Securing a Web Service

Web Services are a very common way to implement a Service Oriented Architecture (SOA).

There are lots of web service standards/specifications (XML, SOAP, WSDL, UUDI, WS-*, ...) coming from organizations like W3C, OASIS, WS-I, ...
And there are java web service standards like JAX-WS 1.x (JSR 181), JAX-WS 2.0 (JSR 224).

OpenEJB provides a standard way to implement web services transport protocol throughout the JAX-WS specification.
Java basic standards for web services (JAX-WS) do lack some features that are required in most real world applications, e.g. standard ways for handling
security and authentication (there is no java specification for Oasis's WS-Security specification).

OpenEJB provides two mechanisms to secure web services - HTTP authentication and WS-Security:

- HTTPS: works at the transport level, enables a point-to-point security. It has no impact on developments. It allows you to:
  1. To secure data over the network with data encrypted during transport
  2. To identify the end user with SSLv3 with client certificate required
  3. OpenEJB supports BASIC authentication over HTTP(S), using the configured JAAS provider. This will honour any EJB security roles you
     have setup using @RolesAllowed. See the webservice-security example in the OpenEJB codebase http://svn.apache.org/repos/asf
     /openejb/trunk/openejb3/examples/

**Warning**

Currently only BASIC is the only HTTP authentication mechanism available when running OpenEJB standalone or in a unit test, but we hope to support
DIGEST in the future.

- WS-Security: works at the message (SOAP) level, enables a higher-level security,
Nowadays, SOAP implementations use other protocols than just HTTP so we need to apply security to the message itself and not only at the
transport layer. Moreover, HTTPS can only be used for securing point-to-point services which tend to decrease with Enterprise Service Bus for
example.

The Oasis organization has defined a standard (part of well-known WS-*) which aims at providing high level features in the context of web services: WS-
Security. It provides a standard way to secure your services above and beyond transport level protocols such as HTTPS. WS-Security relies on other
standards like XML-Encryption. Main features are:

  - Timestamp a message,
  - Pass credentials (plain text and/or ciphered) between services,
  - Sign messages,
  - Encrypt messages or part of messages.

Again, JAX-WS doesn't standardize security for web services. OpenEJB provides a common and highly configurable way to configure WS-Security in
association with the JAX-WS usage without vendor dependence. Internally, OpenEJB integrates Apache WSS4J as the WS-Security implementation. To
use the integration, you will need to configure WSS4J using the `openejb-jar.xml`

**Warning**

the proposed WS-Security integration is only used at server side. Currently, WS-Security client configuration is not managed by OpenEJB. You can use
the JAX-WS API to create a stub and then rely on the implementation to set up WS-Security properties.

This configuration file lets you set up incoming and outgoing security parameters. Incoming and outgoing configuration is independent so that you can
configure either one or the other or both. You can decide to check client credentials for incoming messages and sign outgoing messages (response).

Configuration principles

The configuration is made in the `openejb-jar.xml`. Each endpoint web service can provide a set of properties to customize WS-Security behavior
through the `<properties>` element. The content of this element is consistent with the overall structure of `openejb.xml`. The format for properties is the
same as if you would use a common java property file.

```
[...]
<properties>
  wss4j.in.action = UsernameToken
  wss4j.in.passwordType = PasswordDigest
  wss4j.in.passwordCallbackClass=org.superbiz.calculator.CustomPasswordHandler
</properties>
[...]
```

In order to recover WSS4J properties both for input and output, we use naming conventions. Each property is made of

```
<wss4j>.<in|out>.<wss4j property name>=<wss4j property value>
```

For example: `wss4j.in.action = UsernameToken`
Username Token (Password digest) example

Excerpt from `openejb-jar.xml`.

```xml
<openejb-jar xmlns="http://openejb.apache.org/xml/ns/openejb-jar-2.2">
  <enterprise-beans>
    ...
    <session>
      <ejb-name>CalculatorImpl</ejb-name>
      <web-service-security>
        <security-realm-name/>
        <transport-guarantee>NONE</transport-guarantee>
        <auth-method>WS-SECURITY</auth-method>
        <properties>
          wss4j.in.action = UsernameToken
          wss4j.in.passwordType = PasswordDigest
          wss4j.in.passwordCallbackClass=org.superbiz.calculator.CustomPasswordHandler
        </properties>
      </web-service-security>
    </session>
    ...
  </enterprise-beans>
</openejb-jar>
```

Request sent by the client.

This request contains SOAP headers to manage security. You can see `UsernameToken` tag from the WS-Security specification.
POST /CalculatorImplUsernameTokenHashedPassword HTTP/1.1
Content-Type: text/xml; charset=UTF-8
SOAPAction: ""
Accept: *
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.5.0_05
Host: 127.0.0.1:8204
Connection: keep-alive
Transfer-Encoding: chunked

524
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soap:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">tf7k3a4GREIt1xec/KXVmBdRNIg=</wsse:Password>
        <wsse:Nonce xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">cKhUhmjQ1hGYPsdOLez5kA==</wsse:Nonce>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <ns1:sum xmlns:ns1="http://superbiz.org/wsdl">
      <arg0>4</arg0>
      <arg1>6</arg1>
    </ns1:sum>
  </soap:Body>
</soap:Envelope>

The response returned from the server.

HTTP/1.1 200 OK
Content-Length: 200
Connection: close
Content-Type: text/xml; charset=UTF-8
Server: OpenEJB/??? (unknown os)

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:sumResponse xmlns:ns1="http://superbiz.org/wsdl">
      <return>10</return>
    </ns1:sumResponse>
  </soap:Body>
</soap:Envelope>

JAAS with WS-Security

@RolesAllowed doesn't work straight off with WS-Security, but you can add calls to the OpenEJB SecurityService to login to a JAAS provider to a CallbackHandler. Once you have done this, any permissions configured with @RolesAllowed should be honoured.

Here is a snippet from the webservice-ws-security example demonstrating this:
public class CustomPasswordHandler implements CallbackHandler {
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];

        if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
            // TODO get the password from the users.properties if possible
            pc.setPassword("waterfall");
        } else if (pc.getUsage() == WSPasswordCallback.DECRYPT) {
            pc.setPassword("serverPassword");
        } else if (pc.getUsage() == WSPasswordCallback.SIGNATURE) {
            pc.setPassword("serverPassword");
        }

        if ((pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN)
                || (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN_UNKNOWN)) {
            SecurityService securityService = SystemInstance.get().
                    getComponent(SecurityService.class);
            Object token = null;
            try {
                securityService.disassociate();
                token = securityService.login(pc.getIdentifer(), pc.getPassword());
                securityService.associate(token);
            } catch (LoginException e) {
                e.printStackTrace();
                throw new SecurityException("wrong password");
            } finally {
            }
        }
    }
}

Examples
A full example (webservice-ws-security) is available with OpenEJB Examples.