authentication request for authentication with an External Authorization flag set.
Return an error indicating that External Authorization is not enabled on the home realm.	→	Send an authentication failed response to the client.

Setting Up External Authorization

This section explains the External Authorization setup tasks.

Substituting Customized Routines for the Stubbed Routines

Review the source makefile (named **makefile**) and the source code in the stubbed routines **iSDExtAuthorInit()**, **iSDExtAuthorCheck()**, **iSDExtAuthorGet HomeData()**, and **iSDExtAuthorShutdown()**. Then, create your customized External Authorization routines named **sdxauthr.c**, **sdxauthr.h**, and **makefile**, and substitute them for the stubbed routines.

Compiling and Linking Routines

Windows: use the project file **msvc60.dsp** and the workspace file **msvc60.dsw** to compile your customized routines and to create a dynamically linked library named **libsdxauthr.dll**.

Use **makefile** to compile and customize your routines. Verify that an ANSI C compiler is installed and configured, and that the Lib environment variable is set appropriately. Invoke the makefile by using either of the following commands:

```
cd WORK_AREA/xauth_sdk/nt
NMAKE makefile "CGF=exauth - Win32 Release"
```

or

```
cd WORK_AREA/xauth_sdk/nt
NMAKE makefile "CGF=exauth - Win32 Debug"
```

where **WORK_AREA** is a directory of your choice.

UNIX: use **makefile** located in the **xauth_sdk\unix** directory to compile your customized routines, and to create a shared library named l**ibsdxauthr.so** (Solaris), **libsdxauthr.sl** (HP), or **libsdxauthr.so** (AIX). Verify that an ANSI C compiler is installed and configured. Invoke the compiler as appropriate depending on your environment. Invoke the makefile as follows:

```
cd WORK_AREA /xauth_sdk/unix
make PLATFORM=<plat>
```

where **<plat>** is either SOL, AIX, or HP (case-sensitive), and **WORK_AREA** is a directory of your choice.

Creating Customized Status and Error Messages

Templates for creating messages for your customized External Authorization criteria are in the section for each stubbed routine in "API Routines" on page 14.

Testing Customized Routines

You should test your customized routines to ensure that they function properly with the RSA ACE/Server. Before testing your routines, it is recommended that you

- Create an isolated test environment that depicts the actual environment in which the RSA ACE/Server will function with external authorization enabled. For example, if you intend to use external authorization in conjunction with enterprise authentication, your test environment should include multiple realms.
- Create status messages for each routine indicating success or failure.

When your external authorization routines complete, a response packet is sent to a backend on the RSA ACE/Server. If, during testing, any of your routines terminates unexpectedly, it will cause a backend on the Server to terminate. If this occurs, you should verify that the RSA ACE/Server can still function properly. To do so, perform the following steps:

1. Stop the RSA ACE/Server services.
2. Replace your substituted library with the original one provided by RSA Security.
3. Restart the RSA ACE/Server, and repeat your testing procedure.

API Routines

This section describes the API routines that you use to create customized External Authorization. It also includes a sample audit log. The API routines

- Set up and initialize the External Authorization API.
- Determine whether or not an authenticated user is permitted access.
- Pass additional authorization data from the home server to a remote server (if the **Enable Authorization of Remote Login Requests** option is enabled).
- Shut down the External Authorization API.

Important: Your External Authorization routines must execute as quickly as possible. If either **iSDExtAuthorCheck()** or **iSDExtAuthorGetHomeData()** takes a long time to process, the performance of the RSA ACE/Server severely degrades. If the API takes too long to process, the RSA ACE/Server times out the authentication request and returns a timeout error.

iSDExtAuthorInit()

Performs any required initialization in the External Authorization Module.

Synopsis

```
#include <sdxauthr.h>

int iSDExtAuthorInit(
  SD_U32                uExtAuthorFlags,
  P_SD_ERRSTRING_S      pErrorString);
```

Parameters

Parameter	Explanation
uExtAuthorFlags	Flags indicating the following External Authorization options for this RSA ACE/Server: SD_EXTAUTH_CHECK_ON - Enables External Authorization SD_EXTAUTH_HOMEDATA_ON - Enables external remote authorization; allows users to retrieve data from the home realm.
pErrorString->iStringType	0=ASCII
pErrorString->cErrString	Pointer to location to receive a null–terminated user error string to be appended to syslog or event log.

Description

This routine lets the External Authorization Module initialize itself prior to the first authorization request. If this routine returns an error, **iSDExtAuthorCheck()**, and **iSDExtAuthorGetHomeData()** are not called.

Return Values

0	Initialization succeeded.
nnnnn	Initialization error; the non–zero error value is stored in the audit log record.

iSDExtAuthorCheck()

Authorizes a user to log in to the client.

Synopsis

```
#include <sdxauthr.h>
int iSDExtAuthorCheck (
char  *            pszLoginName,
char  *            pszClientName,
unsigned int             InetAddr,
 P_SD_HOMEDATA_S         pHomeData,
P_SD_ERRSTRING_S         pErrorString);
```

Parameters

Parameter	Explanation
pszLogin	Pointer to null-terminated string buffer containing the login name of the user authenticating on the RSA ACE/Server.
pszClientName	Pointer to null-terminated string buffer containing the name of the client machine the user is accessing.
InetAddr	32-bit unsigned integer containing the IP address of the client in network byte order. For example, address 12.1.68.4 is stored as 0x0C014404.
pHomeData	Pointer to the data structure retrieved from the home server during enterprise authentication. If pHomeData is null, the remote **iSDExtAuthorGetHomeData()** routine completed successfully but did not supply additional data.
pHomeData->iLen	The number of bytes returned in cData.
pHomeData->cData	Data to be returned.
pErrorString->iStringType	0=ASCII
pErrorString->cErrString	Pointer to a location to receive a null-terminated user error string to be appended to syslog or event log.

Description

If External Authorization is enabled on the RSA ACE/Server, this routine is called after a user successfully authenticates.

Important: Your External Authorization routines must execute as quickly as possible. If either **iSDExtAuthorCheck ()** or **iSDExtAuthorGetHomeData()** takes a long time to process, the performance of the RSA ACE/Server severely degrades. If the API takes too long to process, the RSA ACE/Server times out the authentication request and returns a timeout error.

If **iSDExtAuthorInit()** or **iSDExtAuthorGetHomeData()** fails during an enterprise authentication, a user authorization failure is logged and this routine is not called.

Return Values

0	Authorization succeeded.
nnnnn	Authorization error; the non–zero error value is stored in the audit log record.

iSDExtAuthorGetHomeData()

Gets local information to be returned as part of an enterprise authentication.

Synopsis

```
#include <sdxauthr.h>

int iSDExtAuthorGetHomeData (
char  *             pszLogin
char  *             pszClientName
unsigned int        InetAddr
P_SD_HOMEDATA_S     pHomeData
P_SD_ERRSTRING_S    pErrorString);
```

Parameters

Parameter	Explanation
pszLogin	Pointer to a null–terminated string buffer containing the login name of the user authenticating on the RSA ACE/Server.
pszClientName	Pointer to a null–terminated string buffer containing the name of the client machine the user is accessing.
InetAddr	32–bit unsigned integer containing the IP address of the client in network byte order. For example, address 12.1.68.4 is stored as 0x0C014404.
pHomeData	Upon successful authentication, a pointer to the location where data is copied to be returned to remote realm's **iSDExtAuthorCheck()** routine.
pHomeData->iLen	Length of data returned in bytes, 0=no data returned.
pHomeData->cData	Data to be returned
pErrorString->iStringType	0=ASCII

Description

This routine is called if the **Enable Authorization of Remote Login Requests** option is enabled on the RSA ACE/Server. If enabled, the RSA ACE/Server receives an enterprise authentication request with external remote authorization flagged.

Important: Your External Authorization routines must execute as quickly as possible. If either **iSDExtAuthorCheck()** or **iSDExtAuthorGetHomeData()** takes a long time to process, the performance of the RSA ACE/Server severely degrades. If the API takes too long to process, the RSA ACE/Server times out the authentication request and returns a timeout error.

If **iSDExtAuthorInit()** fails:

- An error is returned to the home realm.
- This routine is not called.
- A status of "authorization failed" is returned to the remote realm.

If external remote authorization is disabled in the home realm:

- A user message is logged.
- This routine is not called.
- A status of "authorization disabled" is returned to the remote realm.
- The remote realm skips further authentication.

Return Values

0	External remote authorization succeeded.
nnnnn	External remote authorization error; the non–zero error value is stored in the audit log record.

iSDExtAuthorShutdown()

Shuts down the External Authorization Module.

Synopsis

```
#include <sdxauthr.h>

int  iSDExtAuthorShutdown(
P_SD_ERRSTRING_S pErrorString);
```

Parameters

Parameter	Explanation
pErrorString->iStringType	0=ASCII
pErrorString->cErrString	Pointer to location to receive a null–terminated user error string to be appended to syslog or event log.

Description

This routine is called just before the RSA ACE/Server is shut down or terminated. It allows the External Authorization Module to perform any required shutdown or cleanup procedures before it exits. If **iSDExtAuthorInit()** failed, a user authorization failure is logged and this routine is not called.

Disabling External Authorization while it is running on the RSA ACE/Server generates a call to this routine.

Return Values

0	Shutdown succeeded.
nnnnn	Shutdown error; the non–zero error value is stored in the audit log record.

External Authorization Log Messages

External Authorization log messages are generated on the event log (Windows), and on the system log (UNIX). Each routine returns a null-terminated error string of up to 48 characters that is appended to the log message.

Event Log or syslog Messages

These messages appear in the Windows event log, or in the UNIX system log.

iSDExtAuthorInit()

External Authorization Initialized
External Authorization Initialization error *errstring*

iSDExtAuthorCheck()

External Authorization Check error *errstring*

iSDExtAuthorHomeData()

External Authorization HomeData error *errstring*

iSDExtAuthorShutdown()

External Authorization Shutdown
External Authorization Shutdown error *errstring*

Audit Log Messages

These messages appear in the RSA ACE/Server Audit Log.

General

date time ACCESS DENIED, Ext-auth failed login *user agent host server token*
date time XR ACCESS DENIED, Ext-auth failed login *user agent host server token*
date time Ext-Auth conflict w/ remote realm

iSDExtAuthorCheck

date time Ext-auth check login *user agent host server token*
date time ACCESS DENIED, Ext-auth failed login *user agent host server token*
date time XR ACCESS DENIED, Ext-auth failed login *user agent host server token*

iSDExtAuthGetHomeData

date time Ext-auth home-data login *user agent host server token*
date time ACCESS DENIED, Ext-auth failed login *user agent host server token*
date time XR ACCESS DENIED, Ext-auth failed login *user agent host server token*