webMethods EntireX

Security

Version 10.1

October 2017

WEBMETHODS

This document applies to webMethods EntireX Version 10.1 and all subsequent releases.

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

Document ID: EXX-SECURITY-101-20191129

Table of Contents

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **http://documentation.softwareag.com**. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.asp** and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at **http://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Which EntireX Security Solution?

Choosing a Security Solution

The security solutions for your distributed applications using EntireX are described below.

Security Solutions

- **EntireX Security**
  This is Software AG's standard security solution providing user authentication and user authorization, delivered with EntireX. Most organizations that use 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.
- **Sample Security Exits for Broker**
  This is an alternative, user-written security solution for use only in exceptional processing situations.
- **SSL/TLS and Certificates with EntireX**
  For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See ***SSL/TLS and Certificates with EntireX***.

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*. ■ UNIX local security or LDAP repository for Broker running under UNIX. See *Setting up EntireX Security under UNIX*. ■ Windows local security or LDAP repository for Broker running under Windows. See *Setting up EntireX Security under Windows*. The major advantages of EntireX Security: ■ **Comprehensive Security** EntireX Security provides comprehensive security for EntireX Broker, that is user authentication and user authorization ■ **Protection of Application Systems** EntireX Security protects client and server application systems.

Security Choice	Criteria for Choosing a Security Solution
	■ **No User Exits to Write/Debug** EntireX Security is fully supported (that is, object code only). There are no user exits to write and debug. 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. ■ **Standard Security Definitions** EntireX Security enables security definitions, based on class/name/service (client and server), to be validated. All definitions are managed using existing security procedures and software. ■ **Protected Investment in SAF-based Security Repositories** On z/OS security definitions are accessed using industry standard SAF interface. Your investment in SAF-based security repositories is therefore protected. This includes not only the security systems RACF, CA ACF2 and CA Top Secret, but also the infrastructure to administer security profiles.
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 when operating EntireX Broker. Sample security exits are provided as skeleton programs only and must be completely customized before they can be deployed.
SSL/TLS and Certificates with EntireX	For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol.

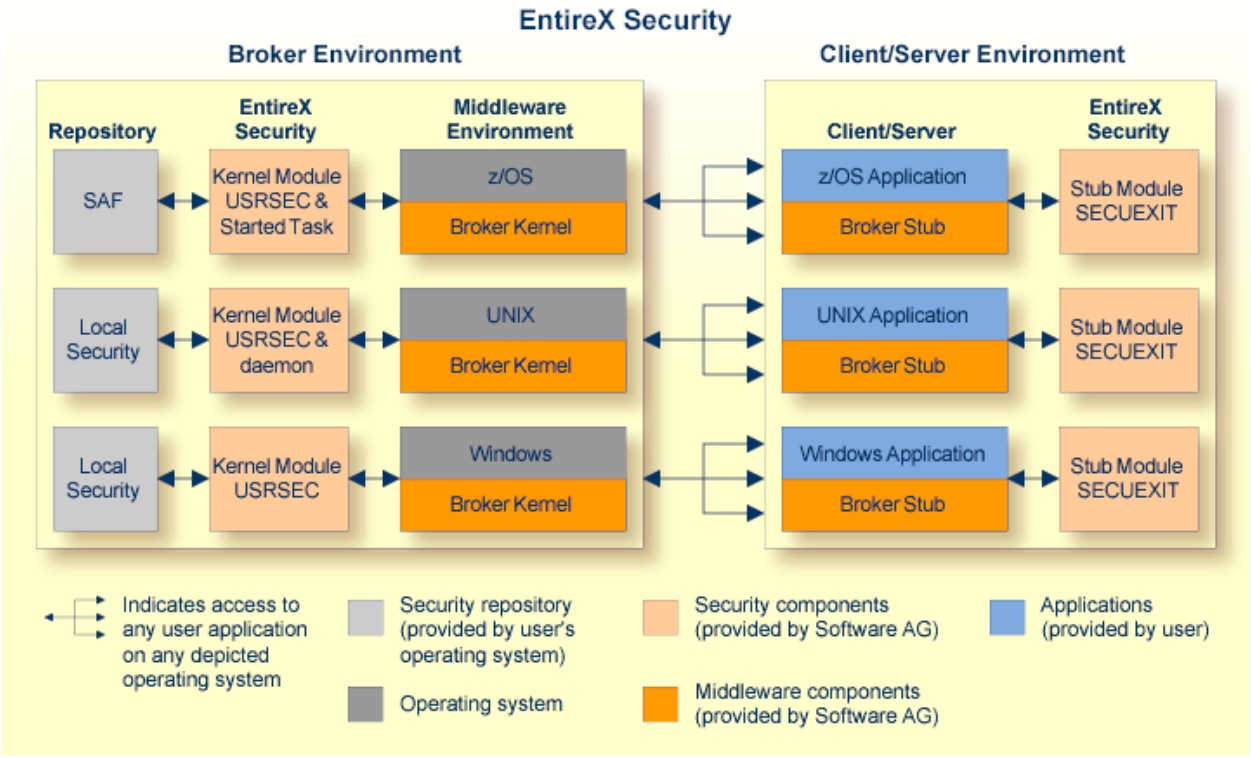
Notes:

1. Do not mix the security solutions **EntireX Security** and ***Sample Security Exits for Broker Security***. It is an either/or choice. Do not use a stub secured with a sample exit against a kernel secured with EntireX Security or vice versa.
2. *SSL/TLS and Certificates* can be used in combination with **EntireX Security** or ***Sample Security Exits for Broker Security***.

Overview of Security Configurations

EntireX Security: Standard Security Solution

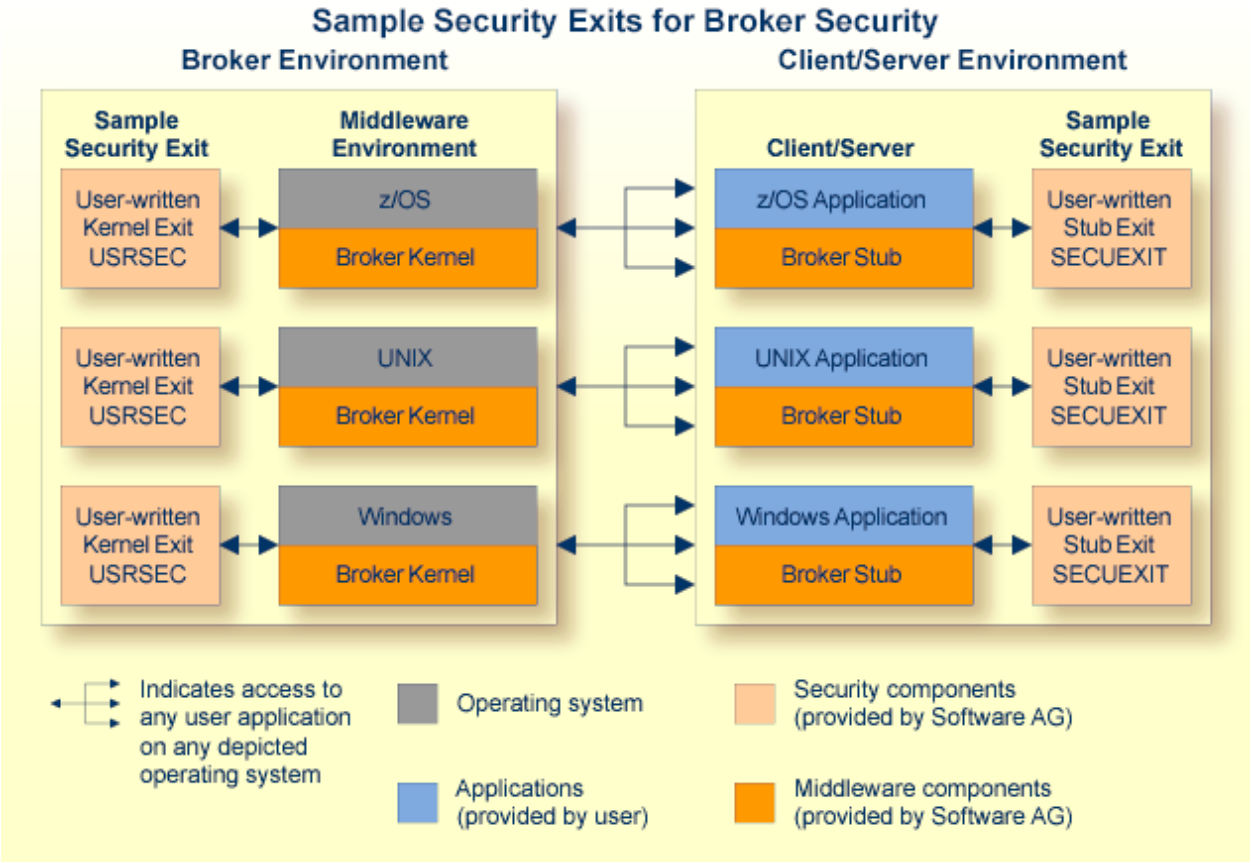
This diagram shows the locations where the broker kernel and broker stubs can be installed; it also shows the locations of the security components of the kernel and stubs.



The *List of Components per Platform* shows where broker kernel and stubs are supported.

Sample Security Exits for Broker Security

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.



The *List of Components per Platform* shows where broker kernel and stubs are supported.

3 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.

See also *Which EntireX Security Solution?*

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 application components is protected, using definitions located in the security repository of your organization: for example 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, and Command and Information Services

See *Functionality of EntireX Security*.

Overview of EntireX Security Features

EntireX Security provides comprehensive security for EntireX Broker:

- user authentication
- user authorization
- supplied in object code only

The major advantages of EntireX Security:

- **Comprehensive Security**
  EntireX Security provides comprehensive security for EntireX Broker, that is user authentication and user authorization
- **Protection of Application Systems**
  EntireX Security protects client and server application systems.
- **No User Exits to Write/Debug**
  EntireX Security is fully supported (that is, object code only). There are no user exits to write and debug. 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.
- **Standard Security Definitions**
  EntireX Security enables security definitions, based on class/name/service (client and server), to be validated. All definitions are managed using existing security procedures and software.
- **Protected Investment in SAF-based Security Repositories**
  On z/OS security definitions are accessed using industry standard SAF interface. Your investment in SAF-based security repositories is therefore 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 *Authorization Rules*.

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. Specify `INFO` for the full information service: all clients, servers and conversations are listed.
`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. `USER-INFO` is an information service limited to user-specific information: only the user's own resources are listed.

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

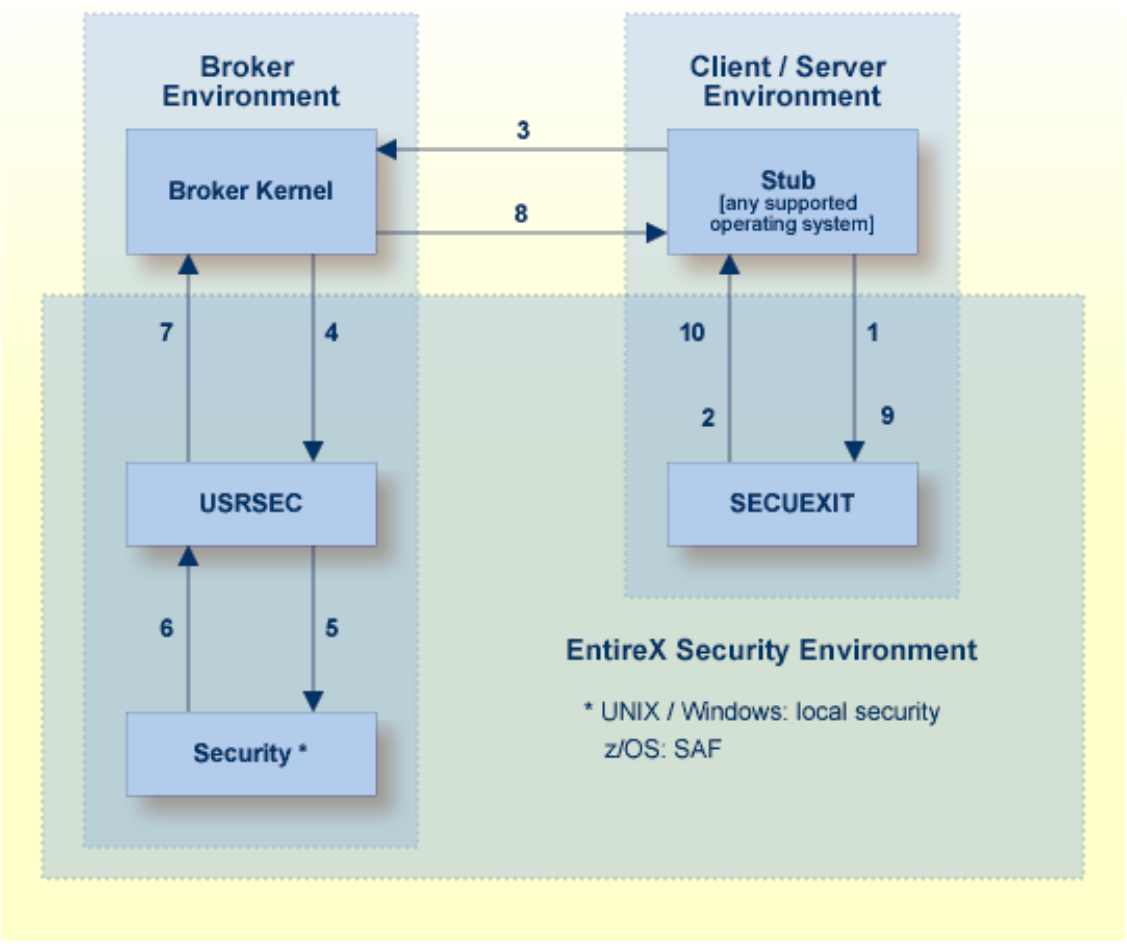
- To shutdown a *service*, users must have the required authorization to register this class, server and service themselves.
- To shutdown a *server*, users must have the required authorization to register all the services registered by that server.

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

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 *Authorization Rules*.

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 and authentication.



Note: This diagram depicts the operation of the broker stub for Natural and other third-generation programming languages. It is not intended to show the mechanism used by the Java ACI and *EntireX Adapter* with regard to EntireX Security. See *Using Security with Java-based EntireX Applications* under *Writing Advanced Applications - EntireX Java ACI*.

Description of Steps in Data Flow

1. Broker stub calls security module `SECUEXIT`, if present.
2. Security module `SECUEXIT` encrypts the password.
3. Broker stub communicates the call to the broker kernel.
4. Broker kernel calls security module `USRSEC`, which provides the functionality, based on the *EntireX Security Configuration Options for Broker*:
   - 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`.
10. Security module `SECUEXIT` returns call to Broker stub.

For installation see *Installing/Setting up EntireX Security* in the respective section of the Installation documentation.

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
- 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.

See *List of Components per Platform* 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* for where broker stubs are supported.

4 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*.

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 ***EntireX Security Configuration Options for Broker***.

Security Functionality	z/OS	UNIX	Windows	BS2000	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	An authorization rule is used to perform access checks for authenticated user IDs against lists of services defined within the rule. This feature is available on UNIX and Windows using EntireX Security on these platforms. Authorization rules can be stored in the Broker attribute file or in an LDAP repository. See *Authorization Rules*.
SSL/TLS	(1)	Yes	Yes	No	(2)	Industry standard encryption mechanism. See ***SSL/TLS and Certificates with EntireX***.

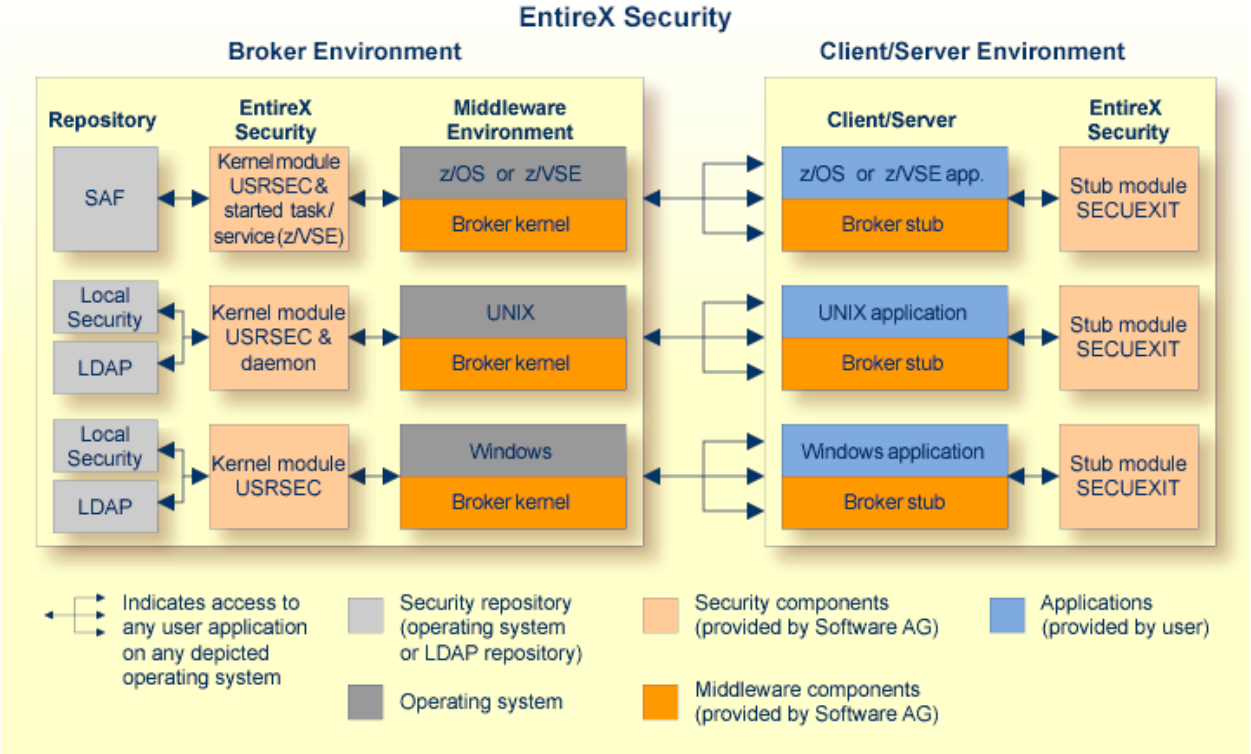
Notes:

1. Establishing an SSL/TLS connection on z/OS is done with IBM's Application Transparent Transport Layer Security (AT-TLS)

2. Establishing an SSL/TLS connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS).

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

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.

EntireX Security 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(SAFIO1O):

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

Make sure the Adabas Security Interface is enabled by specifying the following parameter in the source assembled by job member WAL*vrs*.JOBS(ASMGBLS):

SAF=YES	Adabas Security Interface in use.

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 =<u>{YES</u>,NO}	Include Class in resource check.
INCLUDE-NAME =<u>{YES</u>,NO}	Include Name in resource check.
INCLUDE-SERVICE =<u>{YES</u>,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 servers require CONTROL access in order to REGISTER successfully, otherwise the command is rejected.

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.

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

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.

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

The following parameters can be used to modify the functionality of EntireX Security:

`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.

Notes:

1. In version 8.0, the default value for this parameter was "AUTHORIZATION"

Caution:

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*.

EntireX Broker Client/Server

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.

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 by setting `CHECK-IP-ADDRESS=YES`. 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*.

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 example 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
```

5 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*.

Functionality of EntireX Security

This table lists the security functionality available with EntireX Security running Broker Kernel under the respective operating system.

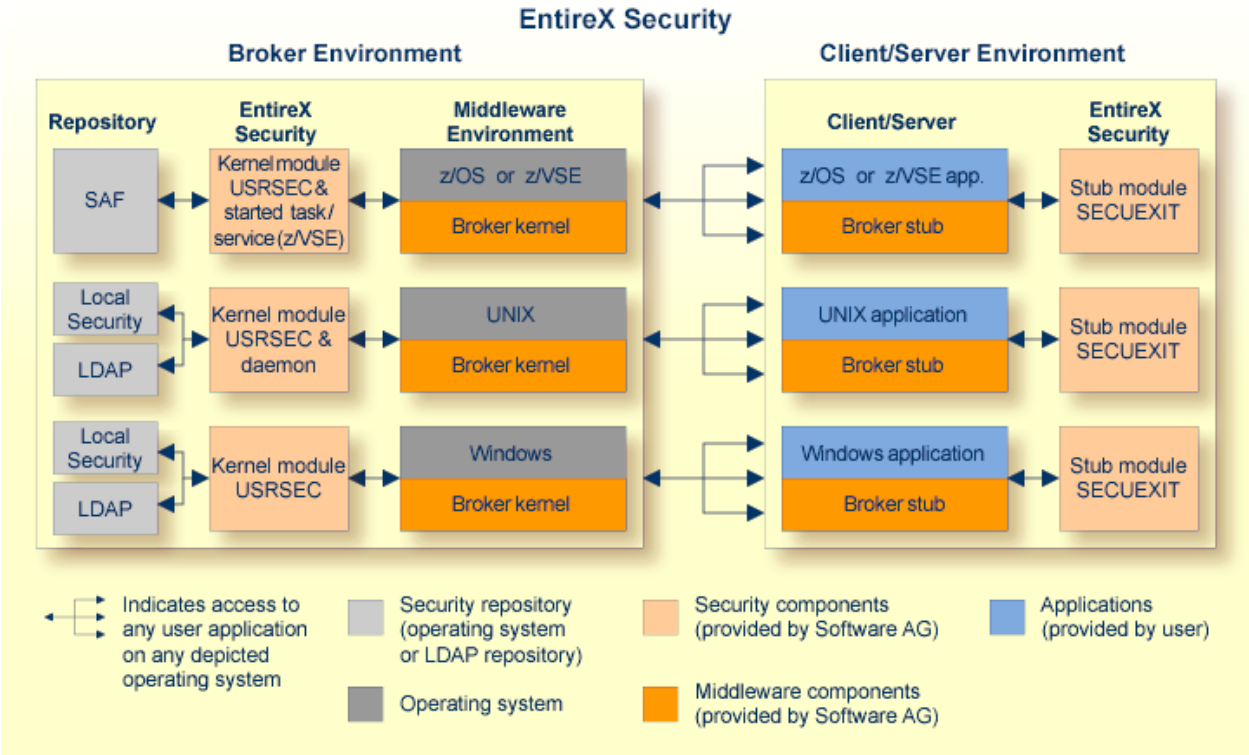
Security Functionality	z/OS	UNIX	Windows	BS2000	z/VSE	Comment
Authentication of user	Yes	Yes	Yes	Yes	Yes	Verify User ID password.
User password change	Yes	No	No	No	No	
LDAP authentication	No	Yes	Yes	No	No	Authenticate using LDAP repository.
Trusted user ID	Yes	No	No	No	No	Trusted computer base, avoiding 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	An authorization rule is used to perform access checks for authenticated user IDs against lists of services defined within the rule. This feature is available on UNIX and Windows using EntireX Security on these platforms. Authorization rules can be stored in the Broker attribute file or in an LDAP repository. See *Authorization Rules*.
SSL/TLS	(1)	Yes	Yes	No	(2)	Industry standard encryption mechanism. See ***SSL/TLS and Certificates with EntireX***.

Notes:

1. Establishing an SSL/TLS connection on z/OS is done with IBM's Application Transparent Transport Layer Security (AT-TLS)
2. Establishing an SSL/TLS connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS).

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 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*.

Functionality of EntireX Security

This table lists the security functionality available with EntireX Security running Broker Kernel under the respective operating system.

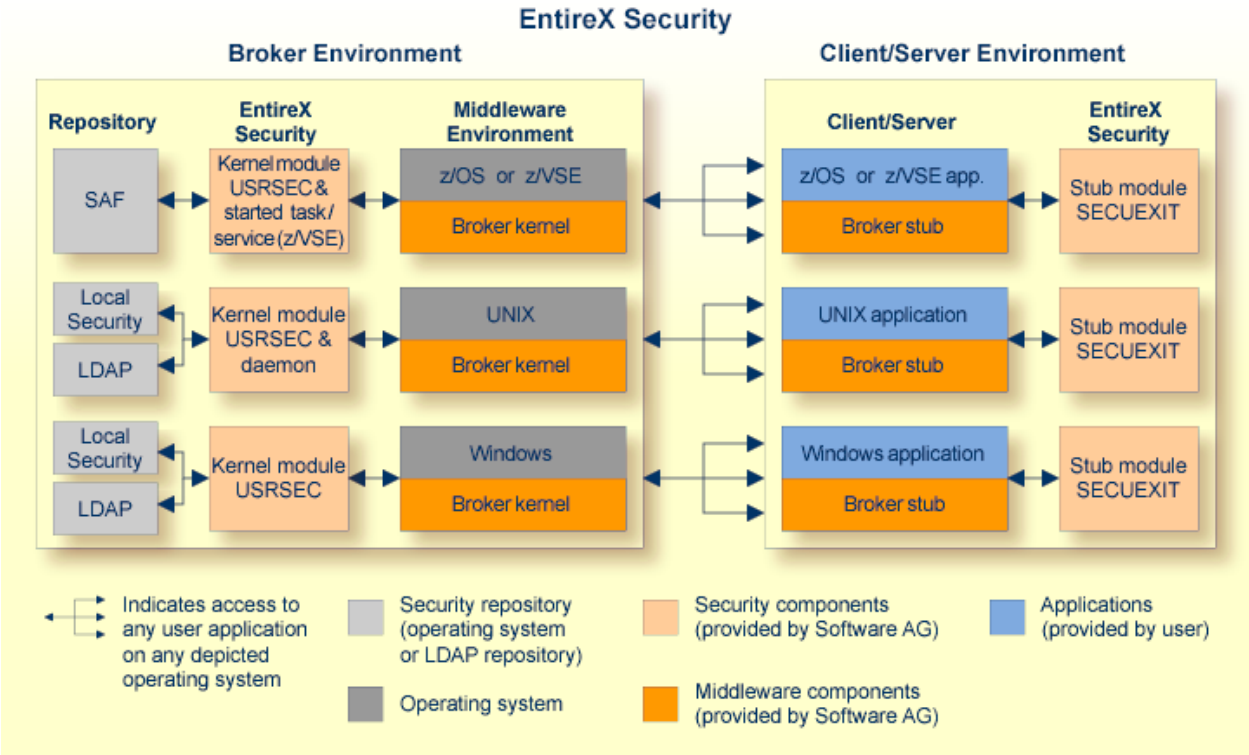
Security Functionality	z/OS	UNIX	Windows	BS2000	z/VSE	Comment
Authentication of user	Yes	Yes	Yes	Yes	Yes	Verify User ID password.
User password change	Yes	No	No	No	No	
LDAP authentication	No	Yes	Yes	No	No	Authenticate using LDAP repository.
Trusted user ID	Yes	No	No	No	No	Trusted computer base, avoiding 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	An authorization rule is used to perform access checks for authenticated user IDs against lists of services defined within the rule. This feature is available on UNIX and Windows using EntireX Security on these platforms. Authorization rules can be stored in the Broker attribute file or in an LDAP repository. See *Authorization Rules*.
SSL/TLS	(1)	Yes	Yes	No	(2)	Industry standard encryption mechanism. See ***SSL/TLS and Certificates with EntireX***.

Notes:

1. Establishing an SSL/TLS connection on z/OS is done with IBM's Application Transparent Transport Layer Security (AT-TLS)
2. Establishing an SSL/TLS connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS).

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.



7 EntireX Security under z/VSE

This chapter introduces EntireX Security under z/VSE 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/VSE*.

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	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	An authorization rule is used to perform access checks for authenticated user IDs against lists of services defined within the rule. This feature is available on UNIX and Windows using EntireX Security on these platforms. Authorization rules can be stored in the Broker attribute file or in an LDAP repository. See *Authorization Rules*.
SSL/TLS	(1)	Yes	Yes	No	(2)	Industry standard encryption mechanism. See ***SSL/TLS and Certificates with EntireX***.

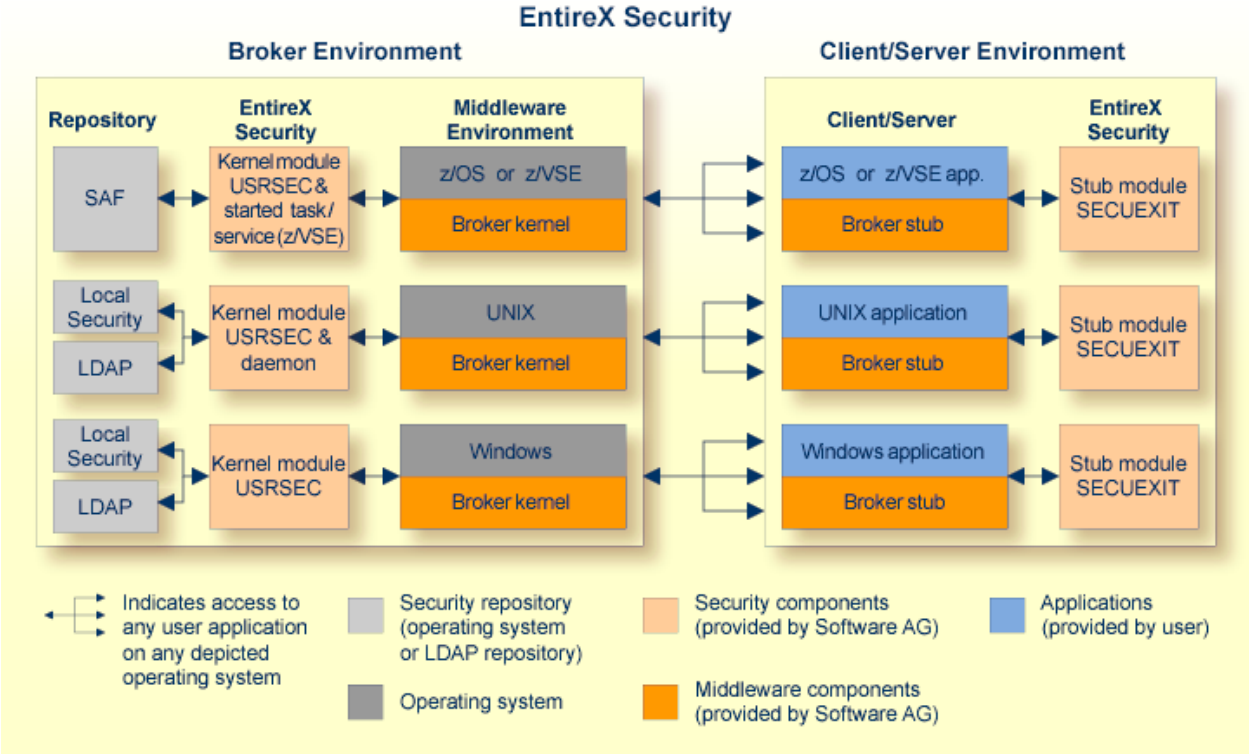
Notes:

1. Establishing an SSL/TLS connection on z/OS is done with IBM's Application Transparent Transport Layer Security (AT-TLS)

2. Establishing an SSL/TLS connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS).

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

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.

Configuration Options for Broker

This section describes the parameters for configuring EntireX Security under z/VSE. You may either accept or modify the default settings which are specified in the Broker attribute file `DEFAULTS=SECURITY`. See also *Security-specific Attributes* and *Operator Commands*.

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.

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

The following parameter can be used to modify the functionality of EntireX Security:

`SECURITY-LEVEL=AUTHENTICATION` User authentication is performed but without any resource authorization (the normal default operation).

8 SSL/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/Transport Layer Security (SSL/TLS) and Certificates within an EntireX context. The term “SSL” in this chapter refers to both SSL and TLS.

See also ***Which EntireX Security Solution?*** and *Running Broker with SSL/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).

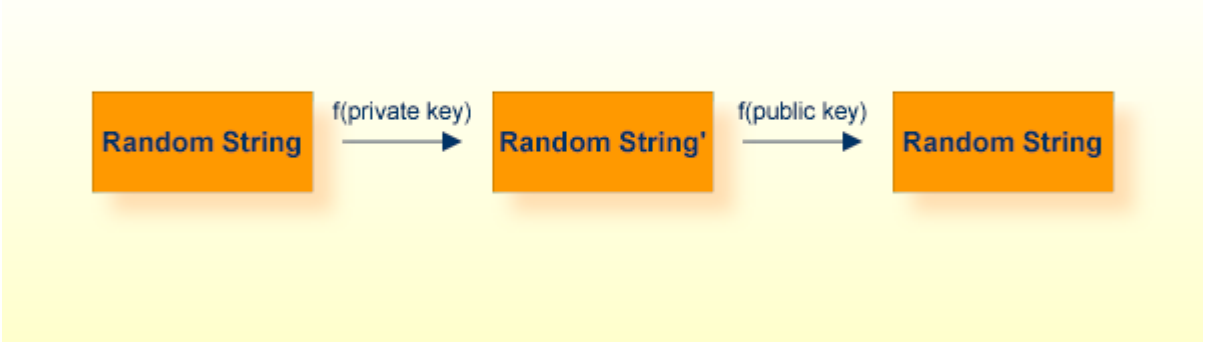
In addition to the above, a certificate also contains a random number that is called the subject's public key. Together with this public key, the subject must also be in possession of a private key. As their names suggest, the public key can be viewed by anyone, whereas the private key must be strictly 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 at least 2048 bytes of random data. As humans are rather limited in their ability to "generate" random data, we suggest using the OpenSSL tool for this task (see *Creating Certificates with OpenSSL (z/OS, UNIX, Windows)* below).
- On Windows systems, the seed is automatically taken.

SSL/TLS Sample Certificates Delivered with EntireX

Certificates play an important role with SSL. The term "SSL" in this section refers to both SSL and TLS. In order to use SSL as the transport method for EntireX, you need to have certificates available at various locations and for various purposes. The sample certificates come as two types: a trust store (containing a public key), and a keystore (containing a private key). EntireX provides the following default certificates for preliminary test purposes:

- Default Certificates for z/OS

We strongly recommended you create your own certificates. See below for how to create your own certificates with ***OpenSSL*** and *keytool*.

Default Certificates for z/OS

After the installation process, you will find certificates in the data set EXX101.CERT ready to use for preliminary testing of the SSL transport:

Certificate	Description	Notes
APPP12	No keys can be stored directly in RACF. The pkcs12 format member APPP12 was generated as a container for the necessary keys and the APPCERT member. The password to unlock this private key is ExxAppPkcs12.	1
CACERT	The CA certificate. This certificate can be used to verify the application certificate. See ***Using SSL/TLS with EntireX Components***.	2
CAKEY	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).	
APPCERT	To be used as the SSL server certificate. If your SSL server is EntireX Broker, see SSL-specific broker attribute KEY-STORE. This certificate is signed with the private key within CAKEY.	
APPKEY	The private key of the application certificate. The password to unlock the key is ExxAppKey. If your SSL server is EntireX Broker see SSL-specific broker attributes KEY-FILE and KEY-PASSWD-ENCRYPTED.	

Notes:

1. See also the README with step-by-step description for setting up an environment that enables an SSL-secured communication with a mainframe Broker and certificates stored in RACF.
2. To allow for multiple CAs, import multiple times the various CA certificates into the keystore.

Default Certificates for UNIX and Windows

After the installation process, you will find certificates in directory *etc* ready to use for preliminary testing of the SSL transport.

Certificate	Description	Notes
`ExxCACert.pem`	The CA certificate. This certificate can be used to verify the application certificate. Use the SSL parameter `trust_store`. See *Using SSL/TLS with EntireX Components*.	1
`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. If your SSL server is EntireX Broker, see SSL-specific broker attribute `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`. If your SSL server is EntireX Broker see SSL-specific broker attributes `KEY-FILE` and `KEY-PASSWD-ENCRYPTED`.	

Notes:

1. To allow for multiple CAs, concatenate all of the CAs' .pem files into a single new .pem file.

Default Certificates for Java

After the installation process, you will find certificates in *etc* directory for preliminary testing of the SSL transport:

Certificate	Explanation	Notes
`ExxCACert.jks`	The truststore containing the default CA certificate. Use SSL parameter `trust_store`. See *Using SSL/TLS with EntireX Components*.	1
`ExxJavaAppCert.jks`	The keystore containing the application certificate. The password to unlock this container is `ExxJavaAppCert` (use SSL parameters `key_store` and `key_passwd` for Java).	

Notes:

1. To allow for multiple CAs, import multiple times the various CA certificates into the keystore.

SSL/TLS Parameters for Broker as SSL Server (One-way SSL)

The term “SSL” in this section refers to both SSL and TLS. EntireX clients and servers are always SSL clients. The SSL server can be either the EntireX Broker, EntireX Broker SSL Agent or direct RPC in webMethods Integration Server (IS inbound).

SSL usually requires a certificate on the SSL server side of a communication. In order to validate the certificate, the SSL client needs to accept the issuer of the server certificate, that is, it needs to

trust the same instance that the certificate has signed. (Customs do not trust your passport - which could be forged - but instead verify its authenticity electronically!) If you are using EntireX Broker as your SSL server, use the following SSL-specific broker attributes:

Broker Attribute	Description
KEY-STORE	The server certificate is specified using the broker attribute KEY-STORE.
KEY-FILE	The appropriate private key is found using the broker attribute KEY-FILE.
KEY-PASSWD-ENCRYPTED	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 with the broker attribute KEY-PASSWD-ENCRYPTED.

The SSL 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. For this purpose, specify SSL parameter trust-store (see below) with the EntireX client or server. Checking the SSL server certificate by an SSL client is also known as *one-way* SSL.

SSL/TLS Parameters for SSL Clients

SSL Parameter	Description
trust_store	The trust_store parameter is mandatory. It specifies the file name of a keystore that must contain the list of trusted certificate authorities for the certificate of the SSL server. By default a check is made that the certificate of the SSL server is issued for the hostname specified in the Broker ID. The common name of the subject entry in the server's certificate is checked against the hostname. If they do not match, the connection will be refused. This check can be disabled by specifying SSL subparameter verify_server=no.
verify_server	Possible values: yes Default. The common 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 broker attribute KEY-STORE):

SSL Parameter	Description
	`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 broker attribute `TRUST-STORE`). The default application certificate (see ***SSL/TLS Sample Certificates Delivered with EntireX***) is issued to "`localhost`". This enables you to use a Broker ID of "`localhost`" together with `verify_server=y`.
`key_store` `key_passwd`	If the SSL server requests a client certificate (known as *two-way* SSL; `verify_client=yes` is defined in the configuration of the SSL server) two additional parameters have to be specified: `key_store` and `key_passwd`. This keystore must contain the private key of the SSL client. The password that protects the private key is specified with `key_passwd`. The ampersand (&) character cannot appear in the password.

How you provide SSL parameters depends on the EntireX component in use. See table ***Using SSL/TLS with EntireX Components*** below for platform and language-specific information. SSL parameters are separated by ampersand (&).

Using SSL/TLS with EntireX Components

This table provides references to available SSL documentation. Select the RPC or ACI components in use from column **SSL Client** and the communication partner such as EntireX Broker, Direct RPC, etc. from column **SSL Server**:

	SSL Client	SSL Server
	In an SSL context, SSL clients are ■ RPC clients and RPC servers ■ EntireX Adapter service and EntireX Adapter listener ■ Bridge components ■ ACI clients and ACI servers	In an SSL context, SSL servers are ■ EntireX Broker ■ EntireX Broker SSL Agent ■ *Direct RPC* in the EntireX Adapter documentation
RPC-based Components	■ For RPC clients generated by a wrapper, see *Using SSL/TLS* (C | COBOL | .NET | Java | PL/I). ■ For webMethods Integration Server, see *Support for SSL/TLS* in the EntireX Adapter documentation.	■ *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation ■ *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation

	SSL Client	SSL Server
	■ For RPC servers, see *Using SSL/TLS with the RPC Server* in the platform-specific administration or RPC server documentation. ■ For Bridge components, see *Using SSL/TLS* in the respective documentation section.	■ *Configuring Direct RPC* in the EntireX Adapter documentation
ACI-based Programming	■ For ACI clients and ACI servers, see *Using the Broker ACI with SSL/TLS* (Assembler | C | COBOL | Java | PL/I) of the programming language in use ■ For webMethods Integration Server, see *Support for SSL/TLS*	■ *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation ■ *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation
Administration	■ For `ETBCMD`, see *Using SSL/TLS* in section *Broker Command-line Utilities* in the respective section of the documentation ■ For `ETBINFO`, see *Using SSL/TLS* in section *Broker Command-line Utilities* in the respective section of the documentation	*Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation

SSL/TLS Certificate Creation and Handling

This section covers the following topics:

Creating Certificates with OpenSSL (z/OS, UNIX, Windows)

This section contains step-by-step instructions on how to create your own certificates. The OpenSSL tool is installed together with EntireX and can be found in directory *<install_root>/common/security/openssl/bin*.

To set up all necessary paths when working with the OpenSSL tool

■ Call the installed `tlsenv` script, which is provided in the following locations:

- Under UNIX: *<install_root>/common/security/openssl/extras/tlsenv.sh*. Source this once with the `dot` command in the POSIX shell (bash, ksh, etc.) where the OpenSSL tool will be used.
- Under Windows: *<install_root>\common\security\openssl\extras\tlsenv.bat*. Call this once in the command line interpreter window (cmd.exe) where the OpenSSL tool will be used.

Note: Certificates adhere to a standard format and can also be created with other tools; OpenSSL is installed 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* with a text editor 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* with at least 2048 bytes of random data in your new directory. You can use the OpenSSL tool to generate this file:

```
openssl rand 2048 > .rand
```

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. Take care not to mix filename separators: Always use the UNIX-style forward slash "/", even on Windows.

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 to point to the same file.)
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).
8. Review the `req_distinguished_name` section and fill in the *_default variables, if sensible. Empty defaults will be prompted for.

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:4096 -x509 -keyout <NewCAKey.pem> -out ↩
<NewCACert.pem> -days 365
```

Do not forget the passphrase for the key file! You will need it whenever a new certificate is generated.

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:2048 -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 (Java)

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
```

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
```

Importing Certificates into RACF (z/OS)

This section applies to operating system z/OS only.

To import certificates into RACF

1 Create a certificate with OpenSSL. See *Creating Certificates with OpenSSL (z/OS, UNIX, Windows)*.

2 Create the PKCS#12 import format for RACF. Enter the following command to create a file containing the application certificate and application key files for import into RACF:

```
openssl pkcs12 -export -inkey <EXXAppKey.pem> -in <EXXAppCert.pem> -certfile ↩
<EXXCACert.pem> -out <EXXPkcs12.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. You can use FTP to transfer the output file in binary mode to the IBM host.

3 Import certificates and private keys with `RACDCERT` into RACF. See readme file EXX101.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 constraints. (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 probably necessary to modify the LDAP settings. For example, to allow use of a self-signed certificate in OpenLDAP, the client needs access to the CA's certificate. Add the following line to file */etc/openldap/ldap.conf*:

```
TLS_CACERT <YourCaCert.pem>
```

9 Sample Security Exits for Broker Security

Sample security exits are a user-written security solution for use only in exceptional processing situations. Example: If your organization wants to access its own user-written security system when operating EntireX Broker.

Note: See *Using Sample Security Exits for Broker Security* in the EntireX Broker documentation. This describes implementation issues and how to use sample security exits on the operating where Broker executes.

See also *Which EntireX Security Solution?*

Sample Security Exits as Alternative Security Solution

Software AG intends sample security exits for Broker security to be only an alternative to **EntireX Security**, which is Software AG's standard security solution. 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.

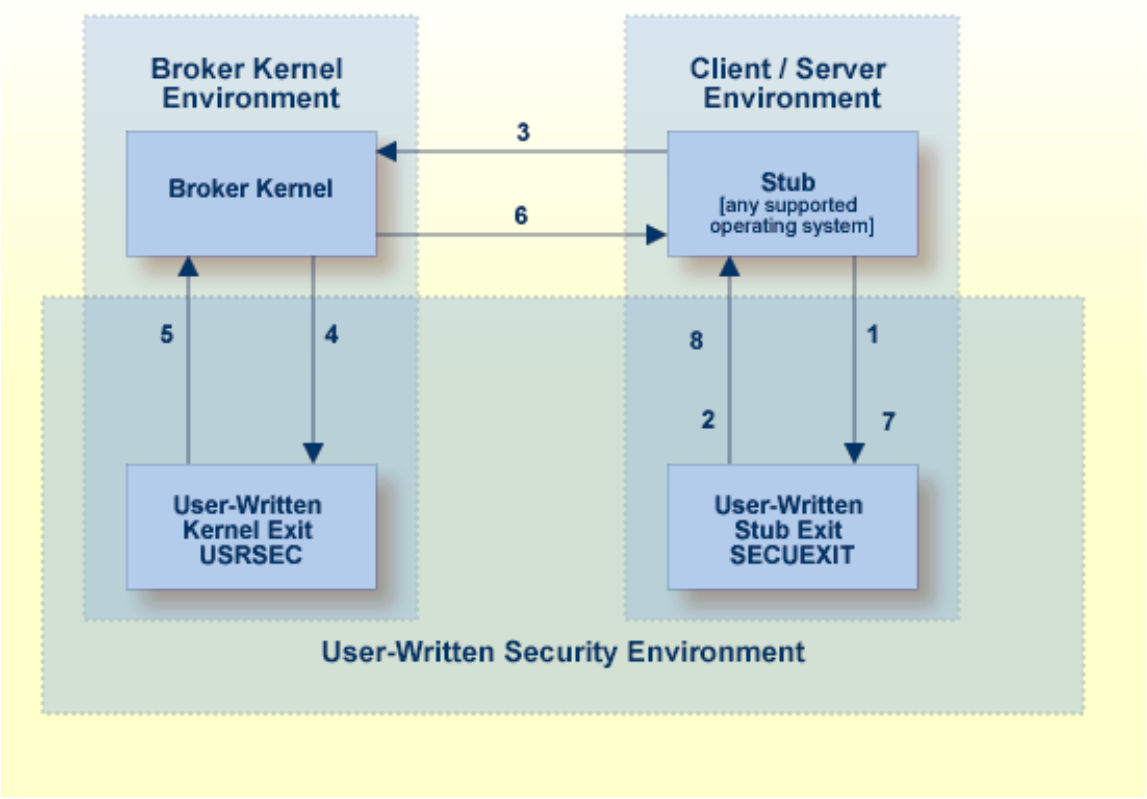
Most organizations that use Software AG's EntireX choose **EntireX Security** instead of sample security exits for Broker security. If your organization is deploying distributed computer systems encompassing mainframe, UNIX and Windows environments, you will use EntireX Security instead of sample security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

Lightweight USRSEC

For compatibility with previous versions (API level 3 and below), a “lightweight” security exit is supplied as load module USRSEC in library EXX101.LOAD for Broker under z/OS. This “lightweight” version of USRSEC performs authentication only against RACF, CA ACF2 and CA Top Secret. It does not include the full functionality of the standard EntireX Security installation of USRSEC (e.g. resource authorization, etc.). The “lightweight” version of USRSEC does not require any security components, i.e. SECUEXIT, to be installed into the application (stub) environment. If you are using ACI version 1 to 7 and you intend to use the “lightweight” version of USRSEC, please ensure you do not have any security components installed into the application (stub) environment.

Implementation of Sample Security Exits

Sample security exits are a user-written security solution for use only in exceptional processing situations. The diagram below depicts the data flow which users can implement in their own user exits for Broker security.



Note: To activate your user-written security exits, specify `SECURITY=YES` in the broker attribute file.

Description of Steps in Data Flow

1. Broker stub calls security exit *`SECUEXIT`*, if present.
2. Security exit *`SECUEXIT`* encrypts the password and optionally the application data. See ***Encryption / Decryption***. `SECUEXIT` accesses the ACI control block and the `SEND/RECEIVE` buffers. `SECUEXIT` returns call to the broker stub.
3. Broker stub communicates the call to the broker kernel.
4. Broker kernel calls security exit *`USRSEC`* for each specific event type:

- Create security context for user; authentication is usually performed in this event. See *Authentication*.
- Destroy security context for user.
- Perform authorization for server to register a service. See *Authorization*.
- Perform authorization for client to send request.
- Optionally perform encryption of application data. See note below.
- Optionally perform decryption of application data. See note below.
- Perform optional processing if a user acquires a new physical user ID. Re-authentication can also be performed.
- Perform optional processing if the value of a user's ACI security token changes. Re-authentication can also be performed.

5. Security exit `USRSEC` passes call to broker kernel.
6. Broker kernel communicates the call to the broker stub of the partner application.
7. The broker stub calls `SECUEXIT`. `SECUEXIT` determines whether decryption is to be performed, if correspondingly coded by user.
8. Security exit `SECUEXIT` returns call to broker stub.

Notes:

1. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.
2. See *Using Sample Security Exits for Broker Security*. This section describes implementation issues and how to use sample security exits on the operating system where Broker executes.

Glossary of Terms

- Authentication
- Authorization
- Broker and Kernel
- Broker Stub
- Encryption / Decryption

- Exits

Authentication

Authentication verifies whether the identity specified by the user ID in the ACI control block is the actual identity. Authentication is usually performed by checking the user's ID and password against a security system. The details of this check are specific to the specific operating system and security system.

Authentication is not needed with every call. It is required when the user's security context is created within the Broker kernel; it is also required, optionally, if the user's physical user ID or ACI security token changes.

Authorization

Authorization can be 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;

Broker and Kernel

It is the location of the Broker kernel that determines the point at which the authentication and authorization checks can be performed. ***Authentication*** and ***Authorization*** can be performed in the kernel exit `USRSEC`.

See *List of Components per Platform* for where 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 stub. Stubs are installed on the client and the server side.

See *Transport: Broker Stubs and APIs* 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.

In the case of user-written security exits, encryption/decryption can be implemented in

- the stub security exits (SECUEXIT or ETBUPRE / ETBUEVA)
- the kernel security exit (USRSEC)

Note: We recommend *not* implementing your own encryption/decryption mechanism. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See ***SSL/TLS and Certificates with EntireX***.

Exits

- **Kernel Exit** `USRSEC`
  USRSEC is the name of the security exit which is invoked if `SECURITY=YES` is specified in the attribute file.

  In the case of user-written security exits, this exit will include functionality for authentication, authorization and optionally encryption/decryption [(2)].
- **Stub Exit** `SECUEXIT`
  SECUEXIT is the stub security exit for use with the broker stub for transports SSL and TCP. See *Implementing Security Exits* z/OS | UNIX | Windows This module is executed during a Broker command if `SECUEXIT` is present in the path of execution.

  In the case of user-written security exits, this exit can optionally include functionality for encryption/decryption [(2)].
- **Stub Exit** `ETBUPRE /ETBUEVA`
  `ETBUPRE` / `ETBUEVA` are the stub security exits for use with the broker stub under z/OS for transport NET. See *Implementing Security Exits under z/OS*. These modules are executed during a Broker command if they are linked to the stub.

Notes:

1. See also *List of Components per Platform* for where Broker kernel and Broker Stub are supported.
2. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See ***SSL/TLS and Certificates with EntireX***.