Java Enterprise Edition Security Explained

MUNI, Brno 2015

JBoss by Red Hat

Peter Škopek, pskopek@redhat.com, twitter: @pskopek

Apr 27, 2015

Abstract

This lecture will guide you through various aspects of security in Java Enterprise Edition Applications. It will start with plain JAAS and continue with JEE security concepts and explanation of their usage in your application. Next comes JAAS and its usage in WildFly 8. Then we will finish with login modules in WildFly.

redhat

Agenda

1. **Java Authentication and Authorization Service**
   - Overview
   - JAAS - putting it all together
2. **Java EE Security**
   - Overview
   - Simple Example
   - Java EE Security Architecture
   - Servlet Container Security
   - EJB Container Security
   - EE and Java Security Manager
3. **JBoss Specific**
   - Annotations
   - LoginModules

Section 1

Java Authentication and Authorization Service

What is that?

The Java Authentication and Authorization Service (JAAS) was introduced as an optional package (extension) to the Java 2 SDK, Standard Edition (J2SDK), v 1.3. JAAS was integrated into the J2SDK 1.4.

JAAS can be used for two purposes:

- for authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet
- for authorization of users to ensure they have the access control rights (permissions) required to do the actions performed

JAAS as PAM

- JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework as we know it from UNIX environment.
  This permits applications to remain independent from underlying authentication technologies. New or updated authentication technologies can be plugged under an application without requiring modifications to the application itself.
- Applications enable the authentication process by instantiating a LoginContext object, which in turn references a Configuration to determine the authentication technology(ies), or LoginModule(s), to be used in performing the authentication.
  Typical LoginModules may prompt for and verify a username and password. Others may read and verify a voice or fingerprint sample.

Common Classes

Common classes are those shared by both the JAAS authentication and authorization components.

- javax.security.auth.Subject
  - the key JAAS class
  - the term subject to represent the source of a request.
    A subject may be any entity, such as a person or a service.
  - Once the subject is authenticated, a Subject object is populated with associated identities (Principals).
  - A Subject may also own security-related attributes, which are referred to as credentials (public and private).

Common Interfaces

- Principals
  - Must implement java.security.Principal interface.
- Credentials
  - Public and private credential classes are not part of the core JAAS class library.
  - Any class can represent a credential.
  - Developers, however, may elect to have their credential classes implement two interfaces related to credentials: javax.security.auth.Refreshable and javax.security.auth.Destroyable.

Authentication Classes and Interfaces

- javax.security.auth.login.LoginContext
  - The class provides the basic methods used to authenticate subjects, and provides a way to develop an application independent of the underlying authentication technology.
  - The LoginContext consults a Configuration to determine the authentication services, or LoginModule(s), configured for a particular application. Therefore, different LoginModules can be plugged in under an application without requiring any modifications to the application itself.
- javax.security.auth.spi.LoginModule
  - The LoginModule interface gives developers the ability to implement different kinds of authentication technologies that can be plugged in under an application.
  - For example, one type of LoginModule may perform a username/password-based form of authentication. Other LoginModules may interface to hardware devices such as smart cards or biometric devices.

Authentication Classes and Interfaces

- javax.security.auth.callback.CallbackHandler
  - In some cases a LoginModule must communicate with the user to obtain authentication information. LoginModules use a CallbackHandler for this purpose.
  - Applications implement the CallbackHandler interface and pass it to the LoginContext, which forwards it directly to the underlying LoginModules.
- javax.security.auth.callback.Callback
  - The javax.security.auth.callback package contains the Callback interface as well as several implementations.
  - LoginModules may pass an array of Callbacks directly to the handle method of a CallbackHandler (in their login() method).

Authorization Classes

- java.security.Policy
- javax.security.auth.AuthPermission
- javax.security.auth.PrivateCredentialPermission

Basic work flow

- LoginContext instance takes two arguments:
    - configuration name - simple string denoting name of configuration used to initialize login context
    - callback handler object
- Call to LoginContext method login() which performs actual authentication / authorization according to configuration passed to the LoginContext and LoginModule(s) in our config file.
- After successful login, LoginContext is populated with Subject and it contains Principal(s) and Credential(s).
- LoginContext logout() method for obvious reasons.

LoginContext configuration

A login configuration contains the following information. Note that this example only represents the default syntax for the configuration file. There is possibility to create a subclass of the implementations class with alternative syntax and may retrieve the configuration from any source such as files, databases, or servers.

```
Name1 {
ModuleClass Flag ModuleOptions;
ModuleClass Flag ModuleOptions;
ModuleClass Flag ModuleOptions;
};
Name2 {
ModuleClass Flag ModuleOptions;
ModuleClass Flag ModuleOptions;
};
other {
ModuleClass Flag ModuleOptions;
ModuleClass Flag ModuleOptions;
};
```

LoginModule stack configuration

The Flag value controls the overall behavior as authentication proceeds down the stack.

1. Required - The LoginModule is required to succeed.
   If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
2. Requisite - The LoginModule is required to succeed.
   If it succeeds, authentication continues down the LoginModule list.
   If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).

LoginModule stack configuration

The Flag value controls the overall behavior as authentication proceeds down the stack.

3. Sufficient - The LoginModule is not required to succeed.
   If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list).
   If it fails, authentication continues down the LoginModule list.
4. Optional - The LoginModule is not required to succeed.
   If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

Section 2

Java EE Security

redhat

Security Definition

- What are the aspects of secure applications?
  - Authentication - The means by which communicating entities (for example, client and server) prove to one another that they are acting on behalf of specific identities that are authorized for access.
  - Access control for resources - The means by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints.
  - Data integrity - The means used to prove that information has not been modified by a third party (some entity other than the source of the information). For example, a recipient of data sent over an open network must be able to detect and discard messages that were modified after they were sent.

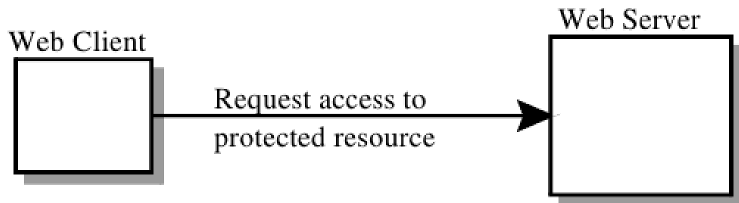
redhat

Security Definition

- Another security aspects of good application.
  - Confidentiality - The means used to ensure that information is made available only to users who are authorized to access it.
  - Non-repudiation - The means used to prove that a user performed some action such that the user cannot reasonably deny having done so.
  - Auditing - The means used to capture a tamper-resistant record of security related events for the purpose of being able to evaluate the effectiveness of security policies and mechanisms.

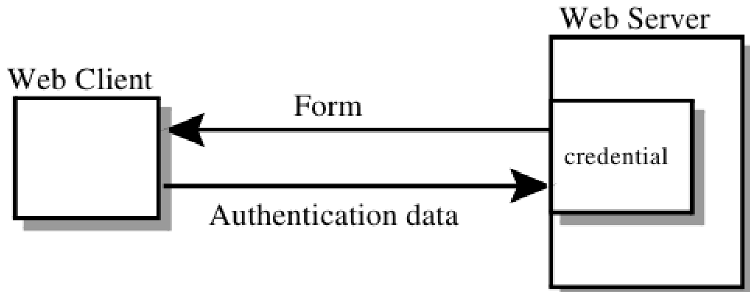
redhat

Step 1: Initial Request



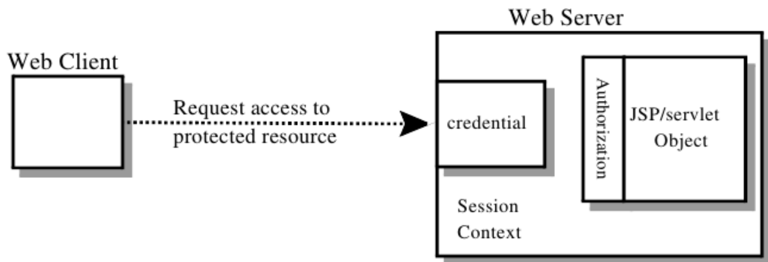
The web client requests the main application URL.

Step 2: Initial Authentication



The web server returns a form that the web client uses to collect authentication data (for example, username and password) from the user. The web client forwards the authentication data to the web server, where it is validated by the web server.

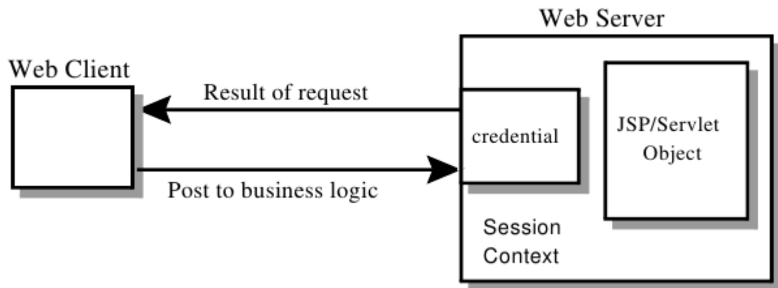
Step 3: URL Authorization



The web container then tests the user's credential against each role to determine if it can map the user to the role.

redhat

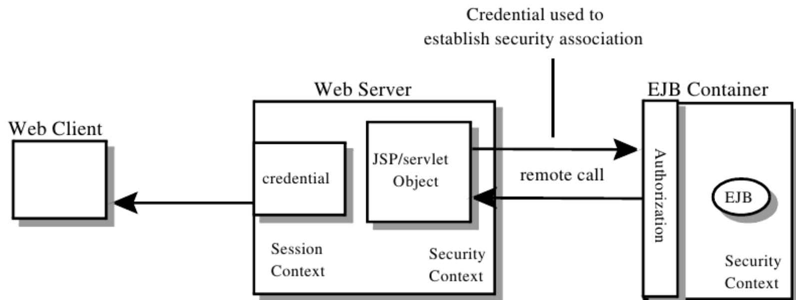
Step 4: Fulfilling the Original Request



If the user is authorized, the web server returns the result of the original URL request.

redhat

Step 5: Invoking Enterprise Bean Business Methods



The servlet performs the remote method call to the enterprise bean, using the user's credential to establish a secure association between the servlet and the enterprise bean. The association is implemented as two related security contexts, one in the web server and one in the EJB container.

redhat

Selected Goals of Java EE Security Architecture

- Transparency: Application Component Providers should not have to know anything about security to write an application.
- Isolation: Divorcing the application from responsibility for security ensures greater portability of Java EE applications.
- Flexibility: The security mechanisms and declarations used by applications under this specification should not impose a particular security policy, but facilitate the implementation of security policies specific to the particular Java EE installation or application.
- Abstraction: An application component's security requirements will be logically specified using deployment descriptors.

redhat

Selected Goals of Java EE Security Architecture

- Independence: Required security behaviors and deployment contracts should be implementable using a variety of popular security technologies.
- Secure interoperability: Application components executing in a Java EE product must be able to invoke services provided in a Java EE product from a different vendor.

redhat

Terminology

- Principal - is an entity that can be authenticated by an authentication protocol in a security service that is deployed in an enterprise.
- Security Policy Domain - is a scope over which a common security policy is defined and enforced by the security administrator of the security service (also known as security domain).
- Security Attributes - a set of security attributes is associated with every principal.
- Credential - contains or references information (security attributes) used to authenticate a principal for Java EE product services.

Container Based Security

Security for components is provided by their containers in order to achieve the goals for security specified above in a Java EE environment. A container provides two kinds of security:

- Declarative security
  - Declarative security refers to the means of expressing an application's security model or requirements, including roles, access control, and authentication requirements in a form external to the application. The deployment descriptor is the primary vehicle for declarative security in web applications.
- Programmatic security
  - Programmatic security is used by security aware applications when declarative security alone is not sufficient to express the security model of the application.

redhat.

Programmatic Security

Programmatic security consists of the following methods of the HttpServletRequest interface:

- authenticate
- login
- logout
- getRemoteUser
- isUserInRole
- getUserPrincipal

redhat

Declarative Security

Following annotations are part of Servlet 3.0 specification and provide alternative to defining access control via declarative deployment descriptor.

- @ServletSecurity
- @HttpConstraint - The annotation is used within the @ServletSecurity annotation to represent the security constraint to be applied to all HTTP protocol methods for which a corresponding @HttpMethodConstraint does NOT occur within the @ServletSecurity annotation.
- @HttpMethodConstraint - The @HttpMethodConstraint annotation is used within the @ServletSecurity annotation to represent security constraints on specific HTTP protocol messages.

redhat

Servlet Security Annotation Reference

Detailed descriptions could be found at:

- http://jcp.org/en/jsr/detail?id=315
- http://docs.oracle.com/javaee/7/api/javax/servlet/annotation/package-summary.html

redhat

Example is worth thousand words

For all HTTP methods, no constraints

```
@ServletSecurity
public class Example1 extends HttpServlet {
  ...
}
```

For all HTTP methods, no auth-constraint, confidential transport required

```
@ServletSecurity(@HttpConstraint(transportGuarantee =
         TransportGuarantee.CONFIDENTIAL))
public class Example2 extends HttpServlet {
  ...
}
```

redhat

Example is worth thousand words

For all HTTP methods, all access denied

```
@ServletSecurity(@HttpConstraint(EmptyRoleSemantic.DENY))
public class Example3 extends HttpServlet {
}
```

For all HTTP methods, auth-constraint requiring membership in Role R1

```
@ServletSecurity(@HttpConstraint(rolesAllowed = "R1"))
public class Example4 extends HttpServlet {
}
```

redhat

Example is worth thousand words

For All HTTP methods except GET and POST, no constraints; for methods GET and POST, auth-constraint requiring membership in Role R1; for POST, confidential transport required

```
@ServletSecurity((httpMethodConstraints = {
 @HttpMethodConstraint(value = "GET", rolesAllowed = "R1"),
 @HttpMethodConstraint(value = "POST", rolesAllowed = "R1",
    transportGuarantee = TransportGuarantee.CONFIDENTIAL)
})
public class Example5 extends HttpServlet {
}
```

For all HTTP methods except GET auth-constraint requiring membership in Role R1; for GET, no constraints

```
@ServletSecurity(
   value = @HttpConstraint(rolesAllowed = "R1"),
   httpMethodConstraints = @HttpMethodConstraint("GET"))
public class Example6 extends HttpServlet {
}
```

redhat.

Example is worth thousand words

For all HTTP methods except TRACE, auth-constraint requiring membership in Role R1; for TRACE, all access denied

```
@ServletSecurity(
 value = @HttpConstraint(rolesAllowed = "R1"),
 httpMethodConstraints = @HttpMethodConstraint(
      value="TRACE",
      emptyRoleSemantic = EmptyRoleSemantic.DENY))
public class Example7 extends HttpServlet {
}
```

redhat

Mapping @ServletSecurity to security-constraint

Mapping @ServletSecurity with no contained @HttpMethodConstraint

```
@ServletSecurity(@HttpConstraint(rolesAllowed = "R1"))

<security-constraint>
   <web-resource-collection>
      <url-pattern>...</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <security-role-name>R1</security-role-name>
   </auth-constraint>
</security-constraint>
```

redhat

Mapping @ServletSecurity to security-constraint

Mapping @ServletSecurity with contained @HttpMethodConstraint

```
@ServletSecurity(value=@HttpConstraint(rolesAllowed = "Role1"),
   httpMethodConstraints = @HttpMethodConstraint(value = "TRACE",
   emptyRoleSemantic = EmptyRoleSemantic.DENY))

<security-constraint>
   <web-resource-collection>
      <url-pattern>...</url-pattern>
      <http-method-omission>TRACE</http-method-omission>
   </web-resource-collection>
   <auth-constraint>
      <security-role-name>Role1</security-role-name>
   </auth-constraint>
</security-constraint>
<security-constraint>
   <web-resource-collection>
      <url-pattern>...</url-pattern>
      <http-method>TRACE</http-method>
   </web-resource-collection>
   <auth-constraint/>
</security-constraint>
```

redhat

Roles

A servlet container enforces declarative or programmatic security for the principal associated with an incoming request based on the security attributes of the principal.

- A deployer has mapped a security role to a user group in the operational environment.
- A deployer has mapped a security role to a principal name in a security policy domain.

redhat

Basic idea of EJB security

- Business methods of Enterprise Java Beans contain no security-related logic.
- Security policies for the application can be configured in a way that is most appropriate for the operational environment of the enterprise.
- A security role is a semantic grouping of permissions that a given type of users of the application must have in order to successfully use the application.

redhat

Security Permission Specification

The Bean Provider can use metadata annotations or the deployment descriptor to specify whether the caller's security identity or a run-as security identity should be used for the execution of the bean's methods.

- By default, the caller principal will be propagated as the caller identity. The Bean Provider can use the RunAs annotation to specify that a security principal that has been assigned to a specified security role be used instead.
- If the deployment descriptor is used to specify the security principal, the Bean Provider or the Application Assembler can use the security-identity deployment descriptor element to specify or override the security identity.

redhat

Programmatic Access to Caller's Security Context

The javax.ejb.EJBContext interface provides two methods (plus two deprecated methods that were defined in EJB 1.0) that allow the Bean Provider to access security information about the enterprise bean's caller.

```
public class MyBusinessBean {

   @Resource
   EJBContext ctx;

   ...
}
```

redhat.

Security Related Annotations

Annotation	Corresponding DD Element
@DeclareRoles	security-role
@RolesAllowed	method-permission
@PermitAll	unchecked
@DenyAll	exclude-list
@RunAs	security-identity run-as

Security Related Annotations

Annotation	Class	Method
@DeclareRoles	Yes	No
@RolesAllowed	Yes	Yes
@PermitAll	Yes	Yes
@DenyAll	Yes	Yes
@RunAs	Yes	Yes

redhat

EE 7: Java Security Manager

Java EE application components are able to run with Java Security Manager.
Permission declarations must be stored in META-INF/permissions.xml file within an EJB, web, application client, or resource adapter archive in order for them to be located and subsequently processed by the deployment machinery of the Java EE Application server.

redhat

Permissions Allowed in Web, EJB, and Resource Adapter Components

- java.lang.RuntimePermission loadLibrary.*
- java.lang.RuntimePermission queuePrintJob
- java.net.SocketPermission * connect
- java.io.FilePermission * read,write
- java.io.FilePermission file:${javax.servlet.context.temdir} read, write
- java.util.PropertyPermission

Section 3

JBoss Specific

redhat

PicketBox Authorization Annotations

- We can reduce our boiler plate code using PicketBox Annotations on POJOs.
  - @SecurityDomain Annotation
  - @Authentication Annotation
  - @Authorization Annotation
  - @SecurityMapping Annotation
  - @SecurityAudit Annotation
  - @Module Annotation
  - @ModuleOption Annotation
  - @SecurityConfig Annotation
- More at `http://community.jboss.org/wiki/PicketBoxSecurityAnnotations`

redhat.

Annotated POJO example

```
import org.jboss.security.annotation.Authentication;
import org.jboss.security.annotation.Authorization;
import org.jboss.security.annotation.Module;
import org.jboss.security.annotation.ModuleOption;

import org.jboss.security.auth.spi.UsersRolesLoginModule;
import org.picketbox.plugins.authorization.PicketBoxAuthorizationModule;

@Authentication(modules={@Module(code = UsersRolesLoginModule.class, options =
  {@ModuleOption})})
@Authorization(modules ={@Module(code = PicketBoxAuthorizationModule.class, options =
  {@ModuleOption(key="roles",value="validuser")})})
public class AuthAuthorizationAnnotatedPOJO {

  ....

}
```

redhat

PicketBox LoginModules

- It is based on JAAS which is available as part of the JDK.
- PicketBox provides simple various authentication and authorization modules
  - Advanced LDAP based Authentication using LdapExtLoginModule
  - LDAP based Authentication using LdapLoginModule
  - Database based Authentication using DatabaseServerLoginModule
  - File based Authentication using UsersRolesLoginModule
- More about PicketBox Authentication you can find at `http://community.jboss.org/wiki/PicketBoxAuthentication`

redhat

Pasword Stacking Option

- To use password stacking, each login module should set the <module-option>password-stacking attribute to useFirstPass. If a previous module configured for password stacking has authenticated the user, all the other stacking modules will consider the user authenticated and only attempt to provide a set of roles for the authorization step.
- When password-stacking option is set to useFirstPass, this module first looks for a shared user name and password under the property names javax.security.auth.login.name and javax.security.auth.login.password respectively in the login module shared state map.

redhat

Password Stacking Simple Example

```
<application-policy name="todo">
   <authentication>
      <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
      <!-- LDAP configuration -->
         <module-option name="password-stacking">useFirstPass</module-option>
      </login-module>
      <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule" flag="required">
      <!-- database configuration -->
         <module-option name="password-stacking">useFirstPass</module-option>
      </login-module>
   </authentication>
</application-policy>
```

The end.

Thanks for listening.