OpenSSO QA Test Automation

IDFF module Testing setup & development

Prepared by Mrudula Gaidhani

1. Introduction

This document describes in detail the QA test automation focusing on IDFF. 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 IDFF testing, framework assumes that there are two separate deployments of Federated Access Manager 8.0 samples war.
2. 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 IDFF 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 Federation, SSO, Name registration, SLO & Termination
- AutoFederation Tests
- Default URLs from extended metadata
- ForceAuthN
- IsPassive
- NameIDPolicy

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 IDFF
   1. <TEST_HOME>/xml/testng contains IDFF testng xml files
   2. <TEST_HOME>/resources/idffcontains all properties files for IDFF testing
   3. <TEST_HOME>/source/com/sun/identity/qatest/idff contains all sources of Java implementations for IDFF.
   4. <TEST_HOME>/source/com/sun/identity/qatest/common/IDFFCommon.java contains the common function required by IDFF module.
   5. <TEST_HOME>/servername1_servername2/built/classes will contain xml files which will be generated at run time for IDFF testing.
3. IDFF 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 IDFF tests?

This section describes how to execute the IDFF tests.

1. Deploy Federated Access Manager 8.0 wars on two different machines.
2. Before running the IDFF 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 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 idff
   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 idff module:
   ant -DSERVER_NAME1=host1 -DSERRVER_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 IDFFDefaultURLsTests.properties file. Change the different URL's 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 idff module:
   ant -DSERVER_NAME1=host1 -DSERRVER_NAME2=host2 module
   host1 is treated as IDP while host2 is treated as SP. Based on the information from these two configurator files IDFFConfigData.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 ConfigureIDFF.java. This class always runs before any of the tests are executed. That is governed by BeforeSuite annotation.
4. ConfigureIDFF.java also creates metadata templates for SP & IDP, and loads on both sides. This will make both war's ready for IDFF test execution.
5. The tests are run for different protocols. The protocols are changed in the standard metadata. The parameters are passed through testng xml. ConfigureIDFFProtocols.java does the metadata manipulation based on these parameters. After the tests are run, original metadata are loaded back to get the system back to the original state. That is governed by BeforeTest & AfterTest annotations.
6. 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.
7. 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.
8. Each test is run using the combination of SP & IDP. All the testing will be performed using htmlunit api's. It doesnt depend on AMConfig.properties.
9. After all the testcases are run, UnconfigureIDFF.java is run. It deletes the entities created on SP & IDP along with Circle of trusts. It brings system back to original state.

10. All the tests refer to IDFFConfigData.properties which contains configuration data. This file is auto generated based on the Configurator-hostname1.properties & Configurator-hostname2.properties. User should not edit this file.
11. IDFFTestData.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 IDFFTestData.properties file.

5. Tests details

5.1 Current Tests classes:

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

Test Class	**Properties files used**	**Description**
IDFFSmokeTests.java	IDFFSmkokeTest.properties IDFFConfigData.properties IDFFTestData.properties	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.
IDFFAutoFederation.java	idffautofederationtests.properties IDFFConfigData.properties IDFFTestData.properties	contains testcases related to auto federation attribute in extended metadata. Setup method is changing the metadata & loading it on SP & IDP side.
IDFFForceAuthNTests.java	IDFFForceAuthNTests.properties IDFFConfigData.properties IDFFTestData.properties	It contains testcases related to ForceAuthN attribute from extended metadata. When this attribute is set to true, during federation existing IDP session will be destroyed & user will be forced to login at IDP again.
IDFFIsPassiveTests.java	IDFFIsPassiveTests.properties IDFFConfigData.properties IDFFTestData.properties	It contains testcases related to IsPassive attribute from extended metadata. When this attribute is set, IDP will not display user authentication page, first federation should fail. With existing IDP session federation will pass.
IDFFNameIDPolicyTests.java	IDFFNameIDPolicyTests.properties IDFFConfigData.properties	It contains testcases related nameIDPolicy attribute in extended metadata. By default this attribute is

	IDFFTestData.properties	set to federation. The tests include setting this attribute to none & onetime.
IDFFDefaultURLsTests.java	IDFFDefaultURLsTests.properties IDFFConfigData.properties IDFFTestData.properties	It contains testcases related to different URL attribute in SP and IDP extended metadata. These URL are used when the particular event is occurred.

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 IDFFDefaultURLsTestscontain 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 IDFF tests.

1. IDFFCommon.java
   This file contains the IDFF 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 IDFF 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="idp11" />
   <input name="IDToken2" value="sidp11" />
   <result text="Federation Access Manager" />
   </form>
   </url>
   ```
   Test code is as follows:
   ```
   xmlfile = baseDir + "SPSLOSOAPdefaultRS.xml";
   IDFFCommon.getxmlSPSLO(xmlfile, configMap, "soap");
   log(Level.FINE, "SPSLOSOAPdefaultRS", "Run " + xmlfile);
   task = new DefaultTaskHandler(xmlfile);
   page = task.execute(webClient);
   ```

6. Interpreting the IDFF automated testing results

After execution of idff 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>-idff" 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>-idff". 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 backgroup 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.

Execution output is captured in <REPORT_DIR>/<HOSTNAME>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>/idff.output file.

7. Debugging the IDFF automated test failures

- To learn more about the specific tests click on the link "<EXECUTION_MODE>-idff". 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. IDFF 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 IDFF related testng xml files.
5. Run idff 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 IDFFCommon class to generate such xml's. So that other test also can use it. These xml's should go under <QATEST_HOME>/built/classesdir. Each xml should have a unique name prefixed with test name.