webMethods EntireX

Security

Version 9.5 SP1

November 2013

This document applies to webMethods EntireX Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-SECURITY-95SP1-20140628

Table of Contents

1 Which EntireX Security Solution

Choosing a Security Solution

On each operating system where the Broker kernel resides, you have the choice of installing one of Software AG's two security solutions. These solutions, which provide security for your distributed applications using EntireX Broker, are described below.

Security Alternatives

- **EntireX Security**
  This is Software AG's standard security solution, delivered with EntireX. Most organizations that use EntireX choose EntireX Security, Software AG's standard security solution.
- **Sample Security Exits for Broker**
  This is an alternative, user-written security solution for use only in exceptional processing situations.

Note: Do not mix these two security solutions: do not use a stub secured with a sample exit against a kernel secured with EntireX Security or vice versa.

Criteria for Choosing a Security Solution

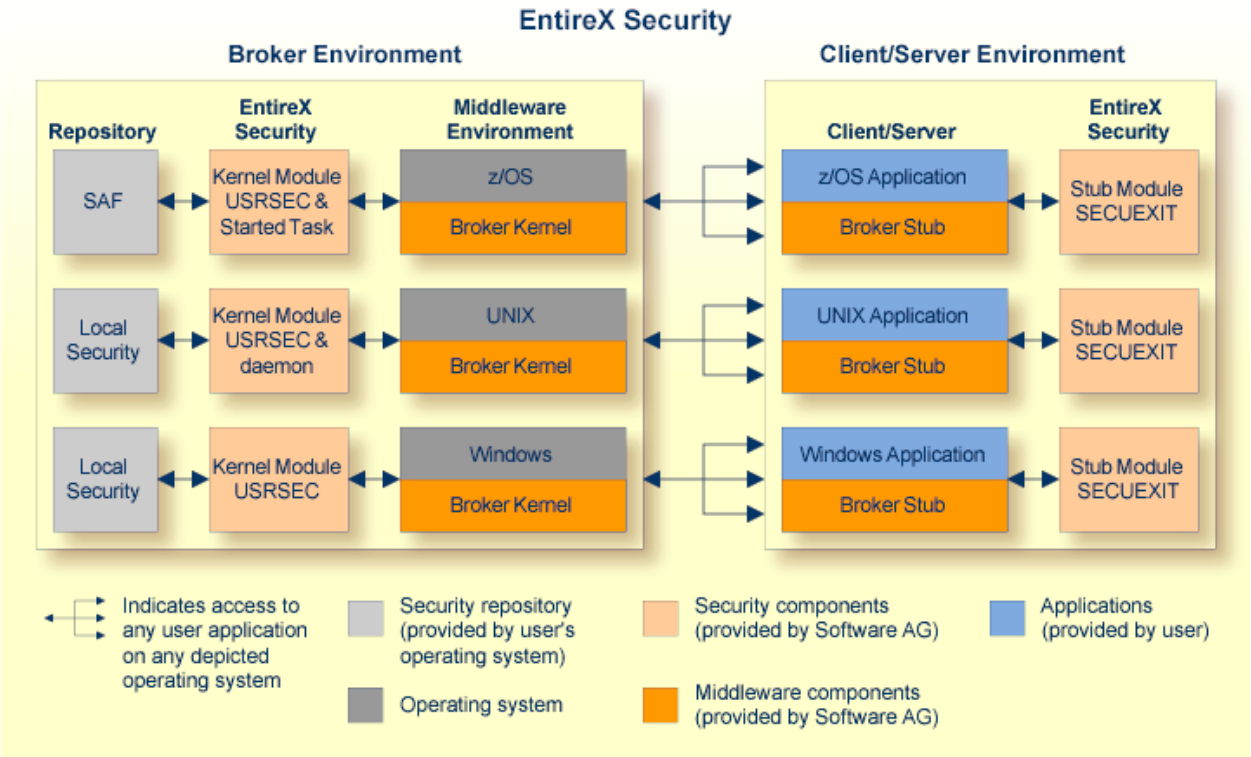
Security Choice	Criteria for Choosing a Security Solution
EntireX Security	Choose this option if you want to use the standard security functionality already provided with EntireX and your organization uses one of the following security repositories: ■ SAF-based security (e.g. RACF, CA ACF2, CA Top Secret, or LDAP repository) for Broker running under z/OS. See *Installing EntireX Security under z/OS under z/OS* in the z/OS installation documentation. ■ UNIX local security or LDAP repository for Broker running under UNIX. See *Setting up EntireX Security under UNIX* under *Post-installation Steps under UNIX*. ■ Windows local security or LDAP repository for Broker running under Windows. See *Post-installation Steps under Windows*. EntireX Security is fully supported (that is, object code only), and there are no user exits to write and debug. EntireX Security provides user authentication (logon/logoff security), user authorization (z/OS only) and application data encryption. This functionality is provided for EntireX Broker kernel running on all supported operating systems and for the corresponding stubs. In most installations EntireX Security operates without altering runtime applications.
Sample Security Exits for Broker Security	Choose this option only if your organization requires an alternative to standard SAF-based security on z/OS or local UNIX / Windows security on these platforms. Writing sample security exits is recommended only in exceptional processing situations - for example, if your organization wants to access its own user-written security system

Security Choice	Criteria for Choosing a Security Solution
	when operating EntireX Broker. Sample security exits are provided as skeleton programs only and must be completely customized before they can be deployed. See *Sample Security Exits for Broker Security*.

Overview of Security Configurations

EntireX Security: Standard Security Solution

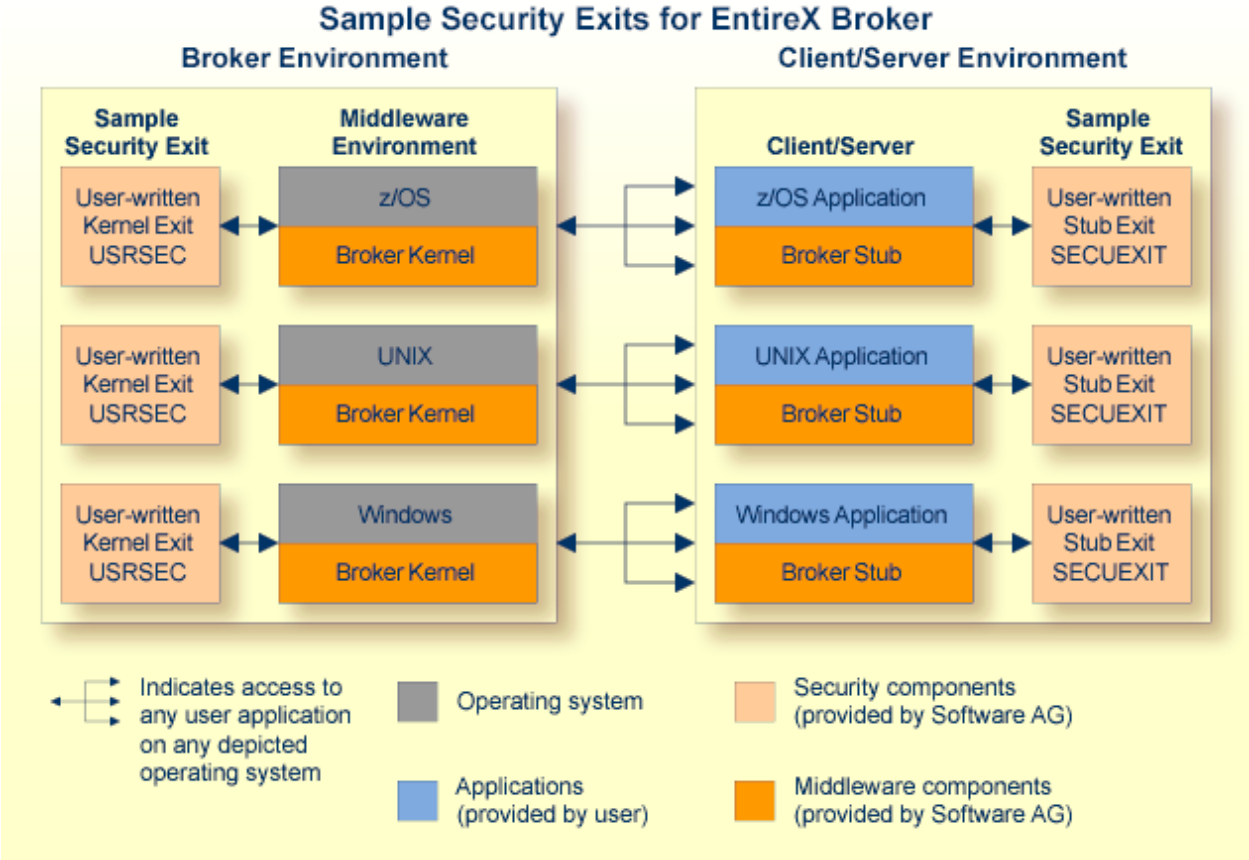
This diagram shows the locations where the broker kernel and broker stubs can be installed; it also shows the locations of the security components of the kernel and stubs.



See *Platform Coverage* in the EntireX Release Notes for where EntireX Security for broker kernel and stubs is supported.

Sample Security Exits for EntireX Broker

This diagram shows the locations where the broker kernel and broker stubs can be installed; it also shows the locations of the security components of the kernel and stubs.



See *Platform Coverage* in the EntireX Release Notes for where EntireX Security for broker kernel and stubs is supported.

Features of EntireX Security

EntireX Security provides comprehensive security for EntireX Broker:

- user authentication
- user authorization
- application-data encryption
- supplied in object code only

See also ***Overview of EntireX Security Features***.

User-written Sample Security Exits for EntireX Broker

Software AG intends security supplied by EntireX Broker to be only an alternative to EntireX Security, which is Software AG's standard security solution and shipped with EntireX. See ***Overview of EntireX Security***. Do not mix these two security solutions: do not use a stub secured with a sample exit against a kernel secured with EntireX Security or vice versa.

Most organizations that use Software AG's EntireX choose EntireX Security instead of sample security exits for EntireX Broker security. If your organization is deploying distributed computer systems encompassing mainframe, UNIX and Windows environments, you will use EntireX Security instead of sample security exits for EntireX Broker security.

See *Sample Security Exits for Broker Security*.

2 Overview of EntireX Security

EntireX Security is the standard security solution provided with EntireX. It provides centralized security for EntireX Broker under z/OS, UNIX and Windows. EntireX Security operates with your organization's security repository.

Introduction to EntireX Security

EntireX Security secures distributed application components running with EntireX Broker. EntireX Security software is installed at specific points where communication between client and server / publish and subscribe application components is protected, using definitions located in the security repository of your organization: e.g., SAF-based security (RACF, CA ACF2 or CA Top Secret) under z/OS, and for UNIX and Windows either the local security system of the machine or an LDAP repository.

The basic functionality of EntireX Security covers

- authentication of user
- authorization of client and server, publish and subscribe, and Command and Information Services
- encryption of application

See *Functionality of EntireX Security*.

Overview of EntireX Security Features

Comprehensive Security

EntireX Security provides comprehensive security for EntireX Broker:

- user authentication
- user authorization
- application-data encryption
- supplied in object code only

Protection of Application Systems

EntireX Security protects client and server and publish and subscribe application systems, and, in most installations, EntireX Security operates without altering runtime applications.

One User=One Definition

EntireX Security allows your organization to control the use of all applications, including distributed components, from a central point, enabling flexible control with a “one user = one definition” approach.

No User Exits to Write/Debug

There are no user exits to write and debug when using EntireX Security. Compare *Sample Security Exits for Broker Security*.

Standard Security Definitions

EntireX Security enables security definitions, based on class/name/service (client and server) or topic (publish and subscribe), to be credentialized within your SAF Security system. All definitions are managed using existing security procedures and software.

Protected Investment in SAF-based Security Repositories

Your investment in SAF-based security repositories is protected. This includes not only the security systems RACF, CA ACF2 and CA Top Secret, but also the infrastructure to administer security profiles.

Functionality of EntireX Security

This section covers the following topics

Authentication of User

Authentication verifies whether the identity specified by the user application is the actual identity. Authentication is performed for application components executing on different platforms against the security repository where the broker kernel resides. See ***EntireX Security: Standard Security Solution***. It is the responsibility of the application to supply the ACI user ID and password on the first command. See `USER-ID` and `PASSWORD` under *Broker ACI Fields*.

Note: There is an uppercase translation when the `USER-ID` field is propagated to the `CLIENT-UID` field under EntireX Security when the broker kernel is running under z/OS.

Authorization of Client and Server

Authorization determines whether client and server application components are allowed to execute with EntireX Broker. The class, server and service associated with the user's command form the basis for the check. Separate authorization checks are performed, depending on the role of the application as either client or server. The checks differentiate between the client's `SEND` command and a server's `REGISTER` command. Therefore your security administrator should allow only the level of access required for the user to operate in the intended role. The authorization checks are performed on the same platform as the broker kernel resides (see ***EntireX Security: Standard Security Solution***) regardless of location of the individual application components.

This authorization functionality is available only with EntireX Broker running under z/OS. Under UNIX and Windows, limited functionality is available through authorization rules. See also *Administering Authorization Rules using System Management Hub* in the UNIX and Windows administration documentation.

Authorization of Publish and Subscribe

Authorization determines whether publisher and subscriber application components are allowed to execute with EntireX Broker. The topic name associated with the user's command forms the basis for the check. Separate authorization checks are performed, depending on the role of the application as either publisher or subscriber. The checks differentiate between a publisher's `PUBLISH` command and a subscriber's `SUBSCRIBE` command. Therefore your security administrator should allow only the level of access required for the user to operate in the intended role. The authorization checks are performed on the same platform as the broker kernel resides. See ***EntireX Security: Standard Security Solution***.

This authorization functionality is available only with EntireX Broker running under z/OS. Under UNIX and Windows, limited functionality is available through authorization rules. See also *Administering Authorization Rules using System Management Hub* in the UNIX and Windows administration documentation.

Authorization for Command and Information Services

Authorization determines whether a user is permitted to issue commands to the EntireX Broker Command and Information Services. See *Broker Command and Information Services*. The following resource definitions, derived from the user's intended activities, form the basis for the check. The level of authorization needed for accessing these services is identical to that of a "client". These services are automatically started by broker kernel without performing a check for `REGISTER`:

Resource Definition	Using
`SAG.ETBCIS.CMD`	`ETBCMD`
`SAG.ETBCIS.INFO`	`ETBINFO` to retrieve general information.
`SAG.ETBCIS.SAGCCV5`	For RPC CIS command services.
`SAG.ETBCIS.SAGCIV5`	For RPC CIS information services.
`SAG.ETBCIS.SECURITY-CMD`	For security related requests: (1) reset user [ACEE]; (2) change security trace level.
`SAG.ETBCIS.USER-INFO`	`ETBINFO` to retrieve information specific to the user issuing the command

In addition, a separate authorization check is made when a user attempts to perform third party actions affecting other users:

- To shutdown a *service*, users must have the required authorization to register this class, server and service themselves.
- To shutdown a *server*, users must have the required authoriszation to register all the services registered by that server.

This authorization is required in addition to the requesting user's ability to use `SAG.ETBCIS.CMD` in general.

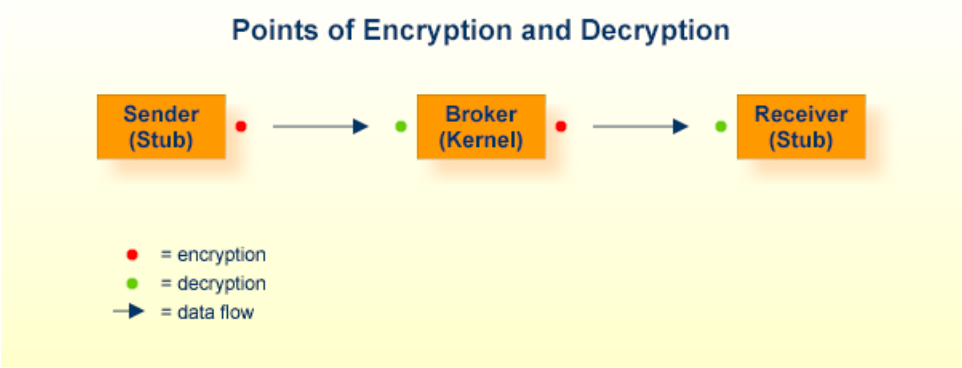
Similarly, Command and Information Services allows for third party subscription of users to a specified topic. In this case, a separate authorization check is made to ensure the issuing user is indeed authorized to subscribe to this same topic. This authorization is required in addition to the requesting user's ability to use `SAG.ETBCIS.CMD` in general.

This authorization functionality is available only with EntireX Broker running under z/OS. Under UNIX and Windows, limited functionality is available through authorization rules. See also *Administering Authorization Rules using System Management Hub* in the UNIX and Windows administration documentation.

Encryption of Application Data

In EntireX Security, a client application can create and encrypt a message before sending it via a broker to a server application or vice versa. Similarly, publisher applications can encrypt messages before communicating them via a broker to subscriber applications. The following message flow illustrates an example client application sending a message to a server application. In a reverse message flow - that is, server application to client application - the points of encryption and decryption are also reversed.

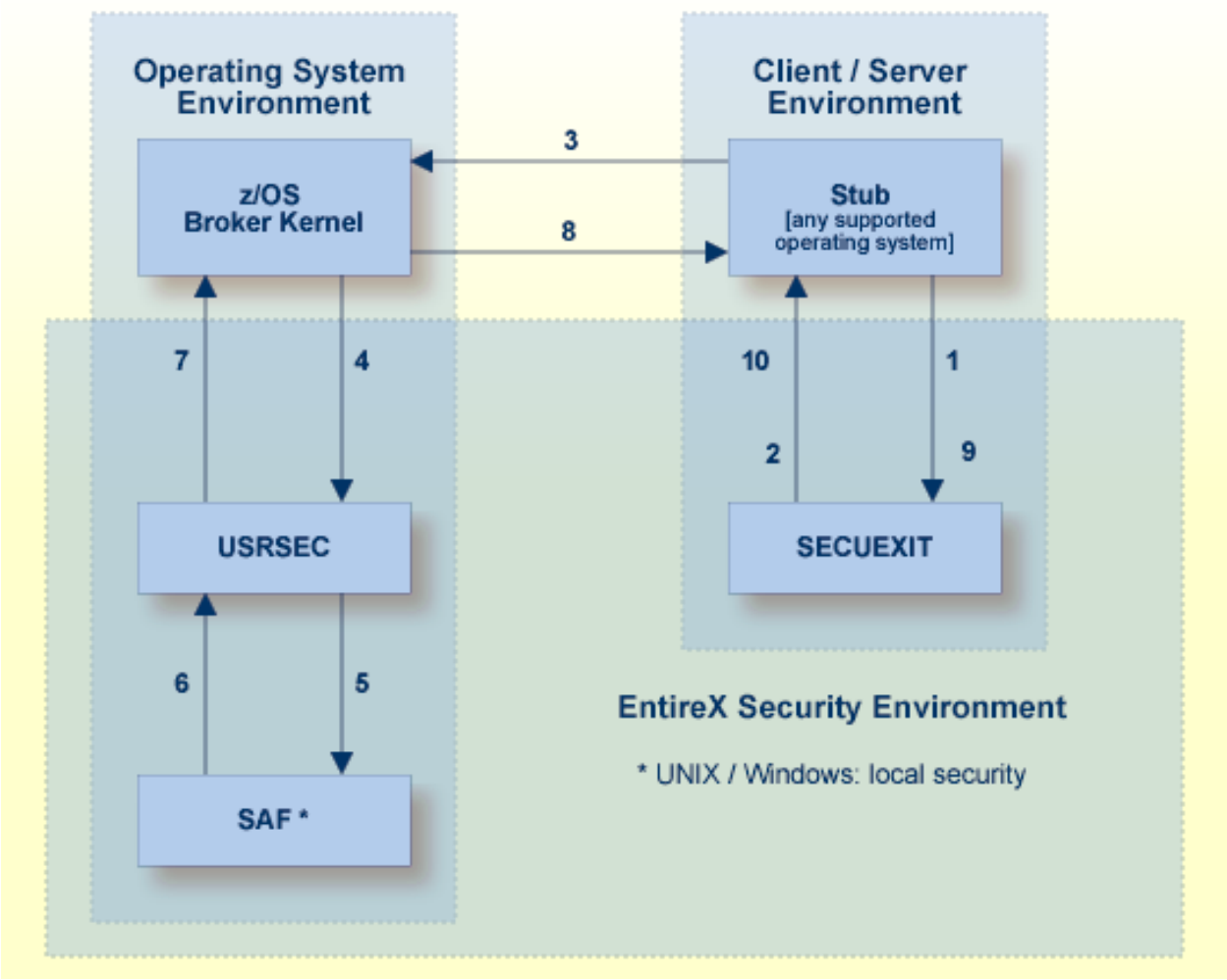
The setting of encryption by an application component activates encryption between that application and the broker kernel; it does not activate encryption between the broker kernel and the partner application component. However, the behavior of encryption from client via the broker to the server is controlled by the broker attribute `ENCRYPTION-LEVEL` and the control block field `ENCRYPTION-LEVEL`.



In the graphic, Sender refers to the message producer; Receiver refers to the message consumer.

Data Flow of EntireX Security (Client and Server)

The diagram shows the location of the security components of the kernel and stubs of EntireX Broker. Each step in the table below represents a specific step in the data flow sequence.This table describes the functionality of the security components of the kernel / stubs of broker: authorization; authentication; encryption/decryption.



Note: This diagram depicts the operation of the broker stub for Natural and 3GL. It is not intended to show the mechanism used by the Java ACI with regard to EntireX Security. See *Using EntireX Security with Java-based EntireX Applications* in the Java ACI documentation.

Description of Steps in Data Flow

1. Broker stub calls security module SECUEXIT, if present.
2. Security module SECUEXIT encrypts the password and optionally the application data, based on the value assigned to the ENCRYPTION-LEVEL field of the broker control block, or through configuration options for: *z/OS* | UNIX | Windows.
3. Broker stub communicates the call to the broker kernel.
4. Broker kernel calls security module USRSEC, which provides the following functionality, based on the configuration options applicable to EntireX Security (see *Configuration Options for Broker*):
   - user **authentication** of publish and subscribe application components;
   - **authorization** checking required for subscriber to issue SUBSCRIBE command for specified topic;
   - **authorization** checking required for publisher to send publication;
   - **encryption** of application data;
   - **decryption** of application data;
   - Re-authentication if a user acquires a new physical user ID.
   - Re-authentication if the value of a user's ACI security token changes.

   All functionality is available on z/OS only.
5. Security module USRSEC references local security system where the broker is located.
   - z/OS

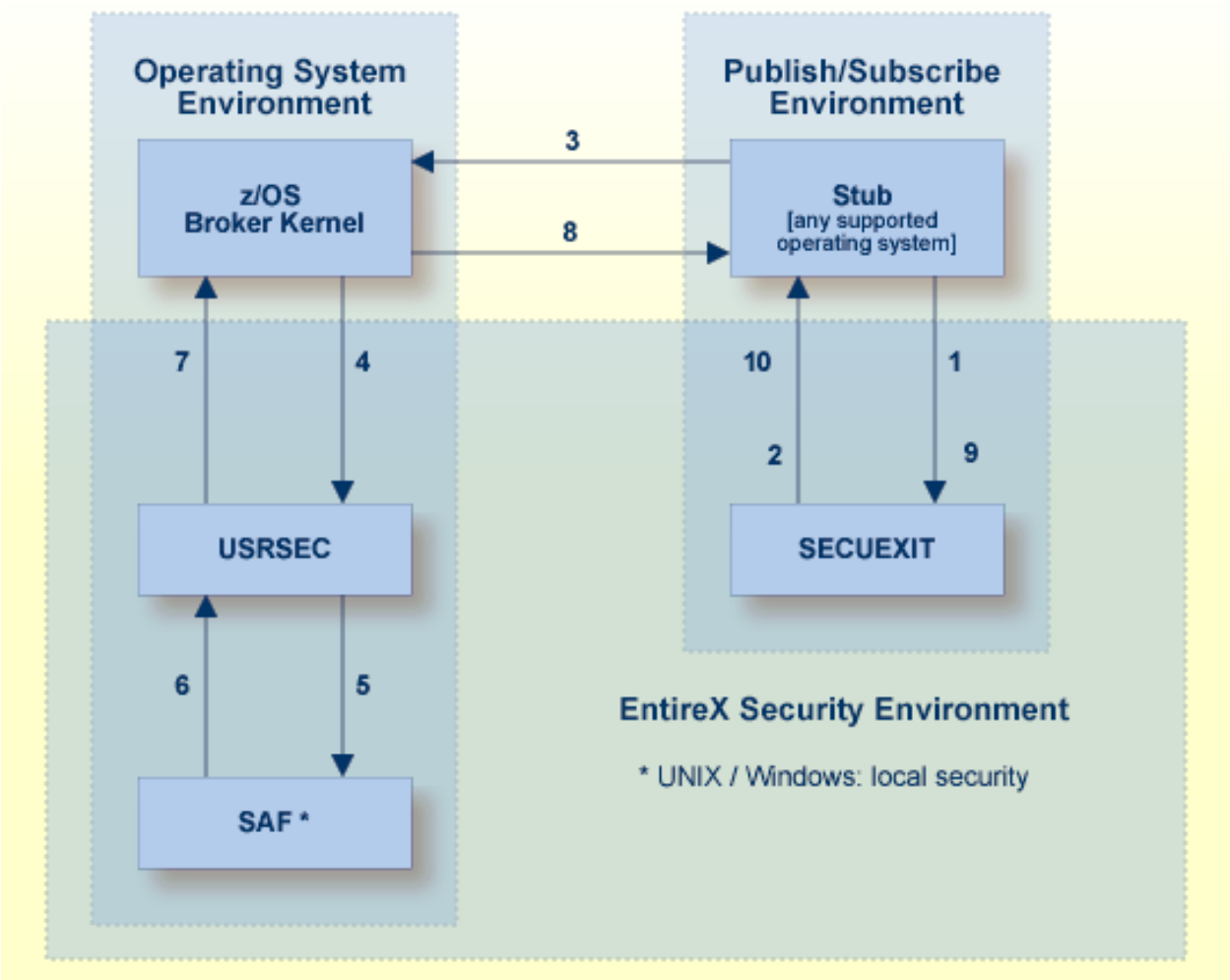
     Security module USRSEC calls SAF (RACF, CA ACF2 or CA Top Secret).
   - UNIX

     Security module USRSEC calls the UNIX security system or LDAP.
   - Windows

     Security module USRSEC calls the Windows security system or LDAP.
6. The result of the security check is communicated back to security module USRSEC.
7. Security module USRSEC passes call to Broker kernel.
8. Broker kernel communicates the call to Broker stub of the partner application.
9. The Broker stub calls SECUEXIT. SECUEXIT performs decryption, where this is necessary to receive the data.
10. Security module SECUEXIT returns call to Broker stub.

Data Flow of EntireX Security (Publish and Subscribe)

The diagram shows the location of the security components of the kernel and stubs of Broker. Each step in the list below represents a specific step in the data flow sequence. The steps describe the functionality of the security components of the kernel / stubs of the broker: authorization; authentication; encryption/decryption.



Description of Steps in Data Flow

1. Broker stub calls security module `SECUEXIT`, if present.
2. Security module `SECUEXIT` encrypts the password and optionally the application data, based on the value assigned to the `ENCRYPTION-LEVEL` field of the broker control block, or through configuration options for: ***z/OS*** | UNIX | Windows.
3. Broker stub communicates the call to the broker kernel.
4. Broker kernel calls security module `USRSEC`, which provides the following functionality, based on the configuration options applicable to EntireX Security. See ***Configuration Options for Broker***.
   - user **authentication** of client and server application components;
   - **authorization** checking required for server to register a service;
   - **authorization** checking required for client to send request;
   - **encryption** of application data;
   - **decryption** of application data;
   - Re-authentication if a user acquires a new physical user ID.
   - Re-authentication if the value of a user's ACI security token changes.

   All functionality is available on z/OS only.
5. Security module `USRSEC` references local security system where the broker is located.
   - z/OS

     Security module `USRSEC` calls SAF (RACF, CA ACF2 or CA Top Secret).
   - UNIX

     Security module `USRSEC` calls the UNIX security system or LDAP.
   - Windows

     Security module `USRSEC` calls the Windows security system or LDAP.
6. The result of the security check is communicated back to security module `USRSEC`.
7. Security module `USRSEC` passes call to Broker kernel.
8. Broker kernel communicates the call to broker stub of the partner application.
9. The Broker stub calls `SECUEXIT`. `SECUEXIT` performs decryption, where this is necessary to receive the data.
10. Security module `SECUEXIT` returns call to Broker stub.

Glossary of Terms

See also *EntireX Glossary*.

Authentication

Authentication verifies whether the identity specified by the user ID in the ACI control block is the actual identity. Authentication is performed by checking the user's ID and password against a security system, except where Trusted user ID automatically acquires the identity of the logged-on user or batch job, obviating the requirement for a password in the ACI control block. See ***Trusted User ID***. Trusted user ID is applicable only where the application component and the broker kernel reside under z/OS.

Authentication is not performed with every call. It is performed when a user is first presented to the kernel of EntireX Broker. The broker kernel recognizes the identity of the user on subsequent occasions by combination of user ID and physical user ID (or user ID and TOKEN where supplied). Broker kernel also verifies the correctness of the ACI security token on all subsequent commands and, if this is not as expected, the application must provide the correct user ID and PASSWORD again (unless configured otherwise).

An application identifying itself by combination of user ID and TOKEN can change its physical user ID without needing to provide the user ID and PASSWORD again provided it maintains the value of ACI security token in the broker control block. This functionality is recommended for multithreading applications or applications executing within a Web server. Caution should be exercised to ensure the user ID and TOKEN combination is unique.

Authorization

Authorization is performed when:

- a client issues a request to a service in the case of the first `SEND` command in a conversation, or of each SEND command if `CONV-ID=NONE`;
- a server registers a service to the broker;
- a publisher communicates a publication via broker for a specified topic;
- a subscriber issues the subscribe command to broker kernel for a specified topic;
- an application connects to broker through TCP/IP, an optional authorization check is performed based on the address.

Full authorization functionality is available only under z/OS.

Broker Kernel

It is the location of the broker kernel that determines the point at which the authentication and authorization checks are performed. ***Authentication*** and ***Authorization*** are performed in the kernel. ***Encryption / Decryption*** is performed in the kernel (as well as in the stub).

See *List of Components per Platform* under *Platform Coverage* in the EntireX Release Notes for where EntireX Broker kernel is supported.

Broker Stub

In EntireX Broker, a module that implements the ACI (Advanced Communication Interface) is commonly referred to as “broker stub” or simply “stub”. Stubs are installed on the client side or server side.

See *List of Components per Platform* under *Platform Coverage* in the EntireX Release Notes for where broker stubs are supported.

Encryption / Decryption

Encryption is the process by which the information or data being sent back and forth between two computers (including the password submitted when logging on) is “encoded”, shielding it from view by unauthorized persons. With EntireX Security, the algorithms for encryption/decryption are present in the broker stubs and also in the kernel of broker.

See ***Encryption of Application Data***.

3 EntireX Security under z/OS

This chapter introduces EntireX Security under z/OS through overviews of the functionality and components of EntireX Security. The location where Broker Kernel is installed determines the functionality made available for EntireX Security.

Note: Installation of the security software is described under *Installing EntireX Security under z/OS under z/OS* in the z/OS installation documentation.

Introduction

Functionality of EntireX Security

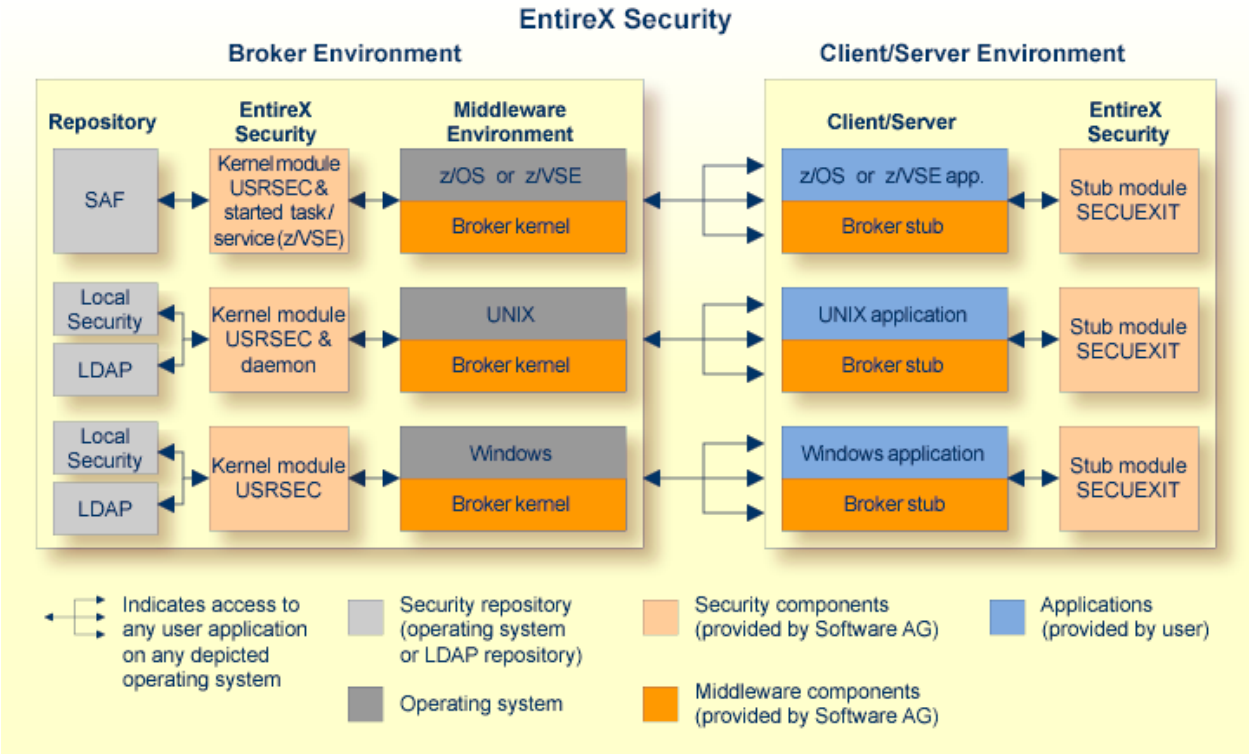
This table lists the security functionality available with EntireX Security running Broker Kernel under the respective operating system. See also ***Configuration Options for Broker***.

Security Functionality	z/OS	UNIX	Windows	BS2000/OSD	z/VSE	Comment
Authentication of user	Yes	Yes	Yes	Yes	Yes	Verify User ID password.
User password change	Yes	No	No	No	No	
LDAP authentication	No	Yes	Yes	No	No	Authenticate using LDAP repository.
Trusted user ID	Yes	No	No	No	No	Trusted computer base, avoiding plain text password.
Verified client user ID	Yes	No	No	Yes	Yes	Provide verified identity of client to server.
Authorization of client request	Yes	No	No	No	No	
Authorization of server register	Yes	No	No	No	No	
Authorize IP connection	Yes	No	No	No	No	
Authorization rules	No	Yes	Yes	No	No	Check rules stored in an LDAP repository. These rules are maintained using an agent of System Management Hub, and are independent of the LDAP authentication mechanism. **Note:** These rules can be stored either in the same or a different LDAP repository.
Encryption of application data	Yes	Yes	Yes	No	Yes	RC4-compatible algorithm.
Guaranteed encryption	Yes	Yes	Yes	No	Yes	Allows administrator to require encryption for specific services.

Security Functionality	z/OS	UNIX	Windows	BS2000/OSD	z/VSE	Comment
SSL	Yes	Yes	Yes	No	No	Industry standard encryption mechanism.

EntireX Security Components

This diagram depicts the location where the Broker kernel must be installed and where the Broker stubs can be installed. It also depicts the location of the security components of the kernel and stubs of Broker.



EntireX Security for EntireX Broker

- Introduction

Introduction

EntireX Broker acts as an agent to make the creation and operation of client/server applications simpler and more effective. Any number of server applications can be built for use by any number of clients. EntireX Security allows you to protect your server applications and clients independently.

Clients and servers are authenticated by user ID and password on their first contact with the system. Authorization is sought for specific server applications before a client is allowed access. This enables control of your distributed application systems - at both the application level and the client level.

Authorization is also required for server applications to register services. Unauthorized servers can be intercepted when trying to register. Facilities exist to establish connection authorization when client and server applications first establish contact with the Broker.

Considerations for Mainframe Application Components

Application components running in a mainframe environment that communicate using EntireX Broker interact with EntireX Security in the following ways:

- No password is required for applications executing under mainframe where the trusted user ID option is implemented. This is true for both client and server application components. EntireX Security automatically acquires the logged-on user ID. Utilizing the “trusted” user ID avoids having to supply the password again. It also requires customers to configure security for their mainframe environment(s), for example ensuring that the CICS system is protected by RACF.
- Applications can override the trusted user ID by supplying a valid user ID/password combination in the ACI control block. This causes EntireX Security to ignore the trusted user ID in favor of the supplied credentials. Applications must therefore ensure that they do not assign an incorrect user ID or spurious password to the ACI control block, where “trusted” user ID is implemented. The `CLIENT-ID` as conveyed in the ACI to the server component of the application now represents the client's verified user ID, derived either from valid user ID/password credentials or from trusted user ID itself.

Configuration Options for Broker

This section describes the parameters for configuring EntireX Security under z/OS. You may either accept or modify the default settings which are specified in the Broker attribute file `DEFAULTS=SECURITY`. Always check installation options against the corresponding resource profile. See ***Resource Profiles in EntireX Security***.

Authentication

Authentication is mandatory and performed for both client and server applications based on user ID and password. First contact with the Broker results in the host security system being referenced. If authentication fails, access is denied and the application is informed with a suitable error message.

It is the responsibility of both client and server applications to supply a valid user ID and password when calling the Broker. The user ID must be supplied with all commands. The password is required only for the first command and should not be supplied subsequently, except when executing multiple instances of the same application.

Authentication expires after a period of non-activity after which it must be repeated. User ID and password must be resupplied before further access is possible. The time limits `CLIENT-NONACT` and `SERVER-NONACT` determine these timeout periods and are defined in the Broker attribute file.

Note: Applications must not assign a password to the ACI control block if they intend to use trusted user ID. This applies to all applications, including EntireX RPC Server.

Trusted User ID

This allows z/OS-based applications to communicate securely without having to supply user ID and password. Activate this option by specifying the following parameter in job member WAL&*vrs*..JOBS (SAFI010):

TRUSTED-USERID=N	Require user ID/password for z/OS application components.
TRUSTED-USERID=Y	Leverage trusted user ID mechanism for z/OS applications.

Request Authorization

Clients request distributed processing using the SEND command, indicating the class, name and service to be invoked. The Broker transmits the request to the server only if the client has access to the relevant resource profile. Similarly, servers are allowed to REGISTER services only if the server has access to the resource profile. The default profile make-up comprises class.name.service of the service. The following system parameters will modify the resource profile if required:

INCLUDE-CLASS ={YES,NO}	Include Class in resource check.
INCLUDE-NAME ={YES,NO}	Include Name in resource check.
INCLUDE-SERVICE ={YES,NO}	Include Service in resource check.

Note: At least one option must be "YES" for authorization to be performed.

For example: if INCLUDE-CLASS=YES, INCLUDE-NAME=YES and INCLUDE-SERVICE=YES, the structure of the resource profile checked is:

```
<class_name>.<server_name>.<service_name>
```

But with INCLUDE-CLASS=YES, INCLUDE-NAME=NO and INCLUDE-SERVICE=YES, the profile would look like:

```
<class_name>.<service_name>
```

Alternatively, with INCLUDE-CLASS=NO, INCLUDE-NAME=YES and INCLUDE-SERVICE=YES, the resource profile to be checked is:

```
<server_name>.<service_name>
```

Clients require READ access to obtain processing from a server application and severs require CONTROL access in order to REGISTER successfully, otherwise the command is rejected.

Authorization checks are also performed for publish-and-subscribe procesing. Subscribers are allowed to SUBSCRIBE only if they have READ access to the resource profile representing the TOPIC

to which they are subscribing. A publisher requires `CONTROL` access in order to successfully `PULBLISH` to a topic.

Discrete or generic resource profiles can be defined for this purpose.

Note: If you set `SECURITY-NODE`, the Broker ID is used as a prefix for all authorization checks.

Request Authorization for Command and Info Services

If you are using one of the following RPC servers with a broker protected by, for example, RACF or CA Top Secret, at least READ access is required to resources `SAG.ETBCIS.INFO` and `SAG.ETBCIS.CMD`.

- CICS ECI RPC Server
- IMS Connect RPC Server
- Micro Focus RPC Server
- RPC-ACI Bridge
- WebSphere MQ RPC Server
- Java RPC Server
- XML/SOAP RPC Server

Ignore Security Token

A security token is generated by EntireX. It is the responsibility of the application to clear the security token before making the first call and thereafter to maintain the contents of the field for the duration of communication for the user.

If validation of security token is not required - for example, where applications or packages do not maintain the security token in the ACI control block - this option may be switched off. The default setting is "NO" (do not ignore Security Token).

`IGNORE-STOKEN={YES,NO}` Do not ignore Security Token.

Guaranteed Encryption / Decryption Mechanism

EntireX Security ensures message encryption consistency regardless of any configuration errors. This means, if the relevant assembly parameters or environment variables are incorrectly or inconsistently specified, the integrity of the Broker message is honored. This feature requires upgrade of all EntireX Security Broker stub and kernel components in all places.

See Broker attribute `ENCRYPTION-LEVEL` and control block field `ENCRYPTION-LEVEL`.

Authorize IP Connection

Communication between distributed application components and the Broker via TCP/IP can be subject to an authorization check at connection time. Define the following system parameter if this option is required:

`CHECK-IP-ADDRESS={YES,NO}` Authorize IP connection.

Access to Undefined Resources

The normal mode of operation is to prevent access to resources not defined to the security system. Profiles representing services are added to the security repository with either a default access or by granting access to specific users and groups. Access to undefined resources can be permitted using the following system parameter:

`UNIVERSAL={YES,NO}` Allow access to undefined resources.

Note: This option does not permit access to resources defined with universal access “none”. See also note on defining resources to ACF.

Alternate Resource Class/Type Names

By default, the resource class/type `NBKSAG` is used when performing authorization checks. The name of an alternate resource class can be specified using the following system parameter:

`SAF-CLASS=NBKSAG` Resource class for Broker.

Length of Resource Class/Type Profile

By default, the maximum length of the resource class/type profiles is 80 characters when performing authorization checks. Longer resource profiles can be checked by increasing the maximum resource profile length as follows. Make sure you also increase the maximum profile length in the SAF Class/Type Descriptor table in z/OS.

`MAX-SAF-PROF-LENGTH=<nn>` Max resource profile length.

Password to Uppercase

To cater for situations where a site is in transition from uppercase to mixed case passwords setting this parameter can convert all passwords to uppercase. It is not recommended you use this option by default.

`PASSWORD-TO-UPPER-CASE={NO,YES}` Convert password to uppercase.

Security Level

By default, EntireX Security furnishes authentication with optional encryption of send/receive buffers. The following parameter can be used to modify the functionality of EntireX Security:

`SECURITY-LEVEL=ENCRYPTION`	No authentication or authorization checks performed. The only functionality available in this mode is message privacy.
`SECURITY-LEVEL=AUTHENTICATION`	User authentication is performed but without any resource authorization (the normal default operation).
`SECURITY-LEVEL=AUTHORIZATION`	User authentication and resource authorization are both applied.

Caution: In version 8.0, the default value for this parameter was "AUTHORIZATION"

Verified Client User ID

It is often important for server applications to know the identity of the client issuing the request. For this reason, the Broker kernel communicates the ACI field `CLIENT-UID` to the server application during the `RECEIVE` function. EntireX Security guarantees that the `CLIENT-UID` has been formally authenticated. EntireX Security automatically substitutes the value from trusted user ID where this is applicable.

`PROPAGATE-TRUSTED-USERID=YES`	Set ACI field `CLIENT-UID` to the user ID of the client. This will be authenticated by EntireX Security and may be obtained according to the trusted user ID mechanism where installed.

PROPAGATE-TRUSTED-USERID=NO	Do not set this value unless explicitly instructed to do so by Software AG support.

Security Node

This parameter can be used to specify a prefix which is added to all authorization checks, hence enabling broker kernels in different environments to perform authorization checks on different sets of resource profiles. For example, it is often important to distinguish among production, test, and development environments when performing authorization checks. The following settings are available:

SECURITY-NODE=YES	This causes the Broker ID - i.e., ETB113 - to be used as a prefix for all authorization checks.
SECURITY-NODE=<*node_name*>	This will utilize the string “node_name” (maximum 8 characters) as the prefix for all authorization checks.
SECURITY-NODE=NO	This causes the actual text (max 8 characters) to be prefixed onto all authorization checks..

Client RPC Authorization

For services supporting Natural RPC or other applications that know RPC, you can optionally perform authorization checks on the client making the RPC request by defining the “per service” attribute CLIENT-RPC-AUTHORIZATION=YES in the Broker attribute file. Setting this parameter to “YES” will cause the RPC library and program names to be appended to the profile associated with the authorization check. The resource profile would then appear as follows:

```
Class.server.service.rpc-library.rpc-program
```

Note: Natural Security performs its resource authorization checks as follows:
<prefix-character>.rpc-library.rpc-program
To allow conformity with Natural Security, the CLIENT-RPC-AUTHORIZATION parameter can optionally be defined with a prefix character as follows: CLIENT-RPC-AUTHORIZATION=(YES,<*prefix-character*>).

Considerations for Mainframe Natural Application Components

Application components running in a mainframe Natural environment which communicate using EntireX Broker interact with EntireX Security in the following ways:

- No password is required for applications executing under mainframe Natural where the trusted user ID option is implemented. This is true for both client and server application components. EntireX Security automatically acquires the logged-on user ID. Utilizing the trusted user ID avoids having to supply the password again. It also requires the customers to configure security for their mainframe environment(s), for example,ensuring that the CICS system is protected by RACF.
- Applications can override the trusted user ID by supplying a valid user ID/password combination in the ACI control block. This causes EntireX Security to ignore the trusted user ID in favor of the supplied credentials. Applications must therefore ensure that they do not assign an incorrect user ID or spurious password to the ACI control block, where trusted user ID is implemented. The `CLIENT-ID` as conveyed in the ACI to the server component of the application now represents the client's verified user ID, derived either from valid user ID/password credentials or from trusted user ID itself.

Resource Profiles in EntireX Security

This section describes the definitions required in the SAF repository according to the underlying security system used (RACF, CA ACF2, CA Top Secret).

Introduction

EntireX Security enables the secure deployment of EntireX Broker. This involves defining the resource profiles in the SAF repository to protect all distributed and mainframe application components. This philosophy is consistent with maintaining a single user ID and password.

Each SAF security system provides the facilities required for maintaining resource profiles.

RACF enables the grouping of similar resource profiles into a resource Class. CA ACF2 provides resource types which give equivalent functionality.

The name of the SAF class/type used to hold the EntireX-related resource profiles is specified with the Security-specific attribute `SAF-CLASS`. Default is `NBKSAG`.

The default length of the resource profile is 80 bytes, and this can be increased if necessary. See `MAX-SAF-PROF-LENGTH`. If you increase the maximum profile length, you must also increase the maximum profile length defined in the RACF class descriptor table.

Format of Resource Profiles

This section describes the format of various resource profiles. Note that the specific contents of resource files themselves will vary, depending upon the configuration options specified in the *Security-specific Attributes* (`DEFAULTS=SECURITY`) under *Broker Attributes* in the administration documentation.

EntireX Broker

- **Client Server Example**
  Resource profiles protecting Broker client and server applications normally comprise Broker class, name and service. It is possible to omit any of these components from the resource profile. See also ***Request Authorization***. The following resource profile shows an example service:

```
ETB.POLICY.QUOTE1
```

Client applications must execute with a user ID that has READ access to allow them to send to the given service. Registration of services is also secured. Server applications require CONTROL access to register a service with the Broker.

- **Publis and Subscribe Example**
  Resource profiles protecting Broker publish and subscribe applications are always defined in terms of the 96 character topic name. The following resource profile would be used to protect a topic used for publish and subscribe:

```
NYSE
```

Subscriber applications must execute with a user ID that has READ access to this resource to allow them to issue the subscribe command for this topic. Publisher applications require CONTROL access in order to send publications to the topic.

EntireX Broker TCP/IP Address Verification

If optional TCP/IP address checking is required at authentication time, the relevant resource profiles must be defined in the security system. Users will require READ access in order to connect, using TCP/IP, from a particular address. A typical TCP/IP address would be entered in the security system as follows:

```
247.72.46.239
```

Note: You can perform TCP/IP address checking independently of user authentication and authorization, by setting `CHECK-IP-ADDRESS=YES` and `SECURITY-LEVEL=ENCRYPTION`, i.e. not authentication or authorization. This results in an authorization check for the IP address for the user ID under which EntireX Broker itself executes.

Command and Information Services

Access to Command and Information Services is controlled by permitting, or denying, access to Software AG supplied services which implement Command and Information Services.

For a complete list of profiles representing these services, see ***Authorization for Command and Information Services***.

For more information see *Security with Command and Information Services* under *Writing Applications: Command and Information Services* in the ACI documentation.

Resource Definitions

This section describes the definitions required in the various supported security systems in order to enable Security for EntireX resources. These definitions are described in the following subsections:

- Defining Resources to RACF
- Defining Resources to CA Top Secret
- Defining Resources to CA ACF2

Note: Define resources using uppercase characters only.

Defining Resources to RACF

This section defines how the EntireX resources are defined to RACF. For exact details of the procedures to be followed for the installed RACF version, consult the relevant IBM manuals.

Overview of tasks:

- Add classes to class descriptor table
- Update z/OS router table
- Activate new classes
- Assign user ID for the Broker started task, if you have not done so already
- Permit user access to resource profiles
- Optimize the performance of RAC authorization checks

To add classes to class descriptor table

1 Add the resource classes to the RACF class descriptor table. Refer to the IBM SPL RACF manual.

For an example, see IBM `SYS1.SAMPLIB`, member `RACINSTL`.

2 You must allocate a class descriptor length for class NBKSAG of 80 bytes in order to prevent the possibility of a system 282 abend, which could occur if the length of your resource (class/server/service) exceeds the length known to RACF. The maximum length allowed by EntireX is 80 bytes, so allow 80 bytes in the RACF class descriptor table.

3 Define the classes to enable discrete and generic profile use.

4 Check further attributes controlling the level of RACF messages generated when performing `RACROUTE` calls, as well as the required level of SMF recording. Sample definitions are provided in source member `RACFCLSX`.

To update z/OS router table

- Update the z/OS router table as described in the IBM SPL RACF manual. For an example, see the IBM `SYS1.SAMPLIB`, member `RACINSTL`, section `RFTABLE`.

To activate new classes

- Activate new resource classes with `SETROPTS` (see *IBM RACF Command Language Reference manual*). For an example, activate class `NBKSAG`:

```
SETROPTS CLASSACT(NBKSAG)
SETROPTS GENCMD(NBKSAG)
SETROPTS GENERIC(NBKSAG)
```

To assign user ID for the Broker started task

- The EntireX Security functions are performed within the address space of EntireX Broker. Assign a user ID to the Broker started task with the relevant RACF authorizations, including the ability to perform `RACROUTE, TYPE=EXTRACT`, `TYPE=AUTH` and `TYPE=VERIFY` calls on profiles belonging to the defined classes.

To permit user access to resource profiles

- After adding profiles to protect the different resources, permits users the required level of access, using the relevant RACF commands. The following example adds resource profile `ETB.POLICY.QUOTE1` and grants read access to user ID USER2 and control access to USER3. USER2 represents a client and requires read access to execute while USER3 represents a server component which needs control access to register:

```
RDEFINE NBKSAG ETB.POLICY.QUOTE1 UACC(NONE)
PERMIT ETB.POLICY.QUOTE1 CLASS(NBKSAG) ACCESS(READ) ID(USER2)
PERMIT ETB.POLICY.QUOTE1 CLASS(NBKSAG) ACCESS(CONTROL) ID(USER3)
```

- If you utilize authorization checks based upon TCP/IP address (TCP transport only) then define these resource definitions (`RDEFINE`) as follows and `PERMIT` the appropriate user read access as shown:

```
RDEFINE NBKSAG 247.72.46.239 UACC(NONE)
PERMIT 247.72.46.239 CLASS(NBKSAG) ACCESS(READ) ID(USER42)
```

To optimize the performance of RACF authorization checks

- Use `SETROPTS RACLIST(NBKSAG)` to cache in memory the RACF general resource profiles belonging to class `NBKSAG`. If you use a RACF resource class other than `NBKSAG`, make sure this RACF general resource class is cached in memory.

Defining Resources to CA Top Secret

This section defines how the EntireX classes are defined to CA Top Secret. For exact details of the procedures to be followed for the installed version of CA Top Secret, consult the relevant CA Top Secret manual.

Overview of tasks:

- Add CA Top Secret Facility
- Assign user ID for the Broker started task, if you have not done so already
- Add procedure name for the started task
- Add resource type to resource definition table
- Assign ownership of resources
- Permit defined resources to users

To add CA Top Secret Facility

- CA Top Secret enables a set of authorization checks to be made against a certain facility. For example, this can be used to secure the development environment `SAGDEV` separately from the production environment `SAGPROD`. Alternatively, a default facility of batch can be used.

  To add additional facilities, use the following commands:

  ```
  AUTHINIT,MULTIUSER,NONPWR,PGM=ETBNUC,NOABEND
  ```

To assign a user ID for the Broker started task

- Add one user ID for each instance of the Broker started task.

  If required, different facilities can be assigned to development and production started tasks.

  The designated facility is assigned to the started task user ID:

```
TSS CRE(user-id) DEPT(dept) MASTFAC(fac)
```

To add a procedure name for the Broker started task

- The procedure name under which the Broker started task executes must be defined to CA Top Secret.

```
TSS ADD(STC) PROC(proc) ACID(user-id)
```

To add resource type to resource definition table

- Add the resource types to the CA Top Secret resource definition table (RDT). Resource definitions relating to EntireX are kept in resource type `NBKSAG`. Refer to the *CA Top Secret Reference Guide* for a detailed explanation of the following commands and arguments:

```
TSS ADD(RDT) RESCLASS(NBKSAG)
RESCODE(HEXCODE)
ATTR(LONG)
ACLST(NONE,READ,CONTROL)
DEFACC(NONE)
```

To assign ownership of resources

- Assign ownership to a particular resource as shown in the following example. This must be done before permitting access to defined resource profiles:

```
TSS ADD(user1) NBKSAG(etb.policy.quote1)
```

This makes user *user1* the owner of the Broker service *etb.policy.quote1*.

Similarly, to add ownership to profiles used to control access based on TCP/IP address (TCP communications only) follow the steps below. This makes *user4* the owner of this resource profile:

```
TSS ADD(user4) NBKSAG(247.72.46.239)
```

To permit defined resource to users

- Permit access to a resource profile as in the following example. In the example, user *user2* is permitted read access to the Broker service *etb.policy.quote1*. This enables the user to execute as a client and issue requests to this Broker service:

```
TSS PER(user2) NBKSAG(etb.policy.quote1) FAC(fac) ACCESS(READ)
```

Similarly, to permit access to profiles used to control access based on TCP/IP address, use the `PER` command as shown:

```
TSS PER(user42) NBKSAG(247.72.46.239) FAC(fac) ACCESS(READ)
```

Defining Resources to CA ACF2

See also your CA ACF2 documentation.

Note: CA ACF2 provides insufficient return codes to determine whether a resource profile does not exist or simply the user does not have access to it. Therefore, if access is denied by CA ACF2, EntireX Security will always report "Access denied resource not allowed" in the error message.

To define resources to CA ACF2

1 The Broker or Broker Services started task executes as a normal started task in z/OS. Define the user ID of started task to CA ACF2 with the following attributes:

```
MUSASS, STC
```

2 Insert SAFDEF records as follows:

```
SAFDEF.EXS1
FUNCRET(4) FUNCRSN(0) ID(ENTIREX) MODE(GLOBAL)
RACROUTE(REQUEST=VERIFY SUBSYS=ETBNUC REQSTOR=-)
RETCODE(4)

SAFDEF.EXS2
FUNCRET(4) FUNCRSN(0) ID(ENTIREX) MODE(GLOBAL)
RACROUTE(REQUEST=AUTH SUBSYS=ETBNUC REQSTOR=-)
RETCODE(4)

SAFDEF.EXS3
FUNCRET(4) FUNCRSN(0) ID(ENTIREX) MODE(GLOBAL)
RACROUTE(REQUEST=EXTRACT SUBSYS=ETBNUC REQSTOR=-)
RETCODE(4)
```

3 For the general resource class name used by EntireX Security, define a 3-character CA ACF2 resource type code by inserting a `CLASMAP` record as follows:

```
CLASMAP
ENTITYLN(0) MUSID() RESOURCE(NBKSAG) RSRCTYPE(NBK)
```

4 Define the required security profiles to CA ACF2 using the new type code.

The following example shows the addition of a Broker service *etb.policy.quote1*, allowing read access only for user ID *user2*:

```
$KEY(ETB) TYPE(NBK)
policy.quote1 UID(user2) SERVICE(READ)   ALLOW
policy.quote1 UID(-)                     PREVENT
```

A service level of DELETE is required for a service to register (this is functionally the same as CONTROL access in RACF).

The following examle secures the TCP/IP connection for checking at authentication time for the TCP/IP address of 247.72.46.239 granting access to all users, except U402451.

```
$KEY(247) TYPE(NBK)
72.46.239 UID(user42)  SERVICE(READ) ALLOW
72.46.239 UID(u402451) SERVICE(READ) ALLOW
```

4 EntireX Security under UNIX

This chapter introduces EntireX Security under UNIX through overviews of the functionality and components of EntireX Security. The location where Broker Kernel is installed determines the functionality made available for EntireX Security.

Note: Setting up EntireX Security is described under *Setting up EntireX Security under UNIX* under *Post-installation Steps under UNIX*.

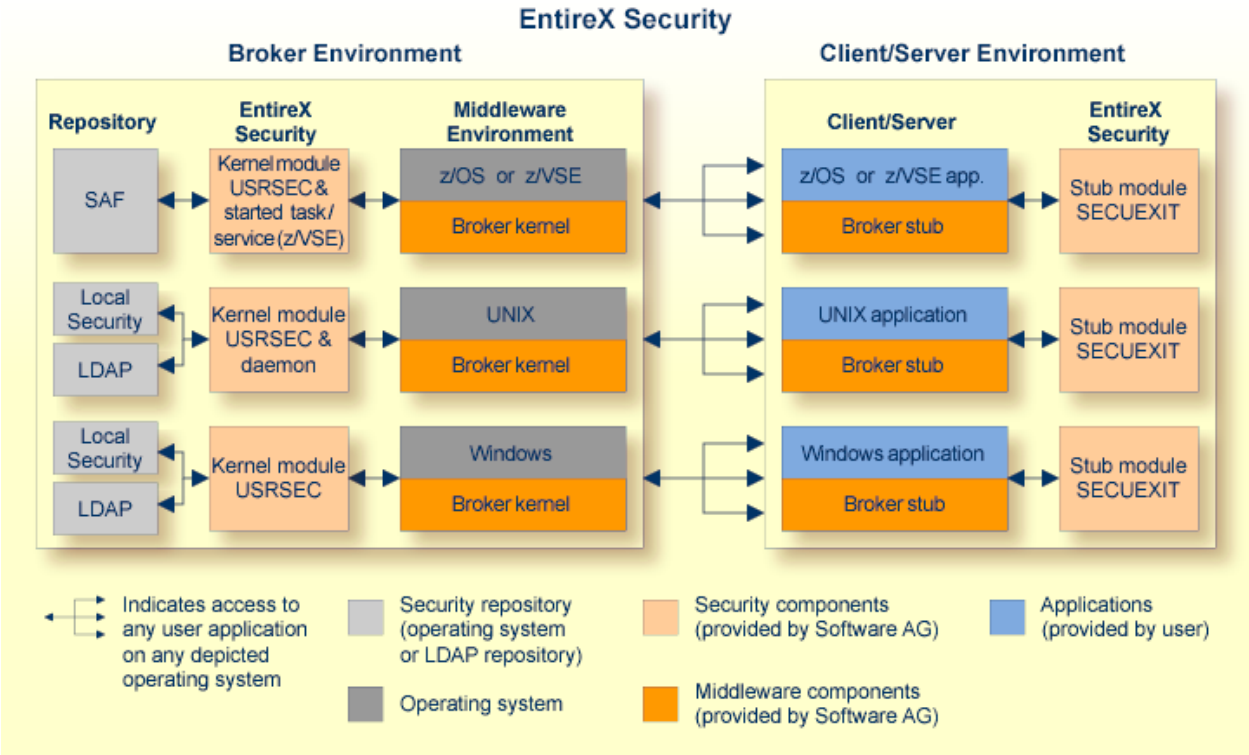
Functionality of EntireX Security

This table lists the security functionality available with EntireX Security running Broker Kernel under z/OS, UNIX or Windows.

Security Functionality	z/OS	UNIX	Windows	BS2000/OSD	z/VSE	Comment
Authentication of user	Yes	Yes	Yes	Yes	Yes	Verify User ID password.
User password change	Yes	No	No	No	No	
LDAP authentication	No	Yes	Yes	No	No	Authenticate using LDAP repository.
Trusted user ID	Yes	No	No	No	No	Trusted computer base, avoiding plain text password.
Verified client user ID	Yes	No	No	Yes	Yes	Provide verified identity of client to server.
Authorization of client request	Yes	No	No	No	No	
Authorization of server register	Yes	No	No	No	No	
Authorize IP connection	Yes	No	No	No	No	
Authorization rules	No	Yes	Yes	No	No	Check rules stored in an LDAP repository. These rules are maintained using an agent of System Management Hub, and are independent of the LDAP authentication mechanism. **Note:** These rules can be stored either in the same or a different LDAP repository.
Encryption of application data	Yes	Yes	Yes	No	Yes	RC4-compatible algorithm.
Guaranteed encryption	Yes	Yes	Yes	No	Yes	Allows administrator to require encryption for specific services.
SSL	Yes	Yes	Yes	No	No	Industry standard encryption mechanism.

EntireX Security Components

This diagram depicts the location where the broker kernel must be installed and where the broker stubs can be installed. It also depicts the location of the security components of the kernel and stubs of broker.



5 EntireX Security under Windows

This chapter introduces EntireX Security under Windows through overviews of the functionality and components of EntireX Security. The location where the broker kernel is installed determines the functionality made available for EntireX Security.

Note: Installation of the security software is described under *Setting up EntireX Security under Windows* under *Post-installation Steps under Windows*.

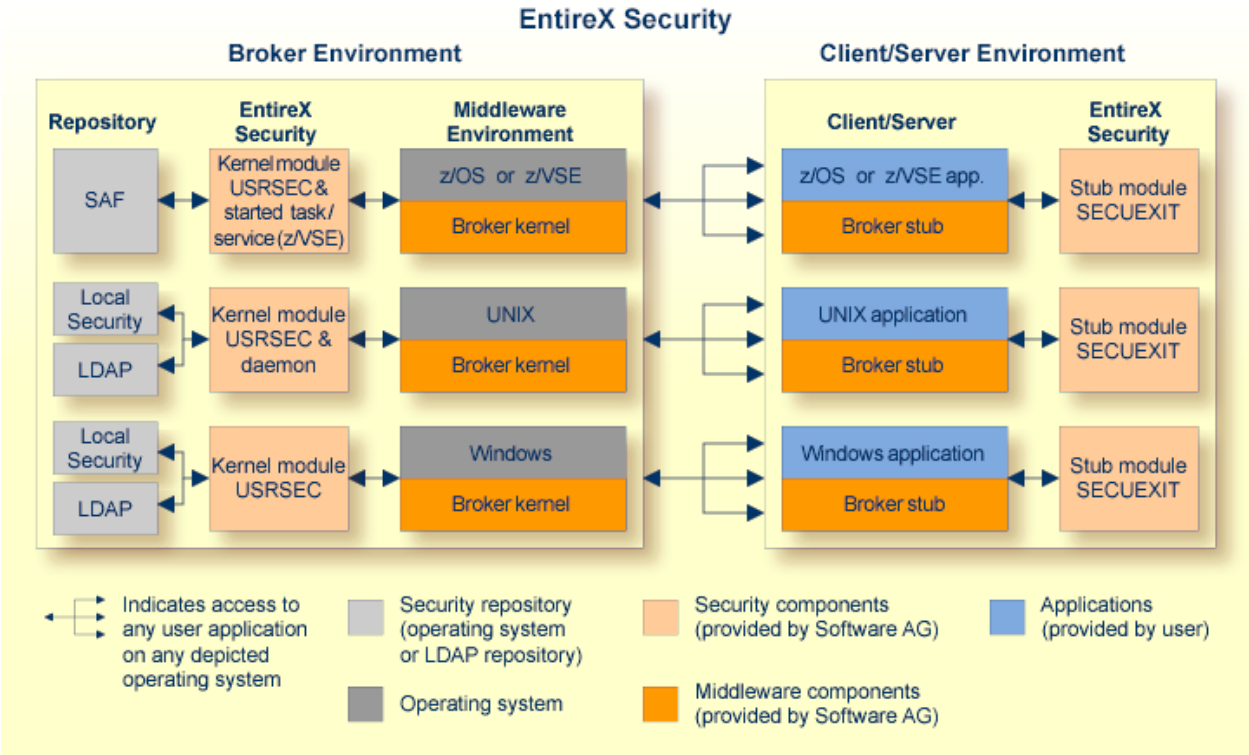
Functionality of EntireX Security

This table lists the security functionality available with EntireX Security running Broker Kernel under z/OS, UNIX or Windows.

Security Functionality	z/OS	UNIX	Windows	BS2000/OSD	z/VSE	Comment
Authentication of user	Yes	Yes	Yes	Yes	Yes	Verify User ID password.
User password change	Yes	No	No	No	No	
LDAP authentication	No	Yes	Yes	No	No	Authenticate using LDAP repository.
Trusted user ID	Yes	No	No	No	No	Trusted computer base, avoiding plain text password.
Verified client user ID	Yes	No	No	Yes	Yes	Provide verified identity of client to server.
Authorization of client request	Yes	No	No	No	No	
Authorization of server register	Yes	No	No	No	No	
Authorize IP connection	Yes	No	No	No	No	
Authorization rules	No	Yes	Yes	No	No	Check rules stored in an LDAP repository. These rules are maintained using an agent of System Management Hub, and are independent of the LDAP authentication mechanism. **Note:** These rules can be stored either in the same or a different LDAP repository.
Encryption of application data	Yes	Yes	Yes	No	Yes	RC4-compatible algorithm.
Guaranteed encryption	Yes	Yes	Yes	No	Yes	Allows administrator to require encryption for specific services.
SSL	Yes	Yes	Yes	No	No	Industry standard encryption mechanism.

EntireX Security Components

This diagram depicts the location where the Broker kernel must be installed and where the Broker stubs can be installed. It also depicts the location of the security components of the kernel and stubs of Broker.



6 SSL or TLS and Certificates with EntireX

Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are program layers for managing the security of message transmissions in a network. The idea is to contain the programming required to keep messages confidential in a program layer between an application (such as your Web browser or HTTP) and the internet's TCP/IP layers. The term sockets refers to the method of passing data back and forth between a client and a server program in a network or between program layers in the same computer. SSL and TLS use the public-and-private key encryption system from RSA, which also includes the use of a digital certificate.

This chapter describes Secure Sockets Layer (SSL) or Transport Layer Security (TLS) and Certificates within an EntireX context. When "SSL" is used in the documentation, this refers to both SSL and TLS.

See also *Running Broker with SSL or TLS Transport* in the platform-specific administration documentation.

Introduction

One of the major components when using SSL is the certificate. One of the tasks of certificates is to ensure that communication, which runs atop TCP/IP, adheres to an industrial-strength encryption.

Certificates can be described as electronic passports. They contain information about someone (or a machine or location), generally called the Subject. The authenticity of the subject's information is digitally signed by a trustworthy instance, called the Issuer. With certificates, this issuer is also known as a Certificate Authority (CA).

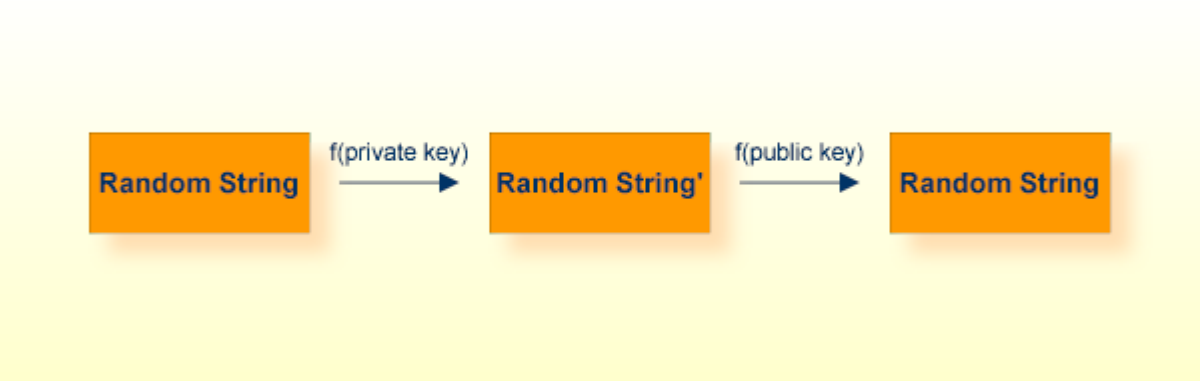
In addition to the above, a certificate also contains a random number that is called the subject's public key. Together with this public key, the subject must also be in possession of a private key. As their names suggest, the public key can be viewed by anyone, whereas the private key must be strictly secured. The public and the private keys together always form a key pair, i.e. they are always created together and complement each other.

Here are some typical scenarios of their usage:

Encryption:

In the image above, a public key has been used to encrypt a document. Only the owner of the private key is able to decrypt this text.

Authentication



To verify that the instance that presented a certificate is really who they claim to be (authentic), I can choose a random string, encrypt it with their public key, send it to the subject, have it decrypted with their private key and sent back. I then compare it with my original random string. Only the owner of the appropriate private key is able to perform this operation.

Random Number Generator

Another of the major components with SSL is called the Random Number Generator (RNG). To ensure genuinely random keys with each new session, SSL uses its own random number generator.

This requires a "seed", which should be unique for each installation.

- On UNIX systems, make sure you have defined the environment variable `RANDFILE`, which refers to a file that contains some random text (e.g. some 20 lines of 50 characters each should suffice).
- On Windows systems, the seed is automatically taken by using the pixels of the current desktop.

Using SSL or TLS as the Transport for EntireX

As we have seen, certificates play an important role with SSL. In order to use SSL as the transport method for EntireX, you need to have certificates available at various locations and for various purposes.

After the installation process, you will find the following certificates in the *etc* directory (UNIX and Windows) or in *EXX951.CERT* (z/OS) ready to use for preliminary testing of the SSL transport:

Certificate	Explanation
For C-based programs	
ExxCACert.pem	The CA certificate. This certificate can be used to verify the application certificate (see TRUST_STORE parameter).
ExxCAKey.pem	The private key of the CA certificate, above. The password to unlock this private key is ExxCAKey. You will need this password only if you want to sign more certificates with this CA (not recommended).
ExxAppCert.pem	To be used as the SSL server certificate (see KEY-STORE). This certificate is signed with the private key within ExxCAKey.pem
ExxAppKey.pem	The private key of the application certificate. The password to unlock the key is ExxAppKey (see KEY-FILE and KEY-PASSWD-ENCRYPTED).
For Java-based programs	
ExxCACert.jks	The truststore containing the default CA certificate (see TRUST_STORE).
ExxJavaAppCert.jks	The keystore containing the application certificate. The password to unlock this container is ExxJavaAppCert (see KEY-STORE and KEY-PASSWD-ENCRYPTED).

SSL usually requires a certificate on the server side of a communication (which in EntireX is always the Broker kernel). In order to validate the certificate, the other side needs to accept the issuer of the server certificate, i.e. needs to trust the same instance that the certificate has signed. (Customs authorities do not trust your passport - which could be forged - but instead verify its authenticity electronically!)

The server certificate is specified using the KEY-STORE attribute.

The appropriate private key is found using the KEY-FILE attribute.

(Generally, the private key is not stored in the open, it is further encrypted with a password, which - because it is often more than a single word - is sometimes also called passphrase. To use the private key properly, the application must be able to re-create the original private key. Therefore you have to provide the appropriate password in the KEY-PASSWD-ENCRYPTED attribute.)

The client must now present the CA (i.e. its certificate, which includes the public key), so that SSL can determine whether to accept a server certificate or not. The following possibilities are available:

1. For non-Java *ACI-based Programming*, this is done with the SETSSLPARMS command for Broker ACI programs. Specify TRUST_STORE in the 2nd parameter, for example:
   - For z/OS except CICS [(1)]:

```
'broker' etbcb "VERIFY_SERVER=N&TRUST_STORE=<racf_uid>/<racf_keyring>"
```

(on Z/OS certificates are stored in RACF)

- For UNIX and Windows:

```
'broker' etbcb "VERIFY_SERVER=N&TRUST_STORE=c:\\certs\\CaCert.pem"
```

2. For *Java ACI* in the Developer's Kit documentation, SSL parameters are appended [(2)] to the Broker ID, for example:

```
ssl://localhost:22101?trust_store=C:\SoftwareAG\EntireX/etc/ExxCACert.jks&verify_server=n
```

3. In general for most EntireX Components such as RPC servers, RPC clients, agents etc., SSL parameters are also appended [(2)] to the Broker ID, as for Java ACI above. See the documentation of the component for further information.
4. Under z/OS, IBM's Application Transparent Transport Layer Security (AT-TLS) can be used as an alternative to direct SSL support inside the broker stub. Refer to the IBM documentation for further information.

See also *Transport: Broker Stubs and APIs*.

Notes:

1. Under z/OS CICS, use option 4.
2. See *URL-style Broker ID* for more information.

The parameter `VERIFY_SERVER` can have the following (Boolean) values:

Value	Explanation
`YES` (default)	The name of the server certificate (the field CN of the Subject) must be equal to the Broker ID (excluding port number and transport). (Example: `broker_id="pc001.my-company.com:1958:ssl"` and Broker kernel certificate (see `KEY-STORE`): `Subject, CN=pc001.my-company.com`)
`NO`	Accept any Common Name (CN) in the server certificate, but still check that the certificate is signed by a trusted CA (see `TRUST_STORE`).

The default application certificate that we deliver in the *etc* directory is issued to “localhost”. This enables you to use a Broker ID of “localhost” together with `VERIFY_SERVER=Y`.

Refer also to the sections on how to create your own certificates with *`openssl`* and with *`keytool`*.

Note: In order to allow for multiple CAs, concatenate all of the CAs' .pem files into a single new .pem file and specifiy this new .pem file in the `TRUST_STORE` parameter. If you are using

Java and a keystore, then import multiple times the various CA certificates into the keystore (`-> TRUST_STORE`).

Creating Certificates with openssl

This section contains step-by-step instructions on how to create your own certificates.

Note: Certificates adhere to a standard format and can also be created with other tools; `openssl` is delivered with EntireX and can be used as an example.

To create your own certificates

1 Create a new directory in which the new certificates will be created and where all of the other required files will be stored.

2 In your new directory create a file named *genca.cnf* and cut and paste the contents of the file gencacnf.html (delivered with this documentation) to your new file.

3 Create a file called *.rand* in your new directory. Edit it, adding about 20 lines (each about 40 to 50 characters long) of random text. (This file is used by the random number generator, so do not use it in more than one location to create certificates!)

4 Create an empty directory newcerts in your new directory.

5 Create an empty directory certs in your new directory.

6 Create an empty file called *index.txt* in the current directory.

7 Create a file called *serial* in the current directory and enter a number in column 1, line 1, for example: 1000. This serial number will be incremented for each certificate that you create.

8 Now edit the *genca.cnf* file which you cut and pasted into your new directory in step 2, above. Please read the comments carefully. There are a few defaults that you will probably want to adapt to your own environment.

   Below is a list of the important variables that should be checked:

   1. Set the variable `RANDFILE` to point to the *.rand* file. (This appears twice in the file; adjust both occurrences.)
   2. Set the variable `database` to point to the *index.txt* file.
   3. Set the variable `serial` to point to the *serial* file.
   4. Set the variable `new_certs_dir` to point to the newcerts directory.
   5. Set the variable `certs_dir` to point to the certs directory.
   6. Set the variable `certificate` to point to the CA certificate file (see *NewCACert.pem* in the example below).
   7. Set the variable `private_key` to point to the CA certificate's private key file (see *NewCAKey.pem* in the example below).

9 Save the configuration file.

You can now start creating certificates.

First, you need to define a Certificate Authority (CA); create a key pair and a self-signed certificate to represent this CA.

Enter the following command in a shell and follow the instructions (be patient, loading the screen state takes several seconds)

```
openssl req -config genca.cnf -newkey rsa:1024 -x509 -keyout <NewCAKey.pem> -out ↩
<NewCACert.pem> -days 365
```

Do not forget the passphrase for the key file! You will need it soon.

Now you have a CA certificate and a CA key file.

Next, create a certificate that can be used by various products (for example the Broker kernel) to start an SSL server session.

With the CA cert and key files described above you can create any number of certificates. We will sign all of them with the same CA (used from the *genca.cnf* file).

Create a certificate request:

```
openssl req -config genca.cnf -newkey rsa:1024 -out <ExxAppCertReq.pem> -keyout ↩
<ExxAppKey.pem> -days 365
```

You will be prompted for a new passphrase. Again, this will be the passphrase to lock the *MyAppKey.pem* file. Remember it well.

You must then sign this certificate request with your CA to create a proper certificate:

```
openssl ca -config genca.cnf -policy policy_anything -out <ExxAppCert.pem> -infiles ↩
<ExxAppCertReq.pem>
```

Note: The passphrase you are prompted with is the one used to unlock the CA key.

Creating Certificates with keytool

A certificate management tool is also supplied with the standard JDK kit, i.e. it is part of J2SE kit, not the JSSE kit. Certificate requests can be generated and keystores and truststores can be built with this tool. The steps for building keystores and truststores are outlined below.

To create a keystore

1 Create a keystore containing a self-signed certificate and key (example yourkeystore).

The following command will prompt you for identification information.

```
 keytool -genkey -v -alias yourJavaApp -keyalg RSA -validity 900 -keypass ↩
yourkeypsw -keystore yourkeystore -storepass  yourkeypsw
```

The private key password and keystore password must be identical for XTS 1.2 JSSE implementation. The keytool provides commands to change either the private key or keystore password.

2 Import any CA certificates of CAs which will sign the certificate generated above.

```
keytool -import -v -alias yourcacert -file yourcacert.pem -keystore yourkeystore ↩
-storepass yourkeypsw
```

3 (Optional) List certificate chain present in keystore.

```
keytool -list -v -keystore yourkeystore -storepass yourkeypsw
```

4 Extract certificate for signing by a CA.

```
keytool -certreq -v -alias yourJavaApp -file yourJavaAppreq -keypass yourkeypsw ↩
-keystore yourkeystore -storepass yourkeypsw
```

5 Sign Java certificate request with openssl tool.

```
openssl ca -config yourca.cnf -policy policy_anything -out yourjavaapp.pem ↩
-notext -days 365 -infiles yourJavaAppreq
```

Note: The `-notext` parameter is required. Without it, the import of a signed certificate to keystore will fail. The error will be either a `Not an X.509 certificate` or a `Tag sequence` error. The reason for the error is that the openssl signing tool will write both a text version and an encoded version of the signed certificate to the output file if the `-notext` parameter is not specified.

6 Import signed certificate.

```
keytool -import -v -alias yourJavaApp -file yourjavaapp.pem -keypass yourkeypsw ↩
-keystore yourkeystore -storepass yourkeypsw
```

Notes:

1. *yourjavaapp.pem* is the signed certificate returned by the CA.
2. Import will only work if a signed CA certificate is already present in the keystore.

To create a truststore

- Import the CA certificates that were used to sign the client and server certificates.
  - Import signed CA certificates.

```
keytool -import -v -alias yourcacert -file yourcacert.pem -keystore ↩
yourtruststore -storepass yourstorepsw
```

  - (Optional) List truststore.

```
keytool -list -v -keystore yourtruststore -storepass yourstorepsw
```

Exporting Certificates and Private Keys with openssl

Note: Applies to operating system z/OS only.

Enter the following command to create a file containing the application certificate and application key files for import into RACF:

```
openssl pkcs12 -export -in <ExxAppCert.pem> -inkey <ExxAppKey.pem> -out ↩
<ExxAppCert.p12>
```

You will be prompted for the passphrase of the private key and for an export password. The output file is created in PKCS#12 format. FTP can be used to transfer the output file in binary mode to the IBM host.

Importing Certificates and Private Keys with RACDCERT

Note: Applies to operating system z/OS only.

To import certificates and private keys with `RACDCERT` into RACF

- See readme file EXX951.CERT(README) in the product distribution for detailed instructions.

Additional Considerations for PKI (Public Key Infrastructure)

When using a PKI, there are usually more than two certificates involved. Typically, there is one (self-signed) root certificate, one or more CA certificates, and several application certificates, usually one for every server.

For the SSL server side (Broker) you need a suitable application certificate.

To check the certificate

- Execute the command:

```
openssl x509 -in <YourSSLCert.pem> -text
```

  This will display relevant information about the certificate such as key extensions with key usage and basic contraints. (For example, the Basic Contraint CA should be "FALSE".)

Given a specific server certificate, it is also possible to verify the certificate chain.

To verify the certificate chain

- Execute the command:

```
openssl verify -CAfile <YourCaCert.pem> -purpose sslserver <YourSSLCert.pem>
```

  If you receive an OK, then `<YourSSLCert.pem>` should work on the SSL server side together with the `<YourCaCert.pem>` on the SSL client side.

Note: If there is a chain of CA certificates defined, copy the contents of the appropriate CA*xxx*.pem files into one new file and use this as the `<YourCaCert.pem>` on the client side to verify the SSL server certificate against.

Support of Self-signed Certificates

To support self-signed certificates it is sometimes necessary to modify the LDAP settings. For example, to allow use of self-signed certificates in OpenLDAP, add the following line to file */etc/openldap/ldap.conf*:

```
TLS_REQCERT never
```

7 IAF Considerations

The Integrated Authentication Framework (IAF) is a token-based infrastructure that enables Software AG's enterprise single sign-on. In addition, it allows usage of a configurable authentication system (user database) with Software AG products across platforms. IAF is part of the Software AG Security Infrastructure.

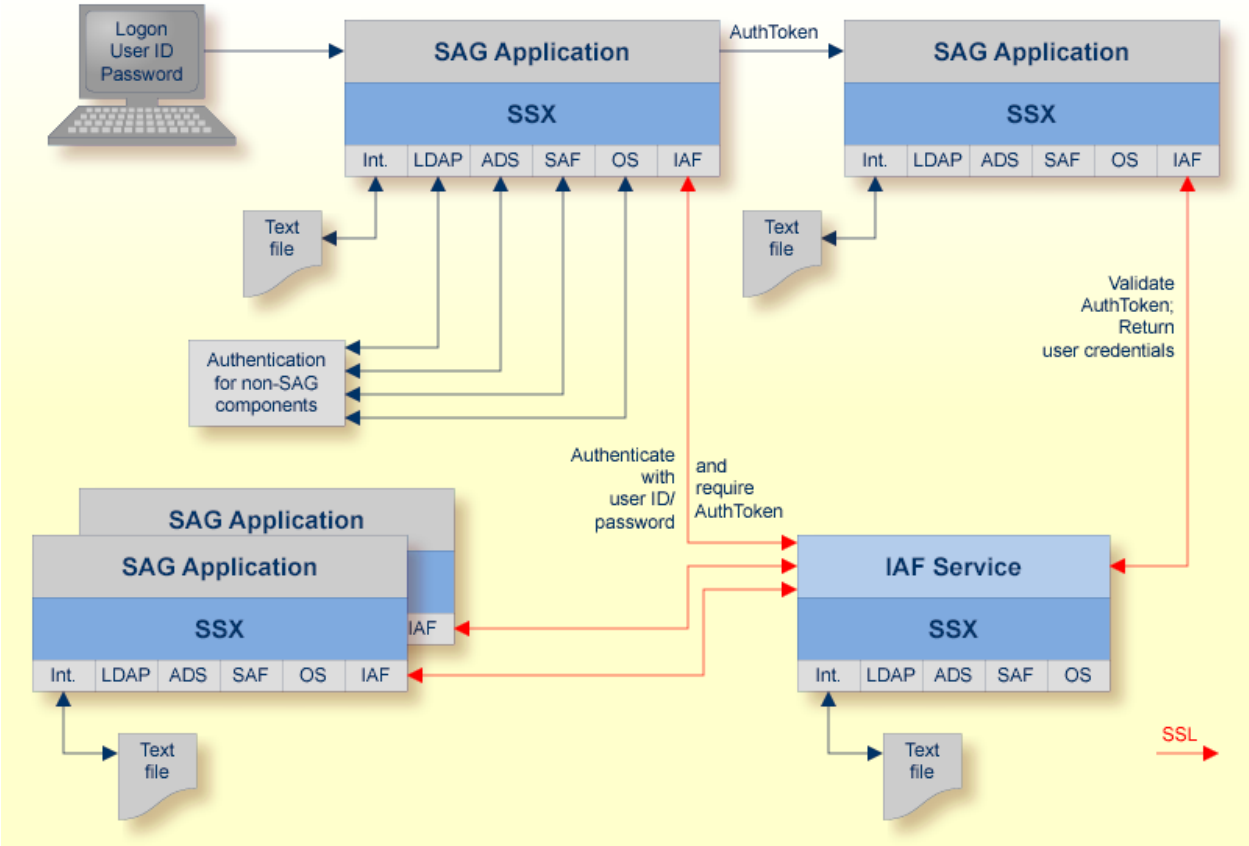
See also *Integrated Authentication Framework (IAF)* under *Shared Components - Security Infrastructure* on the **Software AG Product Documentation** website.

IAF Architecture

The architecture of IAF defines a central service (the IAF service) that is contacted by multiple clients in order to:

- authenticate a user
- create a unique and fraud resistant token
- validate tokens
- pass information about the token and the authenticated user to the owner of the token

More and more Software AG products will be equipped with the IAF client-side modules. These will be configured through the configuration process of the application itself. Mainly, the application must know the "address" of the IAF server. Under UNIX and Windows the IAF server can be started and stopped using a System Management Hub agent. Under z/OS it runs as a started task and can be started and stopped via operator commands.



IAF Implementation Details

The IAF server is configured using an attribute file, where you can define parameters that influence the scalability (multiple parallel threads) and internal cache sizes (max. buffers) of the server.

The main functionality of an IAF server is also configured using the attribute file. You have to select an existing user repository (such as a RACF database or a Windows Active Directory) that is triggered with authentication requests.

No license file is required, since IAF is a common infrastructure that can only be used by Software AG products.

The access module for the IAF server is called SSX [Software AG Security eXtension]. SSX is a shared library that offers authentication for a variety of different user repositories. The number of repositories depends on the platform.

All SSX implementations offer authentication type "IAF". The `serverHost` parameter is then a Broker ID that points to a running IAF server.

For EntireX, the notation of an IAF server is as follows:

```
iaf://<IAF-server-machine-IP-address>:<port>?<sslparms>
```

Calling IAF for authentication serves two purposes:

1. The authentication is performed by a remote server, i.e. can reach out to user repositories that may not be available on the local machine.
2. IAF creates a unique and secure token for each successful user authentication. The server returns this token to the client process, where it can be included in the message flow. The advantages are:
   - The user ID is not readable because the token is encrypted.
   - The token cannot be changed (e.g. the UserId cannot be altered) on transit because it is signed.
   - The token guarantees to the receiver that a user has successfully signed in to a user repository. Using an SSX library and the connection to the IAF server makes it possible to reveal the user name and some more statistical data about the authentication process.

IAF and EntireX Broker

IAF has been added as an authentication method to the Broker.

In the `DEFAULTS=BROKER` section of the broker attribute file, specify

```
SECURITY = YES
```

In the `DEFAULTS=SECURITY` section, specify the address of the IAF service:

```
AUTHENTICATION-TYPE="iaf://<server-name>:<server-port>?<ssl-parms>"
SECURITY-LEVEL=AUTHENTICATION
```

At the moment, IAF on the mainframe is not capable of performing authorization calls against RACF resource definitions. As the default SECURITY-LEVEL on the mainframe sets both authentication and authorization, it must be explicitly restricted to `SECURITY-LEVEL=AUTHENTICATION`.

See *Security-specific Attributes* (`DEFAULTS=SECURITY`) under *Broker Attributes* in the administration documentation.