Seam Security

1. Security - Introduction 1
1.1. Overview 1
1.1.1. Authentication 1
1.1.2. Identity Management 1
1.1.3. External Authentication 1
1.1.4. Authorization 1
1.2. Configuration 2
1.2.1. Maven Dependencies 2
1.2.2. Third Party Dependencies 3
2. Security - Authentication 5
2.1. Basic Concepts 5
2.2. Built-in Authenticators 6
2.3. Which Authenticator will Seam use? 6
2.4. Writing a custom Authenticator 7
3. Security - Identity Management 11
3.1. TO DO 11
4. Security - External Authentication 13
4.1. TO DO 13
5. Security - Authorization 15
5.1. TO DO 15

Security - Introduction

1.1. Overview

The Seam Security module provides a number of useful features for securing your Java EE application, which are briefly summarised in the following sections. The rest of the chapters contained in this documentation each focus on one major aspect of each of the following features.

1.1.1. Authentication

Authentication is the act of establishing, or confirming, the identity of a user. In many applications a user confirms their identity by providing a username and password (also known as their *credentials*). Seam Security allows the developer to control how users are authenticated, by providing a flexible *Authentication API* that can be easily configured to allow authentication against any number of sources, including but not limited to databases, LDAP directory servers or some other external authentication service.

If none of the built-in authentication providers are suitable for your application, then it is also possible to write your own custom Authenticator implementation.

1.1.2. Identity Management

Identity Management is a set of useful APIs for managing the users, groups and roles within your application. The identity management features in Seam are provided by PicketLink IDM, and allow you to manage users stored in a variety of backend security stores, such as in a database or LDAP directory.

1.1.3. External Authentication

Seam Security contains an external authentication sub-module that provides a number of features for authenticating your application users against external authentication services, such as OpenID and SAML.

1.1.4. Authorization

While *authentication* is used to confirm the identity of the user, *authorization* is used to control which actions a user may perform within your application. Authorization can be roughly divided into two categories; coarse-grained and fine-grained. An example of a coarse-grained restriction is allowing only members of a certain group or role to perform a privileged operation. A fine-grained restriction on the other hand may allow only a certain individual user to perform a specific action on a specific object within your application.

There are also rule-based permissions, which bridge the gap between fine-grained and coarse-grained restrictions. These permissions may be used to determine a user's privileges based on any type of business logic.

1.2. Configuration

1.2.1. Maven Dependencies

The Maven artifacts for all Seam modules are hosted within the JBoss Maven repository. Please refer to the *Maven Getting Started Guide* [http://community.jboss.org/wiki/MavenGettingStarted-Users] for information about configuring your Maven installation to use the JBoss repository.

To use Seam Security within your Maven-based project, it is advised that you import the Seam BOM (Bill of Materials) which declares the versions for all Seam modules. First declare a property value for `${seam.version}` as follows:

```
<properties>
  <seam.version>3.0.0.Final</seam.version>
</properties>
```

You can check the *JBoss Maven Repository* [https://repository.jboss.org/nexus/content/groups/public/org/jboss/seam/seam-bom/] directly to determine the latest version of the Seam BOM to use.

Now add the following lines to the list of dependencies within the `dependencyManagement` section of your project's `pom.xml` file:

```
<dependency>
  <groupId>org.jboss.seam</groupId>
  <artifactId>seam-bom</artifactId>
  <version>${seam.version}</version>
  <type>pom</type>
  <scope>import</scope>
</dependency>
```

Once that is done, add the following dependency (no version is required as it comes from `seam-bom`):

```
<dependency>
  <groupId>org.jboss.seam.security</groupId>
  <artifactId>seam-security</artifactId>
</dependency>
```

It is also possible to import the security module as separate API and implementation modules, for situations where you may not want to use the default implementation (such as

testing environments where you may wish to substitute mock objects instead of the actual implementation). To do this, the following dependencies may be declared instead:

```
<dependency>
  <groupId>org.jboss.seam.security</groupId>
  <artifactId>seam-security-api</artifactId>
</dependency>

<dependency>
  <groupId>org.jboss.seam.security</groupId>
  <artifactId>seam-security-impl</artifactId>
</dependency>
```

If you wish to use the external authentication module in your application to allow authentication using OpenID or SAML, then add the following dependency also:

```
<dependency>
  <groupId>org.jboss.seam.security</groupId>
  <artifactId>seam-security-external</artifactId>
</dependency>
```

1.2.2. Third Party Dependencies

Security - Authentication

2.1. Basic Concepts

The majority of the Security API is centered around the `Identity` bean. This bean represents the identity of the current user, the default implementation of which is a session-scoped, named bean. This means that once logged in, a user's identity is scoped to the lifecycle of their current session. The two most important methods that you need to know about at this stage in regard to authentication are `login()` and `logout()`, which as the names suggest are used to log the user in and out, respectively.

As the default implementation of the `Identity` bean is named, it may be referenced via an EL expression, or be used as the target of an EL action. Take the following JSF code snippet for example:

```
<h:commandButton action="#{identity.login}" value="Log in"/>
```

This JSF command button would typically be used in a login form (which would also contain inputs for the user's username and password) that allows the user to log into the application.

Note

The bean type of the `Identity` bean is `org.jboss.seam.security.Identity`. This interface is what you should inject if you need to access the `Identity` bean from your own beans. The default implementation is `org.jboss.seam.security.IdentityImpl`.

The other important bean to know about right now is the `Credentials` bean. Its' purpose is to hold the user's credentials (such as their username and password) before the user logs in. The default implementation of the `Credentials` bean is also a session-scoped, named bean (just like the `Identity` bean).

The `Credentials` bean has two properties, `username` and `credential` that are used to hold the current user's username and credential (e.g. a password) values. The default implementation of the `Credentials` bean provides an additional convenience property called `password`, which may be used in lieu of the `credential` property when a simple password is required.

Note

The bean type of the `Credential` bean is `org.jboss.seam.security.Credentials`. The default implementation for this

bean type is `org.jboss.seam.security.CredentialsImpl`. Also, as credentials may come in many forms (such as passwords, biometric data such as that from a fingerprint reader, etc) the `credential` property of the `Credentials` bean must be able to support each variation, not just passwords. To allow for this, any credential that implements the `org.picketlink.idm.api.Credential` interface is a valid value for the `credential` property.

2.2. Built-in Authenticators

The Seam Security module provides the following built-in `Authenticator` implementations:

- `org.jboss.seam.security.jaas.JaasAuthenticator` - used to authenticate against a JAAS configuration defined by the container.
- `org.jboss.seam.security.management.IdmAuthenticator` - used to authenticate against an Identity Store using the Identity Management API. See the Identity Management chapter for details on how to configure this authenticator.
- `org.jboss.seam.security.external.openid.OpenIdAuthenticator` (provided by the external module) - used to authenticate against an external OpenID provider, such as Google, Yahoo, etc. See the External Authentication chapter for details on how to configure this authenticator.

2.3. Which Authenticator will Seam use?

The `Identity` bean has an `authenticatorClass` property, which if set will be used to determine which `Authenticator` bean implementation to invoke during the authentication process. This property may be set by configuring it with a predefined authenticator type, for example by using the Seam Config module. The following XML configuration example shows how you would configure the `Identity` bean to use the `com.acme.MyCustomerAuthenticator` bean for authentication:

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:ee"
  xmlns:security="urn:java:org.jboss.seam.security"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee    http://jboss.org/schema/cdi/
beans_1_0.xsd">

  <security:IdentityImpl>
    <s:modifies/>
                              <security:authenticatorClass>com.acme.MyCustomAuthenticator</
security:authenticatorClass>
  </security:IdentityImpl>
```

```
</beans>
```

Alternatively, if you wish to be able to select the `Authenticator` to authenticate with by specifying the name of the `Authenticator` implementation (i.e. for those annotated with the `@Named` annotation), the `authenticatorName` property may be set instead. This might be useful if you wish to offer your users the choice of how they would like to authenticate, whether it be through a local user database, an external OpenID provider, or some other method.

The following example shows how you might configure the `authenticatorName` property with the Seam Config module:

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:ee"
  xmlns:security="urn:java:org.jboss.seam.security"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee    http://jboss.org/schema/cdi/
beans_1_0.xsd">
  <security:IdentityImpl>
    <s:modifies/>
    <security:authenticatorName>openIdAuthenticator</security:authenticatorName>
  </security:IdentityImpl>
</beans>
```

If neither the `authenticatorClass` or `authenticatorName` properties are set, then the authentication process with automatically use a custom `Authenticator` implementation, if the developer has provided one (and only one) within their application.

If neither property is set, and the user has not provided a custom `Authenticator`, then the authentication process will fall back to the Identity Management API to attempt to authenticate the user.

2.4. Writing a custom Authenticator

All `Authenticator` implementations must implement the `org.jboss.seam.security.Authenticator` interface. This interface defines the following methods:

```
public interface Authenticator {
  void authenticate();
  void postAuthenticate();
  User getUser();
  AuthenticationStatus getStatus();
```

```
}
```

The `authenticate()` method is invoked during the authentication process and is responsible for performing the work necessary to validate whether the current user is who they claim to be.

The `postAuthenticate()` method is invoked after the authentication process has already completed, and may be used to perform any post-authentication business logic, such as setting session variables, logging, auditing, etc.

The `getUser()` method should return an instance of `org.picketlink.idm.api.User`, which is generally determined during the authentication process.

The `getStatus()` method must return the current status of authentication, represented by the `AuthenticationStatus` enum. Possible values are `SUCCESS`, `FAILURE` and `DEFERRED`. The `DEFERRED` value should be used for special circumstances, such as asynchronous authentication as a result of authenticating against a third party as is the case with OpenID, etc.

The easiest way to get started writing your own custom authenticator is to extend the `org.jboss.seam.security.BaseAuthenticator` abstract class. This class implements the `getUser()` and `getStatus()` methods for you, and provides `setUser()` and `setStatus()` methods for setting both the user and status values.

To access the user's credentials from within the `authenticate()` method, you can inject the `Credentials` bean like so:

```
@Inject Credentials credentials;
```

Once the credentials are injected, the `authenticate()` method is responsible for checking that the provided credentials are valid. Here is a complete example:

```
public class SimpleAuthenticator extends BaseAuthenticator implements Authenticator {
  @Inject Credentials credentials;

  @Override
  public void authenticate() {
    if ("demo".equals(credentials.getUsername()) &&
        credentials.getCredential() instanceof PasswordCredential &&
        "demo".equals(((PasswordCredential) credentials.getCredential()).getValue()))  {
      setStatus(AuthenticationStatus.SUCCESS);
      setUser(new SimpleUser("demo"));
    }
  }
}
```

Note

The above code was taken from the simple authentication example, included in the Seam Security distribution.

In the above code, the `authenticate()` method checks that the user has provided a username of *demo* and a password of *demo*. If so, the authentication is deemed as successful and the status is set to `AuthenticationStatus.SUCCESS`, and a new `SimpleUser` instance is created to represent the authenticated user.

Warning

The `Authenticator` implementation *must* return a non-null value when `getUser()` is invoked if authentication is successful. Failure to return a non-null value will result in an `AuthenticationException` being thrown.

Security - Identity Management

3.1. TO DO

This chapter coming soon.

Security - External Authentication

4.1. TO DO

This chapter coming soon.

Security - Authorization

5.1. TO DO

This chapter coming soon.