NCICB

Ekagra

High Impact – High Value – Business Results

Common Security Module Web Services

Application Developers Guide

Version No: 1.0

Last Modified: 12/03/06

Author: Vijay Parmar

Team : Common Security Module (CSM)

Purchase Order# 34552

Client : National Cancer Institute - Center for Bioinformatics,

National Institutes of Health,

US Department of Health and Human Services

Document History

Document Location

The most current version of this document is located on the CSM website:
http://ncicb.nci.nih.gov/core/CSM

Revision History

Version Number	Revision Date	Author	Summary of Changes
1.0	12/03/06	Vijay Parmar	Draft

Review

Name	Team/Role	Version	Date Reviewed	Reviewer Comments
Kunal Modi	Team Lead	1.0	12/03/06	

Related Documents

More information can be found in the following related CSM documents:

Document Name
CSM's Design Document

These and other documents can be found on the CSM website: http://ncicb.nci.nih.gov/core/CSM

Table of Contents

1. Introduction to Common Security Module's Security Web Service 4
1.1 Purpose 4
1.2 Scope 4
1.3 Using This Guide 4

2. CSM Web Services Overview 5
2.1 Explanation **Error! Bookmark not defined.**
2.2 Web Service Operations 5
2.2.1 Login 5
2.2.2 Checkpermission 6
2.3 Workflow for CSM Web Services 7

3. CSM Web Services Deployment 9
3.1 Deployment Steps 9

CSM Guide for Application Developers

1. Introduction to Common Security Module's Security Web Service

1.1 Purpose

This document provides all the information application developers need to successfully integrate with NCICB's Common Security Module (CSM) Security Web Services (Security WS). The CSM Security WS was chartered to expose CSM's Authentication and Authorization services. Clients, either Java or non-Java, can easily consume the exposed services via Web Services

1.2 Scope

This document demonstrates the features of CSM Security WS. It also shows how to deploy CSM Web Services.

1.3 Using This Guide

Begin by reading the CSM Security Web Services Overview section to learn the CSM Web Services concepts and how they apply to your own application. Next read the Workflow for CSM Web Services Integration section to understand how to successfully integrate with CSM Security WS. Finally, the Deployment Models section describes how to deploy CSM Security Web services.

2. CSM Web Services

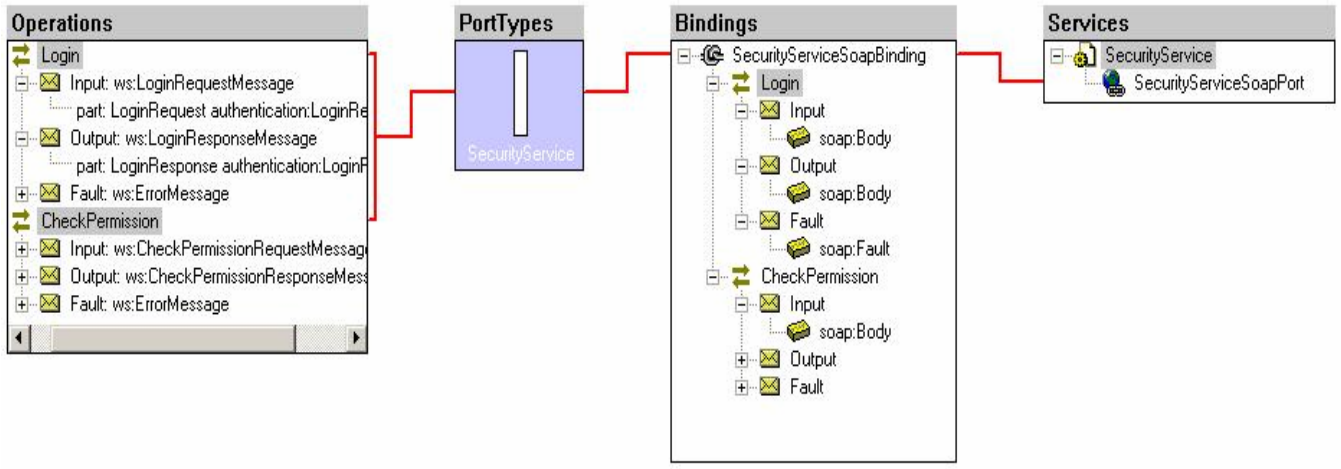
2.1 Overview

The Common Security Module Security Web Services are introduced to expose the CSM authentication and Authorization service features. The Security Web Services currently provide only two operations; namely Login and CheckPermission. The operations are exposed versions available in CSM API's.

2.2 Web Service WSDL and Operations

2.2.1 Security Web Services WSDL

The CSM Security Web Service WSDL is shown in the below. The name of the exposed web service is 'SecurityService'. Currently two operations are available namely Login and CheckPermission. The web service operations are explained in detail in the following sections.



2.2.2 Login Operation

The Login web service operation is a request/response operation. This operation receives a LoginRequestMessage, performs authentication and responds with LoginResponseMessage to the web service consumer. If there are any problems with the processing the LoginRequestMessage and/or performing authentication on the user credentials then the web service operation will return a SOAP Fault response error message indicating an error code and the error details.

```
<xs:schema targetNamespace="http://security.nci.nih.gov/ws/authentication"
  xmlns:authentication="http://security.nci.nih.gov/ws/authentication"
  elementFormDefault="qualified"
  attributeFormDefault="qualified"
  version=".1">
<xs:element name="LoginRequest" type="authentication:LoginRequest"/>
 <xs:complexType name="LoginRequest">
  <xs:sequence>
   <xs:element name="UserName" type="xs:string"/>
   <xs:element name="Password" type="xs:string"/>
   <xs:element name="ApplicationContext" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="LoginResponse" type="authentication:LoginResponse"/>
  <xs:complexType name="LoginResponse">
   <xs:sequence>
    <xs:element name="Result" type="xs:boolean"/>
   </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 1 Schema (XSD) for Authentication

As displayed in the Figure 1. The LoginRequest message consists of three parameters, namely Username, password and ApplicationContext. The Apache AXIS framework validates all request and response messages against the Schema specified in the Security WS WSDL. When the LoginRequest message is received by the web service operation, the User credentials from the LoginRequest message are used by the CSM API to authenticate the user against privilege for the 'ApplicationContext'. If the User is authenticated and has privilege to access the ApplicationContext then a LoginResponse is returned with result value of 'true'. If the user is not authenticated and does not have access privilege for the 'ApplicationContext' then a LoginResponse is returned with the result value of 'false'.

2.2.3 Checkpermission

The Checkpermisson web service operation is a request/response operation. This operation receives a CheckPermissionRequestMessage, performs a permission check and responds with CheckPermissionResponseMessage. If there are any problems then the web service operation will return a SOAP Fault response error message indicating an error code and the error details.

```
<xs:schema targetNamespace="http://security.nci.nih.gov/ws/authorization"
  xmlns:authorization="http://security.nci.nih.gov/ws/authorization"
  elementFormDefault="qualified"
  attributeFormDefault="qualified" version=".1">
<xs:element name="CheckPermissionRequest"
type="authorization:CheckPermissionRequest"/>
<xs:complexType name="CheckPermissionRequest">
   <xs:sequence>
     <xs:choice>
          <xs:element name="UserName" type="xs:string"/>
          <xs:element name="GroupName" type="xs:string"/>
     </xs:choice>
     <xs:element name="ObjectId" type="xs:string"/>
     <xs:element name="Attribute" type="xs:string" nillable="true"/>
     <xs:element name="Privilege" type="xs:string"/>
     <xs:element name="ApplicationContext" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:element name="CheckPermissionResponse"
type="authorization:CheckPermissionResponse"/>
<xs:complexType name="CheckPermissionResponse">
   <xs:sequence>
     <xs:element name="Result" type="xs:boolean"/>
```

Figure 2 Schema for Authorization

As displayed in the Figure 2. The CheckPermission request message consists of User name or Group name, ObjectId, Attribute, Privilege and ApplicationContext. The Apache AXIS framework validates all request and response messages against the Schema specified in the Security WS WSDL. When the CheckPermission request message is received by the web service operation, the CSM API's checkpermission method is invoked to check permission. If the User or Group has permission then a CheckPermissionResponse is returned with result value 'true' otherwise result value is 'false'.

2.3 Workflow for CSM Web Services

This workflow section outlines the basic steps, both strategic and technical, for successful CSM Security Web Services integration.

1) Read the deployment steps from this document and also read the CSM Guide for Application Developers. It provides an overview, workflow, and specific deployment and integration steps.

2) Determine the security requirements and provision security with CSM's UPT.

3) After the Security Web Service is deployed and user security provisioned with UPT. The Security Web Service is ready operable and consumption

4) Using the CSM Web Services Interface use the authentication and authorization operation exposed.

5) Using the LoginRequestMessage invoke and consume Login Web Service Operation.

6) Using the CheckPermissionRequestMessage invoke and consume the CheckPermission Web Service operation.

3. Deployment Model

3.1 Deployment Steps

Step 1: Create and Prime MySQL Database

1. Log into the database using an account id which has permission to create new databases. As you follow the deployment steps, use the files containing the name corresponding with your database. Make sure that the database you are about to create doesn't already exist. If it does, then drop it to recreate new one.

2. In the AuthSchemaMySQL.sql file replace the `<<database_name>>` tag with the target applications scheme – **csmupt**.

3. Run this script on the database prompt. This should create a database with the given name.

4. In the `DataPrimingMySQL.sql` file, replace:

   The `<<super_admin_login_id>>` with the login id of the user who is going to act as the Super Admin for that particular installation. For example **"**doej" for John Doe admin.

   Also provide the first name and last name for the same by replacing
   `<<super_admin_first_name>>` with `Doe` and
   `<<super_admin_last_name >>` with `Joe.`

5. Replace the <<application_context_name>> with a test application entry – '**abc_app'.** For example: Application name is 'abc_app' and application schema name is 'abc_app'. For the sake of this document we will use schema 'abc_app' and the application as 'abc_app'.

6. Run the script on the database prompt. This should populate the database with the initial data. Verify by querying the `application`, `user`, `protection_element` and `user_protection_element` tables. They should have one record each. The database will include CSM Standard Privileges and the privilege table should have 7 entries.

Step 2: Configure Datasource

1. Modify the `mysql-ds.xml` file which contains information for creating a data source. One entry is required for each database connection. Edit this file to replace:

   The --database_user_name-- with the user id. . `--database_user_password--` with the password of the user account.

   The `--database_url--` with the URL needed to access the Authorization Schema residing on the database server - **jdbc:mysql://<<stage_database_server_name>>:<<port>>/<<database_name>>**
   Shown in Figure is an example `mysql-ds.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
  <local-tx-datasource>
    <jndi-name>abc_app_ds</jndi-name>
    <connection-url>
jdbc:mysql://<<database_server_name>>:<<port>>/<<database_name>></connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>name</user-name>
    <password>password</password>
  </local-tx-datasource>
</datasources>
```

Figure 3 *Example mysql-ds.xml file*

2. Place the `mysql-ds.xml` file in the JBoss **deploy** directory - `{jboss-home}/server/default/deploy/`

Step 3: Create Directory

1. Create a directory on the server where all the configuration files pertaining to the UPT will be kept. This directory can have any name and can reside anywhere on the server. However, it should be accessible to the JBoss id running the UPT.

Step 4: Configure Hibernate

1. The provided `hibernate.cfg.xml` file needs to be modified to include configuration details to connect to the appropriate UPT Authorization Schema. For the property connection.datasource, replace the `<<abc_app_ds>>` with the application name for the UPT. For example, the property may contain `java:/abc_app_ds or java:/abc_app.` This application name should be the same as the one created in Step 1.

2. Replace the `<<database_dialect>>` with "***`MySQLDialect`***" if you are using MySQL as the authorization database or "***`OracleDialect`***" if you are using Oracle as the authorization database.

3. Rename this file to <<abc_app>>`.hibernate.cfg.xml` (for example add 'abc_app' or any preferred Prefix). Place this file in the directory created in Step 3. Make sure that the JBoss id has access to it.

Step 5: Modify ApplicationSecurityConfig.xml

1. Edit the provided ApplicationSecurityConfig.xml.

2. Replace the `<<upt_context_name>>` with the application name for the UPT. This application name should be same as the one created in Step 1.. For example 'abc_app'.

3. Replace the `<<hibernate_cfg_file_path>>` with the fully qualified path of hibernate configuration file created in Step 3. For example C:\foo\bar\abc_app`.hibernate.cfg.xml or`

`/foo/bar/abc.app.hibernate.cfg.xml depending on the target environment.`

4. Place this file in the directory. Make sure that the JBoss id has access to it.

Step 6: Make an Addition to the JBoss Startup Properties File

1. Edit the `JBoss properties-service.xml` to provide a startup parameter to the JBoss server. This file is located at the following path: `{jboss-home}/server/standard/deploy/properties-service.xml` where `{jboss-home}` is the base directory where JBoss is installed on the server. Add the following entry to the existing properties:

```
<attribute name="Properties"> <!-- could already exist -->
:
gov.nih.nci.security.configFile=/foo/bar/ApplicationSecurityConfig.xml
:
</attribute> <!-- could already exist -->
```

The `gov.nih.nci.security.configFile` is the name of the property which points to the fully qualified path `foo/bar/ApplicationSecurityConfig.xml` where the `ApplicationSecurityConfig.xml` has been created in Step 4. The name of the property must be the `gov.nih.nci.security.configFile` and cannot be modified, as it is a system-wide property.

2. Save this file in a deploy folder (for example, `{jboss-home}/server/default/deploy/`).

Note: When deploying to JBoss, the `properties-service.xml` file is already located in the folder `{jboss-home}/server/default/deploy/`.

Step 7: Configure the JBoss JAAS Login Parameters

1. In order to configure the CSM Web Service to verify against the LDAP, create an entry in the `login-config.xml` of JBoss as shown in *Figure* . This entry configures a login-module against the ‘abc_app’ application context. The location of this file is `{jboss-home}/server/default/conf/login-config.xml`.

```
<application-policy name = "abc_app">
  <authentication>
    <login-module code = "gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule"
flag = "required" >
      <module-option name="ldapHost">ldaps://ncids4a.nci.nih.gov:636</module-option>
      <module-option name="ldapSearchableBase">ou=nci,o=nih</module-option>
      <module-option name="ldapUserIdLabel">cn</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figure 4 *Example login-config.xml entry*

As shown in *Figure* :

The `application-policy` is the name of the application for defining the authentication policy – in this case, '**abc_app'**.

The `login-module` is the `LoginModule` class which is used to perform the authentication task; in this case, it is - **`gov.nih.nci.security.authentication.loginmodules.LDAPLoginModule`**.

The `flag` provided is "**`required`**".

The `module-options` list the parameters which are passed to the `LoginModule` to perform the authentication task. In this case, they are pointing to the NCICB LDAP Server:

```
<module-option name="ldapHost">ldaps://ncids4a.nci.nih.gov:636</module-
   option>

<module-option name="ldapSearchableBase">ou=nci,o=nih</module-option>

<module-option name="ldapUserIdLabel">cn</module-option>
```

Simultaneously you can also point to a RDBMS database containing the username and password information. The configuration steps for the same are provided in the CSM's Guide for Application's Developers

Step 8: Deploy the war file.

1. Copy the securityws`.war` in the deployment directory of JBoss which can be found at `{jboss-home}/server/default/deploy/.`