Web Services-How to Trust Any SSL Certificate

Francis Labrie

Sometimes, it's would be useful to relax the Java security manager to allow connection to secure HTTP server using self-signed certificate, especially during development phase. To avoid Java exception on HTTPS connection, it's possible to add self signed certificate to the Java trusted X509 certificate repository using Java keystore command line tool:

```
% cd /Library/Java/Home/lib/security
% sudo keytool -import -keystore cacerts -alias anAlias -file aCertificate
```

(The default password for the "cacerts" keystore is "changeit")

But this can be painful, even more if the application must be tested with various servers using self-signed or bad defined certificate. And if a server is using a certificate with a hostname different from the one you use to test it, it will still fail.

It's also possible to change the `TrustManager` and `HostnameVerifier` in Java code, but the API did change from JDK 1.3 and 1.4. Unfortunately, the old deprecated "`com.sun.net.ssl`" package is still available, making setting of a custom `TrustManager` and `HostnameVerifier` a bit difficult.

So I've created an utility class that allow relaxing of the SSL trust rules. Simply add it to a package, an application or a framework, and call:

- `SSLUtilities.trustAllHostnames()` to turn off the default hostname verification on HTTPS connection;
- `SSLUtilities.trustAllHttpsCertificates()` to turn off the default certificate validation on HTTPS connection.

SSLUtilities.java

```
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

/**
 * This class provide various static methods that relax X509 certificate and
 * hostname verification while using the SSL over the HTTP protocol.
 *
 * @author    Francis Labrie
 */
public final class SSLUtilities {

  /**
   * Hostname verifier for the Sun's deprecated API.
   *
   * @deprecated see {@link #_hostnameVerifier}.
   */
  private static com.sun.net.ssl.HostnameVerifier __hostnameVerifier;
  /**
   * Thrust managers for the Sun's deprecated API.
   *
   * @deprecated see {@link #_trustManagers}.
   */
  private static com.sun.net.ssl.TrustManager[] __trustManagers;
  /**
   * Hostname verifier.
   */
  private static HostnameVerifier _hostnameVerifier;
  /**
   * Thrust managers.
   */
  private static TrustManager[] _trustManagers;


  /**
   * Set the default Hostname Verifier to an instance of a fake class that
```

```
 * trust all hostnames. This method uses the old deprecated API from the
 * <code>com.sun.ssl</code> package.
 *
 * @deprecated see {@link #_trustAllHostnames()}.
 */
private static void __trustAllHostnames() {
    // Create a trust manager that does not validate certificate chains
    if(__hostnameVerifier == null) {
        __hostnameVerifier = new _FakeHostnameVerifier();
    } // if
    // Install the all-trusting host name verifier
    com.sun.net.ssl.HttpsURLConnection.
        setDefaultHostnameVerifier(__hostnameVerifier);
} // __trustAllHttpsCertificates

/**
 * Set the default X509 Trust Manager to an instance of a fake class that
 * trust all certificates, even the self-signed ones. This method uses the
 * old deprecated API from the <code>com.sun.ssl</code> package.
 *
 * @deprecated see {@link #_trustAllHttpsCertificates()}.
 */
private static void __trustAllHttpsCertificates() {
    com.sun.net.ssl.SSLContext context;

    // Create a trust manager that does not validate certificate chains
    if(__trustManagers == null) {
        __trustManagers = new com.sun.net.ssl.TrustManager[]
            {new _FakeX509TrustManager()};
    } // if
    // Install the all-trusting trust manager
    try {
        context = com.sun.net.ssl.SSLContext.getInstance("SSL");
        context.init(null, __trustManagers, new SecureRandom());
    } catch(GeneralSecurityException gse) {
        throw new IllegalStateException(gse.getMessage());
    } // catch
    com.sun.net.ssl.HttpsURLConnection.
        setDefaultSSLSocketFactory(context.getSocketFactory());
} // __trustAllHttpsCertificates

/**
 * Return <code>true</code> if the protocol handler property <code>java.
 * protocol.handler.pkgs</code> is set to the Sun's <code>com.sun.net.ssl.
 * internal.www.protocol</code> deprecated one, <code>false</code>
 * otherwise.
 *
 * @return               <code>true</code> if the protocol handler
 * property is set to the Sun's deprecated one, <code>false</code>
 * otherwise.
 */
private static boolean isDeprecatedSSLProtocol() {
    return("com.sun.net.ssl.internal.www.protocol".equals(System.
        getProperty("java.protocol.handler.pkgs")));
} // isDeprecatedSSLProtocol

/**
 * Set the default Hostname Verifier to an instance of a fake class that
 * trust all hostnames.
 */
private static void _trustAllHostnames() {
    // Create a trust manager that does not validate certificate chains
    if(_hostnameVerifier == null) {
        _hostnameVerifier = new FakeHostnameVerifier();
    } // if
      // Install the all-trusting host name verifier:
    HttpsURLConnection.setDefaultHostnameVerifier(_hostnameVerifier);
} // _trustAllHttpsCertificates

/**
 * Set the default X509 Trust Manager to an instance of a fake class that
```

```
 * trust all certificates, even the self-signed ones.
 */
private static void _trustAllHttpsCertificates() {
    SSLContext context;

    // Create a trust manager that does not validate certificate chains
    if(_trustManagers == null) {
        _trustManagers = new TrustManager[] {new FakeX509TrustManager()};
    } // if
    // Install the all-trusting trust manager:
    try {
    context = SSLContext.getInstance("SSL");
    context.init(null, _trustManagers, new SecureRandom());
    } catch(GeneralSecurityException gse) {
        throw new IllegalStateException(gse.getMessage());
    } // catch
    HttpsURLConnection.setDefaultSSLSocketFactory(context.
        getSocketFactory());
} // _trustAllHttpsCertificates

/**
 * Set the default Hostname Verifier to an instance of a fake class that
 * trust all hostnames.
 */
public static void trustAllHostnames() {
    // Is the deprecated protocol setted?
    if(isDeprecatedSSLProtocol()) {
        __trustAllHostnames();
    } else {
        _trustAllHostnames();
    } // else
} // trustAllHostnames

/**
 * Set the default X509 Trust Manager to an instance of a fake class that
 * trust all certificates, even the self-signed ones.
 */
public static void trustAllHttpsCertificates() {
    // Is the deprecated protocol setted?
    if(isDeprecatedSSLProtocol()) {
        __trustAllHttpsCertificates();
    } else {
        _trustAllHttpsCertificates();
    } // else
} // trustAllHttpsCertificates

/**
 * This class implements a fake hostname verificator, trusting any host
 * name. This class uses the old deprecated API from the <code>com.sun.
 * ssl</code> package.
 *
 * @author    Francis Labrie
 *
 * @deprecated see {@link SSLUtilities.FakeHostnameVerifier}.
 */
public static class _FakeHostnameVerifier
    implements com.sun.net.ssl.HostnameVerifier {

    /**
     * Always return <code>true</code>, indicating that the host name is an
     * acceptable match with the server's authentication scheme.
     *
     * @param hostname        the host name.
     * @param session         the SSL session used on the connection to
     * host.
     * @return                the <code>true</code> boolean value
     * indicating the host name is trusted.
     */
    public boolean verify(String hostname, String session) {
        return(true);
    } // verify
```

```
} // _FakeHostnameVerifier


/**
 * This class allow any X509 certificates to be used to authenticate the
 * remote side of a secure socket, including self-signed certificates. This
 * class uses the old deprecated API from the <code>com.sun.ssl</code>
 * package.
 *
 * @author    Francis Labrie
 *
 * @deprecated see {@link SSLUtilities.FakeX509TrustManager}.
 */
public static class _FakeX509TrustManager
    implements com.sun.net.ssl.X509TrustManager {

    /**
     * Empty array of certificate authority certificates.
     */
    private static final X509Certificate[] _AcceptedIssuers =
        new X509Certificate[] {};


    /**
     * Always return <code>true</code>, trusting for client SSL
     * <code>chain</code> peer certificate chain.
     *
     * @param chain           the peer certificate chain.
     * @return                the <code>true</code> boolean value
     * indicating the chain is trusted.
     */
    public boolean isClientTrusted(X509Certificate[] chain) {
        return(true);
    } // checkClientTrusted

    /**
     * Always return <code>true</code>, trusting for server SSL
     * <code>chain</code> peer certificate chain.
     *
     * @param chain           the peer certificate chain.
     * @return                the <code>true</code> boolean value
     * indicating the chain is trusted.
     */
    public boolean isServerTrusted(X509Certificate[] chain) {
        return(true);
    } // checkServerTrusted

    /**
     * Return an empty array of certificate authority certificates which
     * are trusted for authenticating peers.
     *
     * @return                a empty array of issuer certificates.
     */
    public X509Certificate[] getAcceptedIssuers() {
        return(_AcceptedIssuers);
    } // getAcceptedIssuers
} // _FakeX509TrustManager


/**
 * This class implements a fake hostname verificator, trusting any host
 * name.
 *
 * @author    Francis Labrie
 */
public static class FakeHostnameVerifier implements HostnameVerifier {

    /**
     * Always return <code>true</code>, indicating that the host name is
     * an acceptable match with the server's authentication scheme.
     *
```

```
       * @param hostname        the host name.
       * @param session         the SSL session used on the connection to
       * host.
       * @return                the <code>true</code> boolean value
       * indicating the host name is trusted.
       */
      public boolean verify(String hostname,
          javax.net.ssl.SSLSession session) {
          return(true);
      } // verify
  } // FakeHostnameVerifier


  /**
   * This class allow any X509 certificates to be used to authenticate the
   * remote side of a secure socket, including self-signed certificates.
   *
   * @author    Francis Labrie
   */
  public static class FakeX509TrustManager implements X509TrustManager {

      /**
       * Empty array of certificate authority certificates.
       */
      private static final X509Certificate[] _AcceptedIssuers =
          new X509Certificate[] {};


      /**
       * Always trust for client SSL <code>chain</code> peer certificate
       * chain with any <code>authType</code> authentication types.
       *
       * @param chain           the peer certificate chain.
       * @param authType        the authentication type based on the client
       * certificate.
       */
      public void checkClientTrusted(X509Certificate[] chain,
          String authType) {
      } // checkClientTrusted

      /**
       * Always trust for server SSL <code>chain</code> peer certificate
       * chain with any <code>authType</code> exchange algorithm types.
       *
       * @param chain           the peer certificate chain.
       * @param authType        the key exchange algorithm used.
       */
      public void checkServerTrusted(X509Certificate[] chain,
          String authType) {
      } // checkServerTrusted

      /**
       * Return an empty array of certificate authority certificates which
       * are trusted for authenticating peers.
       *
       * @return                a empty array of issuer certificates.
       */
      public X509Certificate[] getAcceptedIssuers() {
          return(_AcceptedIssuers);
      } // getAcceptedIssuers
  } // FakeX509TrustManager
} // SSLUtilities
```

Category:WebObjects