Using SSL with Enhydra 5.1

Table of Contents

Chapter 1. Introduction

This document is a guide to developing Enhydra applications that use Secure Sockets Layer (SSL). Although not a tutorial, if you use the development checklist and the configuration file supplied, you can get the Golf Shop Demo that comes with the Enhydra source code working under SSL.

Chapter 2. System Requirements

- JDK1.2. from Sun Microsystems or Blackdown JDK 1.2 (on Linux)
- Your JDK must have keytool (located in JDK_HOME/jre/bin)
- you will also need to download Sun's Java Secure Socket Extension Kit from the Sun web site http://java.sun.com/products/jsse/ [http://developer.java.sun.com/developer/] .

The Java Secure Socket Extension Kit contains implementations of cryptographic algorithms, and is subject to US export restrictions: You cannot download it outside of the US and Canada. Sun has made a weakened encryption version available for export, see the JSSE [http://developer.java.sun.com/developer/] page.

Chapter 3. Background

There are two ways to use SSL with Enhydra:

- Associate Enhydra with a webserver using Enhydra director. This has to be the preferred method, any serious use of encryption will take a large amount of CPU cycles. Its far better to do this with native code than with Java. My preferred setup would be enhydra with Apache and the mod_ssl module. See http://www.modssl.org [http://www.modssl.org].
- In the case that you need a pure Java solution then use SSL support directly built in to Enhydra.

For the first option see the enhydra-director documentation and http://www.modssl.org [http://www.modssl.org]. I'll discuss the pure Java option here.

JSSE is reasonably full featured, there have been 3 releases since September with the final release in January 2000. It is included in jdk 1.4.

Chapter 4. Installation and Configuration

Here are the basic steps to go through to get SSL working with Enhydra.

- Install a version of Enhydra as usual with the built in SSL hooks.
- If building from source configure the build.xml .
- Edit your Java security policy file
- Generate or install X509 Certificates
- Modify Enhydra configuration file to add the SSL connection method

These steps are explained in detail below.

- Install Enhydra

  Make sure that you have the version of Enhydra with SSL support. Enhydra has no implementations of cryptographic algorithms, so its export outside the US is not restricted.

- Edit the Java security file

  You can find your Java security file at JDK_HOME/jre/lib/security/java.security.

  Find the list of security providers. The default is:

  ```
  security.provider.1=sun.security.provider.Sun
  ```

  To add the default JSSE security provider, add the following line:

  ```
  security.provider.2=com.sun.net.ssl.internal.ssl.Provider
  ```

  The numbering refers to the order in which the security providers are used. If you are using a vendor's implementation of JSSE, then the security provider will be something else. See your vendor documentation.

- Generate or install your X509 Certificates

  If you are testing your setup you will want to generate your X509 certificate yourself, but if you are building a production site, you will need to purchase a certificate from a certificate authority such as Thawte or Verisign.

  - Generating your private key

    Using the JDK keytool utility you can generate your own X509 certificates, but be aware this is a memory intensive operation. In generating a certificate you will need to give information to the keytool utility, do not lose this information. At the command prompt (presuming that JDK_HOME/jre/bin is in your path) enter:

    ```
    $ keytool -genkey -alias name -keyalg RSA
    ```

    Important: The program will not work without the -keyalg RSA option. Netscape uses RSA encryption, but the keytool uses DSA by default. At this point you will be prompted if the information is correct, if it is then the program will proceed to generate a self signed certificate and key. This may take some time.You will finally be prompted for a password for the certificate. Make a note of this as you will not be able to use the certificate without it.

    Do not attempt to run this command until you have changed your java.security file as described above. If you

do you will get:

```
keytool error: KeyPairGenerator not available
```

An RSA enabled provider is not provided with the default JDK1.2

Once you run the keytool command you will then be prompted for the following information:

- keystore password - if this is the first time that you are running this, it will create a keystore in your home directory and you will be prompted to create a keystore password. You will need this password every time you use any key management. The alias is the name that will identify the key in the keystore, you can have several keys in your keystore. If you do not specify an alias the default name is mykey.
- First and Last name e.g. "www.whitehouse.com"
- Name of organizational unit. This is not a company name, but the name of an internal department. e.g. "White House".
- Name of your organization e.g "US Government"
- City or location e.g "Washington"
- State or Province eg "District of Columbia"
- Country code e.g. "USA"

I find the use of first and last name to be confusing, it really wants the Fully Qualified Domain Name (FQDN) of the host that you are running your server on. In other SSL implementations they ask for the Common Name. Its important that you get this right, your certificate authority will not issue a certificate without it.

Note: Once you have entered the key information you will be prompted for a password for the private key, with the default option being that the password will be the same as the keystore. You must choose the default option. This is a bug, that should be fixed in the next version of Enhydra.

To verify that the key was properly created in the keystore you can verify it with:

```
$ keytool -list
```

- Generating a certificate request

  If you want a certificate from a recognized certificate authority, or your own self signed certificate you will need to generate a certificate request. Once you have the certificate request you can submit it to your certificate authority, or issue your own self signed certificate using either keytool or OpenSSL.

  At the command prompt type:

  ```
  $ keytool -alias name -certreq -file filename
  ```
  where name is the alias of the key in the keystore that your generating the request against.

  The specified file will be where the certificate request will be written to. If not specified the request will be output to standard out.

  If successful the certificate request should look like:

  ```
  -----BEGIN NEW CERTIFICATE REQUEST-----
  ```

```
MIIBuzCCASQCAQAwezELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExEzARBgNVBAcT
CnNhbnRhIGNydXoxDzANBgNVBAoTBmx1dHJpczEbMBkGA1UECxMSc2VjdXJlIGRldmVsb3BtZW50
MRQwEgYDVQQDEwtzdGV2ZSBsYXRpZjCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAuunCyGrr
wCCZeUAJrCvoN/n82k8IF1OwH7KNzAyaPgMU6L7CcawvWqVQY/TncHZmy5tvIlNaEJR300Ha8Keo
TxWIG7T/GHgwqBcjmt/reZbvKdKxBnT7ocoWx2G5BjHoN8RxMLQtZIc/vd9QUrelfw3WMTSLoT4A
QJiAOQpcSvECAwEAAaAAMA0GCSqGSIb3DQEBBAUAA4GBAEkeC/6FzrLO0EUAg0zaIDHazB7kKqZH
AFFOXitH2CiZVM5458NVECGdlNauRvjpwQvsRcRHC2rEpfTD0db9ISH/lN0JmDUz
-----END NEW CERTIFICATE REQUEST-----
```

The first and last lines with BEGIN and END are part of the certificate request and should not be removed.

If you want to validate the certificate request use OpenSSL or keytool. Why use OpenSSL if its not java based ? OpenSSL is more robust and has a more refined set of command line tools than keytool. OpenSSL is an open source implementation of the SSL, and TLS protocols. Its widely used with the apache mod_ssl package to provide SSL servers. If you have any doubts about open source crypto packages, note that openssl and mod_ssl are used by three commercial Apache vendors as the basis for their secure servers. See http://www.openssl.org [http://www.openssl.org] and http://www.modssl.org [http://www.modssl.org] for downloads and documentation. Once OpenSSL is installed run the command:

```
$ openssl req -noout -text -in csr
```

where csr is the name of the file containing your Certificate request.

- Submitting your certificate request

  To submit your certificate to a recognized certificate authority, see the instruction on their web page. Two well known certificate authorities are: http://www.thawte.com [http://www.thawte.com] and Verisign [http://www.verisign.com].See the instructions on their respective pages.

  If you are doing development and are creating your own self signed certificate run the command: of the key you want to associate with the certificate. Once run the certificate will be stored in the keystore.

  To validate the certificate, first export the export from the keystore:

  ```
  $ keytool -export -alias name -file filename
  ```

  where name is the alias of the associated key, and filename is the name of the file that the certificate will be written into.

  Now, to read the certificate information:

  ```
  $ keytool -printcert -file file
  ```

  where file is the name of the file with the exported certificate.

- Importing a certificate

  If you are using a certificate authority to issue your certificate, you will receive a file that looks like:

  ```
  -----BEGIN CERTIFICATE-----
  MIIC3DCCAkWgAwIBAgIDATZXMA0GCSqGSIb3DQEBBAUAMIHEMQswCQYDVQQGEwJa
  QTEVMBMGA1UECBMMV2VzdGVybiBDYXBlMRIwEAYDVQQHEwlDYXBlIFRvd24xHTAb
  BgNVBAoTFFRoYXd0ZSBDb25zdWx0aW5nIGNjMSgwJgYDVQQLEx9DZXJ0aWZpY2F0
  ```

```
aW9uIFNlcnZpY2VzIERpdmlzaW9uMRkwFwYDVQQDExBUaGF3dGUgU2VydmVyIENB
MSYwJAYJKoZIhvcNAQkBFhdzZXJ2ZXItY2VydHNAdGhhd3RlLmNvbTAeFw0wMDA3
MjQyMjMyNDBaFw0wMTA4MDcyMjMyNDBaMHoxCzAJBgNVBAYTAlVTMRMwEQYDVQQI
EwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpTYW50YSBDcnV6MSIwIAYDVQQKExlMdXRy
aXMgVGVjaG5vbG9naWVzLCBJbmMuMR0wGwYDVQQDExRiZWVsemVidWIubHV0cmlz
LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA4pMbXgVD0jBrQHW5Xqpj
jfSQ70HzCwagrUyHPtV5LbvLffInJ2mAhihlqwPxCmr0HnYIioDxtJgr/3gqfL9C
IC1/L1xlEx06IKBkFs9X4XVXPay2DzFFGnpvCvSlEjCYobHpK+QqwF8bJrnEa9Bd
oyLyxkGBGthaQkxUJARus+MCAwEAAaMlMCMwEwYDVR0lBAwwCgYIKwYBBQUHAwEw
DAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQQFAAOBgQC6xEHb6Is9jUJUf06XfWiD
wrZ4/IOYnA52bg54NVTTyjjl3qxcQpanAwajp6aAnWUYb34MuRZ8dpsYVu3TUjNF
xxgv0MWQByb4LIjv+l2JcTO4a5ZmFp7Kqp6U2XgdgcS2YYxG+mMQmTdJ3PjCB4Od
g3TILQ8TdSHnSG4YaQgNPw==
-----END CERTIFICATE-----
```

You can verify that with openssl:

```
 $ openssl x509 -noout -text -in enhydra.crt
```

Or else by using keytool:

```
 $ keytool -printcert -v -file crt
```

where crt is the name of the file containing the certificate.

Note: keytool and openssl will handle certificates in different ways. keytool will complain that a certificate is unreadable if it does contain a new line at the end of the file, while openssl will not have such a problem.

Once you have verified your certificate you can import it into your keystore by issuing the command:

```
 $ keytool -import -alias name -file certfile -trustcacerts
```

where certfile is the name of the issued certificate file, name is the name of the alias that you want to associate with the certificate. The trustcacerts option tells keytool to look in the cacerts file that can be found in the JDK/jre/lib/security directory. This file contains the root certificates for Thawte and Verisign and keytool uses them to verify the certificates you input into the keystore.

Note: If you are using a Certificate authority other than Thawte or Verisign you will have to import their root certificates into the JDK/jre/lib/security/cacerts file. To do this download the root certificate files from your Certificate Authority. then run keytool:

```
 $ keytool -import -alias name -file filename -keystore cacerts
```

where name is the alias that you want to associate with the certificate and filename is the name of the file containing the root certificate.

Chapter 5. Modifying Your Application

Now you can alter your Enhydra application configuration file so that it can find the certificates and keys. For example, here is the configuration file for the GolfShop demo shipped with the Enhydra source code. This configuration file is in the directory:

```
GOLF_SHOP_HOME/output/
```

Add the following lines to the configuration file:

```
# begin --------------------------------
Connection.golfPortSSL.Type = https
Connection.golfPortSSL.Port = 8443
Connection.golfPortSSL.SecureRandomAlgorithm =SHA1PRNG
Connection.golfPortSSL.SecureRandomProvider = SUN
Connection.golfPortSSL.SSLContextProvider = SunJSSE
Connection.golfPortSSL.SSLContextProtocol =TLS
Connection.golfPortSSL.KeyStoreLocation="/home/steve/.keystore"
Connection.golfPortSSL.KeyStoreProvider=JKS
Connection.golfPortSSL.KeyManagerAlgorithm = SUNX509
Connection.golfPortSSL.KeyManagerProvider = SunJSSE
Connection.golfPortSSL.TrustManager=JSSE
Connection.golfPortSSL.Password = your_password_here
Connection.golfPortSSL.ClientAuthentication=false
#
# Connect the port to the application
#
Channel.golfPortSSL.golfChannel.Servlet = GolfShopSSL
Channel.golfPortSSL.golfChannel.Url= /
Channel.golfPortSSL.golfChannel.Enabled = yes
#
# Specify applications (no admin).
#
Application.GolfShopSSL.ConfFile = GolfShopXMLC.conf
Application.GolfShopSSL.Description =
     "Enhydra Demo Secure Shopping Cart Application(SSL)."
Application.GolfShopSSL.Running = yes
#end ----------------------------------------
```

This example uses the XMLC implementation of the GolfShop demo.

The remainder of this section explains each line in detail.

```
Connection.golfPortSSL.Type = https
```

Define the connection method, this is required if you want to use SSL.

```
Connection.golfPortSSL.Port = 8443
```

Define the port to connect to with the HTTPS method. The default port for SSL is 443, but that is a privileged port on Unix and you will need to be root to use it. The HTTP alternative ports are in the 8000 range. If you are testing your application on a port other than 443, Internet Explorer will not be able to connect to it. Netscape does not have a problem with SSL on non-standard ports. A workaround is to use SSH port-forwarding to bind port 44 on your local machine to the port on which Enhydra is running. For example, if your Enhydra application is running on foo.bar.org on port 8443 where I am user steve, then I invoke ssh:

```
ssh -x -l steve -L 443:foo.bar.org:8443 steve@foo.bar.org
```

```
Connection.golfPortSSL.SecureRandomAlgorithm =SHA1PRNG
```

Java security provides a cryptographically strong Pseudo Random Number Generator (PRNG). This specifies the algorithm.

```
Connection.golfPortSSL.SecureRandomProvider = SUN
```

The provider refers to the providers in the java.security file, in our case SUN or SSL

```
Connection.golfPortSSL.SSLContextProvider = JSSE
```

The SSLContext Provider currently defaults to JSSE. The SSLContext holds the state of the SSL implementation. It is used to generate the factories for the sockets.

```
Connection.golfPortSSL.SSLContextProtocol =TLS
```

This currently has two defaults SSL or TLS. TLS is a protocol that is likely replacement for SSL 3.0.

```
Connection.golfPortSSL.KeyStoreLocation="/home/steve/.keystore"
```

The keystore is generated and managed by the keytool utility. The default is to have it in your home directory.

```
Connection.golfPortSSL.KeyStoreProvider=JKS
```

The key store provider

```
Connection.golfPortSSL.KeyManagerAlgorithm = SUNX509
```

Currently SUNX509 is the only default value

```
Connection.golfPortSSL.KeyManagerProvider = JSSE
```

This is currently the only provider. The name may change to SunJSSE in the future.

```
Connection.golfPortSSL.TrustManager=JSSE
```

The Trust manager

```
Connection.golfPortSSL.Password =
```

When you generated the key and certificate or the certificate request you will have had to specify the password for it. At this point, assuming that everything is correct you should be able to start up enhydra and connect to it on port 8443.

Chapter 6. Other sources of information

- For general Java security see Java Security by Scott Oakes, published by O'Reilly
- An interesting book is Java2 Network security by Marco Pistoia published by Prentice Hall. This has interesting and topical information, but the sample code had many bugs.
- The best source for JSSE are the javadoc files that come with the jar files. In particular, there is an overview.html, API_users.html, and addtional.html files which are very useful
- The comp.lang.java.security mailing list has occasional things on JSSE
- Sun has a mailing list archive at http://java.sun.com/security/hypermail/java-security-archive [http://java.sun.com/security/hypermail/java-security-archive/]. This has many interesting things related to JSSE.
- An excellent resource to learn about SSL is the open source OpenSSL libraries at http://www.openssl.org. It forms the basis of most of the commercial SSL version of Apache when combined with mod_ssl http://www.modssl.org [http://www.modssl.org].
- An excellent paper on SSL scalability issues see http://www.awe.com/mark/apcon2000/ [http://www.awe.com/mark/apcon2000/]. The authors are two of the main openssl developers.