Master Note for Troubleshooting Java SSL issues (Doc ID 1906321.1)

In this Document

APPLIES TO:

Java SE JDK and JRE - Version 6 and later
Oracle Internet Directory - Version 11.1.1.7.0 to 11.1.1.7.0 [Release 11g]
Information in this document applies to any platform.

PURPOSE

This document provides references and pointers to troubleshoot Java SSL related issues.

TROUBLESHOOTING STEPS

Introduction

The Java Secure Socket Extension (JSSE) provides a framework and an implementation for a Java version of the SSL and TLS protocols. It includes functic data encryption, server authentication, message integrity, and optional client authentication.

Java applications using the JSSE framework for client/server communications may encounter runtime issues. These problems can be caused by client/serve misconfiguration (e.g. server certificate not installed in client's store and vice versa), incompatibility between client and server side libraries, user errors, or Java library bugs.

Most often, these problems happen during the initial SSL handshake between client and server. The best way to troubleshoot is to add this diagnostic flag, -*Djavax.net.debug=all* , on both the client and the server. Once the problem has been reproduced, collect the log file. Next, compare the connection protocol with the example session documented in "Debugging SSL/TLS Connections" to discover clues.

How to interpret SSL logs

- Debugging SSL/TLS Connections

Known/common SSL issues

- SSL Handshake Error "**ValidatorException: PKIX path building failed**" Note 1327811.1
  - server certificate not found in client's truststore
- javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: **No trusted certificate found**" Note 785327.1
  - Intermediate certificate not imported to client store
- Application Code gets **bad_certificate** Error When Using SSL Client and Server Authentication Code Note 1305106.1
  - incorrect use of "com.sun.net.ssl.internal.ssl.*" method
- Java SE 7 SSL Connections Generate "SSLException: **Received fatal alert: illegal_parameter**" Note 1598639.1
  - SNI or ECC Extensions formats being interpreted incorrectly
- Applications Using SSL Throw 'Javax.net.ssl.SSLHandshakeException: **Received Fatal Alert: Handshake_failure'** with Java SE 7 Update 85 Note 2037473.1
  - In 7u85 and 8u51, RC4 based ciphersuites were removed from the default enabled list

- SSL handshake failed with **javax.net.ssl.SSLException: Received fatal alert: handshake_failure**
  Unmatched Server's certificate signature with Client's ciphersuite.
  Example: If the sever is using an RSA based signature algorithm in its certificate, then an RSA based ciphersuite is required for SSL connection.
- SSL Connections Using ECDH Fail with **javax.net.ssl.SSLHandshakeException: Invalid Padding length** Note 1991236.1
  - Solaris' PKCS11 library bug

- SSL Call from WLS 12.1.1 to WLS 10.3.5 Fails "NEW ALERT With Severity: **FATAL, Type: 70**" When Using TLS 1.2 Protocol Note 1544573.1
  - incompatible TLS protocol version

- Application Fails To Run On GlassFish Server With The Error, "java.lang.RuntimeException: **Could not parse key values**" When Using JDK 1.6.0_43 or Higher Updates Note 1587984.1
  - incompatitble ECC algorithm

- Login Attempts May Hang at "Loading..." When Users Login to Oracle Secure Global Desktop (SGD) After Upgrading Client Java Plug-in to Version 1.7 Note 1496595.1
  - incompatible SNI extension

- Getting "javax.net.ssl.SSLHandshakeException: **Remote Host Closed Connection During Handshake**" Error Note 1572454.1
  - Proxy is blocking the requests from the application
  (Note: This error also happens if client's authentication certificate is not imported to server's truststore in a two way SSL scenario)

- SSL not getting enabled showing java.lang.SecurityException: **Strong RSA key pair generation requires bit length** >= XXX and multiple of YYY Note 1608820.1
  - For certain types of certificates, which are called Strong RSA KeyPairs, it's necessary to enable FIPS in order to make the server work correctly.

- The Java Program Group's blog entry "Diagnosing TLS, SSL, and HTTPS"
  - Handshake failure example related to unsupported algorithm

- Remote host close unexpectedly: **jaxax.net.ssl.SSLProtocolException: Remote host closed coonnection incorrectly.**
  - Remote server fails to handle CBC protection mechanism. There is a java.lang.System property called "jsse.enableCBCProtection" (defaults to true) that enables the BEAST countermeasure.

- **javax.net.ssl.SSLHandshakeException: server certificate change is restricted during renegotiation**
  - Unsafe Server Certificate Change in SSL/TLS Renegotiations Not Allowed.
  If unsafe server certificate change is really required, please set the system property, jdk.tls.allowUnsafeServerCertChange, to "true" before JSSE is initialized. Note that this would re-establish the unsafe server certificate change issue.

- Starting with JDK 7u75 release, the SSLv3 protocol (Secure Socket Layer) has been deactivated and is not available by default. See the java.security.Security property jdk.tls.disabledAlgorithms in <JRE_HOME>/lib/security/java.security file. If SSLv3 is absolutely required, the protocol can be reactivated by removing "SSLv3" from the jdk.tls.disabledAlgorithms property in the java.security file or by dynamically setting this Security property to "true" before JSSE is initialized. It should be noted that SSLv3 is obsolete and should no longer be used.
  To re-enable SSLv3 protocol on deploy level: edit the deployment.properties file and add the following:
  deployment.security.SSLv3=true.

Resources for Troubleshooting SSL issues

- How to Decrypt and View SSL Snoop Data With Wireshark for Oracle iPlanet Web Server Note 1455999.1
- How to Enable TLSv1.2 for a Client Side SSLSocket for Java SE 7 Note 1910270.1
- There are certain known reported compatibility issues when moving from Java SE 6 to 7.
  To confirm, test if the problem disappear when the following flags are used:

  ```
        -Djsse.enableSNIExtension=false -Dcom.sun.net.ssl.enableECC=false
  ```

- For a proof of concept testing, you can follow these KM notes to set up either a one/two way SSL tomcat server
  - How To Use Client Authentication with Java Web Start Note 1507952.1
  - How to set up a Testcase Framework for Deployment of Java Web Start applications Note 1511357.1
  Next, you may use a simple test client like the following to access any document placed in the tomcat server's document directory

```
import javax.net.ssl.*;
import java.net.*; import java.io.*;

public class HTTPS {
```

```
    public static void main(String[] args) throws Exception {
        System.out.println(args[0]);
        URL url = new URL(args[0]);
        HttpsURLConnection uc = (HttpsURLConnection)url.openConnection();
        uc.connect();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(uc.getInputStream()));

        String inputLine;

        while ((inputLine = in.readLine()) != null) {
              System.out.println(inputLine);
        }
        in.close();
    }
}
```

Usage:

```
 javac HTTPS.java

 java -Djavax.net.ssl.trustStore=<keystore that contains server's cert>
 -Djavax.net.ssl.keyStore=<keystore that contains the client cert> -Djavax.net.ssl.keyStorePassword=
<password of client keystore>
 -Djavax.net.debug=all   HTTPS  <https URL to a document hosted by the server>
```

CAUTION

This sample code is provided for educational purposes only and not supported by Oracle Support Services. It has been tested internally, however, and works as documented. We do not guarantee that it will work for you, so be sure to test it in your environment before relying on it.

Proofread this sample code before using it! Due to the differences in the way text editors, e-mail packages and operating systems handle text formatting (spaces, tabs and carriage returns), this sample code may not be in an executable state when you first receive it. Check over the sample code to ensure that errors of this type are corrected.

Didn't find what you are looking for?