OSTwoUserManJavaSOAPClientExample

Here is a basic example of how to to use the OpenSAML 2.x SOAP client. The example is for a SAML 1.1 assertion artifact resolution message, but is applicable to any SOAP client needs. It also illustrates use of client TLS, using an included impl static key manager of javax.net.ssl.X509KeyManager.

```
package brent.test;

import java.io.IOException;
import java.net.Socket;
import java.security.KeyException;
import java.security.Principal;
import java.security.PrivateKey;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.X509KeyManager;

import org.opensaml.DefaultBootstrap;
import org.opensaml.saml1.core.AssertionArtifact;
import org.opensaml.saml1.core.Request;
import org.opensaml.ws.soap.client.BasicSOAPMessageContext;
import org.opensaml.ws.soap.client.http.HttpClientBuilder;
import org.opensaml.ws.soap.client.http.HttpSOAPClient;
import org.opensaml.ws.soap.client.http.TLSProtocolSocketFactory;
import org.opensaml.ws.soap.common.SOAPException;
import org.opensaml.ws.soap.soap11.Body;
import org.opensaml.ws.soap.soap11.Envelope;
import org.opensaml.xml.Configuration;
import org.opensaml.xml.ConfigurationException;
import org.opensaml.xml.XMLObject;
import org.opensaml.xml.XMLObjectBuilderFactory;
import org.opensaml.xml.parse.BasicParserPool;
import org.opensaml.xml.security.SecurityException;
import org.opensaml.xml.security.SecurityHelper;
import org.opensaml.xml.security.x509.X509Credential;
import org.opensaml.xml.security.x509.X509Util;
import org.opensaml.xml.util.DatatypeHelper;
import org.opensaml.xml.util.XMLHelper;

import edu.internet2.middleware.shibboleth.DelegateToApplicationX509TrustManager;

public class SAML1ArtifactResolveExample {

    public static void main(String[] args) {

        String base64Artifact = "...base64encodedSAML1ArtifactData...";

        String serverEndpoint = "https://idp.example.org:8443/idp/profile/SAML1/SOAP/ArtifactResolve";

        String clientTLSPrivateKeyResourceName = "client.key";
        String clientTLSCertificateResourceName = "client.crt";

        try {
            DefaultBootstrap.bootstrap();
        } catch (ConfigurationException e) {
            e.printStackTrace();
        }

        BasicParserPool parserPool = new BasicParserPool();
        parserPool.setNamespaceAware(true);

        // Build the outgoing message structures
        Request request = buildSAML1ArtifactResolve(base64Artifact);

        Envelope envelope = buildSOAP11Envelope(request);

        // SOAP context used by the SOAP client
```

```
        BasicSOAPMessageContext soapContext = new BasicSOAPMessageContext();
        soapContext.setOutboundMessage(envelope);

        // This part is for client TLS support
        X509Credential clientTLSCred = getClientTLSCred(clientTLSPrivateKeyResourceName,
clientTLSCertificateResourceName);
        StaticClientKeyManager keyManager =
            new StaticClientKeyManager(clientTLSCred.getPrivateKey(), clientTLSCred.getEntityCertificate());

        // Build the SOAP client
        HttpClientBuilder clientBuilder = new HttpClientBuilder();
        clientBuilder.setHttpsProtocolSocketFactory(
                new TLSProtocolSocketFactory(keyManager, new DelegateToApplicationX509TrustManager()));

        HttpSOAPClient soapClient = new HttpSOAPClient(clientBuilder.buildClient(), parserPool);

        // Send the message
        try {
            soapClient.send(serverEndpoint, soapContext);
        } catch (SOAPException e) {
            e.printStackTrace();
        } catch (SecurityException e) {
            e.printStackTrace();
        }

        // Access the SOAP response envelope
        Envelope soapResponse = (Envelope) soapContext.getInboundMessage();

        System.out.println("SOAP Response was:");
        System.out.println(XMLHelper.prettyPrintXML(soapResponse.getDOM()));

    }

    private static Envelope buildSOAP11Envelope(XMLObject payload) {
        XMLObjectBuilderFactory bf = Configuration.getBuilderFactory();
        Envelope envelope = (Envelope) bf.getBuilder(Envelope.DEFAULT_ELEMENT_NAME).buildObject(Envelope.
DEFAULT_ELEMENT_NAME);
        Body body = (Body) bf.getBuilder(Body.DEFAULT_ELEMENT_NAME).buildObject(Body.DEFAULT_ELEMENT_NAME);

        body.getUnknownXMLObjects().add(payload);
        envelope.setBody(body);

        return envelope;
    }

    private static Request buildSAML1ArtifactResolve(String base64Artifact) {
        XMLObjectBuilderFactory bf = Configuration.getBuilderFactory();
        Request request = (Request) bf.getBuilder(Request.DEFAULT_ELEMENT_NAME).buildObject(Request.
DEFAULT_ELEMENT_NAME);
        AssertionArtifact assertionArtifact =
            (AssertionArtifact) bf.getBuilder(AssertionArtifact.DEFAULT_ELEMENT_NAME).buildObject
(AssertionArtifact.DEFAULT_ELEMENT_NAME);

        assertionArtifact.setAssertionArtifact(base64Artifact);
        request.getAssertionArtifacts().add(assertionArtifact);

        // add other data to Request as appropriate

        return request;
    }

    /* ---------------------------------------------------------- */

    private static X509Credential getClientTLSCred(String clientTLSPrivateKeyResourceName,
            String clientTLSCertificateResourceName) {
        PrivateKey privateKey = null;
        X509Certificate cert = null;

        try {
            privateKey = SecurityHelper.decodePrivateKey(DatatypeHelper.inputstreamToString(
                    SAML1ArtifactResolveExample.class.getResourceAsStream(clientTLSPrivateKeyResourceName),
```

```
null).getBytes(), null);
            cert = X509Util.decodeCertificate(DatatypeHelper.inputstreamToString(
                    SAML1ArtifactResolveExample.class.getResourceAsStream(clientTLSCertificateResourceName),
null).getBytes()).iterator().next();
        } catch (KeyException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (CertificateException e) {
            e.printStackTrace();
        }
                return SecurityHelper.getSimpleCredential(cert, privateKey);
    }

}

class StaticClientKeyManager implements X509KeyManager {

    private static final String clientAlias = "myStaticAlias";

    private PrivateKey privateKey;
    private X509Certificate cert;

    public StaticClientKeyManager(PrivateKey newPrivateKey, X509Certificate newCert) {
        privateKey = newPrivateKey;
        cert = newCert;
    }

    /** {@inheritDoc} */
    public String chooseClientAlias(String[] as, Principal[] aprincipal, Socket socket) {
        System.out.println("chooseClientAlias");
        return clientAlias;
    }

    /** {@inheritDoc} */
    public String chooseServerAlias(String s, Principal[] aprincipal, Socket socket) {
        System.out.println("chooseServerAlias");
        return null;
    }

    /** {@inheritDoc} */
    public X509Certificate[] getCertificateChain(String s) {
        System.out.println("getCertificateChain");
        return new X509Certificate[] {cert};
    }

    /** {@inheritDoc} */
    public String[] getClientAliases(String s, Principal[] aprincipal) {
        System.out.println("getClientAliases");
        return new String[] {clientAlias};
    }

    /** {@inheritDoc} */
    public PrivateKey getPrivateKey(String s) {
        System.out.println("getPrivateKey");
        return privateKey;
    }

    /** {@inheritDoc} */
    public String[] getServerAliases(String s, Principal[] aprincipal) {
        System.out.println("getServerAliases");
        return null;
    }

}
```