Spring Security SAML module

Author: Vladimir Schäfer
E-mail: vladimir.schafer@gmail.com

The package contains the implementation of SAML v2.0 support for Spring Security framework. Following features are available:

- SP and IDP initialized WebSSO profile of the SAML v2 protocol stack
- HTTP-POST and HTTP-Redirect bindings
- Automatic generation of SP metadata
- User selection of IDP to federate with
- Multiple IDPs in the circle of trust with metadata loading from filesystem or URL
- Custom loading/storing of user data using UserDetails interface
- Fully configurable using Spring context
- Sample pre-configured web application

Internal processing of the SAML messages, marshalling and unmarshalling is handled by OpenSAML (http://www.opensaml.org/).

1.1 Prerequisites

The library is written in Java 1.5 and uses Maven as build system. For the run of the library any Java EE 5 application server or container with support for Java Servlets 2.4 should be sufficient.

Make sure that the XML parsing libraries used at the server are compatible with OpenSAML (https://spaces.internet2.edu/display/OpenSAML/OSTwoUsrManJavaInstall).

The sample web application uses JSTL 1.2 library which must be present at the application server.

1.2 Compilation

To compile modules type "mvn package". All dependencies are located in http://shibboleth.internet2.edu/downloads/maven2/ and http://repo1.maven.org/maven2/ repositories.

1.3 IDP server

Library should be able to authenticate users against any SAML 2.0 compatible Identity Provider.

Currently the NameIDs urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress, urn:oasis:names:tc:SAML:1.1:nameid-format:x509SubjectName, urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified, urn:oasis:names:tc:SAML:2.0:nameid-format:persistent and urn:oasis:names:tc:SAML:2.0:nameid-format:transient are stated as supported in the generated metadata. The implementations may use NameIDs in the UserDetails implementation. Application programmers can utilize various NameIDs with custom processing code by implementing and plugging SAMLUserDetailsService interface.

The library respects WantAuthnRequestsSigned attribute in IDP metadata and signs the AuthNRequests only if this property is true.

1.4 Application design

Library consists of two parts:

- **saml2-shared** – contains all the common classes and must be present in all projects
- **saml2-webapp** – sample web application containing recommended Spring configuration and jsp files presenting correct use of the library

Key components of the library are SAMLEntryPoint and SAMLProcessingFilter.

SAMLEntryPoint

Implements AuthenticationEntryPoint and SpringSecurityFilter interfaces and can be invoked either directly by access to a monitored URL or by ExceptionTranslationFilter upon AuthenticationException being thrown while accessing an application URL.

Entry point can function in one of the two modes of IDP discovery:

a. A default IDP has been configured which is then always used
b. User is allowed to select IDP from the list of configured ones; entry point then redirects user to the selection page.

When the IDP discovery process has finished the AuthNRequest SAML message is sent to the IDP using either HTTP-POST or HTTP-Redirect binding and the request is stored in the cache for subsequent matching with the response.

SAMLProcessingFilter

A SAML response created by the IDP is sent to the URL monitored by this filter. Processing filter parses the SAML message, wraps it into SAMLAuthenticationToken and invokes the SAMLAuthenticationProvider. By default the processing filter expects calls at /saml/SSO.

SAMLAuthenticationProvider

Performs validation using WebSSOProfileConsumer class. In case the assertion is present in the response and is valid the UsernamePasswordAuthenticationToken is returned and stored.

The UsernamePasswordAuthenticationToken contains following information:

- **Name:** value of the Subject in the SAML response
- **Credential:** SAMLCredential object (containing NameID, assertion used to authenticate the user and name of IDP who performed the authentication).
- **GrantedAuthority:** are currently not implemented, in future could be loaded from SAML attributes

Optionally the authentication provider invokes custom SAMLUserDetailsService which can load/store information about the user. For example urn:oasis:names:tc:SAML:2.0:nameid-format:persistent NameID could be stored to custom datastore for future user matching.

Details about SAML processing can be found in the implementation.

The SAML Single log-out profile is currently not supported, invocation of logout filter at /saml/logout context will only destroy the local session.

1.5 Configuration

Application is fully configurable using Spring context. At the moment no custom namespaces are implemented, but may be added in the future.

Sample configuration is located in the **saml2-webapp/src/main/resources/security/securityContext.xml**, all bean names used in the following text are referring to this document.

Keystores

The library requires one JKS keystore with at least one private key. There is a sample JKS bundled in the webapp module, information about creation of custom one can be found at http://wiki.eclipse.org/Generating_a_Private_Key_and_a_Keystore.

1) Configure *keyStore* bean:
   - Set path to the key store
   - Set password used to open the keystore
2) Configure *keyResolver* bean:
   - For each used key in the keystore set a map entry:
     <entry key="keyAlias" value="privateKeyPassword" />
3) Configure *webSSOprofile* bean:
   - Set *keyAlias* of the key you want to use for encryption/signing as constructor argument with index 2

Metadata

First step in enabling SAML Single Sign-On is exchange of SAML metadata. This process is specific for each IDP. For configuration of Spring SAML please have the IDP metadata file ready beforehand. Metadata of the hosted SP will be generated automatically after deployment.

All configured IDPs are considered to be members of a single circle of trust.

To configure metadata:

1) Prepare metadata of the IDP you want to federate with in a file or URL
2) Modify the *metadata* bean in configuration file:
   a. In case you have metadata in a file use the org.opensaml.saml2.metadata.provider.FilesystemMetadataProvider class
   b. In case you have metadata as URL use the org.opensaml.saml2.metadata.provider.HTTPMetadataProvider.
   c. Delete other unused metadata configurations
3) Optionally set the *defaultIDP* property to the value of entityID attribute of one of the configured IDPs. In case the defaultIDP value is set and custom selection of IDP is disabled user will be automatically forwarded to this IDP upon invocation of entryPoint. In case only one IDP is configured this value can be removed completely.

In case the SP metadata generator is disabled, there must be exactly one metadata document describing values of the hosted service provider.

Metadata generator

The metadata for the hosted SP (our deployed application) can be generated automatically. Beans *metadataFilter* and *metadataGenerator* are responsible for this task.

By default the SP metadata can be accessed at:

protocol://server:port/appContext/saml/metadata

The generated XML contains one assertion consumer with HTTP-POST support.

The generated metadata must be uploaded to the IDPs circle of trust.

In case the metadata generator is not needed (hand configured SP metadata is present in the *metadata* bean) both *metadataFilter* and *metadataGenerator* can be removed from the configuration.

User IDP selection

In case the *idpSelectionPath* property of *samlEntryPoint* bean is set the EntryPoint will present user with an IDP selection view instead of directly sending a SAML request. The selected IDP is sent with a GET request including parameter **login=true** and **idp=entityID** to the filter URL of *samlEntryPoint*, which then issues the SAML request to the selected IDP.

Please consult WEB-INF/security/idpSelection.jsp for an example or customization.

Custom user data loading

In order to enable custom processing of user after login create a class implementing SAMLUserDetailsService interface and a bean in the Spring configuration.

Reference to the bean must be added to the *samlAuthenticationProvider* bean as *userDetails* property and will be called after each successful authentication attempt.

Custom paths

By default all the components have default paths:

- /saml/SSO – processing filter
- /saml/metadata – metadata generator
- /saml/logout – logout filter
- /saml/login – entry point

All paths can be changed in the Spring context configuration file.

1.6 Sample application

The bundled web application demonstrates usage of the library. Please follow the above mentioned configuration steps before deployment. In particular the keystore must be set, metadata list can be left empty in case you only want to generate the SP metadata.

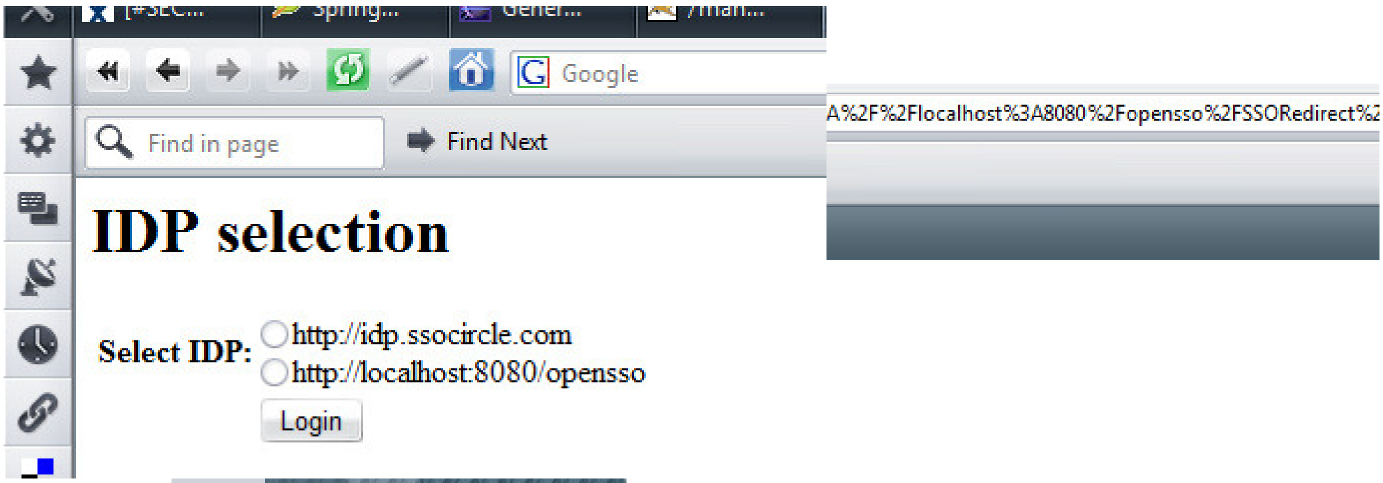
For execution of single sign-on at least one IDP must be configured.

The JSTL 1.2 is used in the application; you can get the JSTL jar at https://maven-repository.dev.java.net/repository/jstl/jars/ or more info at http://www.mularien.com/blog/2008/02/19/tutorial-how-to-set-up-tomcat-6-to-work-with-jstl-12/

The sample configuration uses user IDP selection, so upon access to the root context you will be presented with the list of configured IDPs to select from and login button will redirect you to the IDP. After login at the IDP assertion will be sent back and user logged in.

1.7 Tested environment

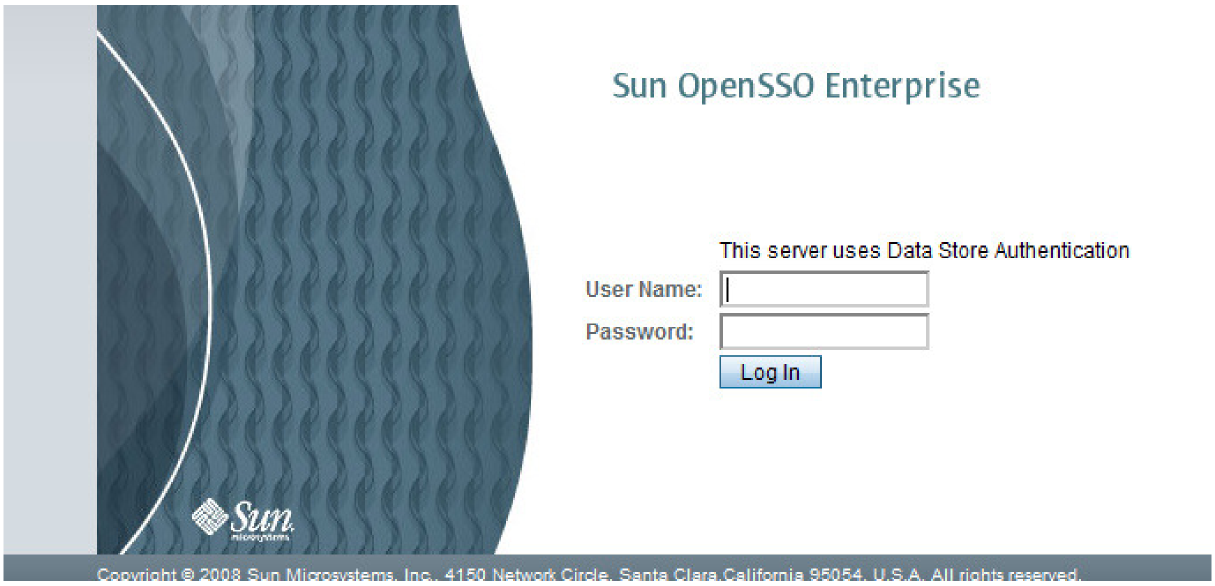
The library has been tested against OpenSSO and Weblogic 10 as IDPs. Application has been successfully deployed and used on JBoss 4.2.2 GA, Tomcat 6.0.18 and Weblogic 10.1. **Weblogic requires updates to the XML parsing libraries in order to work with OpenSAML.**



A%2F%2Flocalhost%3A8080%2Fopensso%2FSSORedirect%2

IDP selection

Select IDP: ○ http://idp.ssocircle.com ○ http://localhost:8080/opensso



This server uses Data Store Authentication

User Name:

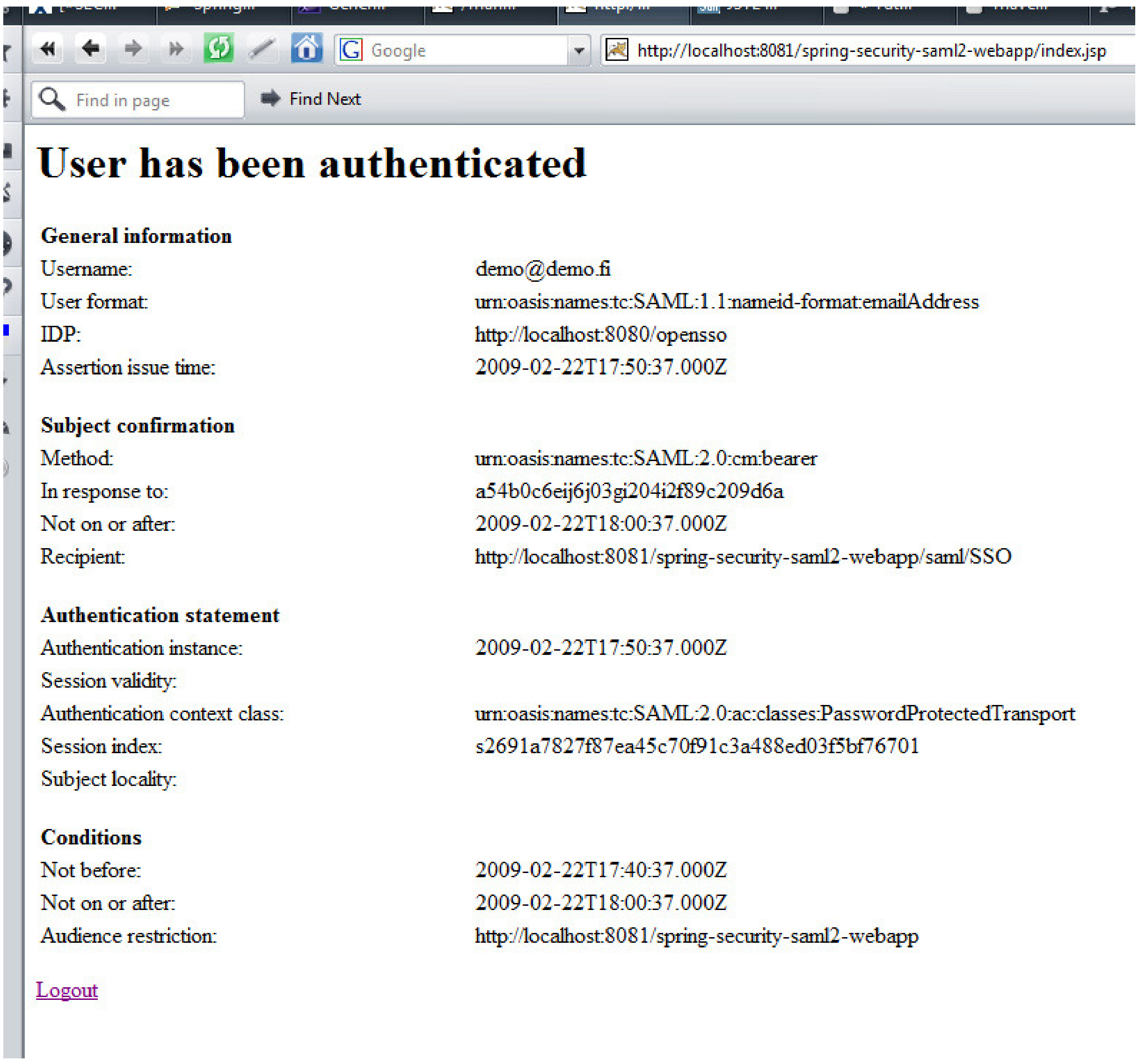
Password:

Copyright © 2008 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,California 95054, U.S.A. All rights reserved.

1.8 Future development and problems

At the moment the application is in the progress of extensive unit testing, although no bugs are known right now it may change soon ☺

- Currently the most wanted feature seems to be the single logout support
- the simplified configuration with a custom namespace would also be nice.
- cache of sent SAML requests needs a cleaning thread
- the SAMLCredential is currently not serializable but should be
- there’s never enough documentation, some more should be written


Google
http://localhost:8081/spring-security-saml2-webapp/index.jsp
Find in page
Find Next

User has been authenticated

General information

Username:	demo@demo.fi
User format:	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
IDP:	http://localhost:8080/opensso
Assertion issue time:	2009-02-22T17:50:37.000Z

Subject confirmation

Method:	urn:oasis:names:tc:SAML:2.0:cm:bearer
In response to:	a54b0c6eij6j03gi204i2f89c209d6a
Not on or after:	2009-02-22T18:00:37.000Z
Recipient:	http://localhost:8081/spring-security-saml2-webapp/saml/SSO

Authentication statement

Authentication instance:	2009-02-22T17:50:37.000Z
Session validity:	
Authentication context class:	urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
Session index:	s2691a7827f87ea45c70f91c3a488ed03f5bf76701
Subject locality:	

Conditions

Not before:	2009-02-22T17:40:37.000Z
Not on or after:	2009-02-22T18:00:37.000Z
Audience restriction:	http://localhost:8081/spring-security-saml2-webapp

Logout