Using Sample Security Exits for Broker Security

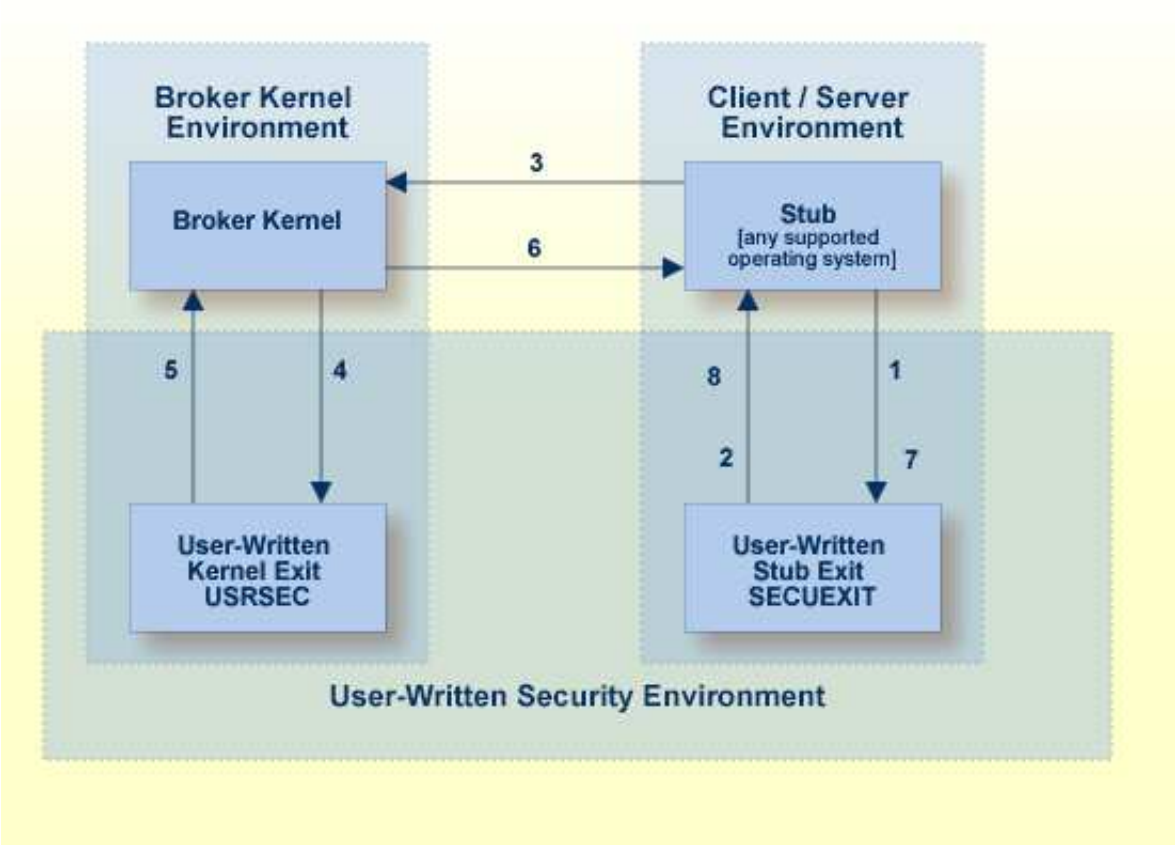
This page describes implementation issues and how to use sample security exits in EntireX Broker. It assumes you are familiar with EntireX Broker from both an administrative and an application perspective, and with the ACI programming interface in particular. See *Introduction to ACI-based Programming*.

This chapter covers the following topics:

- Overview of Security Data Flow
- Prerequisites for Running EntireX Broker in a Secure Environment
- General Security Recommendations
- Writing Security Exits
- Security-Related Parameters
- Programming Broker Stub Exits
- Layout of Security Parameter Block ETB_SECPAR
- Layouts of Type-dependent Security Parameter Blocks

Overview of Security Data Flow

The diagram shows a data flow for sample security exits, with Broker Kernel located, for example, on z/OS. See also *Description of Steps in Data Flow*.



Prerequisites for Running EntireX Broker in a Secure Environment

To run EntireX Broker in a secure environment, the following prerequisites must be met:

- The security system in the EntireX Broker kernel must be activated by setting `SECURITY=YES` in the broker attribute file.
- The security routines must be accessible to the Broker. The method you use to achieve this depends on the operating system where your user-written USRSEC is implemented.

Note:
EntireX Broker will not start if `SECURITY=YES` is specified but the security routines cannot be activated.

General Security Recommendations

If you run a secure environment, we recommend you performing an explicit LOGON with the `AUTOLOGON=NO` definition in the attribute file. All security violations are logged to the EntireX Broker log file.

- Implementing the Kernel Security Exit under z/OS

- Implementing Security for Broker Stubs under z/OS
- Implementing Security Exits for Broker Stubs on UNIX
- Implementing Security Exits for Broker Stubs on Windows

Implementing the Kernel Security Exit under z/OS

To implement the kernel security exit under z/OS

1. Write the exits USRSEC. The code must always be reentrant and reusable.
2. The kernel security exit USRSEC is loaded automatically during startup of Broker. Use module and entry name USRSEC for this exit. A security module sample source is delivered with the ETB source library.
3. Under z/OS, link the exit as reentrant and reusable.
4. Ensure that the security exit is accessible in the Broker STEPLIB.

Implementing Security for Broker Stubs under z/OS

To implement security exits for Broker stubs under z/OS

1. Write the stub security exits ETBUPRE and ETBUEVA. The code must always be reentrant, except for batch, where the code must be reusable.
2. Link these exits ETBUPRE and ETBUEVA to the stub of the target application. The stub contains weak externals for both entries.

Implementing Security Exits for Broker Stubs on UNIX

To implement security exits for Broker stubs under UNIX

1. Write your own *usrsec.c* and *secuexit.c*, based on the samples delivered with EntireX.
2. Build your own *secuexit.s*[o|l] and *usrsec.s*[o|l], using the provided makefiles. (A sample makefile, makexa, is provided.)
3. Ensure that *usrsec.s*[o|l] is made available to the Broker kernel at execution time. The attribute file parameter SECURITY-PATH must be used to specify the location of *usrsec.s*[o|l].
4. Ensure that *secuexit.s*[o|l] is made available to the application in the same directory as the Broker stub.

Implementing Security Exits for Broker Stubs on Windows

To implement security exits for Broker stubs under Windows

1. Write your own *usrsec.c* and *secuexit.c*, based on the samples delivered with EntireX.

2. Build your own *secuexit.dll* and *usrsec.dll*, using the provided makefiles.

Writing Security Exits

This section describes how to write your own security exits. It describes the interfaces, indicates what can be modified and what has to be done within an exit. It also provides some helpful tips.

This section covers the following topics:

- Requirements
- Error Checking for Incomplete Security Installation

Requirements

You must provide the following functions:

- The Preparation exit `etbupre()` and the Evaluation exit `etbueva()` for the Broker stub. These two functions are linked statically to the Broker stub routines.
- The Kernel exit `usrsec()` which is loaded by the kernel. This exit is more generic than the other two. It is called with the function that has been performed and a function-dependent Broker ACI control block that provides all the necessary information. This function is loaded dynamically by EntireX Broker during startup. One parameter of the kernel exit is the function that is performed.

The functions map to the exit type is as follows:

Exit Type	Function	Function to perform
Authentication exit	ETB_SEC_LOGON	Checks authentication for the user.
	ETB_SEC_LOGOFF	Release user-specific information if necessary.
	ETB_SEC_NEWPUID	Application call with different physical USER ID.
	ETB_SEC_NEWST	Application call with a different SECURITY TOKEN
Authorization exit	ETB_SEC_SEND	Check whether user is allowed to use the addressed service.
	ETB_SEC_REGISTER	Check whether the user is allowed to offer that service.
Encryption exit	ETB_SEC_ENCRYPT	Encrypt the given data.
	ETB_SEC_DECRYPT	Decrypt the given data.

In the following text, "encryption" or "authentication" exit refers to the functions listed above.

Error Checking for Incomplete Security Installation

With `ACI_VERSION=4` or above, the security configuration of the caller's stub is checked against the security configuration of the broker kernel. The request will be rejected with the error message `00200379 - API: Inconsistent Security Installation`, if security

- is present in the stub and it is not present in the kernel;

  or

- is not present in the stub and it is present in the kernel.

Note:
If you have written your own security - instead of using *Security Solutions in EntireX* - and it is implemented only on the kernel, you will have to add a dummy security exit to the stub.

Security-Related Parameters

The following security-related parameters are provided. These are fields in the *Broker ACI Fields*:

- USER-ID
- PASSWORD
- SECURITY-TOKEN
- CLIENT-UID
- ERROR-CODE
- ERROR-TEXT
- KERNELSECURITY
- ENCRYPTION-LEVEL

USER-ID

The USER ID is defined by the application. It is available in all ACI exits as well as in the kernel exits, except the encryption and decryption exits. Theoretically the preparation exit can be used to retrieve the login name by an operating system specific mechanism. This would allow a user identification without the application being involved. See the description of the `USER-ID` field in the Broker ACI control block.

PASSWORD

The `PASSWORD` is defined by the application. It is available in all ACI and kernel exits except the encryption exit. The `PASSWORD`, if provided by the application in plain text, should be encrypted in the Preparation exit before sending it across insecure network connections. If the `PASSWORD` is needed in the application again, it must be decrypted after receipt in the Evaluation exit. The authentication exit must ensure that the `PASSWORD` is properly decrypted if necessary before sending it to an external security system.

The EntireX Broker provides minimal encryption of the `PASSWORD` field, that is, the field is not transmitted in plain text. If your environment requires absolute security, however, you will need to provide both Broker stub and EntireX Broker kernel exits to perform encryption and decryption. See the description of the `PASSWORD` field in the Broker ACI control block.

SECURITY-TOKEN

The SECURITY TOKEN can be created by the application and sent to EntireX Broker. That allows for a kind of credential algorithm. The security token is passed to all kernel exits and can therefore contain security information which is also important for the authorization and encryption exits. The SECURITY TOKEN can be altered in the authentication exit to provide a context token for that application and that session. It is transmitted back to the application and can then be used in all subsequent calls. For each subsequent call, the EntireX Broker checks whether the SECURITY TOKEN is identical to the one returned from the last call to the authentication exit, which could be the ETB_SEC_LOGON, the ETB_SEC_NEWPUI or the ETB_SEC_NEWST function. After an ETB_SEC_LOGOFF call, a subsequent call is always a ETB_SEC_LOGON call. See the description of the SECURITY-TOKEN field in the Broker ACI control block.

CLIENT-UID

CLIENT-UID is returned to a server application after a RECEIVE and contains the user ID of the sending client. This allows for further security checks (logging, separate checks, etc.). See the description of the CLIENT-UID field in the Broker ACI control block.

ERROR-CODE

All security-related ERROR CODEs start with the ERROR CLASS 0008. The following four digits in the ERROR CODE can be assigned by any exit if a security violation occurs. These digits only reach the application if the current operation is aborted by the security exit with a security violation, i.e. an appropriate return code. See ERROR-CODE under *Broker ACI Fields*.

ERROR-TEXT

The security exits can also pass an error message back to the application. This error text must not be longer than 40 bytes.

KERNELSECURITY

See KERNELSECURITY under *Broker ACI Fields*.

ENCRYPTION-LEVEL

See ENCRYPTION-LEVEL under *Broker ACI Fields* or *Encryption* under *Writing Applications using EntireX Security*.

Programming Broker Stub Exits

The exits in the stub have the following interface:

- Preparation Exit
- Evaluation Exit
- Programming the Kernel Exit Routine

Preparation Exit

Synopsis

```
int etbupre (ETBCB *pEtbCb,
    void *pSendBuf,
    void *pReserved,
    char *pErrText)
```

Parameter	Format	Direction	Description
Address of ETBCB	Pointer to ETBCB control block.	I/O	ETBCB's user_id and password are used to generate the credential which will be saved in field security_token for function LOGON.
Address of send buffer	void pointer	I/O	Send buffer supplied by caller, only available for function SEND, length of send buffer is member of ETBCB.
Reserved	void pointer	I	Must be NULL.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text that will be returned by the stub routine.

Return value

0 (okay) or non-zero (error)

The real error code must be written to the Broker control block as an 8-byte character array (without trailing 0 byte!) . The error class 0008 (security / encryption error class) is reserved for all errors in function `etbupre`. The error number is user-defined. Additionally, the error text can be returned to the user in the error text array.

Required Actions in the Exit

If no data encryption is desired, no action is required.

Recommended Actions in the Exit

- Generate a credential if function is `LOGON` and move it to the field security_token.
- Encrypt the send buffer if function is `SEND`. The encryption process must not change the length of the buffer.

The exit gets control for each function of ACI version 2 and above. The exit must exist.

Evaluation Exit

Synopsis

```
int etbueva (ETBCB *pEtbCb,
    void *pRecBufEncr,
    void *pReserved,
    char *pErrText)
```

Parameters

Parameter	Format	Direction	Description
Address of ETBCB	Pointer to ETBCB control block.	I/O	ETBCB's security token is used for data decryption.
Address of receive buffer.	void pointer	I/O	Receive buffer provided by EntireX Broker. Only available for functions `RECEIVE` and `SEND WAIT=X` (implicit receive). Length of receive buffer is member of ETBCB.
Reserved	void pointer	I	Must be NULL.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text which will be returned by the stub routine.

Return Value

0 (okay) or non-zero (error)

The real error code must be written to the Broker control block as an 8-byte character string (without trailing 0 bytes!). The error class 0008 (security / encryption error class) is reserved for all errors in function `etbueva`. The error number is user-defined.

In addition, the error text can be returned to the user.

Required Actions in the Exit

If no data decryption is wanted, no action is required.

Recommended Actions in the Exit

- Decrypt the receive buffer if functions are `RECEIVE` or `SEND` with `WAIT`. The decryption process must not change the length of the buffer.

The exit gets control for each function of ACI Version 2 and above. The exit must exist.

Use of a Single Send/Receive Buffer

A single send/receive buffer is used to perform encryption in place. This means that encrypted data is provided in the send buffer. After the completion of a send/nowait command, the application should ignore the contents of the send buffer, i.e. the encrypted data.

Programming the Kernel Exit Routine

All authentication, authorization, encryption and decryption exits are combined within one exit module named `USRSEC`. The various security checks are indicated by a type parameter in the CALL interface. `USRSEC` is provided with EntireX Broker as the default security exit. It is invoked if `SECURITY=YES` is set in the attribute file. Prior to EntireX, the `USRSEC` exit was available only with the SAF Gateway security package.

The general syntax of this user exit is defined as follows:

Synopsis

```
long usrsec (ETB_SECPAR *pParSec,
    void  *pVarious,
    char  *pErrText,
    char  *pWorkArea,
    long  lWorkArea)
```

Parameters

Parameter	Format	Direction	Description
Address of security parameter block	Pointer to structure ETB_SECPAR	I	Contains the security type flag.
Address of type-dependent security parameter block	void pointer	I	See control block structures `ETB_SECPAR_<type>`.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text which will be returned to the user.
Address of work area	char pointer	O	Volatile work area.
Length of work area	long integer value	I	Size of the work area in number of bytes.

Return Value

0 (okay) or user-defined error number

Error class 0008 (security / encryption error class) and the error number will be returned to the user. In addition, the error text can be returned to the user.

Layout of Security Parameter Block ETB_SECPAR

```
typedef struct _ETB_SECPAR
{
 unsigned long vers;    /* I: interface version number   */

#define ETB_SEC_VERSION_1 (1) /* ETBCB version1 (no stub exits)*/
#define ETB_SEC_VERSION_2 (2) /* ETBCB version2 (stub exits)   */

 unsigned long type;          /* I: security type             */
#define ETB_SEC_LOGON     (1) /* user authentication (LOGON)   */
#define ETB_SEC_LOGOFF    (2) /* destroy user env (LOGOFF)     */
#define ETB_SEC_REGISTER  (3) /* authorization for REGISTER    */
#define ETB_SEC_SEND      (4) /* authorization for SEND        */
#define ETB_SEC_ENCRYPT   (5) /* encrypt message (RECEIVE)     */
#define ETB_SEC_DECRYPT   (6) /* decrypt message (SEND)        */

 char id[3];                /* I:ID e.g. W01 for worker task 1 */
 void *pNetAddr             /* I: pointer to network address   */
} ETB_SECPAR;
```

Parameter	Direction	Description
version	I	The interface version number.
type	I	Unsigned long type.
char id	I	Identifier for the task.
pNetAddr	I	Pointer to the network address. A TCP/IP address contains 0001 in the first two bytes, followed by the actual address in the next four bytes. If the pointer is 0000, there is no address.

Layouts of Type-dependent Security Parameter Blocks

This section describes the following security parameter blocks:

- DECRYPT
- LOGOFF
- LOGON
- NEWST
- REG
- SEND

```
typedef struct _ETB_SECPAR_

 /* decrypt message of sender           */
{
 unsigned char *pSecTok;                /* I:   Security Token            */
 unsigned char *pBufECry;               /* I:   Encrypted buffer          */
 unsigned char *pBufDCry;               /* O:   Decrypted buffer          */
 long *plBufECry;                       /* I:   length of encrypted buffer*/
 long *plBufDCry;                       /* I/O: length of decrypted buffer*/
} ETB_SECPAR_DECRYPT;

typedef struct _ETB_SECPAR_

 /* encrypt message for receiver        */
{
 unsigned char *pSecTok;                /* I:   Security Token            */
 unsigned char *pBufDCry;               /* I:   Decrypted buffer          */
 unsigned char *pBufECry;               /* O:   Encrypted buffer          */
 long *plBufDCry;                       /* I:   length of decrypted buffer*/
 long *plBufECry;                       /* I/O: length of encrypted buffer*/
} ETB_SECPAR_ENCRYPT;

typedef struct _ETB_SECPAR_

 /* destroy security environment         */
{
 char *pUid;                            /* I: UserID                      */
 unsigned char *pSecTok;                /* I: Security Token              */
 unsigned long *pnSecHndl;              /* I: Security handle             */
} ETB_SECPAR_LOGOFF;
```

```
typedef struct _ETB_SECPAR_

 /* user authentication                   */
{
 char *pUid;                     /* I:   UserID                          */
 unsigned char *pPasswd;         /* I:   Password (encoded)             */
 unsigned char *pNewPasswd;      /* I:   New Password (encoded)         */
 unsigned char *pSecTok;         /* I/O: Security Token                  */
 unsigned long *pnCode;          /* I:   Character set of user           */
 unsigned long *pnSecHndl;       /* O:   Security handle                 */
} ETB_SECPAR_LOGON;

typedef struct _ETB_SECPAR_

 /* reauthentication due to new         */
                                 /* physical user ID                     */
{
 char *pUid;                     /* I:   UserID                          */
 unsigned char *pPasswd;         /* I:   Password (encoded)             */
 unsigned char *pNewPasswd;      /* I:   New Password (encoded)
*/
 unsigned char *pSecTokOld;      /* I:   Previously used security token */
 unsigned char *pSecTokNew;      /* I/O: New security token              */
 unsigned long *pnCode;          /* I:   Character set of user           */
 unsigned long *pnSecHndl;       /* I/O: Security handle                 */
} ETB_SECPAR_LOGON;

typedef struct _ETB_SECPAR_

   /* reauthentication due to new         */
                                 /* Sec. Tok.                            */
{
 char *pUid;                     /* I:   UserID                          */
 unsigned char *pPasswd;         /* I:   Password (encoded)             */
 unsigned char *pNewPasswd;      /* I:   New Password (encoded)         */
 unsigned char *pSecTokOld;      /* I:   Previously used security token */
 unsigned char *pSecTokNew;      /* I/O: New security token              */
 unsigned long *pnCode;          /* I:   Character set of user           */
 unsigned long *pnSecHndl;       /* I/O: Security handle                 */
} ETB_SECPAR_LOGON;

typedef struct _ETB_SECPAR_

   /* REGISTER authorization              */
{
 char *pUid;                     /* I:   UserID                          */
 unsigned char *pSecTok;         /* I:   Security Token                  */
 char *pSrvCls;                  /* I:   Server Class                    */
 char *pSrvName;                 /* I:   Server Name                     */
 char *pService;                 /* I:   Service                         */
 unsigned long *pnSecHndl;       /* I:   Security handle                 */
} ETB_SECPAR_REG;

typedef struct _ETB_SECPAR_

    /* SEND authorization                  */
{
 char *pUid;                     /* I:   UserID                          */
 unsigned char *pSecTok;         /* I:   Security Token                  */
 char *pSrvCls;                  /* I:   Server Class                    */
```

```
 char *pSrvName;                  /* I:   Server Name                  */
 char *pService;                  /* I:   Service                      */
 unsigned long *pnSecHndl;        /* I:   Security handle              */
} ETB_SECPAR_SEND;
```

Required/ Recommended Actions in the Exit (depending on Security Type)

Security Type	Required Action	Recommended Action	Note
ETB_SEC_ENCRYPT	Copy decrypted to encrypted buffer and set the length of encrypted buffer. This is necessary because exit is called whether the receive data has to be encrypted or not.	Encrypt receive data if needed.	The size of the buffer cannot be changed in this exit.
ETB_SEC_DECRYPT	Copy encrypted to decrypted buffer and set the length of decrypted buffer. This is necessary because exit is called irrespective of whether send data is encrypted or not.	Decrypt receive data if needed.	The size of the buffer cannot be changed in this exit.
ETB_SEC_LOGON		Decrypt the password and check combination of user ID and password against the security system. Generate a context token according to the credentials of the user and EntireX Broker. Create a security handle for a user session (e.g. ACEE on z/OS).	
ETB_SEC_LOGOFF	None	Delete the security handle of the user session.	
ETB_SEC_NEWPUID	None	An application has changed the physical user ID between two calls. If necessary, a new security token can be created.	
ETB_SEC_NEWST	None	For some reason, the security token of an application has changed and no longer matches the original. The security token should be recalculated and approved or the application should be rejected.	

Security Type	Required Action	Recommended Action	Note
ETB_SEC_REGISTER	None	Check whether user_id is authorized to offer the requested SERVICE (check security_token data if necessary).	
ETB_SEC_SEND	None	Check whether user_id is authorized to offer the requested SERVICE (check security_token data if necessary).	