Octopus Framework

Rudy De Busscher

Version 0.3, ??/??/2018

Table of Contents

Release notes

0.3

1. Authorization parts implementation (tags like securedComponent, interceptors for EJB/CDI, filters for URLs)
2. AuthorizationToken to be able to extract AuthorizationInfo from token (MicroProfile, Offline, …)
3. Basic support for authorization annotation in Java SE (with *MethodAuthorizationChecker.checkAuthorization();*)

0.2

1. Split into different modules (Core, JSON, Non-Web [Java SE, …], Web [JSF, JAX-RS])
2. Octopus-jwt-support for handling JSON supporting plain, JWS and JWE.
3. Octopus-json is optimized smart-json code
4. MicroProfile JWT Auth for Rest (POC)
5. OfflineToken for standalone Java SE (POC)

0.1

1. POC integration Apache Shiro into Octopus

Modules

List of Maven modules

Artefact	SE, CDI, EE	info
be.atbash.json:octopus-json-smart	SE	Very light-weight JSON reader/writer (based on json-smart). Can be externalized if needed.
be.atbash.ee.security:octopus-jwt-support	SE, CDI	Java Beans to/from JSON/JWS/JWE. Can be externalized if needed. octopus-keys can be extracted from it.
be.atbash.ee.security:octopus-core	SE, CDI	All Octopus classes usable in Java SE and Java EE environment.
be.atbash.ee.security:octopus-common-se	SE, CDI	All Octopus classes Specific for Java SE
be.atbash.ee.security:octopus-se-standalone	SE, CDI	Specific for Java SE CLI programs
be.atbash.ee.security:octopus-token-generator	SE, CDI	Contains class to generate the Offline Token (for SE usage).
be.atbash.ee.security:octopus-common-web	EE (Web)	All Octopus classes Specific for Java EE (Web - Servlets)

Artefact	SE, CDI, EE	info
be.atbash.ee.security:octopus-rest	EE (JAX-RS)	Specific for JAX-RS
be.atbash.ee.security:octopus-jsf7	EE (JSF)	Specific for JSF
be.atbash.ee.security:octopus-mp	EE (JAX-RS)	Support for MP JWT Auth tokens

octopus-utilities contains for the moment the JavaFX app to maintain JWK files.

Authorization

Interceptors

```
@RequiresPermissions
```

Can be used to protect the execution of an EJB method. User (subject) must have the permission before method is executed.

```
String[] value()
```

Supply the permission(s) wildcard or named permission. See Permission chapter.

```
Class<? extends NamedPermission>[] permission()
```

Supply the permission to check as class instance.

```
Combined combined() default Combined.OR
```

When multiple permissions are supplied, must they all be satisfied or only one (the default)

```
@RequiresRoles
```

Can be used to protect the execution of an EJB method. User (subject) must have the role before method is executed.

```
String[] value()
```

Supply the role name(s). See Permission chapter.

Voters

Voters for a certain permission or role can be created.

```
@Inject
@RequiresPermissions("order:read:*")
private GenericPermissionVoter orderReadVoter;
```

or for a role

```
@Inject
@RequiresRoles("admin")
private GenericRoleVoter adminRoleVoter;
```

Things you can do with a voter

```
voter.checkPermission(AccessDecisionVoterContext, Set<SecurityViolation>);
```

Verifies the permission and add violations to the Set<SecurityViolation> parameter. AccessDecisionVoterContext supplies context

```
voter.verifyPermission();
```

returns true if the current user /subject has the required permission checked by the voter.

These voters can be created programmatically in those environments where no CDI inject is available.

```
GenericPermissionVoter.createInstance(String);
```

or

```
GenericRoleVoter.createInstance(ApplicationRole)
```

Custom voters

When the default checks on permissions or not enough. It can be that more complex logic is required or that multiple checks must be combined.

```
@ApplicationScoped
@Named
public class CustomVoter extends AbstractGenericVoter {
}
```

Typically the injection of voters is performed within these custom voters.

Custom tags

Custom tags are created to perform declarative authorization on JSF components. These are defined in the namespace

```
xmlns:sec="http://www.atbash.be/secure/octopus"
```

<sec:securedComponent>

Defines if a certain JSF component can be viewed (is rendered) for the user/subject.

When defined within another JSF tag (without the for attribute) it controles the parent. With the for attribute one can define the JSF component on which it operates.

```
permission
```

Supply the permission(s) wildcard or named permission. See Permission chapter.

```
role
```

Supply the role name(s). See Permission chapter.

```
voter
```

Supply the names of the custom voter(s)

Combination of the 3 above attributes is allowed

```
not
```

Inverts the result of the check

```
combined (true/false)
```

Do all checks need to be pass on the user/subject or is only 1 enough.

```
for
```

Specifies the id of one (or more) JSF components for which the authorization check is performed.

<sec:securedListener>

Defines the possibility to execute a method when the authorization checks ofr te user are positive based on the supplied permission, role and/or voter. The Java method can update the component to allow correct styling based on the permissions of the users.

When defined within another JSF tag (without the for attribute) it controles the parent. With the for attribute one can define the JSF component on which it operates.

```
listener
```

Defines the EL expression of the method which needs to be executed. The EL expression must point to a Java method with a parameter of type UIComponent and has no return (void)

```
permission
```

Supply the permission(s) wildcard or named permission. See Permission chapter.

```
role
```

Supply the role name(s). See Permission chapter.

```
voter
```

Supply the names of the custom voter(s)

Combination of the 3 above attributes is allowed

```
not
```

Inverts the result of the check

```
combined (true/false)
```

Do all checks need to be pass on the user/subject or is only 1 enough.

```
for
```

Specifies the id of one (or more) JSF components for which the authorization check is performed.

<sec:securePage>

This is an alternative for the usage of the filter definition with the securedURLs.ini file. We can specify the authorization checks (using permission, role and voter) in order that the page is visible for the end user. If (s)he has no permission, the unauthorized page will be shown.

This tag can be placed anywhere on the page, but for optimal performance, it should be in the beginning of the page and within the <h:body> parent.

```
permission
```

Supply the permission(s) wildcard or named permission. See Permission chapter.

```
role
```

Supply the role name(s). See Permission chapter.

```
voter
```

Supply the names of the custom voter(s)

Combination of the 3 above attributes is allowed

```
not
```

Inverts the result of the check

```
combined (true/false)
```

Do all checks need to be pass on the user/subject or is only 1 enough.

Java SE

Methods can be annotated with authorization checks, like @RequiresPermission, and authorization checks are performed by calling the method

```
    @RequiresPermissions("demo:offline:*")
    public String checkPermission() {
        MethodAuthorizationChecker.checkAuthorization();
```

Since we are running in plain Java SE, we have no interceptors available to perform these checks

automatically.

Java EE

JAX-RS

Core

FIXME

MP Auth token

Creation of the token can be done be using the **be.atbash.ee.security.octopus.token.MPJWTTokenBuilder** class.

Maven artefact be.atbash.ee.security:octopus-mp contains the *mpUser* filter.

Java SE

OfflineToken

Offline token can be used for standalone Java SE programs.

A token can be generated which will be only valid for a certain computer.

Besides the Processor Id and the first disk UUID, also a pass phrase is required (when multiple users are using the program on the same laptop/desktop.)

Steps (example flow, final programs not created yet)

1. Program **LocalSecret** (*examples/local-secret*) generates the token which is user dependent for a certain machine(Standalone program run by the end-user)
2. Program **CreateOfflineTokenFile** (*examples/se-cli*) generates the offline token (here stored within the <*user_home*>/*octopus.offline.token* file)
3. Program **SecuredCLI** uses the offline token to authenticate/authorize using Octopus.

Configuration

Core

hashAlgorithmName

default : **(none)**

Name of the MessageDigest algorithm when you use hashed passwords. examples are Md5 and

Sha512.

saltLength

default : **0**

Number of bytes used when creating a salt for the hashing of passwords. 0 means that no salt is used.

hashEncoding

default : **HEX**

Defines how the hashed passwords are encoded (HEX or BASE64) before they are compared to the supplied value which should be identically before access is granted. The value specified in the configuration file is case insensitive compared with the allowed values.

cacheManager.class

default : **be.atbash.ee.security.octopus.cache.MemoryConstrainedCacheManager**

The class responsible for holding/managing the cache of the authentication and authorization data. The developer can supply a custom implementation of be.atbash.ee.security.octopus.cache.AbstractCacheManager when the cache needs different logic.

When the class has the javax.enterprise.context.ApplicationScoped annotation, it is instantiated as a CDI bean, otherwise a classic new is performed.

voter.suffix.permission

default : **PermissionVoter**

The suffix used to determine the CDI named bean which are created dynamically for each Named Permission. See VoterNameFactory.

voter.suffix.role

default : **RoleVoter**

The suffix used to determine the CDI named bean which are created dynamically for each Named Role. See VoterNameFactory.

voter.suffix.check

default : **AccessDecisionVoter**

The suffix used to determine the CDI named bean for the Custom check functionality. See VoterNameFactory and Custom check feature description.

authorization.dynamic

default : false

???

namedPermission.class

default : **(none)**

Defines the Enum class which enumerates all permissions. Within the demo example it is the class **be.c4j.demo.security.permission.DemoPermission**.

namedPermissionCheck.class

default : **(none)**

Defines the annotation which can be used on method and class level to define the security requirements.

customCheck.class

default : **(none)**

Defines the annotation class which can be used to use custom declared Permissions, mostly useful in the case where you want to extend the named permission with some additional information.

namedRole.class

default : **(none)**

Defines the Enum class which enumerates all named roles. It is the role counterpart of the namedPermission.class configuration option.

namedRoleCheck.class

default : **(none)**

Defines the annotations which can be used on method and class level to define the security requirements.