

Atlassian Knowledge Base

# Connecting to SSL services



This Knowledge Base article was written specifically for the **Atlassian Server** platform. Due to the [Restricted functions in Atlassian Cloud apps](#), the contents of this article **cannot be applied to Atlassian Cloud applications**.



The content on this page relates to platforms which are not supported. Consequently, Atlassian Support **cannot guarantee providing any support for it**. Please be aware that this material is provided for your information only and using it is done so at your own risk.

This page describes how to get web applications like JIRA and Confluence connecting to external servers over SSL, via the various SSL-wrapped protocols. For instance, you may want to:

- Refer to an `https://...` URL in a Confluence macro.
- Use an IMAPS server to retrieve mail in JIRA.
- Use SMTP over SSL (SMTPS) to send mail in JIRA.
- Connect to a LDAP directory over SSL.
- Set up **Application Links** over SSL.

This does not cover running your application over SSL. Please see your product's documentation to run it over SSL:

## On this page:

- [Problem Symptoms](#)
- [The Cause](#)
- [Resolution](#)
  - [Obtain and Import the Server's Public Certificate](#)
  - [Alternative KeyStore Locations](#)
  - [Debugging](#)

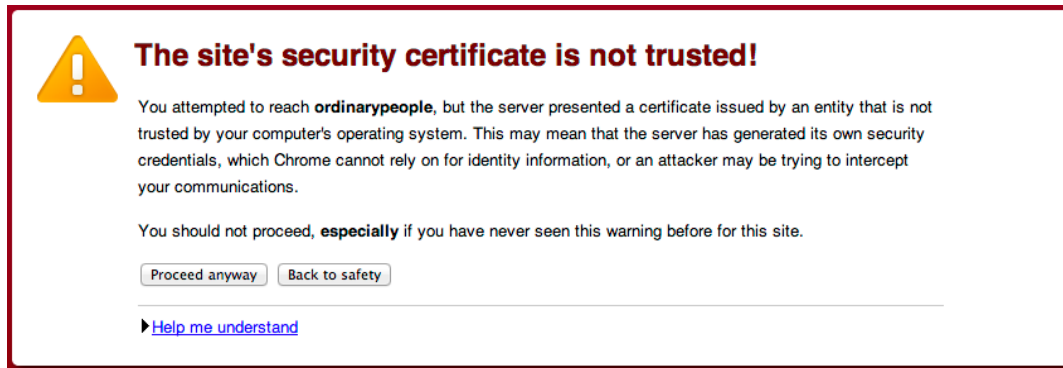
## Problem Symptoms

Attempting to access URLs that are encrypted with SSL (for example HTTPS, LDAPS, IMAPS) throws an exception and your application refuses to connect to it. For example:

```
1 javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path
```

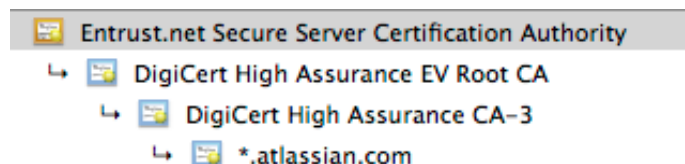
```
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid
certification path to requested target
2  at com.sun.mail.imap.IMAPStore.protocolConnect(IMAPStore.java:441)
3  at javax.mail.Service.connect(Service.java:233)
4  at javax.mail.Service.connect(Service.java:134)
```

This is the same as the following error that's generated in Chrome when visiting a page that's encrypted with a self-signed certificate, except Java can't "Proceed anyway", it just refuses the certificate:



## The Cause

Whenever your application attempts to connect to another application over SSL (e.g.: HTTPS, IMAPS, LDAPS), it will *only* be able to connect to that application if it can trust it. The way trust is handled in the Java world (this is what your application is written in) is that you have a keystore (typically `$JAVA_HOME/lib/security/cacerts`) or also known as the trust store. This contains a list of all the known CA certificates and Java will only trust certificates that are signed by those CA certificate or public certificates that exist within that keystore. For example, if we look at the certificate for Atlassian:




We can see the **\*.atlassian.com** certificate has been signed by the intermediate certificates, **DigiCert High Assurance EV Root CA** and **DigiCert High Assurance CA-3**. These intermediate certificates have been signed by the root **Entrust.net Secure Server CA**. Those three certificates combined are referred to as the certificate chain. As all of those CA certificates are within the Java keystore (`cacerts`), Java will trust any certificates signed by them (in this case, **\*.atlassian.com**). Alternatively, if the **\*.atlassian.com** certificate was in the keystore, Java would also trust that site.

This problem comes from a certificate that is either self-signed (a CA did not sign it) or the certificate chain does not exist within the Java keystore. Subsequently, your application doesn't trust the certificate and fails to connect to the application.

## Resolution

In order to resolve this, the public certificate need to be imported in the Java keystore that your

application uses. In the example above, this is \*.atlassian.com and we cover how to install it below.


 If you're unable to install Portecle on the server or prefer the command line please see our [Command Line Installation](#) section below.

## Obtain and Import the Server's Public Certificate

1. Download and install the [Portecle](#) app onto the server that runs your application.

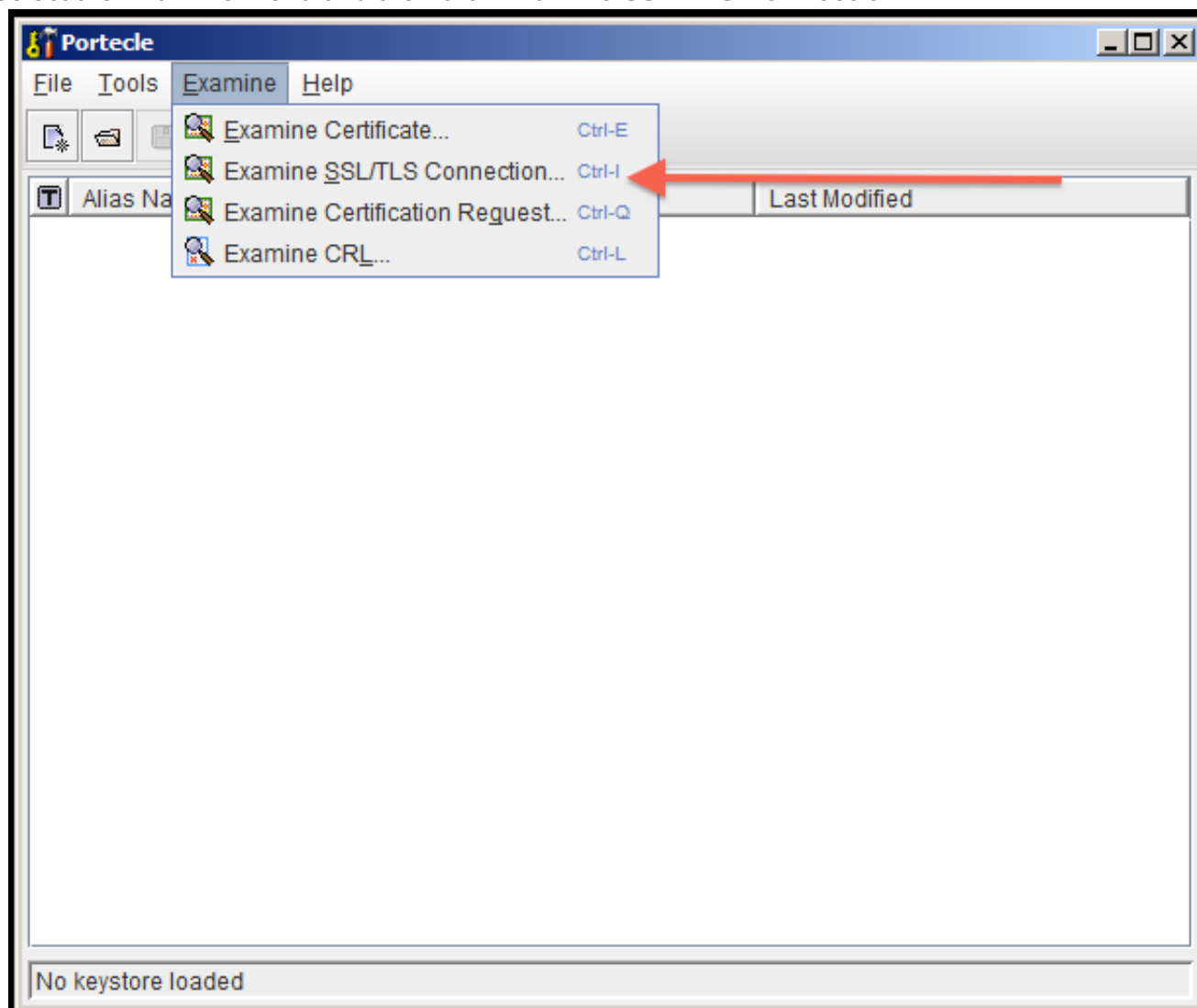
 This is a third-party application and not supported by Atlassian.

2. Ensure the <JAVA\_HOME> variable is pointing to the same version of Java that your application uses. See our [Setting JAVA\\_HOME](#) docs for further information on this.

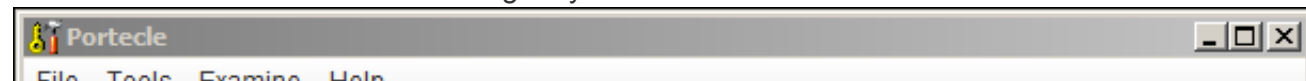
 If running on a Linux/UNIX server, X11 will need to be forwarded when connecting to the server (so you can use the GUI), as below:

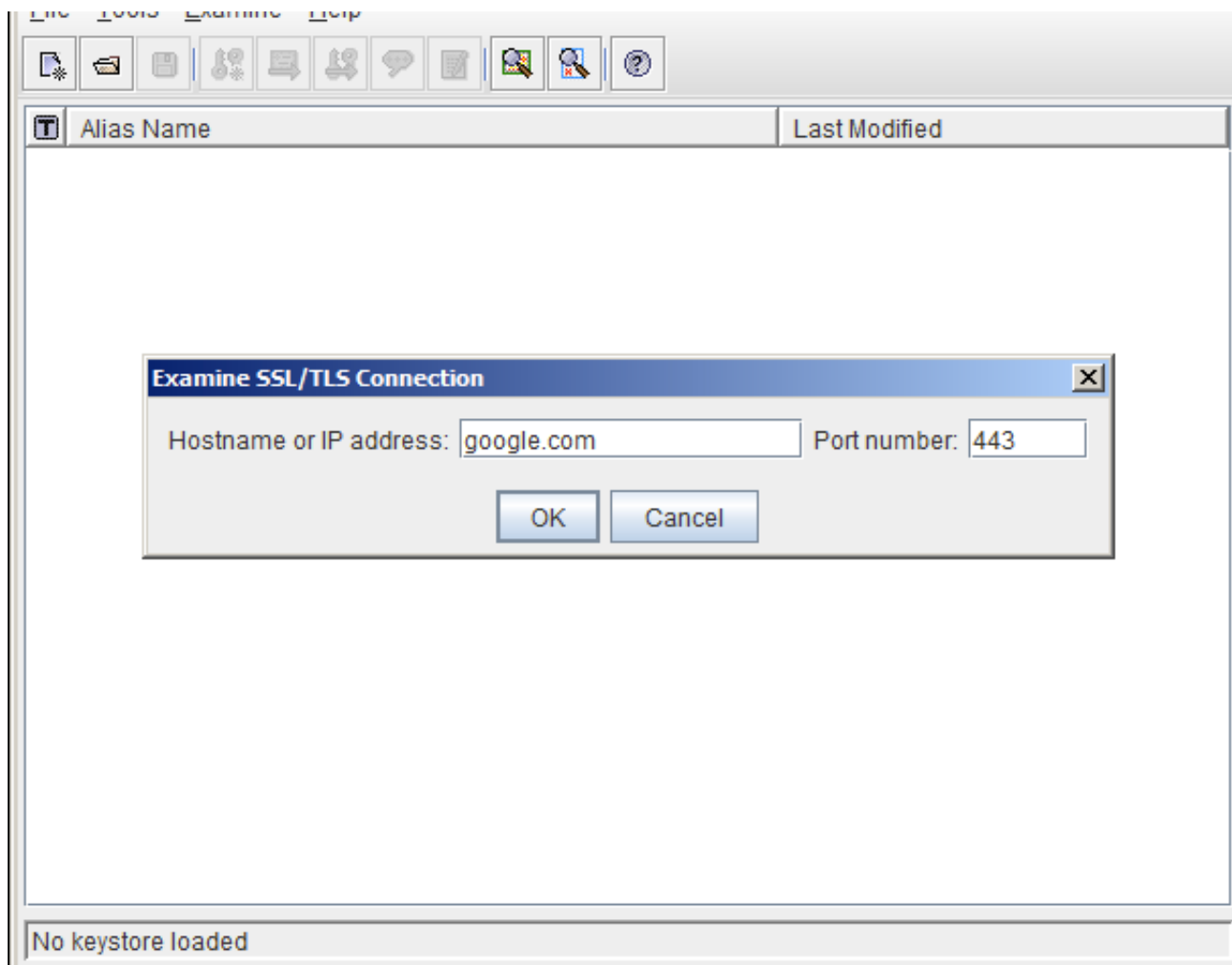
```
1  ssh -X user@server
```

3. Select the **Examine** menu and then click **Examine SSL/TLS Connection**:

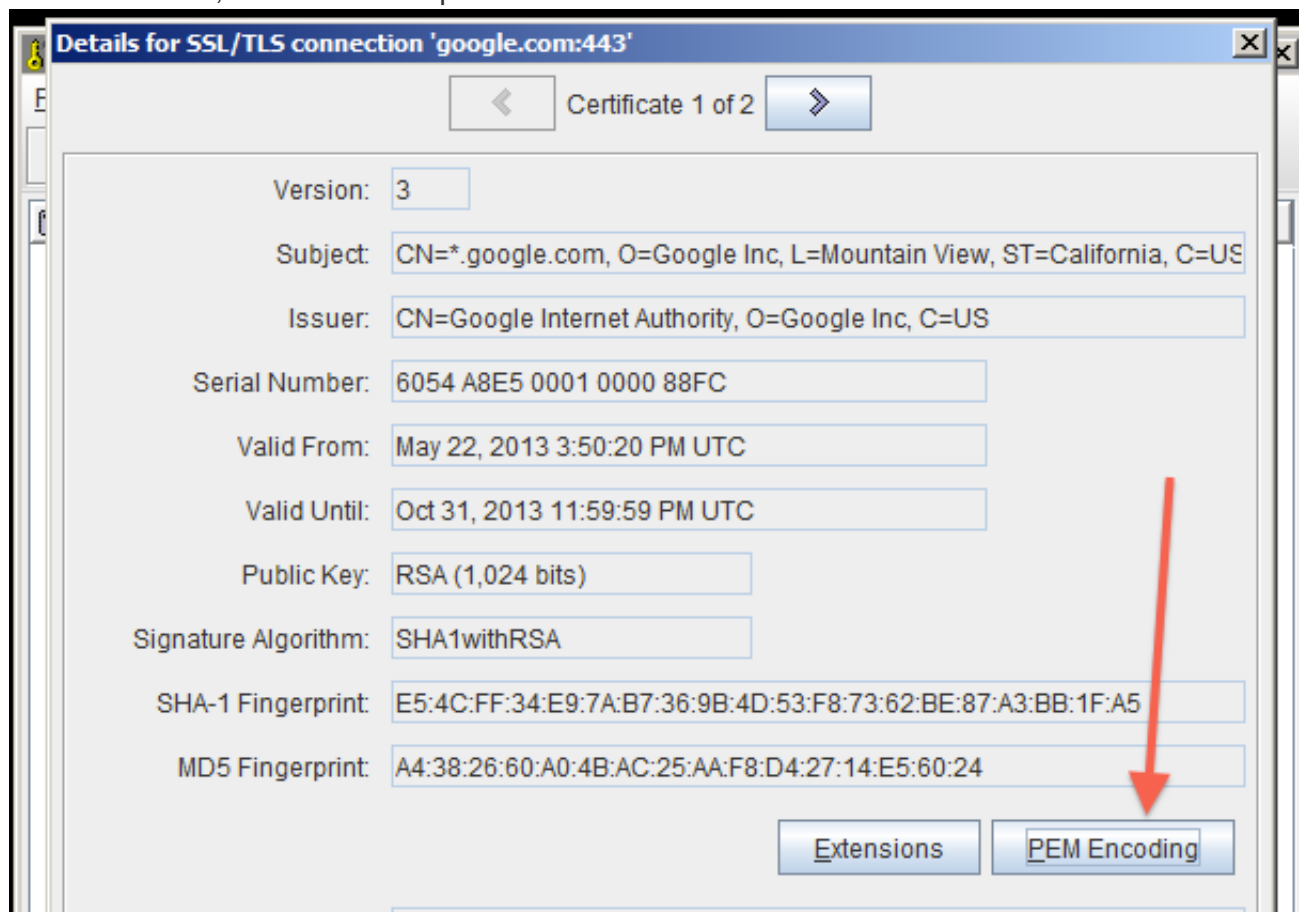


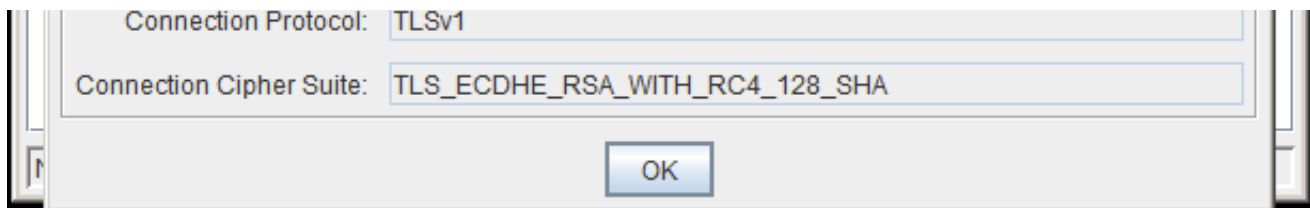
4. Enter the SSL Host and Port of the target system:



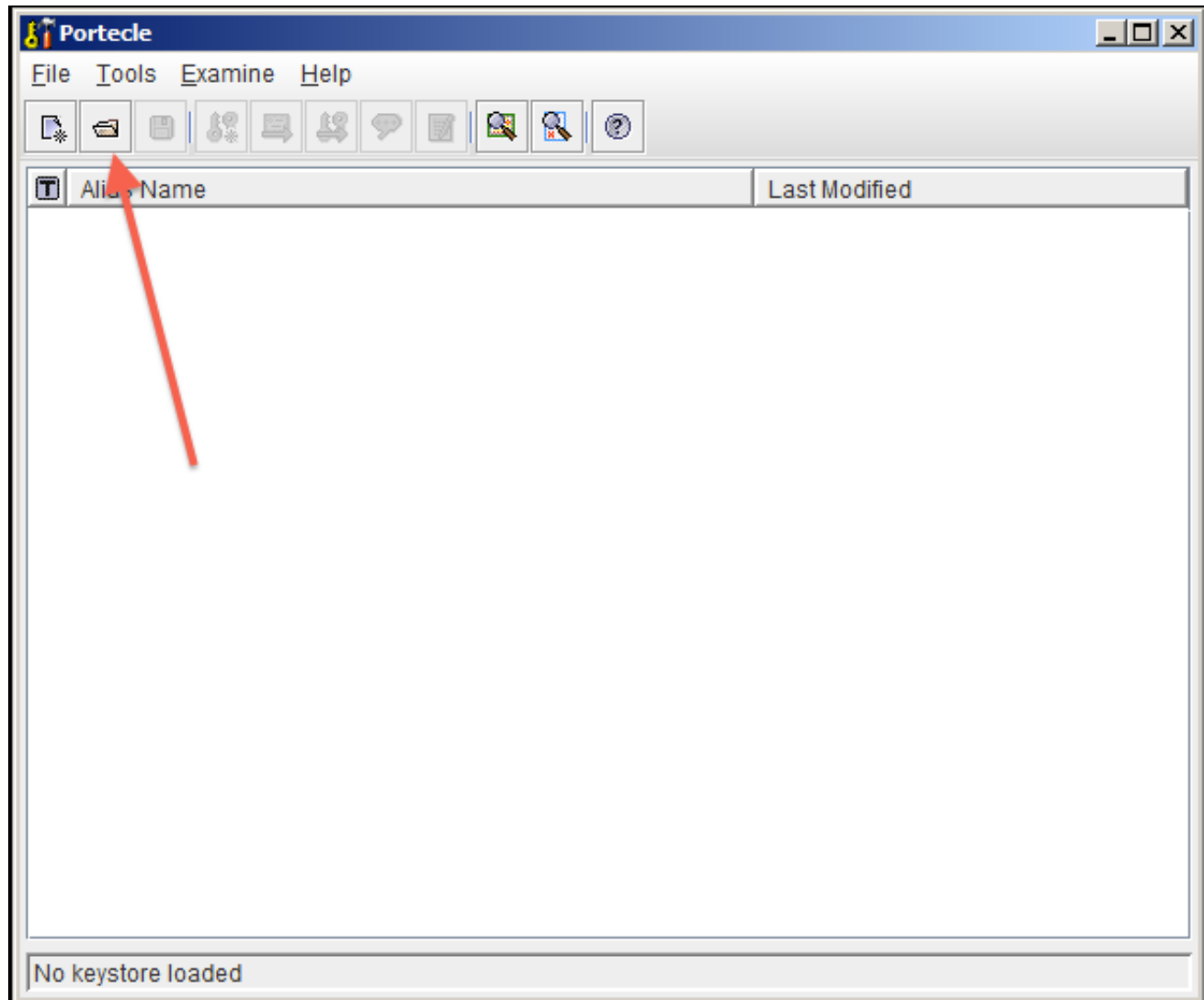


5. Wait for it to load, then select the public certificate and click on PEM:

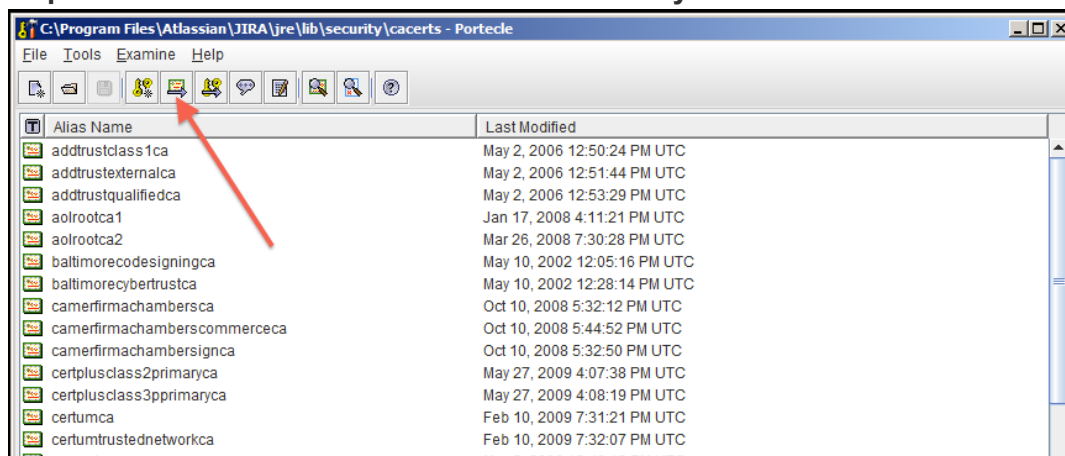


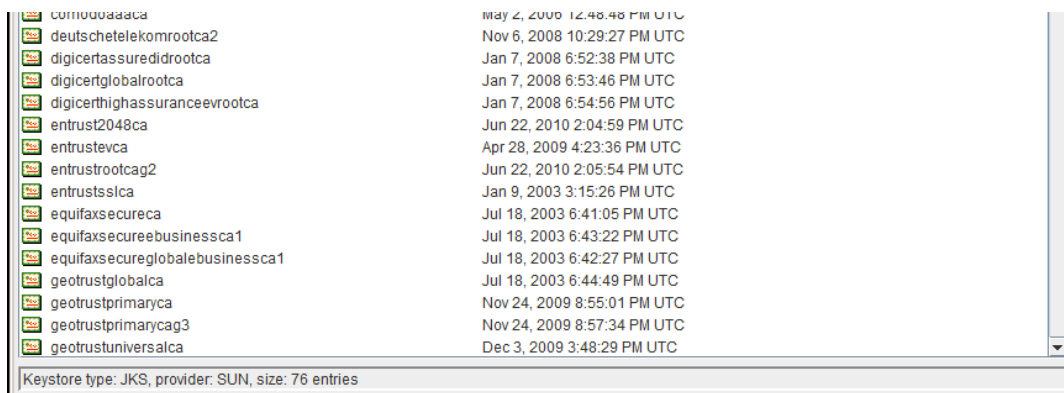


6. Export the certificate and save it.
7. Go back to the main screen and select the **Open an existing keystore from disk** option, select cacerts (for example `$JAVA_HOME/lib/security/cacerts`) then enter the password (the default is changeit).



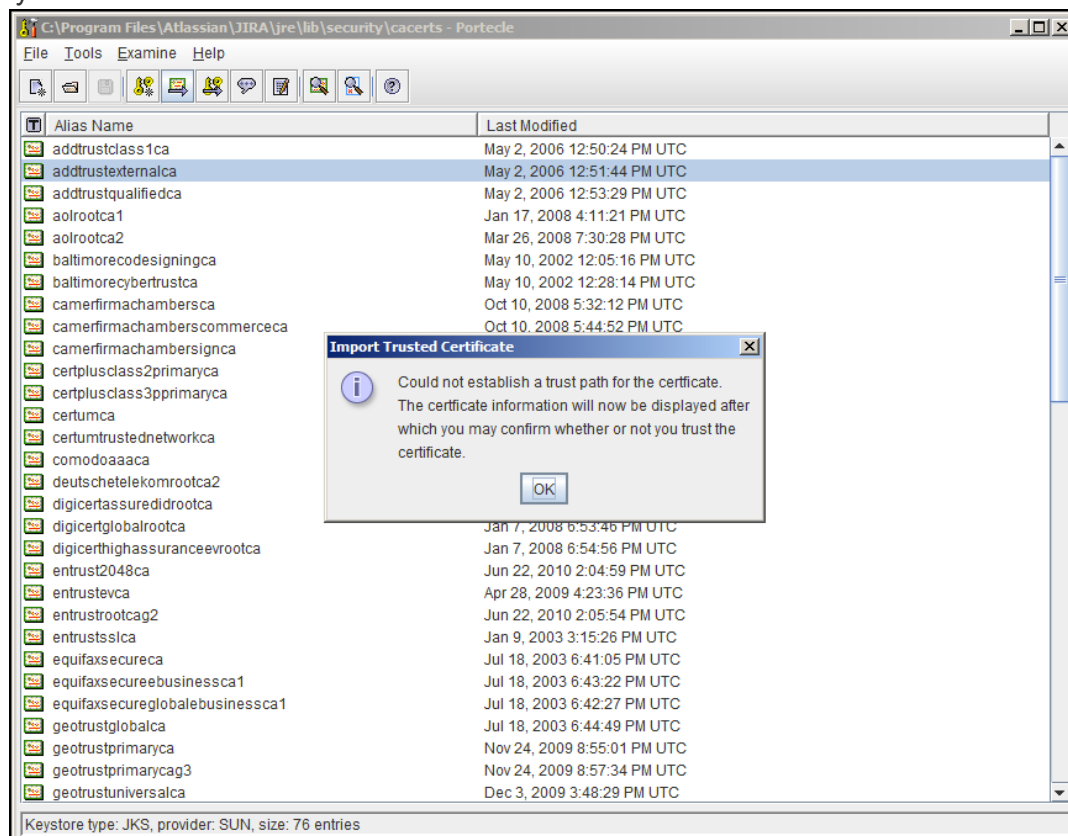
8. Select the **Import a trusted certificate into the loaded keystore** button:





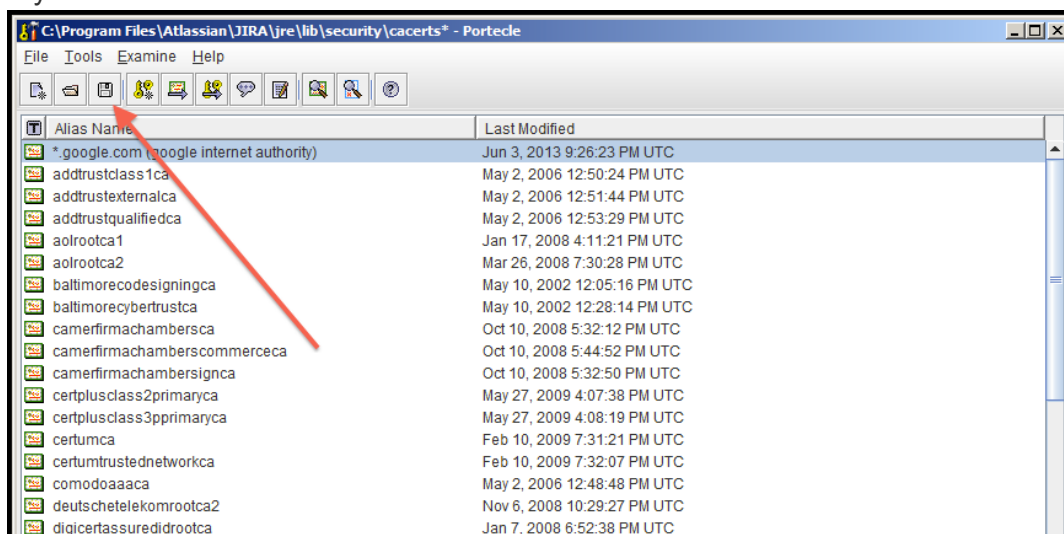
9. Select the certificate that was saved in step 6 and confirm that you trust it, giving it an appropriate alias (e.g.: confluence).













1. You may hit this error:



2. If so, hit OK, and then accept the certificate as trusted.

10. Save the Key Store to disk:



 digicertglobalrootca	Jan 7, 2008 6:53:46 PM UTC
 digicerthighassurancevrootca	Jan 7, 2008 6:54:56 PM UTC
 entrust2048ca	Jun 22, 2010 2:04:59 PM UTC
 entrustevca	Apr 28, 2009 4:23:36 PM UTC
 entrustrootcag2	Jun 22, 2010 2:05:54 PM UTC
 entrustsslca	Jan 9, 2003 3:15:26 PM UTC
 equifaxsecureca	Jul 18, 2003 6:41:05 PM UTC
 equifaxsecurebusinessca1	Jul 18, 2003 6:43:22 PM UTC
 equifaxsecureglobalebusinessca1	Jul 18, 2003 6:42:27 PM UTC
 geotrustglobalca	Jul 18, 2003 6:44:49 PM UTC
 geotrustprimaryca	Nov 24, 2009 8:55:01 PM UTC
 geotrustprimarycag3	Nov 24, 2009 8:57:34 PM UTC

Keystore type: JKS, provider: SUN, size: 77 entries

11. Restart your application.
12. Test that you can connect to the host.

## Command Line Installation

1. Fetch the certificate, replacing google.com with the FQDN of the server JIRA is attempting to connect to:

### Unix:

```
1 openssl s_client -connect google.com:443 < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > public.crt
```

### Windows:

```
1 openssl s_client -connect google.com:443 < NUL | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > public.crt
```

**i** The command above will only be executed if you have [Sed for Windows](#) as well as [OpenSSL](#) installed on your environment. If you don't have Sed or OpenSSL or you don't want to install it, use the instructions below as an alternative. Issue the following command:

```
1 openssl s_client -connect google.com:443
```

Save the output to a file called `public.cert`. Edit the `public.cert` file so it contains only what is between the `BEGIN CERTIFICATE` and `END CERTIFICATE` lines. This is how your file should look like after you edited it:

```
1 -----BEGIN CERTIFICATE-----
2 < Certificate content as fetched by the command line.
3 Don't change this content, only remove what is before
4 and after the BEGIN CERTIFICATE and END CERTIFICATE.
5 That's what your Sed command is doing for you :-> >
6 -----END CERTIFICATE-----
```

2. Import the certificate:

```
1 <JAVA_HOME>/bin/keytool -import -alias <server_name> -keystore
   <JAVA_HOME>/jre/lib/security/cacerts -file public.crt
```

## Alternative KeyStore Locations

Java will normally use a system-wide keystore in `$JAVA_HOME/jre/lib/security/cacerts`, but it is possible to use a different keystore by specifying a parameter, -

**Djvax.net.ssl.trustStore=/path/to/keystore**, where `'/path/to/keystore'` is the absolute file path of the

alternative keystore. Information on how to configure JIRA startup variables can be found [here](#).

However, setting this **is not recommended** because if Java is told to use a custom keystore (eg. containing a self-signed certificate), then Java will not have access to the root certificates of signing authorities found in `$JAVA_HOME/jre/lib/security/cacerts`, and accessing most CA-signed SSL sites will fail. It is better to add new certificates (eg. self-signed) to the system-wide keystore (as above).

## Debugging

Problems are typically one of two forms:

- The certificate was installed into the incorrect keystore.
- The keystore does not contain the certificate of the SSL service you're connecting to.

Was this helpful?  [Yes](#)  [No](#)

### Have a question about this article?

[Ask question](#)[See questions about this article](#)