Common Errors in BW related to SSL communication

Balakrishnan Viswanathan
TIBCO Support

This document outlines some of the common errors that show up as technical support service requests(SR's) when configuring BW for SSL communication. So, the objective of this document is to allow BW users to identify and take a first shot at troubleshooting the problem

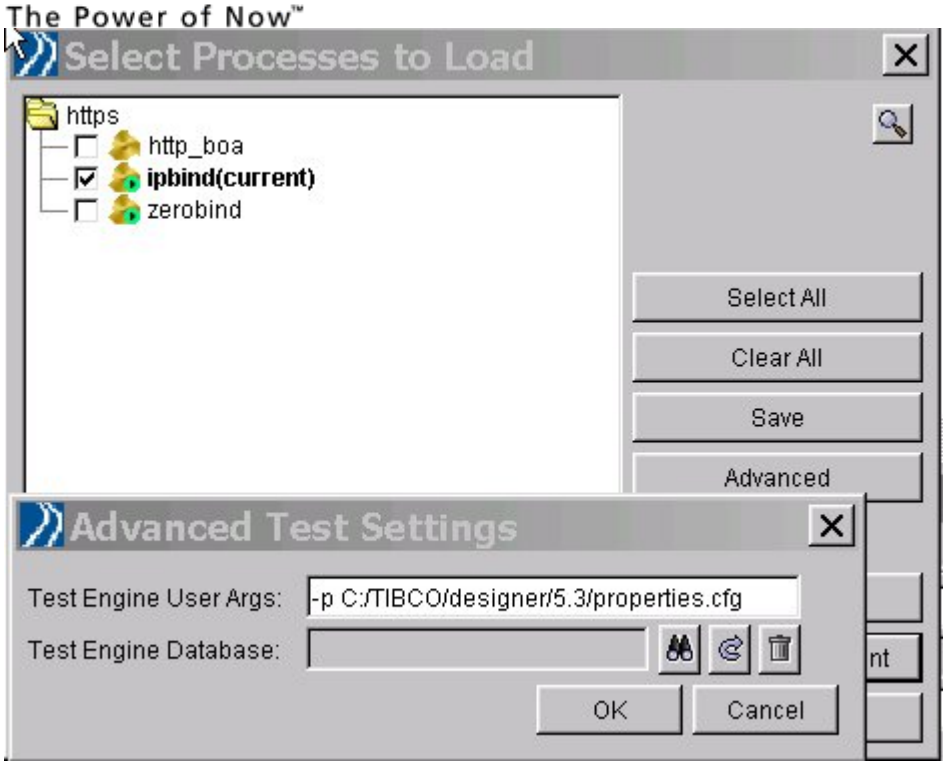
Section-I: Enabling Tracing

Before troubleshooting any problem related to SSL, it helps to have the below tracing properties enabled:-

Trace.Task.*=true (Client side SSL tracing information is made available)
Trace.Startup=true
Trace.JC.*=true
Trace.Engine=true
Trace.Debug.*=true

bw.plugin.http.server.debug: true (Server side SSL tracing information is made available)

If you are debugging from designer, then you can choose to place the above properties in a "properties.cfg" file under <c:/tibco/designer/5.3> folder and then specify the location of this properties.cfg file in the "Advanced Test Settings" window such as below



The other way you can set the properties for the tester engine is to specify the properties in your own custom properties file, for example – c:/test/props.cfg. You can reference this properties file in your designer.tra file (<c:/tibco/designer/5.3/bin/designer.tra>) using property

java.property.testEngine.User.Args –p c:/test/props.cfg

After modifying the .tra file and enabling tracing properties, you would have to exit and restart designer for the properties to take effect.

If you would like to change the default security provider(entrust) to Sun's j2se, then you could set the below property

java.property.TIBCO_SECURITY_VENDOR=j2se

In addition, you can also enable debug level tracing for j2se library by setting an additional property

java.property.javax.net.debug=ssl

Pl. be sure to restart designer or BW engine after changing the above properties.

Section-II: COMMON ERRORS

A) Error message: - “iaik.security.ssl.SSLException: Server certificate rejected by ChainVerifier”

Analysis: - This is a very common error message. Once the tracing is enabled then the problem becomes clearer. For example,

ssl_debug(1): Starting handshake (iSaSiLk 3.03)...
ssl_debug(1): Sending v3 client_hello message, requesting version 3.1...
ssl_debug(1): Received v3 server_hello handshake message.
ssl_debug(1): Server selected SSL version 3.1.
ssl_debug(1): Server created new session 47:E8:48:C2:5A:F9:14:B1...
ssl_debug(1): CipherSuite selected by server: SSL_RSA_WITH_RC4_128_MD5
ssl_debug(1): CompressionMethod selected by server: NULL
ssl_debug(1): Received certificate handshake message with server certificate.
ssl_debug(1): Server sent a 1024 bit RSA certificate, chain has 2 elements.
validating certificate chain
looking in datastore for certificate with DN ou=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign,ou=VeriSign International Server CA - Class 3,ou=VeriSign, Inc.,o=VeriSign Trust Network
No match found
CA certificate with issuer ou=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign,ou=VeriSign International Server CA - Class 3,ou=VeriSign, Inc.,o=VeriSign Trust Network and serial number 03D4 12D3 0672 7D3A 368A 9621 FCD2 357D is not a trusted certificate
server verification failed:
com.tibco.security.AXSecurityException: CA certificate with issuer ou=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign,ou=VeriSign International Server CA - Class 3,ou=VeriSign, Inc.,o=VeriSign Trust Network and serial number 03D4 12D3 0672 7D3A 368A 9621 FCD2 357D is not a trusted certificate
    at com.tibco.security.CertChainVerifier.validateAndCompleteChain(CertChainVerifier.java:171)
    at com.tibco.security.ssl.DefaultCertificateVerifier.authenticateServer(DefaultCertificateVerifier.java:129)
    at com.tibco.security.ssl.ExtendedCertificateVerifier.authenticateServer(ExtendedCertificateVerifier.java:119)
    at com.tibco.security.ssl.entrust6.c.verifyServer(EntrustVerifier.java)
    at com.tibco.security.ssl.entrust6.c.verifyChain(EntrustVerifier.java)

```
        at iaik.security.ssl.x.a(Unknown Source)
        at iaik.security.ssl.x.b(Unknown Source)
        at iaik.security.ssl.x.a(Unknown Source)
        at iaik.security.ssl.r.d(Unknown Source)
        at iaik.security.ssl.SSLTransport.startHandshake(Unknown Source)
        at iaik.security.ssl.SSLTransport.a(Unknown Source)
        at iaik.security.ssl.SSLTransport.renegotiate(Unknown Source)
        at iaik.security.ssl.SSLSocket.renegotiate(Unknown Source)
        at com.tibco.security.ssl.entrust6.b.doHandshake(SSLClientImpl.java)
        at
com.tibco.plugin.share.security.TIBCryptClientSocketFactory.createSocket(TIBCryptCli
entSocketFactory.java:76)
        at
org.apache.commons.httpclient.HttpConnection.open(HttpConnection.java:652)
        at org.apache.commons.httpclient.HttpClient.executeMethod(HttpClient.java:628)
        at org.apache.commons.httpclient.HttpClient.executeMethod(HttpClient.java:497)
        at
com.tibco.plugin.share.http.client.JakartaHttpTransportDriver$RequestExecutor.run(Jaka
rtaHttpTransportDriver.java:227)
        at com.tibco.pe.util.ThreadPool$ThreadPoolThread.run(ThreadPool.java:99)
ssl_debug(1): Sending alert: Alert Fatal: bad certificate
ssl_debug(1): Shutting down SSL layer...
ssl_debug(1): SSLException while handshaking: Server certificate rejected by
ChainVerifier
```

The above trace shows that server returned 2 certificates as part of server authentication and the intermediate certificate corresponding to DN information below

>>looking in datastore for certificate with DN ou=www.verisign.com/CPS Incorp.by ref.
>>LIABILITY LTD.(c)97 VeriSign,ou=VeriSign International Server CA - Class
>>3,ou=VeriSign, Inc.,o=VeriSign Trust Network
>>No match found

was not found in the datastore (trusted certificate folder) configured for this activity. Once this particular certificate is imported then the error will go away.

Resolution: - Once the problem has been identified to have been caused by a missing certificate [**Pl. note that BW requires all intermediate and root certificates to be uploaded into the trusted folder but the leaf certificate by itself is not required to be uploaded**] you can use any of the commonly available tools such as Openssl, cURL or other GUI based open source utilities like “Portecle” to inspect the certificates

To use Portecle, you can follow the steps below:-

1. Download the jar files from http://portecle.sourceforge.net/

2. Download Bouncy Castle (“bc”) cryptographic libraries from http://www.bouncycastle.org/

3. Download “unlimted strength” JCE policy files from http://java.sun.com

4. Navigate to your “java.policy” file located under jre/lib/security folder (for example, C:\j2sdk1.4.2_08\jre\lib\security) and edit it using notepad and add the bouncy castle provider such as

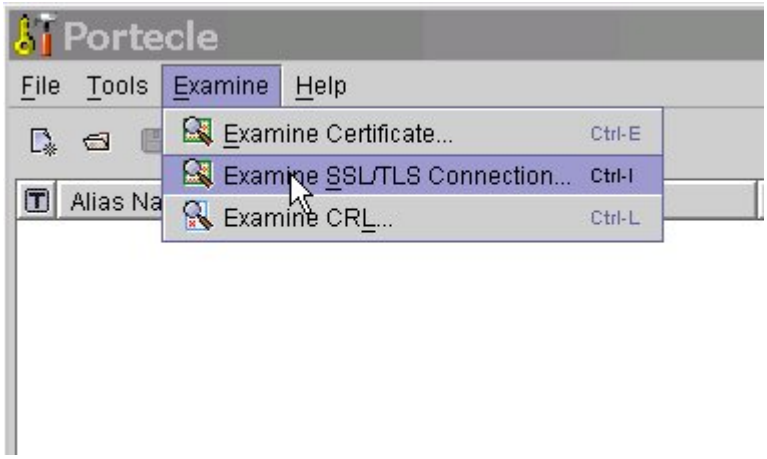
   security.provider.[n]=org.bouncycastle.jce.provider.BouncyCastleProvider

   where [n] should a number such as [1,2,3….n] depending on other security providers you have configured in your environment[For e.g., security.provider.5=org.bouncycastle.jce.provider.BouncyCastleProvider]

5. After you have configured the above steps, you should be able to start Portecle from the command prompt

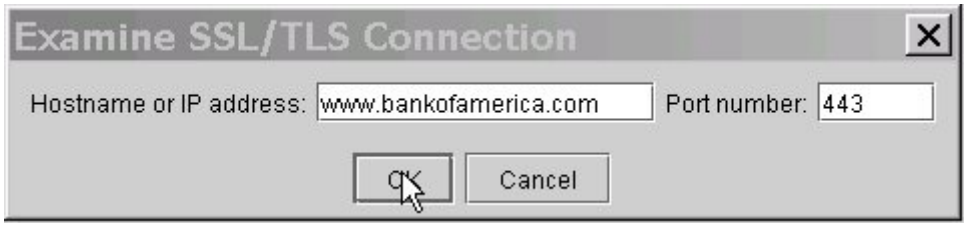
   C:\portecle-1.1>java -jar portecle.jar

6. From the menu option, you can choose to “Examine SSL/TLS connection”

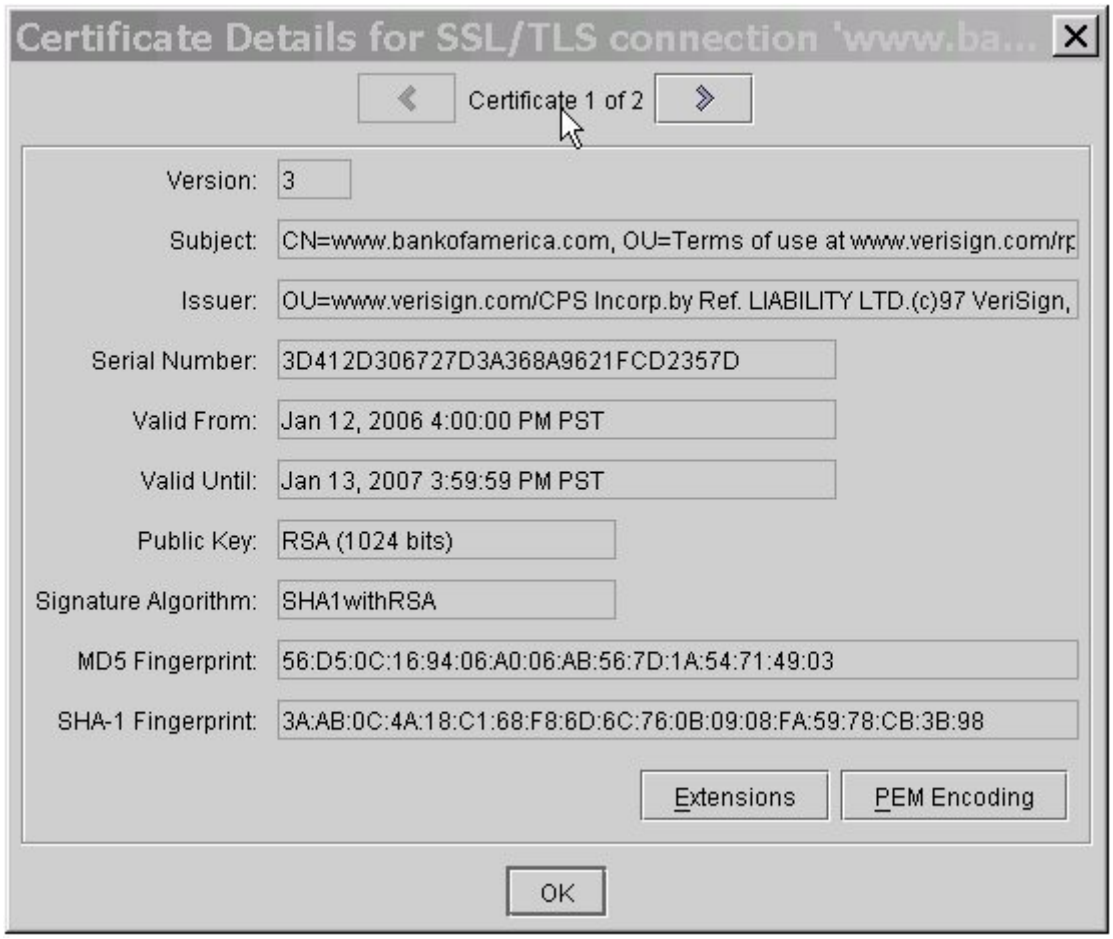


7. Enter the url of the https server that you are trying to connect to from within BW [Pl. note that server address or ip address can be used but the server must be “visible/public” to the machine where you are running Portecle and also to BW. If the server is behind a firewall, then you must request the server administrator to

either allow access or instead send you the .p7b file. Also, the default port for https communication is 443 but this can be different]



8. Once you click OK on the above window, you will see the chain of certificates that the server is presenting. In this particular example, boa site is presenting 2 certificates in the chain



9. In the above window, you can click on “PEM Encoding” button to view the certificate in a PEM encoded format

10. You can copy the entire contents of the above pane between and inclusive of

    -----BEGIN CERTIFICATE-----
    -----END CERTIFICATE-----

    And save it to a local file such as “certboa1.pem”. Similarly, you would have click on the “>” arrow to move to the next certificate in the chain that the server is presenting

11. Later, you can import all the certificates that you have copied to the local file system, using the above steps, into BW from the menu “Tools->Trusted Certificates->Import into PEM format”

Above steps can be used only when the server is publicly accessible and if the server is configured to present the complete chain. In many cases, the server may present only the leaf, i.e. the server certificate, in which case, you would have to talk to the server administrator to get the complete certificate chain

B) Error Message: - No PrivateKeyInfo: iaik.asn1.CodingException: ASN.1 creation error:Length: Too large ASN.1 object

Analysis:- The error message is thrown when we attempt to start a event source that is using SSL identity of the type “certificate/private key” instead of PKCS#12 type like a .p12 or JKS keystore

Looking at the error, it shows that event source could not initialize the SSL server socket because it could not find “PrivateKeyInfo”. So we have to determine why it could not find the private key info. If we look at the private key in a text editor, the key looks like

-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----

Based on which we can tell whether the private key format is PKCS#8 compliant or not [**NOTE- BW supports private keys *only* in PKCS#8 format**]

Resolution: - Since the above key format is not pkcs#8 compliant we would have to convert it using Openssl

Openssl> pkcs8 -in <key file in pem format> -inform PEM -topk8 -out <output pkcs8 file name>

Now if you open the output file from the above command you will notice that certificate has the below beginning and end tags

-----BEGIN ENCRYPTED PRIVATE KEY-----
-----END ENCRYPTED PRIVATE KEY-----

Now when you specify the above pkcs#8 format key file in your BW’s SSL identity, then server socket will get initialized fine

NOTE: - If you have other private key formats like .pvk (Microsoft private key format), then you can use the tool available at

http://www.drh-consultancy.demon.co.uk/pvk.html

to convert to pkcs#8 format

C) Error: - com.tibco.security.AXSecurityException: RSA signature failed to initialize for verifying: Unsupported keysize or algorithm parameters

Analysis: - Looking at the detailed tracing

ssl_debug(5): Starting handshake (iSaSiLk 3.03)...
ssl_debug(5): Sending v3 client_hello message, requesting version 3.1...
ssl_debug(5): Starting handshake (iSaSiLk 3.03)...
ssl_debug(5): Received v3 client_hello handshake message.
ssl_debug(5): Client requested SSL version 3.1, selecting version 3.1.
ssl_debug(5): Creating new session 78:78:DC:06:C4:6B:1C:35...
ssl_debug(5): CompressionMethods supported by the client:
ssl_debug(5): NULL
ssl_debug(5): Sending server_hello handshake message.
ssl_debug(5): Selecting CipherSuite: SSL_RSA_WITH_RC4_128_SHA
ssl_debug(5): Selecting CompressionMethod: NULL
ssl_debug(5): Sending certificate handshake message with server certificate...
ssl_debug(5): Sending server_hello_done handshake message...
ssl_debug(4): Received v3 server_hello handshake message.
ssl_debug(5): Server selected SSL version 3.1.
ssl_debug(5): Server created new session 78:78:DC:06:C4:6B:1C:35...
ssl_debug(5): CipherSuite selected by server: SSL_RSA_WITH_RC4_128_SHA
ssl_debug(5): CompressionMethod selected by server: NULL
ssl_debug(5): Received certificate handshake message with server certificate.
ssl_debug(5): Server sent a 512 bit RSA certificate, chain has 2 elements.
validating certificate chain
server verification failed:
com.tibco.security.AXSecurityException: RSA signature failed to initialize for verifying: Unsupported keysize or algorithm parameters
at com.tibco.security.CertChainVerifier.validateAndCompleteChain(CertChainVerifier.java:106)
at com.tibco.security.ssl.DefaultCertificateVerifier.authenticateServer(DefaultCertificateVerifier.java:129)
at com.tibco.security.ssl.ExtendedCertificateVerifier.authenticateServer(ExtendedCertificateVerifier.java:119)
at com.tibco.security.ssl.entrust6.c.verifyServer(EntrustVerifier.java)
at com.tibco.security.ssl.entrust6.c.verifyChain(EntrustVerifier.java)
at iaik.security.ssl.x.a(Unknown Source)
at iaik.security.ssl.x.b(Unknown Source)
at iaik.security.ssl.x.a(Unknown Source)
at iaik.security.ssl.r.d(Unknown Source)

we can determine that server authentication failed because BW could not verify the signature of one of the certificates (in this case the chain has 2 elements) using the RSA public key. Also, the error “unsupported keysize” indicates that one of the certificates presented by the server has a RSA key size of 2048 bits or higher.

[**NOTE- BW’s default security provider library is entrust and it does not support RSA key sizes higher than 2048 bit. If we change the security provider to Sun’s “j2se”, even that provider will fail for 4096 bit RSA key size**]

Resolution:- Identify the certificate that has a RSA key size greater than 2048 and use a new certificate in place that has a key size of 2048 bits or lower (1024 preferably – 512 bit is considered to be weak)

D) Error: - iaik.security.ssl.SSLException: Peer sent alert: Alert Fatal: unexpected Message

Analysis: - Looking at the SSL debug trace

ssl_debug(1): Starting handshake (iSaSiLk 3.03)...
ssl_debug(1): Sending v3 client_hello message, requesting version 3.1...
ssl_debug(1): Received alert message: Alert Fatal: unexpected message
ssl_debug(1): SSLException while handshaking: Peer sent alert: Alert Fatal: unexpected message
ssl_debug(1): Shutting down SSL layer...
exception: Peer sent alert: Alert Fatal: unexpected message

iaik.security.ssl.SSLException: Peer sent alert: Alert Fatal: unexpected message
    at iaik.security.ssl.r.f(Unknown Source)
    at iaik.security.ssl.x.b(Unknown Source)
    at iaik.security.ssl.x.a(Unknown Source)
    at iaik.security.ssl.r.d(Unknown Source)
    at iaik.security.ssl.SSLTransport.startHandshake(Unknown Source)
    at iaik.security.ssl.SSLTransport.getOutputStream(Unknown Source)

From the above trace we can see that as soon as SSL client (in this case BW) requested the SSL Server for choosing SSL protocol version to “3.1”, the SSL Server rejected and request and closed the connection. The reason being this particular server does not talk SSL v3.1; instead it understands only older versions of the protocol, i.e. SSL v3.0/SSL v2.0

Resolution: - There is a pending enhancement request in BW 5.3.2 version to transparently support lower version of SSL protocol when negotiating the handshake when using the default security provider (to get more info contact support@tibco.com).

However, till the enhancement request is addressed, you would have to change the security provider library to “j2se”. To do this you can set the property below in your deployed .tra file

java.property.TIBCO_SECURITY_VENDOR=j2se

If you are running your project from designer, then you would have to set the below property in your designer.tra file (<c:\tibco\designer\5.3\bin\designer.tra>)

java.property.testEngine.User.Args -p c:/tibco/designer/5.3/properties.cfg

and then inside properties.cfg at location, C:/tibco/designer/5.3 (you can change the location of the properties file but you would have to make sure to change the value in the tester engine user args property too) you can place the security vendor property and set it to “j2se”

You must restart designer for the properties to take effect.

E) Error: - SSLException while handshaking: Peer sent alert: Alert Fatal: decrypt error

Analysis: Looking at the SSL debug trace

ssl_debug(1): Starting handshake (iSaSiLk 3.03)...
ssl_debug(1): Received v2 client hello message.
ssl_debug(1): Client requested SSL version 3.1, selecting version 3.1.
ssl_debug(1): Creating new session 26:A3:07:B5:8B:38:62:E6...
ssl_debug(1): CompressionMethods supported by the client:
ssl_debug(1): NULL
ssl_debug(1): Sending server_hello handshake message.
ssl_debug(1): Selecting CipherSuite: SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
ssl_debug(1): Selecting CompressionMethod: NULL
ssl_debug(1): Temporary domestic DH parameters not set, using defaults.
ssl_debug(1): Sending certificate handshake message with server certificate...
ssl_debug(1): Sending server_key_exchange handshake message...
ssl_debug(1): Sending server_hello_done handshake message...
ssl_debug(1): Received alert message: Alert Fatal: decrypt error
ssl_debug(1): SSLException while handshaking: Peer sent alert: Alert Fatal: decrypt error
ssl_debug(1): Shutting down SSL layer...
ssl_debug(1): Closing transport...

As part of the handshake, the client attempts decrypts some random data with public key of the server (which is present) in the certificate and there is a decryption error, which

indicates that server's private key does not correspond to the public key in its certificate, a possible misconfiguration.

Resolution:- Ensure that if you select "Certificate/private key" as the type of identity, then you would have to make sure that private key(in pkcs#8 format) corresponds to that of the public key in the certificate.

F) Error: - SSLException while handshaking: Certificate verify message signature error!

Analysis: - From looking at the SSL debug trace

ssl_debug(1): Received certificate_request handshake message.
ssl_debug(1): Accepted certificate types: RSA, DSS
ssl_debug(1): Accepted certificate authorities:
ssl_debug(1): (empty list)
ssl_debug(1): Received server_hello_done handshake message.
ssl_debug(1): Sending certificate handshake message with RSA client certificate...
ssl_debug(1): Sending client_key_exchange handshake message (1024 bit)...
ssl_debug(2): Received certificate handshake message with client certificate.
ssl_debug(2): Client sent a 1024 bit RSA certificate, chain has 1 elements.
ssl_debug(2): Received client_key_exchange handshake message.
ssl_debug(1): Sending certificate_verify handshake message...
ssl_debug(2): Received certificate_verify handshake message.
ssl_debug(1): Sending change_cipher_spec message...
ssl_debug(1): Sending finished message...
ssl_debug(2): Sending alert: Alert Fatal: bad certificate
ssl_debug(2): Shutting down SSL layer...
ssl_debug(2): SSLException while handshaking: Certificate verify message signature error!
ssl_debug(2): Closing transport...
ssl_debug(1): Exception sending message: java.net.SocketException: Software caused connection abort: socket write error
ssl_debug(1): Shutting down SSL layer...

we can find out that this error was thrown as part of client authentication phase. As part of the handshake, the client sends a "CertificateVerify" message which comprises of a digitally signed hash of all previous messages exchanged between server and the client until but not including the CertificateVerify message itself. When server tries to verify the signature of the "CertificateVerify" message using the public key of the client which is part of the client certificate, then it fails, indicating the possibility of a mismatch in the public and private keys on the client side.

Resolution: - Ensure that your client side identity, if using the type of "Certificate/Private Key", has the right private key that corresponds to the public key in the certificate.

G) Error:- com.tibco.security.AXSecurityException: Extension error: Certificate [n] does not have a basic constraints extension!

Analysis: - The above error is thrown as part of the handshake when an intermediate or root CA certificate does not have “basic constraints” certificate attribute set to “true” or is absent

According to the spec RFC 2459, section 4.2.1.10:-

4.2.1.10 Basic Constraints

The basic constraints extension identifies whether the subject of the certificate is a CA and how deep a certification path may exist through that CA.

The pathLenConstraint field is meaningful only if cA is set to TRUE. In this case, it gives the maximum number of CA certificates that may follow this certificate in a certification path. A value of zero indicates that only an end-entity certificate may follow in the path. Where it appears, the pathLenConstraint field MUST be greater than or equal to zero. Where pathLenConstraint does not appear, there is no limit to the allowed length of the certification path.

This extension MUST appear as a critical extension in all CA certificates. This extension SHOULD NOT appear in end entity certificates.

id-ce-basicConstraints OBJECT IDENTIFIER ::= { id-ce 19 }

```
BasicConstraints ::= SEQUENCE {
    CA                  BOOLEAN DEFAULT FALSE,
    pathLenConstraint       INTEGER (0..MAX) OPTIONAL }
```

Resolution:- Check the certificates presented during the handshake and ensure that intermediate CA’s and root CA’s have the basic constraints attribute and its value is set to “TRUE”, which indicates that it can sign other leaf certificates in its hierarchy

H) Error: - com.tibco.security.AXSecurityException: Extension error: certificate at index 0 is marked CA certificate

Analysis: - This error is thrown when the leaf certificate has the “basic constraints” extension marked “CA: TRUE”, indicating that it can sign other certificates under it, but the leaf certificates (Server or Client) should either omit this attribute altogether or should have value “CA: FALSE”. You can refer to the previous error description for quote on what RFC 2459 says about “basic constraints” attribute

Resolution: - Identify the certificate that is resulting in the above exception and if it is a leaf certificate then a new certificate would need to be generated omitting the basic constraints attribute

I)Error:- java.io.IOException: Fatal SSL handshake error: java.lang.RuntimeException: Unable to create cipher AES/CBC/NoPadding: java.lang.SecurityException: Unsupported keysize or algorithm parameters

Analysis: - Though this exception “Unsupported keysize or algorithm parameters” is similar to the error that is thrown when RSA keysize is higher than 2048, the clue is in the message “Unable to create cipher AES/CBC/NoPadding”, which indicates that keysize for symmetric encryption is not supported. For bulk encryption US governments restricts the size of the symmetric key and has export regulations for software that is sold to other countries. By default java runtime has only limited strength (less than 128 bit - for symmetric key size) policy files.

Resolution: - You would have to download “unlimited strength” policy files from Sun website.

1. Go to website http://java.sun.com/j2se/1.4.2/download.html and download
   Java Cryptography Extension (JCE)
   Unlimited Strength Jurisdiction Policy Files 1.4.2

   This zip file contains two jar files local_policy.jar, US_export_policy.jar

2. Go to JAVA_HOME installation or if your TIBCO installation is using its bundled jre, then go to <TIBCO_HOME>/jre/1.4.2/lib/security replaces the existing policy files with the new ones from Sun website.

J) Error: - com.tibco.security.AXSecurityException: No certificates encoded in supported ways were found

Analysis: - The above error is thrown in BW when we place a identity or some other activity in the same folder as the trusted certificates folder. When BW loads, it tries to read even the identity file or other resource as a PEM encoded X509 certificate and throws the above exception. Though the above exception thrown in the console, has not impact on running of BW engine, it is good practice to keep only PEM encoded certificates in the trusted certificates folder

Resolution: - Remove any other activities that might be in the "trusted certificates" folder in the BW project and keep only PEM encoded certificates in this folder.

K) Error: - java.io.IOException: An AXSecurityException was thrown while trying to create the server socket on the port: [xxxx]

Analysis: - This exception indicates that a secure server socket could not be opened at the given port [xxxx]. In order to initialize the server socket a valid private key and pass phrase is needed. If you are using JKS keystore as an identity then make sure the keystore also contains the private key for the leaf certificate. For example,

>>keytool –list –v –keystore <keystore name>

Will output the contents of the keystore on the console

Alias name: mykey
Creation date: Feb 7, 2006
Entry type: trustedCertEntry

Owner: CN=x.y.z, OU=Domain Control Validated - Power Server ID(TM),
OU=See www.geotrust.com/resources/cps (c)05, OU=businessprofile.geotrust.com/get.jsp?GT93606005, O=x.y.z, C=SE
Issuer: OU=Equifax Secure Certificate Authority, O=Equifax, C=US
Serial number: 5a733
Valid from: Fri Jan 20 01:06:24 PST 2006 until: Sun Jan 21 01:06:24 PST 2007
Certificate fingerprints:
MD5: 65:88:CC:CC:02:64:64:A8:E6:DE:89:C4:26:BF:E6:B2
SHA1: 3B:49:5A:1F:C8:F1:53:7B:F9:91:56:0F:7C:A4:C6:21:CC:EE:C1:AF

The above keystore does not have "Entry type" as "keyEntry" instead has "trustedCertEntry", which basically means that this keystore does not have the private key corresponding to the above certificate.

Resolution: - Create the key and import the signed certificate in the same keystore using the commands below:-

1. keytool -genkey -v -alias <alias> -keysize 1024 -keypass <key password> -keystore <keystore name> -storepass <keystore password> -validity <no. of days of validity>

2. keytool -certreq -v -alias <alias> -keystore <keystore name> -keypass <password for the keys> -storepass <keystore password> -file <certificate request file name>

3. keytool –import –v –alias <alias> -keystore <keystore name> -storepass <keystore password –file <signed certificate in a file>

L) Error: - java.lang.IllegalArgumentException: Child resource has null name

Analysis: - This is a misleading error that is thrown when an otherwise valid certificate is imported into designer using “Tools->Trusted Certificates->Import into PEM format”. The problem here is that designer somehow does not like “/” being present in the common name attribute. For example, a cert

> s:/C=CA/ST=Ontario/L=Toronto/O=support/OU=tibco/CN=bala.com/rdt/gdr/
> emailAddress=bala@bala.com

has *CN= bala.com/rdt/gdr/* and designer complains about the forward slash with the above exception

Resolution: – It has been marked as defect and will be addressed in future BW releases above 5.3.2 (Pl. contact support@tibco.com for more details).

Current workarounds are

1. Create folder in designer and then manually go to that folder and copy/paste the certificate in .pem format and then rename it to .cert file

2. Use global variable BW_GLOBAL_TRUSTED_CA_STORE and point the location of the PEM encoded certificates (e.g.: file:///c:/certs/bwcert.pem)

M) Error: - ssl_debug(2): No client certificate available, sending empty certificate message...

Analysis: - From SSL debug trace

ssl_debug(2): Received v3 server_hello handshake message.
ssl_debug(2): Server selected SSL version 3.1.
ssl_debug(2): Server created new session F9:46:DB:14:3A:94:A8:0E...
ssl_debug(2): CipherSuite selected by server: SSL_RSA_WITH_RC4_128_SHA
ssl_debug(2): CompressionMethod selected by server: NULL
ssl_debug(2): Received certificate handshake message with server certificate.
ssl_debug(2): Server sent a 1024 bit RSA certificate, chain has 3 elements.
validating certificate chain
looking in datastore for certificate with DN cn=VeriSign Class 3 Secure Intranet Server CA,ou=Terms of use at https://www.verisign.com/rpa (c)03,ou=VeriSign Trust Network,o=VeriSign, Inc.,c=US
match found
looking in datastore for certificate with DN ou=VeriSign Trust Network,ou=(c) 1998 VeriSign, Inc. - For authorized use only,ou=Class 3 Public Primary Certification Authority - G2,o=VeriSign, Inc.,c=US
match found
chain length: 3
chain verifies ok
server verification ok
ssl_debug(2): Received certificate_request handshake message.
ssl_debug(2): Accepted certificate types: RSA, DSS, SSLv3 ephemeral RSA
ssl_debug(2): Accepted certificate authorities:
ssl_debug(2): cn=VeriSign Class 3 Secure Intranet Server CA,ou=Terms of use at https://www.verisign.com/rpa (c)03,ou=VeriSign Trust Network,o=VeriSign, Inc.,c=US
ssl_debug(2): cn=it-wrapper.com,ou=For Intranet Use Only,ou=Terms of use at www.verisign.com/rpa (c)00,ou=18-l2e0,o=ABC,l=PA,st=CA,c=US
ssl_debug(2): ou=VeriSign Trust Network,ou=(c) 1998 VeriSign, Inc. - For authorized use only,ou=Class 3 Public Primary Certification Authority - G2,o=VeriSign, Inc.,c=US
ssl_debug(2): Received server_hello_done handshake message.
ssl_debug(2): No client certificate available, sending empty certificate message...
ssl_debug(2): Sending client_key_exchange handshake message (1024 bit)...
ssl_debug(2): Sending change_cipher_spec message...
ssl_debug(2): Sending finished message...
ssl_debug(2): Exception sending message: java.net.SocketException: Software caused connection abort: socket write error
ssl_debug(2): Shutting down SSL layer...
ssl_debug(2): IOException while handshaking: Software caused connection abort: socket write error

we can see that no client certificate was sent back to the server even though the server was requesting the client to send its certificate. The reason for this is because the identity

that was configured for the SSL client for client authentication only has the leaf certificate inside the .p12 file. Moreover, the signer of the client certificate is not part of the trusted certificates list of the server, the list that is sent as part of the handshake after “ssl_debug(2): Accepted certificate authorities”

As a result BW’s security provider library (entrust or j2se) is unable to complete the certificate chain (leaf->intermediate CA-> Root CA) and therefore no client certificate is sent. Incidentally, this is conformant to what the spec says

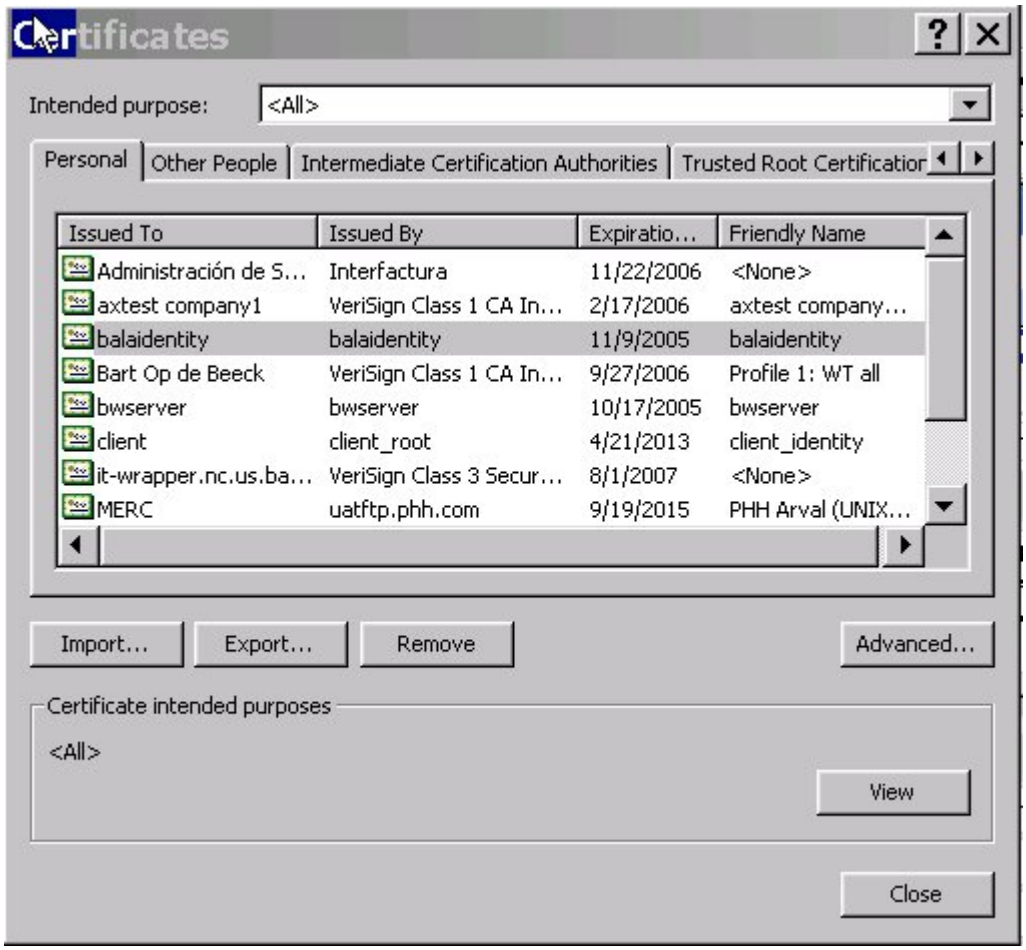
RFC 2246, section 7.4.6, states: "If no suitable certificate is available, the client should send a certificate message containing no certificates. If client authentication is required by the server for the handshake to continue, it may respond with a fatal handshake failure alert."

Resolution: - You will have to have the complete chain of certificates in your client identity file. If you are using Openssl, you can use

>>openssl pkcs12 -export -in <signed_leaf_certificate.pem> -inkey <key_file_of_leaf_cert.pem> -CAfile <file_containing_CA_certs.pem> -out <some_name.p12>

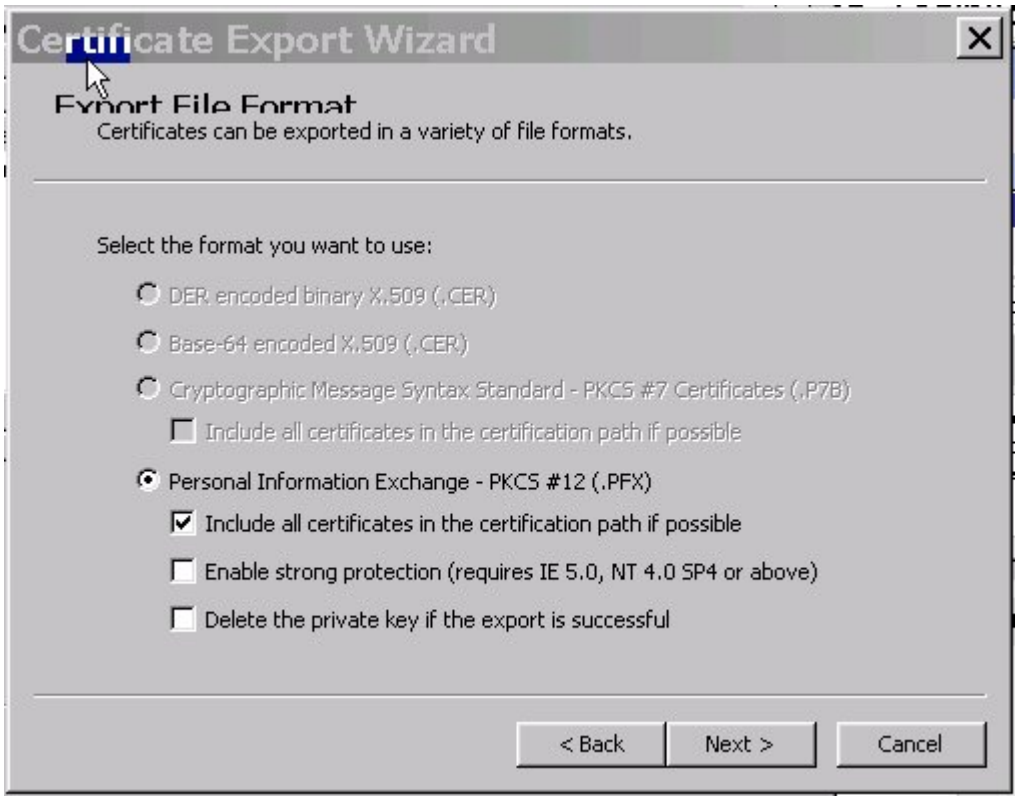
Another option to try if you have .p12 file, but not the individual CA certificates, is the import the p12 into Internet Explorer and then export them out as shown below

- Click on the “Personal” certificate that you want to “Export”



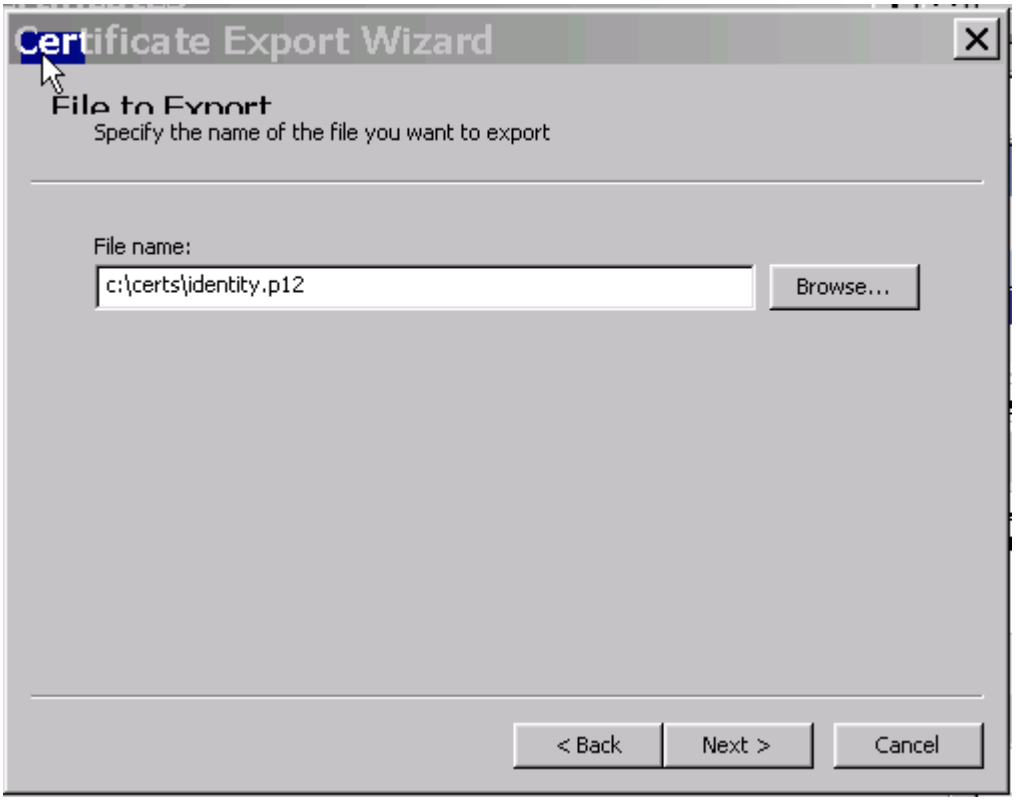
- Click on “export private key” option

- In the “Export File Format” box, make sure to check “Include all certificates in the certification path if possible”



- Enter the password for the PKCS#12 file

- Enter a file name for the exported file



If all of the above steps do not get you the full chain, then please contact your CA and get the signer certificates for your client

Please contact support@tibco.com for issues that are not part of the above list.