OpenSSO QA Test Automation

Authorization Service

Author : Aruna Vempaty

1. Introduction

This document describes the architectural/Implementation details of the automated tests of Authorization (Policy) module of QA test automation framework, that primarily focuses on the policy service feature of OpenSSO. Policy module concentrates on the policy decision feature of the Opensso. The enforcement is automated in agents module. The document is organized in the following sections.

- Requirements for this module
- How tests are organized
- Tests/Features covered in framework
- Execution details
- Interpreting the report
- Debugging the tests failures
- How to add new tests

2. Requirements

The primary requirements are:

- Policy Service automated testing assumes that the user is familiar with the QA automation framework and the related test frameworks. Refer to “Overall Architecture of the Automation Framework” document for more details about the framework and workspace.
- Basic Knowledge about the Policy Frame Work in Federated Access Manager 8.0
- Federated Access Manager 8.0 war is deployed and configured on the supported Web or J2ee Container**.**

3. Organization of Tests

Policy Service Framework of Opensso provides several options for the policy enforcement. Policy module evaluates the policy decision returned by the server using Policy client evaluation API. The policy module of the QA test framework verifies the policy decision from the server for different services with different subjects under different conditions. Currently the following areas are being automated.

1. Different URL resources with wildcard and one level wild card support
2. allow and deny actions for post and get actions
3. LDAP subjects, Authenticated users and AMIdentity subjects
4 . Different conditions with real qualified and non qualified and time conditions
5. Validating static and dynamic attribute fetch via response provider attributes
6. Sub realm referral with explicit referral creation at the parent realm
7. DNS alias referral enabled at the top level realm for all the sub realms
8. Policy service notifications and subject change notifications

Test Organization Details

Most of the policy tests are designed to be plug and play model to ease the effort of new test case additions to the module. The policy tests are driven by the java class and associated property files. Future tests can be added by just adding the property files and adding the test definition the in testng xml file.

- The test driver for the identities and policy creation and evaluation using client.policyevaluator API is implemented in PolicyTests in policy package. This will retrieve the policy definition and evaluation related properties based on the index of tests' parameters passed from the different execution modes of the policy-testng.xml. The test driver will setup the needed infrastructure for the tests (creation of identities, policies and policy service modification etc.) in @BeforeClass annotated method. The evaluation and assertion will happen in @Test annotated method. Finally the driver will clean up the tests to bring the system to the original state in @AfterClass annotated method.
- The tests that can not leverage this plug and play model are developed in separate java files under policy. Currently the following are in separate java files.

  1. Time condition verification test cases
  2. Response Provider Test Cases

- Directory and file structure for policy module is laid out as below :

  - The above listed features are packaged under the "policy"package.
    <TEST_HOME>/source/com/sun/identity/qatest/policy
  - Each test feature implementation has the different java classes representing the feature.
    <TEST_HOME>/source/com/sun/identity/qatest/policy/PolicyTests.java (for main driver)
    <TEST_HOME>/source/com/sun/identity/qatest/policy/<featureName>.java (for other tests)
  - The utility/common classes of policy module are at
    <TEST_HOME>/source/com/sun/identity/qatest/common/policy
  - All properties and configuration files for each feature are at
    <TEST_HOME>/resources/policy
  - Runtime classes and xml files will be generated under
    <TEST_HOME>/<TEST_SERVER1>/built/classes

- Policy tests are divided under the following different groups (Execution Modes) :
  - ff_ds : Flatfile for User Management & Directory Server as the backend repository
  - ds_ds : Directory Server for User management & Directory Server as backend repository
  - ff_ds_sec : Flatfile for User Management & Directory Server as the backend repository with keystore configured (secured)
  - ds_ds_sec: Directory Server for User management & Directory Server as backend repository with keystore configured(secured)

4. Execution Details

4.1 How to execute policy tests?

This section describes how to execute the policy tests.

1. Deploy Federated Access Manager 8.0 war on desired web container.
2. Configure the server to the desired backend repository
3. Change following parameters in <TEST_HOME>/build.properties file
   1. Change the value of QATEST_HOME
   2. Change the value of TEST_MODULE to policy
   3. Change the value of EXECUTION_MODE to appropriate group name as described in section

4. Change the value of REPORT_DIR to desired location
5. Run following command to execute samlv2 module:

```
ant -DSERVER_NAME1=host module
```

4.2 The execution Details

Before running the tests, the following might have to be modified to suite the test system configuration

1. **PolicyGlobal.properties**: This file contains the global properties for policy tests. This has URL resource keys, user, group, role prefixes and suffixes for policy definition. Changing this file is recommended only if needed. Change of the resource names in this file is optional due to the fact that the evaluation is performed using the API which does not require the resource to be existent. The resource URLs can be dummy resources. This is located under <qatest>/resources/policy/

2. **PolicyTests.properties:** This file contains the properties for the identities and policy creation and for the policy evaluation that are needed to be passed as part of the policy evaluation map and expected result for all the tests. This file need not be modified for out of box execution. This is located under <qatest>/resources/policy/

3. **PolicyReferral.properties:** This file contains the referral resources and the referring organization properties. This can be changed to suite the test system

4. **<testclass>.properties:** This file contains the properties for the other tests. These are for the tests that are not part of the main test driver. These might have to be modified to suite the test system. This is located under <qatest>/resources/policy

5. Tests Details

5.1 Current Tests classes:

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

Test Class	Property files Used	Description
PolicyTests.java	PolicyTests.properties PolicyGlobal.properties PolicyReferral.properties	● This is the test driver for the most of the policy test cases. This java class is reads the properties needed for setting up the identities, policies on the server. ● It evaluates the policy for a given subject using Policy API ● It cleans up the policies and indentities set up on the server. ● This class supports one policy definition and multiple policy evaluations. ● Each test is for only one evaluation with one ssotoken. ● This does not have support for multiple evaluations with in the same token. ● This has support to execute the same tests for root realm,

Test Class	Property files Used	Description
		sub realm ● It also has the support for defining explicit referral and dynamic referral. ● Most of the tests are designed to have only one applicable policy for subject on the server at the time of evaluation (with different resource and subject for each policy)unless it is a merge of policies/decisions ● The following are automated ● AmIdentity subject, authenticated users, LDAP subjects ● AuthScheme condition(realm qualified and non qualified) ● AuthLevel condition(realm qualified and non qualified) ● Session property condition ● LDAP filter conditions ● DNS/IP condition ● Normal/ referral policies with dynamic and explicit referral ● Merge of policy decisions ● Merge of policy conditions ● Multiple subjects for the same policy ● New tests can be added by adding the properties and defining the tests in testng.xml file
PolicyRespTest.java	PolicyRespTest.properties PolicyGlobal.properties PolicyReferral.properties	● This class defines the policies with response providers and changes the policy configuration service as needed and evaluates the policy using the policy evaluation API ● This has support for the root and sub realm evaluation ● This has support for the dynamic and explicit referral ● It covers the following tests: ● Dynamic attributes (single

Test Class	Property files Used	Description
		/multi valued, null/empty) ● static attributes (single /multi valued, null/empty) ● Aggregation of the attributes ● New tests can be added by adding the properties and defining the tests in testng.xml file
PolicyTimeTest.java	PolicyTimeTest.properties	● This class defines the policies with time condition)time and date)and evaluates the policy using policy API ● The following are covered ● Day ● Date ● Time ● Combination of the above ● This has no support for the root and sub realm evaluation ● New tests can be added by adding tests in the java file with @Test annotations

5.2 Property Files

- Each Test has a corresponding property file. These files contain the properties related to policy definition and the policy evaluation. Out of box there is no need to modify these. These can be modified the the following scenarios
- To add/ evaluate the policy for the custom/pre existing resource names, change or modify the resource names in the PolicyGlobal. properties or PolicyTimeTest.properties as needed
- To add/ evaluate the policy for the custom/pre existing identity names, change or modify the resource name in the PolicyTests.properties or PolicyRespTest.properties or PolicyTimeTest.properties as required

5.3 Common Files

- There are multiple common files that are used for policy tests
- PolicyCommon.java (Common methods to create identities and policies and evaluation for root realm and sub realm referral)

6.Interpreting the Policy automated testing results

After execution of Policy test module, the results will be located under <REPORT_DIR>/<HOSTNAME>/<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>-policy” that points to the detailed test report
- the number of tests that were passed
- the number of tests that were skipped
- the number of tests that were failed

- 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>-policy".In the left frame of the results page, the individual results of all the tests that 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 test failures

- To learn more about the specific tests click on the link "<EXECUTION_MODE>- policy". In the left frame of the resulting page, the individual results of all the testswhich were executed. Click on the link for the failed tests to see the exception.
- 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.
- General Errors:

  1. **Assertion error:**The policy decision from the server is not matching the expected result defined in the properties file
     - Make sure that the policy definition and the evaluation properties are defined correctly.
     - Now check for the exceptions in the logs file (for famadm.jsp exceptions while loading the policy or creating the Identities, realms or policy service modifications needed for the test setup)
     - Confirm from the logs that the identities and policies are created correctly by running this command
         - grep –i created logs
     - If no errors are found in policy setup, then check the amPolicy debug log for the policy decision.
     - Policy tests are not designed to have multiple policies on the system while evaluating the policy. This could lead to the wrong decision. Make sure that the setup and cleanup is happening properly from the logs.
  2. **Auth loginException:** Configuration is not Found
     - Confirm that the authentication module for the realm for which the user ssotoken
  3. **Authentication is not success**
     - This happens due to the SSOToken is not getting created for the user. Check to make sure that the identity is being created properly
  4. **@BeforeClass Error:** Policy or Identity setup has failed
     - Make sure that the test definition in the < EXECUTION_MODE >policy-testng.xml is correct. For PolicyTests and PolicyRespTest, make sure to have the setup enabled for the first test.
  5. **@BeforeClass Error**: Policy or Identity cleanup has failed
     - Make sure that the test definition in the < EXECUTION_MODE >policy-testng.xml is correct. For PolicyTests and PolicyRespTest, make sure to have the cleanup enabled for the last test.

8. How to add new tests

The following are the steps to add new tests

- To add new test cases that can be executed as part of the test driver.
    - Add the URL resources in the PolicyGlobal properties. Make sure not to delete the existing URLs as they are
    - all referenced in the policy definition and evaluation of the tests
    - Add the URL resources in the PolicyReferral.properties. Make sure not to delete the existing URLs as they are all referenced in the policy definitions and evaluations of the module
    - Add the new policy tests under policytests<i> or can add the tests under any existing policy tests if they logically belong to the existing test group
        - Add identity creation properties under policytests<i>.identity.<property>
        - Add policy creation properties under policytests<i>.policy<j>.<property>
        - Add policy evaluation properties under policytess<i>.evaluation<j>.<property>
        - Add evaluation result and description for each of the policy.
    - Add the new test's definition in < EXECUTION_MODE >policy-testng.xml file under <qatest>/xml/testng and add the following parameters (all of them are required)
        - policy index which is the policytest index in policytests.properties
        - evaluation index which is the evaluation for a given policy. (one policy definition with n evaluations)
        - boolean for setup (true for the first test in the policytests.policy counter)
        - boolean for cleanup (true for the last test in the policytests.policy counter)
        - boolean for dynamic referral (true or false based on the requirement)
        - peAtOrg for the value of the realm where the policy evaluation is performed. The policy will be defined and the evaluated using the subject's ssotoken
- To add new test cases that can not be part of the test driver please follow the steps.
    - Add the <testname>.java under policy or module under source/...../policy
    - The class should have setup, test and cleanup methods. Cleanup should make sure that it will restore the server back to the original state
    - Appropriate groups should be assigned to the tests
    - Create the properties file under <qatest>/resources/policy with <testname>.properties
    - Add the test definition in < ExecutionMode>-policy-testng.xml

- Other general considerations to be taken while writing new testcases.
    - All the policy related common methods should be added to PolicyCommon. java
    - All the Identitiy related common methods should be added to IDMCommon. java
    - All the service related common methods should be added to SMSCommon. java
    - Test related setup & cleanup should be added to the main test case java file. After cleanup, the system should be brought back to the original state.
    - Make sure to have the groups added to @Test annotated methods
    - Make sure to have the test definitions are added to the different test groups testng xml file
    - Make sure to not to have multiple applicable policies for a single subject evaluation while designing the tests.