Implementing the ExternalizableSecurityCheck

Overview

The abstract `ExternalizableSecurityCheck` class implements the `SecurityCheck` interface and handles two important aspects of the security check functionality: externalization and state management.

- Externalization - this class implements the `Externalizable` interface, so that the derived classes don't need to implement it themselves.
- State management - this class predefines a `STATE_EXPIRED` state, which means that the security check is expired and its state is not preserved. The derived classes need to define other states supported by their security check.

Three methods are required to be implemented by the subclasses: `initStateDurations`, `authorize`, and `introspect`.

This tutorial explains how to implement the class and demonstrates how to manage states.

Prerequisites: Make sure to read the Authorization concepts (../) and Creating a Security Check (../creating-a-security-check) tutorials.

Jump to:

The initStateDurations Method

The `ExternalizableSecurityCheck` defines an abstract method called `initStateDurations`. The subclasses must implement that method by providing the names and durations for all states supported by their security check. The duration values usually come from the security check configuration.

```
private static final String SUCCESS_STATE = "success";

protected void initStateDurations(Map<String, Integer> durations) {
    durations.put (SUCCESS_STATE, ((SecurityCheckConfig) config).successStateExpirationSec);
}
```

> For more information about security check configuration, see the configuration class section (../credentials-validation/security-check/#configuration-class) in the Implementing the CredentialsValidationSecurityCheck tutorial.

The authorize Method

The `SecurityCheck` interface defines a method called `authorize`. This method is responsible for implementing the main logic of the security check, managing states and sending a response to the client (success, challenge, or failure).

Use the following helper methods to manage states:

```
protected void setState(String name)
```

```
public String getState()
```

The following example simply checks whether the user is logged-in and returns success or failure accordingly:

```
public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest request,
AuthorizationResponse response) {
   if (loggedIn){
      setState(SUCCESS_STATE);
      response.addSuccess(scope, getExpiresAt(), this.getName());
   } else  {
      setState(STATE_EXPIRED);
      Map <String, Object> failure = new HashMap<String, Object>();
      failure.put("failure", "User is not logged-in");
      response.addFailure(getName(), failure);
   }
}
```

The `AuthorizationResponse.addSuccess` method adds the success scope and its expiration to the response object. It requires:

- The scope granted by the security check.
- The expiration of the granted scope.
  The `getExpiresAt` helper method returns the time at which the current state expires, or 0 if the current state is null:

  ```
  public long getExpiresAt()
  ```

- The name of the security check.

The `AuthorizationResponse.addFailure` method adds a failure to the response object. It requires:

- The name of the security check.
- A failure `Map` object.

The `AuthorizationResponse.addChallenge` method adds a challenge to the response object. It requires:

- The name of the security check.
- A challenge `Map` object.

The introspect Method

The `SecurityCheck` interface defines a method called `introspect`. This method must make sure that the security check is in the state that grants the requested scope. If the scope is granted, the security check must report the granted scope, its expiration, and a custom introspection data to the result parameter. If the scope is not granted, the security check does nothing.
This method might change the state of the security check and/or the client registration record.

```
public void introspect(Set<String> checkScope, IntrospectionResponse response) {
   if (getState().equals(SUCCESS_STATE)) {
      response.addIntrospectionData(getName(),checkScope,getExpiresAt(),null);
   }
}
```

The AuthorizationContext Object

The `ExternalizableSecurityCheck` class provides the `AuthorizationContext authorizationContext` object which is used for storing transient data associated with the current client for the security check.
Use the following methods to store and obtain data:

- Get the authenticated user set by this security check for the current client:

  ```
  AuthenticatedUser getActiveUser();
  ```

- Set the active user for the current client by this security check:

  ```
  void setActiveUser(AuthenticatedUser user);
  ```

The RegistrationContext Object

The `ExternalizableSecurityCheck` class provides the `RegistrationContext registrationContext` object which is used for storing persistent/deployment data associated with the current client.
Use the following methods to store and obtain data:

- Get the user that is registered by this security check for the current client:

  ```
  AuthenticatedUser getRegisteredUser();
  ```

- Register the given user for the current client:

  ```
  setRegisteredUser(AuthenticatedUser user);
  ```

- Get the public persistent attributes of the current client:

  ```
  PersistentAttributes getRegisteredPublicAttributes();
  ```

- Get the protected persistent attributes of the current client:

  ```
  PersistentAttributes getRegisteredProtectedAttributes();
  ```

- Find the registration data of mobile clients by the given search criteria:

  ```
  List<ClientData> findClientRegistrationData(ClientSearchCriteria criteria);
  ```

Sample Application

For a sample that implements the `ExternalizableSecurityCheck`, see the Enrollment (../enrollment) tutorial.