CICS Transaction Gateway Security Exit Samples

Current Version: v1.0, April 6, 2005
Authors: Richard Burton, Simon Knights, Phil Wakelin, IBM UK, Hursley

Description

This SupportPac provides a series of sample security exits for the CICS Transaction Gateway. These samples illustrate how the security exit design can be used to either authenticate clients upon connection, or to encrypt flows between the CICS Transaction Gateway and remote Java clients. Three different security implementations are provided as follows:

1. IP address authentication - This sample illustrates a simple mechanism that can be used to allow only predetermined Java clients to connect to the Gateway daemon based upon the client's IP addresses.
2. Secret key authentication - This sample uses a shared secret key mechanism, which must be known by the Java client and by the Gateway daemon before a connection can be established.
3. JCE encryption - This samples uses Java Cryptography Extension (JCE) to encrypt the network flows between the Java client and Gateway daemon utilizing the AES symmetric encryption algorithm.

It is envisaged that the samples in this SupportPac can be used by Java application programmers to help them become familiar with the design of the CICS TG security user exits.

Skill level required

Systems Specialists and Application Programmers with Java programming skills.

Supplied files

- ce51.pdf - Documentation
- ce51.jar JAR file containing Java source and compiled byte code
- ipcheck.hosts - Sample IP address verification file for use with Gateway daemon
- license2 – directory for translated license files

1. IP address authentication sample

This sample illustrates a simple mechanism that can be used to allow only predetermined Java clients to connect to the Gateway daemon based upon the clients IP addresses. The server side user exit checks the value of the client's IP address flowed in the handshake, and verifies that this address is listed in the `ipcheck.hosts` file. If the client machine's IP address is listed as an authorised IP address, the handshake will be accepted and further ECI or EPI flows will proceed without any need for further authentication. Note that the `ipcheck.hosts` file is only loaded when the server side exit is first called, and the Gateway must be restarted in order to reload the list of valid IP addresses.

Supplied files

- IPCheckClient.java - Java client side source code
- IPCheckServer.java - Gateway daemon side source code
- IPCheckClient.class - Java client side compiled sample
- IPCheckServer.class - Gateway daemon side compiled sample
- ipcheck.hosts - Sample IP address verification file for use with Gateway daemon

Gateway daemon machine configuration:

(1) Edit the supplied `ipcheck.hosts` file to list which client IP addresses are authorized to connect to the Gateway. Each address should be specified in dotted decimal IP format on a separate line. Any line starting with a '#' symbol is treated as a comment, and one IP address should be specified per line.

(2) Put the `ipcheck.hosts` in the bin subdirectory of the CICS TG install. Alternatively the location of `ipcheck.hosts` can be modified using the Java system property *ipcheck* as documented below.

(3) Make sure that ce51.jar is listed in the CLASSPATH used by the CICS TG.

(4) If required, add the parameter *requiresecurity* to the CICS TG protocol handler definition in your CTG.INI, this will ensure that only Java clients using a security exit can connect. An example definition is
`protocol@tcp.parameters=connecttimeout=2000;dropworking;idletimeout=600000;pingfrequency=30000;port=2006;requiresecurity`

(5) Start the Gateway daemon.

NOTE: Optionally the name and location of the `ipcheck.hosts` file can be specified using the JVM directive *ipcheck*. JVM directives can be passed to the Gateway daemon by using: -j-D flags as follows:

```
>ctgstart -j-Dipcheck=/home/user/ipcheck.hosts
```

Or if using CTGBATCH on z/OS with CICS TG v6, JVM directives should by specified using the CTGSTART_OPTS in the STDENV file, for example:

```
CTGSTART_OPTS-j-Dipcheck=/home/user/ipcheck.hosts
```

Java client application usage:

(6) Add the ce51.jar to the CLASSPATH to be used on the client machine.
(7) Add the ctgclient.jar (or for JCA applications the cicseci.rar) to the CLASSPATH to be used on the client machine.
(8) Run the EciB1 sample, which ships with the CICS Transaction Gateway using the following parameters.

```
java com.ibm.ctg.samples.eci.EciB2
    jgate=<gatewayURL>
    jgateport=<jgate_port>
    server=<servername>
    prog0=<program>
    clientsecurity=com.ibm.ctg.samples.security.IPCheckClient
    serversecurity=com.ibm.ctg.samples.security.IPCheckServer
```

Note: The Java client application only needs access to the client side security class. However, the names of both the client side and server side security classes need to be specified to the client application.

If the client machine's IP address is not an authorised IP address, the following exception will be returned to the client application:

```
java.io.IOException: CTG6668E Initial handshake flow failed.
[java.io.IOException: CTG6670E Exception occurred in the Gateway
daemon. [java.io.IOException: Host 1.2.3.4 is not authorised to
connect.]]
```

2. Secret key authentication sample

This sample uses a shared secret key mechanism to authenticate the client and server to each other. The keys are strings which must be defined to the Java client and the Gateway daemon security exits before a connection can be established, and are specified using JVM system properties. The client side system property is called *clientuid* and the server side is called *serveruid*. The client's key is flowed to the Gateway daemon in the handshaking process. If the client key does not match the definition in the Gateway daemon the connection is rejected. In return the Gateway's key is sent back to the client, which then verifies that it matches its definition of the key. Note that the keys are only loaded when the server side exit is first called, and so the Gateway must be restarted in order to update the keys.

For ease of use, these values are best specified using the JVM parameters as detailed in the instruction below, however, they could also be specified programmatically inside the user application using the `System.setProperty()` method.

Supplied files

- SimpleAuthClient.java - Java client side source code
- SimpleAuthServer.java - Gateway daemon side compiled sample
- SimpleAuthClient.class - Java client side compiled sample
- SimpleAuthServer.class - Gateway daemon side compiled sample

Gateway daemon machine configuration:

(1) Make sure that the ce51.jar is listed in the CLASSPATH used by the CICS TG.

(2) If required, add the parameter *requiresecurity* to the CICS TG protocol handler definition in your CTG.INI, this will ensure that only Java clients using a security exit can connect. An example definition is
`protocol@tcp.parameters=connecttimeout=2000;dropworking;idletimeout=600000;pingfrequency=30000;port=2006;requiresecurity`

(3) Start the CICS Transaction Gateway, specifying a unique value for the secret keys, for example:

```
  >ctgstart -j-Dclientuid=clientsecret -j-Dserveruid=
serversecret
```

Or if using the CICS TG on z/OS and CTGBATCH add the following to your STDENV

```
 CTGSTART_OPTS=-j-Dclientuid=clientsecret -j-Dserveruid=
serversecret
```

Java client application usage:

(1) Add the ce51.jar to the CLASSPATH to be used on the client machine.
(2) Add the ctgclient.jar (or for JCA applications the cicseci.rar) to the CLASSPATH to be used on the client machine.
(3) Test the connection using the EciB1 sample, using the following parameters, specifying the client and server's keys using the following JVM system properties. .

```
java -Dclientuid=clientsecret -Dserveruid=serversecret
com.ibm.ctg.samples.eci.EciB2
    jgate=<gatewayURL>
    jgateport=<jgate_port>
    server=<servername>
    prog0=<program>
    clientsecurity=com.ibm.ctg.samples.security.SimpleAuthClient
    serversecurity=com.ibm.ctg.samples.security.SimpleAuthServer
```

Note: the client side only needs access to the client side security class. However the names of both the client side and server side security classes need to be specified to the client application.

If non matching values are specified for either of the keys an exception similar to the following will be returned to the client application:

```
CTG6670E Exception occurred in the Gateway daemon.
[java.io.IOException: Client UIDs do not match]]
```

3. JCE encryption sample

This sample uses Java Cryptography Extension (JCE) to encrypt network flows between the Java client and the Gateway daemon. The Java client first generates an RSA public and private key pair, and sends the public key to the Gateway daemon. The Gateway daemon then generates an AES secret key and encrypts this with the RSA public key sent by the client before flowing the encrypted secret key back to the Client. The Client decrypts the AES secret key with its RSA public key, and then uses the AES cryptographic algorithms for encryption and decryption of all subsequent ECI or EPI requests sent to the Gateway daemon.

Note, this sample provides similar payload encryption function to that provided within the SSL protocol. However, it does not have many of the other features of the SSL protocol, such as message authentication or the secure exchange of keys.

Supplied files

- CryptoClient.java - Java client side source code
- CryptoServer.java - Gateway daemon side source code
- CryptoClient.class - Java client side compiled sample
- CryptoServer.class - Gateway daemon side compiled sample

Gateway daemon machine configuration:

(1) Make sure that ce51.jar is listed in the CLASSPATH used by the CICS TG.
(2) If required, add the parameter *requiresecurity* to the CICS TG protocol handler definition in your CTG.INI, this will ensure that only Java clients using a security exit can connect. An example definition is
```
protocol@tcp.parameters=connecttimeout=2000;dropworking;idletimeo
ut=600000;pingfrequency=30000;port=2006;requiresecurity
```
(3) Start the CICS Transaction Gateway.

Java client application usage:

(1) Add the ce51.jar to the CLASSPATH to be used on the client machine.
(2) Add the ctgclient.jar (or for JCA applications the cicseci.rar) to the CLASSPATH to be used on the client machine
(3) Run the EciB1 sample, which ships with the CICS Transaction Gateway with the following parameters:

```
java com.ibm.ctg.samples.eci.EciB2
   jgate=<gatewayURL>
   jgateport=<jgate_port>
   server=<servername>
   prog0=<program>
   clientsecurity=com.ibm.ctg.samples.security.CryptoClient
   serversecurity=com.ibm.ctg.samples.security.CryptoServer
```

The subsequent ECI and EPI flows sent between the client application and the Gateway daemon will now be encrypted.

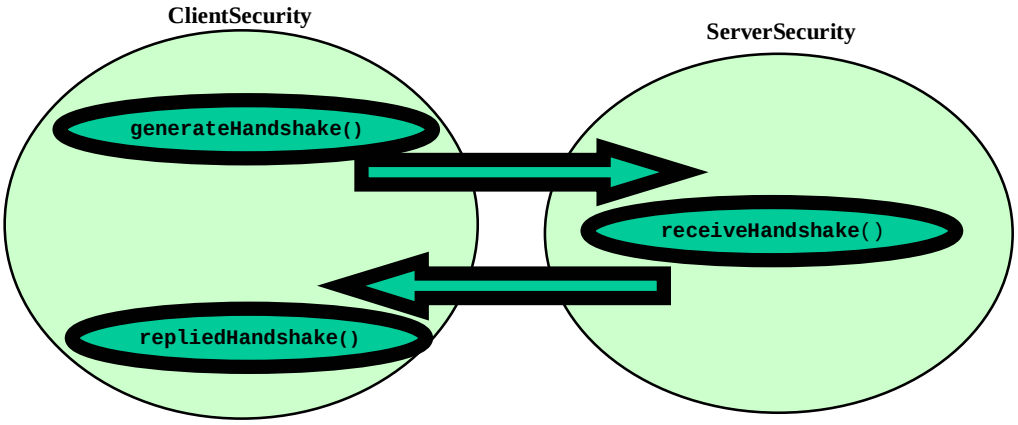
Note: when using this JCE encryption sample the first handshake sent to the Gateway daemon will take considerably longer than usual, due to the overhead of loading the required security classes.

Appendix I - Security exit model

The interactions between the CICS TG security exits are best described by understanding the methods defined in the following interfaces:

- `com.ibm.ctg.security.ClientSecurity`
- `com.ibm.ctg.security.ServerSecurity`

The following figure illustrates the flow upon the establishment of a connection from the Java client to the Gateway daemon, that is upon invocation of the JavaGateway.open().



The following methods are then invoked for every GatewayRequest flow (such as an ECI or EPI request across an established connection.