

OpenSSO QA Test Automation

SAMLv2 module Testing setup & development

Prepared by Mrudula Gaidhani

1. Introduction

This document describes in detail the QA test automation focusing on SAMLv2. It explains the following:

- The Requirements for this module
- How tests are organized.
- Execution details
- Tests in the framework
- Interpreting the report
- Debugging the test failures
- How to add new tests

2. Requirements

The primary requirements are:

1. For SAMLv2 testing, framework assumes that there are two separate deployments of Federated Access Manager 8.0 samples war.
2. Before deploying the fam.war/opensso.war update the war file to include the test jsps in the war deployment :

Copy the

`<your_qatest_workspace>/source/com/sun/identity/qatest/samlv2/attrQuerytest.jsp` to the location where we have fam.war update the war file and deploy the war .

Eg: `jar uf fam.war attrQuerytest.jsp`

The other way to do it manually copy the jsp to the location where you have the FAM related jsps are such as Debug.jsp if you have already deployed the war.

3. Both the war's should be either in two different network domains or the cookie names in AMConfig.properties should be different.

There are two ways to achieve this required scenario:

- To have two war's on different domains:
 - Consider that the first war is deployed on machineA.domain.com & second war is deployed on machineB.domain.com.
 - Now on the driver (from where this QATest automation will be run) machine change machineA.domain.com to machineA.domainA.com in /etc/hosts file for Solaris. Follow the examples given in the hosts file.
192.99.999.9 machineA.domainA.com
 - Make sure to update the SP side /etc/hosts file in case IDP has been given a virtual hostname. Failure to do this, will generate UnknownHostException

- while executing SAMLv2 tests.
- To change the cookie names:
 - Configure both the war's using `http://machineA.domain.com:port/uri/Configurator.jsp`
 - Follow product instructions to proceed with the configuration. After successful configuration, open `AMConfig.properties` residing under `<Config_Dir>`.
 - Change the value of the parameter `com.iplanet.am.cookie.name`. Make sure both war's have different cookie names.
 - Now restart the web containers.

3. Organization of tests

In the section, we shall discuss how tests are organized. Currently there are testcases for following features

- Basic Profile tests
- Scenario Tests: Combination of SP initiated, IDP initiated SSO, SLO & Termination
- AutoFederation Tests
- AutoFederation with Transient User
- RelayState
- ForceAuthN
- Transient User
- Dynamic User profile creation
- Signing & Encryption
- SAMLv2NameIdMngtTests
- SAMLv2AttributeQueryTests

Test organization details:

1. The tests are divided into multiple java classes based on features listed above.
2. This is how the directory and file's are laid out for SAMLv2
 1. `<TEST_HOME>/xml/testng` contains SAMLv2 testng xml files
 2. `<TEST_HOME>/resources/samlv2` contains all properties files for SAMLv2 testing
 3. `<TEST_HOME>/source/com/sun/identity/qatest/samlv2` contains all sources of Java implementations for SAMLv2.
 4. `<TEST_HOME>/source/com/sun/identity/qatest/common/SAMLv2Common.java` contains the common function required by SAMLv2 module.
 5. `<TEST_HOME>/servername1_servername2/built/classes` will contain xml files which will be generated at run time for samlv2 testing.
3. SAMLv2 tests are divided under different groups such as
 1. `ff_ds` : Flatfile for User management & directory Server as the backend repository
 2. `ds_ds` : Directory Server for User management and the backend repository
 3. `ff_ds_sec` : Flatfile for User management & directory Server as the backend repository with Keystore configured
 4. `ds_ds_sec` : Directory Server for User management and the backend repository with Keystore configured

4. Execution details

4.1 How to execute samlv2 tests?

This section describes how to execute the samlv2 tests.

1. Deploy Federated Access Manager 8.0 wars on two different machines.
2. Before running the samlv2 tests please read Section 2 to make sure either cookie names are different or they are in two different domains.
3. Before executing this test module, user should create two Configurator-
<hostname>.properties corresponding to two war deployments. Please refer to OpenSSO QATest automation framework document for details. Update following four properties. Properties such as metaalias, entity_name will be defaulted to hostname if not specified. In case of IDP, cot will be defaulted to idpcot & for SP, it will be spcot. Property certalias should be mentioned if the keystore is configured & if ff_ds_sec or ds_ds_sec group needs to be run.
 - metaalias=@COPY_FROM_CONFIG@
 - entity_name=@COPY_FROM_CONFIG@
 - cot=@COPY_FROM_CONFIG@
 - certalias=testalias
4. Change following parameters in <TEST_HOME>/build.properties file
 1. Change the value of QATEST_HOME
 2. Change the value of TEST_MODULE to samlv2
 3. Change the value of EXECUTION_MODE to appropriate group name as described in section 3
 4. Change the value of REPORT_DIR to desired location
5. Run following command to execute samlv2 module:
ant -DSERVER_NAME1=host1 -DSERVER_NAME2=host2 module
host1 is treated as IDP while host2 is treated as SP

4.2 The execution details

1. If the driver machine (from where the QATest will be run) doesn't have access to internet, then make sure to change samlv2relaystatetests.properties file. Change the relay state url to the url which driver machine can access successfully and change all the results with the text to match on the relay state url page.
2. Run following command to execute samlv2 module:
ant -DSERVER_NAME1=host1 -DSERVER_NAME2=host2 module
host1 is treated as IDP while host2 is treated as SP. Based on the information from these two configurator files SAMLv2ConfigData.properties is created at run time. This file will be used by all the tests.
3. OpenSSO war can be just deployed on two different instances. Configuration of both the war's is handled through ConfigureSAMLv2.java. This class always runs before any of the tests are executed. That is governed by BeforeTest annotation.
4. ConfigureSAMLv2.java also creates metadata templates for SP & IDP, and loads on both sides. This will make both war's ready for SAMLv2 test execution.
5. Framework picks execution of java classes in random order. Plus all the testcases inside the java class are also executed in random order unless the order is forced by dependsOn annotation.
6. Each java class contains multiple tests marked with @Test annotation. Methods annotated by @BeforeClass are setup methods. They prepare the system for the tests to execute. The

examples are adding the users, changing metadata values, etc. Methods annotated by `@AfterClass` are cleanup method. They bring system back to original state. The examples are delete the users created in setup method, revert back the metadata changes. These methods will be run even if the tests fail.

7. Each test is run using the combination of SP & IDP. All the testing will be performed using `htmlunit` api's. It doesn't depend on `AMConfig.properties`.
8. After all the testcases are run, `UnconfigureSAMLv2.java` is run. It deletes the entities created on SP & IDP along with Circle of trusts. It brings system back to original state.
9. All the tests refer to `samlv2ConfigData.properties` which contains configuration data. This file is generated based on the `Configurator-hostname1.properties` & `Configurator-hostname2.properties`. User should not edit this file.
10. `samlv2TestData.properties` contains the data for all the tests. This file mainly contains the result messages after Single-Sign on, Single Logout, etc. If these messages are changed in the product then the user should update this file. But if message change is only for single testcase then it should go in the individual properties file. The individual properties file will override the data specified in `samlv2TestData.properties` file.

5. Tests details

5.1 Current Tests classes:

Current Tests in the QATest Framework for `samlv2` are described in the following table:

Test Class	Properties files used	Description
<code>samlv2SmokeTests.java</code>	<code>samlv2SmkokeTest.properties</code> <code>samlv2ConfigData.properties</code> <code>samlv2TestData.properties</code>	It contains SP & IDP initiated Single Sign-On, Single Logout, Termination requests covering all the profiles. Each action is independent testcase. Each testcase depends on the preceding testcase. If the testcase fails, all the dependent testcases will be skipped.
<code>samlv2Scenarios.java</code>	<code>samlv2ConfigData.properties</code> <code>samlv2TestData.properties</code>	It contains SP & IDP initiated Single Sign-On, Single Logout, Termination in scenario forms. Different combinations of profiles are tested. Currently each scenario is using one properties files.
<code>samlv2AutoFederation.java</code>	<code>samlv2autofederationtests.properties</code> <code>samlv2ConfigData.properties</code> <code>samlv2TestData.properties</code>	contains testcases related to auto federation attribute in extended metadata. Setup method is changing the metadata & loading it on SP & IDP side.
<code>samlv2TransientUserTests.java</code>	<code>samlv2transientusertests.properties</code> <code>samlv2ConfigData.properties</code>	It contains testcases related to transient attribute in extended metadata. Setup method is changing

	samlv2TestData.properties	the metadata & loading it on SP & IDP side. Transient user is set to anonymous. The testcases are to test the one time federation with anonymous user at SP side.
samlv2AutoFedTransientUserTests.java	samlv2autofedtransientusertests.properties samlv2ConfigData.properties samlv2TestData.properties	It contains testcases related to transient attribute along with autofederation in extended metadata. Setup method is changing the metadata & loading it on SP & IDP side. Transient user is set to anonymous. The testcases are to test the one time federation with anonymous user at SP side. There are no database writes on SP & IDP side during federation.
samlv2ForceAuthN.java	samlv2ForceAuthNTests.properties samlv2ConfigData.properties samlv2TestData.properties	It contains testcases related to ForceAuthN parameter to SPSSOInit.jsp. When this attribute is set to true, during federation existing IDP session will be destroyed & user will be forced to login at IDP again.
samlv2RelayStateTests.java	samlv2relaystatetests.properties samlv2ConfigData.properties samlv2TestData.properties	It contains testcases related to RelayState parameter to all jsp's. When this attribute is set, after successful action (SSO, SLO, Termination) browser should be redirected to this url.
SAMLv2AutoFedDynUserCreationTests.java	samlv2AutoFedDynUserCreationTests.properties samlv2ConfigData.properties samlv2TestData.properties	It contains testcases related to dynamic user creation at SP along with autofederation in extended metadata. Setup method is changing the metadata & loading it on SP & IDP side. Dynamic user creation is set to true at SP side. The testcases are to test the auto federation with dynamically created user at SP side.
SAMLv2DefaultRelayStateSPTests.java	samlv2DefaultRelayStateSPTests.properties samlv2ConfigData.properties samlv2TestData.properties	It contains testcases related to defaultRelayState attribute in SP extended metadata. This attribute is changed in the setup and checked if this relay state is used for SP initiated SSO, SLO and termination for both profiles.
SAMLv2SigningEncryptionTests.java	samlv2SigningEncryptionTests.properties samlv2ConfigData.properties	It contains testcases related to multiple attributes present in SP & IDP standard and extended metadata

	samlv2TestData.properties	<p>related to signing and encryption. Each attribute is set to true in the metadata in the setup. After that SP initiated and IDP initiated SSO, SLO & termination is exercised. Also both profiles for each action is used.</p> <p>The execution of this class is controlled through testng.xml. Attribute name and corresponding metadata name (SP/IDP/both) is passed through this.xml.</p>
SAMLv2NameIdMngtTests.java	samlv2TestData.properties samlv2TestConfigData.properties	<p>It contains testcases related to managed Nameid. Setup method is changing the metadata & loading it on SP & IDP side. The samlv2TestData.properties contains all the metadata creation information that is required for the creation of meta data, The values are default if nothing is modified.If any specific values need to be changed should be changed for each attribute in this properties file.</p>
SAMLv2AttributeQueryTests.java	SAMLv2AttributeQueryTests.properties samlv2TestData.properties samlv2TestConfigData.properties attrQuerytest.jsp	<p>It contains testcases related to managed Nameid. Setup method is changing the metadata & loading it on SP & IDP side. The samlv2TestData.properties contains all the metadata creation information that is required for the creation of meta data, The values are default if nothing is modified.If any specific values need to be changed should be changed for each attribute in this properties file.</p> <p>attrQuerytest.jsp is the jsp file that is needed for these tests to run,This jsp needed to be copied to the SP side containers default jsp root directory, This need to be copied to the location where we have fam related jsps such as debug.jsp etc. This setup is must for the tests to run.</p> <p>AttributeQuery also needs the following attributes need to be setup</p>

		<p>on the IDP datastore, these attributes are by default not readable or writeable unless we set these in the datastore. The following need to be set for manual datastore configuration if creating using qatest this is automatically set, for manual configuration go to the data store and set , Add the following attributes to the user attributes list</p> <p>facsimileTelephoneNumber homePhone homePostalAddress mobile pager postofficebox secretary street</p> <p>If automated configuration in the global datastore configuration add the following to IDP datastore configuration data file. "MSGGlobalDatastoreConfig0.sun-idrepo-ldapv3-config-user-attributes.0"</p> <p>add.</p> <p>facsimileTelephoneNumber homePhone homePostalAddress mobile=mobile pager=pager postofficebox secretary street</p>
--	--	---

5.2 Properties Files

- Each test class has corresponding properties file. These files mainly contain the user details used for testing. These details can be changed if needed.
- Some properties files such as SAMLv2DefaultRelayStateSPTests, samlv2relaystatetests contain the relay state URL's and the corresponding result message. If these URL's are changed, then corresponding result message also should be changed.

5.3 Common Files

There are multiple common files which are used by SAMLv2 tests.

1. SAMLv2Common.java
This file contains the SAMLv2 related common methods such as to get xml's for SSO, SLO, termination, etc.
2. MultiProtocolCommon.java

This file contains the common methods related to SAMLv2, IDFF, etc. The common methods include getting metadata from the `htmlPage`.

3. WebTest package:

As samlv2 related tests mostly use http based actions, `htmlunit` is used. The common package is `com.sun.identity.qatest.common.webtest`. This takes a input as a xml file, which contains url to go to, inputs to fill in (if any) and action to perform. It also takes a result string. Based on the occurrence of this result, an assert with pass or fail will be added. Sample xml is as follows:

```
<url href="http://machineA.domainA.com:81/idp">
<form name="Login" buttonName="" >
<input name="IDToken1" value="samlidp11" />
<input name="IDToken2" value="samlidp11" />
<result text="Federation Access Manager" />
</form>
</url>
```

Test code is as follows:

```
xmlfile = baseDir + "SPSLOSOAPdefaultRS.xml";
SAMLv2Common.getxmlSPSLO(xmlfile, configMap, "soap");
log(Level.FINE, "SPSLOSOAPdefaultRS", "Run " + xmlfile);
task = new DefaultTaskHandler(xmlfile);
page = task.execute(webClient);
```

6. Interpreting the SAMLv2 automated testing results

After execution of SAMLv2 test module, the results will be located under `<REPORT_DIR>/<HOSTNAME_1>_<HOSTNAME_2>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>`

Open `index.html` residing in this directory from the browser. It will display the overall result of the test execution with the following details:

- a link named "`<EXECUTION_MODE>-samlv2`" which points to the detailed test report
- the number of tests which passed
- the number of tests which failed
- the number of test skipped
- a link to the TestNG XML file used in this test run

To learn more about the specific tests click on the link "`<EXECUTION_MODE>-samlv2`". In the left frame of the resulting page, the individual results of all the tests which were executed. Passing tests will have a background color of green. Failed tests will have a background color of red.

To find out more information on the results a particular test click on the "Results" link for that test. This will provide you more information about the test such as when the test was executed, the duration of the test in seconds, the test method being executed, and any exception that was thrown during execution of the test.

To view all the log messages which were displayed for a particular test go to the file `<REPORT_DIR>/<HOSTNAME>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>/logs`. In this file, search for the name of the test of interest. Below the name the log records

produced during the three phases of this test's execution, setup, verification, and cleanup, can be viewed.

7. Debugging the SAMLv2 automated test failures

- To learn more about the specific tests click on the link “<EXECUTION_MODE>-samlv2”. In the left frame of the resulting page, the individual results of all the tests which were executed. Click on the link for the failed tests to see the exception reported.
- The details of each test are logged under <REPORT_DIR>/<HOSTNAME>/<EXECUTION_MODE>/<EXECUTION_DATE>/<TIME>/logs. For each test it will list the XML executed. Open those XML's to debug more.
- Also look at the debug logs on IDP and SP to see if any exceptions are reported.

8. How to add new testcases?

This section describes how to add new testcases to the existing or new java class. SAMLv2 testcases are divided into different features. New feature related tests should be added in the new java class.

To add new testcases in existing class:

1. Before adding new testcase in the existing class, make sure you understand all the existing tests in that class.
2. Logically new testcase should belong to that class.
3. If the class has dependencies between the tests then add a testcase in such a way that it doesn't break the existing dependencies. Plus make to add the new dependency for that test too.
4. Add new properties in the properties file if needed.

To add new testcases in new class:

1. Create a new class with appropriate class name. The properties files should have similar name. Please follow naming conventions described in the openSSO QATest document.
2. The class should follow setup, test(s) and cleanup procedures. Cleanup should make sure it restores the server to the default state.
3. Appropriate groups should be assigned to these newly added testcases.
4. Update all the samlv2 related testng xml files.
5. Run samlv2 module with all the tests and make sure all the tests including newly added tests are passing.

Other general considerations to be taken while writing new testcases.

1. To add the new test scenario, based on the feature it belongs to, add it to appropriate test class. Ideally new scenario shouldn't have any dependency on existing tests.
2. In testNG, tests from each class are run in random order. Thus its better not to have dependency on other test. In case it is really needed, to control the order you can

use dependsOnMethod annotation.

3. Test related setup & cleanup should be done in already existing methods. After cleanup, the system should be brought back to the original state.
4. Tests shouldn't use any command line utility, client sdk api, deployment related global properties.
5. Write a common method in SAMLv2Common class to generate such xml's. So that other test also can use it. These xml's should go under <QATEST_HOME>/built/classes dir. Each xml should have a unique name prefixed with test name.