ServletSecurity

Contents

Servlet Container Security Configuration

This page explains how to use the Java Servlet container's mechanism to enforce encryption on the servlets that need it. Please see SecuringDspace for suggestions on how to determine *that*, it is not covered here.

Using this method, you can run a Java Servlet container like Tomcat as a web server, and it will automatically redirect the servlets that need encryption to the HTTPS port.

This was written for DSpace Version 1.4.1.

Background

The Java Servlet 2.4 specification, also known as JSR 154, defines an environment in which *Java Servlets* are run to provide services through a Web server. It standardizes the configuration (at the container level) as well as an API, and a packaging/delivery mechanism (WAR files).

Servlet "container" implementations like Apache Tomcat support the Java Servlet specification, so the specification serves as documentation for them.

DSpace is deployed as a conforming Java Servlet that can be expected to run in any servlet container implementing the correct version of the servlet spec.

See Also

- Consult the Running DSpace on Standard Ports page for details about setting up your server so a Java Servlet container responds to the standard HTTP and HTTPS ports.
- Read the DSpace system documentation for your release:
  - Be familiar with the configuration procedure for the Web UI
  - It also contains some details about installing and configuring Tomcat for SSL.

Configuring Tomcat 5.0

This section has details for Apache Tomcat 5.0, which may need alteration for later releases of Tomcat.

The overall configuration procedure is:

1. Configure Tomcat with an HTTP port and an HTTPS port to which it can automatically redirect servlets that require encryption.
2. Modify the webapp configuration file for the DSpace Web UI to demand encryption on the sensitive pages.
3. Deploy and test.

Tomcat Server Configuration

First get familiar with how a Tomcat server is configured.

The Tomcat configuration file is `conf/server.xml`, relative to the Tomcat installation directory. Consult the Tomcat SSL documentation for all the details about adding a Connector element for HTTPS.

Your configuration should include:

- One Connector element on a NON-SSL port with the attribute `redirectPort` indicating the HTTPS port number, e.g. `redirectPort="8443"`.
- Another Connector element, at that SSL port, marked as secure, i.e. with the `secure="true"` attribute set. It must also have the other attributes needed for HTTPS.

Example of the Connector element for HTTP:

```
          <Connector port="8080"
              maxThreads="6" minSpareThreads="2" maxSpareThreads="5"
              enableLookups="false" redirectPort="8443" acceptCount="100"
              debug="0" connectionTimeout="20000"
              disableUploadTimeout="true" />
```

Example of the Connector element for HTTPS (with keystore in `/tomcat/conf/keystore`):

```
           <Connector port="8443"
              maxThreads="6" minSpareThreads="2" maxSpareThreads="5"
              enableLookups="false" disableUploadTimeout="true"
              acceptCount="100" debug="0" scheme="https" secure="true"
              sslProtocol="TLS"
              keystoreFile="/tomcat/conf/keystore"
              keystorePass="changeit"
              clientAuth="true"
              truststoreFile="/tomcat/conf/keystore"
              trustedstorePass="changeit" />
```

This is all you need to get a request on the HTTP port automatically redirected to the HTTPS port when it requires a secure connection.

Don't forget to check the Tomcat SSL how-to for everything else you'll need to configure to implement SSL on Tomcat.

Webapp Configuration

To configure your webapp, you have to modify the `web.xml` configuration file. In the running server, this will be in the relative path `WEB-INF/web.xml` under the webapp's directory, e.g. `tomcat/webapps/dspace/WEB-INF/web.xml`. However, it comes from the WAR file when that is deployed, so to change it permanently you'll have to go to the DSpace source directory.

In the DSpace source directory, look for `etc/dspace-web.xml`. If you modify this file, and rebuild the WARs, it will get into `dspace.war`.

Add a new `security-constraint` element to the very end of the file, as the last element within the root `web-app` element. It contains a list of resources, and the constraint to apply to them. The resources are in the form of URL patterns as documented in the servlet spec – so you can refine the URLs chosen to have HTTPS required down to the page level.

Here is an example of a `security-constraint` element that requires HTTPS on the servlet classes `EditProfileServlet, LDAPServlet, PasswordServlet, RegisterServlet`. I believe these are the only places where passwords are entered.

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>Pages requiring HTTPS</web-resource-name>
     <url-pattern>/profile</url-pattern>
     <url-pattern>/register</url-pattern>
     <url-pattern>/password-login</url-pattern>
     <url-pattern>/ldap-login</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
```

After modifying the web.xml file, rebuild the WAR files and deploy the new `dspace.war`, and restart the servlet container so you can be sure it is the running version of the webapp.

Final steps

If this proves useful, consider making a patch to include the `security-constraint` element as a commented-out section in the copy of `etc/dspace-web.xml` in the DSpace source, so a site administrator could enable it by removing the comments.