2.0.0

Copyright © 2005-2007 Charles Gay, Inácio Ferrarini

Chapter 1. introduction to jGuard

Warning

this documentation refers to the 2.0.x product line.For earlier releases, please consult the documentation (as a maven™ web site) shipped with the downloaded bundle.

1.1. Overview

jGuard provides EASY security (authentication and authorization) in web and standalone applications. It is built over the stable and mature JAAS framework, which is part of the JAVA J2SE apis.

1.2. License

jGuard is licensed under the GNU Lesser General Public Licence (formerly known as LGPL). you can obtain a copy of the LGPL licence on the gnu project web site [http://www.gnu.org/copyleft/lesser.html].

1.3. Features

- Only requires java 5.0 and j2ee 1.3 or higher
- Can be adapted to any webapp, on any application server
- Permits a user to have more than one role simultaneously
- Does not depend on a web framework, or an AOP framework
- Built on top of the standard, very secure, and flexible JAAS
- Authentication and authorization are handled by pluggable mechanisms
- Authentication data stored in a database, an XML file, a JNDI [http://java.sun.com/products/jndi/] datasource, an LDAP directory,Kerberos [http://java.sun.com/products/jndi/]...
- Changes take effects 'on the fly' (dynamic configuration)
- Permissions, roles, and their associations can be created, updated, deleted on the fly through a webapp (an API is provided too)
- Each webapp has its own authentication and authorization configuration
- A taglib is provided to protect jsp fragments
- Support security manager

jGuard is composed of 9 libraries :

- jGuard-core: it contains the main jGuard features.
- jGuard-ext : it handles specific authentication and authorization managers such as XML or JDBC based managers. It also embeds login modules like JDBC or jCaptcha.
- jGuard-ext-java-6: a java 6 library

- jGuard-jee : library is dedicated for web applications.
- jGuard-jee-4: a jee 4 library
- jGuard-jee-extras: library dedicated to add security features to some specific web thrid part libraries like DWR (ajax library).
- jGuard-struts-example: webapp example to ilustrate jGuard use if you've chosen Struts as your web framework
- jGuard-jsf-example: webapp example to ilustrate jGuard use if you've chosen JSF MyFaces as your web framework

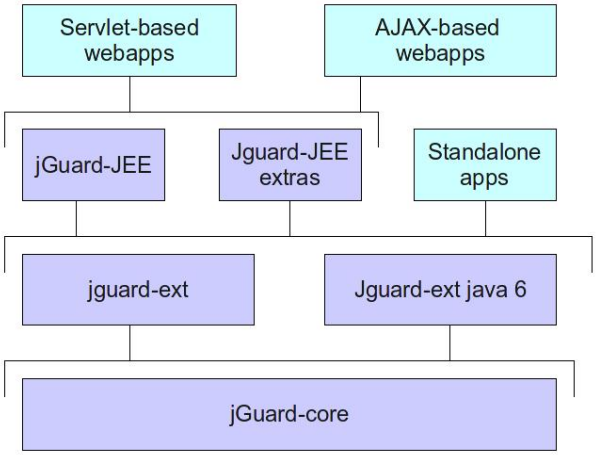


Figure 1.1. jguard libraries dependencies and purposes

Chapter 2. controlling jGuard log output

2.1. logging in jGuard

jGuard relies on the SLF4J™ [http://www.slf4j.org/] library (Simple Logging Facade for Java). it acts as an indirection between jGuard and the logging library you use (i.e log4J™ [http://logging.apache.org/log4j/] , java.util.logging, logback™ [http://logback.qos.ch/] , x4juli™ [http://www.x4juli.org/]).

SLF4J requires for jGuard to use the `slf4j-api.jar` archive, *and* for the end-user of jGuard, to import in its application one more jar depending the logging library he uses (`slf4j-logj12.jar` for log4j, `slf4j-jdk14.jar` for java.util.logging, `logback-classic.jar` for logback).
it is required to import in your project the `slf4j-api.jar` archive AND a bridge library related to the logging framework you use.

Each logging library has its own advantages over others. for example, LOG4J is the older one and can be evaluated as a very *stable* library. java.util.logging is the *standard* implementation (shipped in J2SE 1.4 or higher).x4juli permits to *remove some limitations* of java.util.logging on the JEE field , and can be used in conjunction with java.util.logging.LogBack add some new features among others. it seems to be the *most innovative* library.

Note

one 'advanced' feature present in the java.util.logging' package which is not present in other libraries, is to restrict logging output based on LoggingPermission [http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/LoggingPermission.html]: if a user or a library want to output some informations with the SecurityManager enabled on the java platform, it needs to have granted the related `LoggingPermission` [http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/LoggingPermission.html] .

Note

x4juli permits to extends limitations of the logging facility shipped in java .4 and beyond, especially by controlling logging configuration per classloader. Like each webapp has its own classloader in a JEE application server (if you use the JEE official 'parent-last' class loading mechanism in your application server), you have the ability to customize logging feature per webapp.

2.2. debugging Java Security

Note

thi section is excerpted from chapter 1 of *java Security published* by O'Reilly editions [http://www.oreilly.com/catalog/javasec2/chapter/ch01.html]

the Java Security contains a logging mechanism which permits to trace easily all the underlying security events which occurs.This mechanism is especially very useful to debug Authentication and Authorization decisions made in jGuard through JAAS.This mechainsm can be enabled byt a System property called `java.security.debug` which can have the following values:

- all

  Turn on all the debugging options.

- access

  Trace all calls to the `checkPermission( )` method of the `AccessController`. This allows you to see which permissions your code is requesting, which calls are succeeding, and which ones are failing.This option has the following sub-options separated by a semi-colon (:). If no sub-option is specified, then all are in force:

  - stack

    Dump the stack every time a permission is checked.

  - failure

    Dump the stack only when a permission is denied.

  - domain

    Dump the protection domain in force when a protection is checked.

  - jar

    When processing a signed jar file, print the signatures in the file, their certificates, and the classes to which they apply.

  - policy

    Print information about policy files as they are parsed, including their location in the filesystem, the permissions they grant, and the certificates they use for signed code.

  - scl

    Print information about the permissions granted directly by a secure class loader (rather than granted through a policy file).

Example 2.1. parameter with access option activated with the failure sub-option.

```
-Djava.security.debug=access:failure
```

Example 2.2. parameter with scl and access

```
-Djava.security.debug=scl,access
```

Caution

this facility should only used for debug purpose , because it will generate so many traces and will slow your application server.

Chapter 3. security architecture

3.1. securing an application

securing an application should be done with an*Access Control Model*. widely used access control models are described in a dedicated chapter.

To apply an access control model in a java application, you have these choices:

- use java security infrastructure (through jAAS)
- use security implementation by the application server if you're in a jee context
- reinvent the wheel

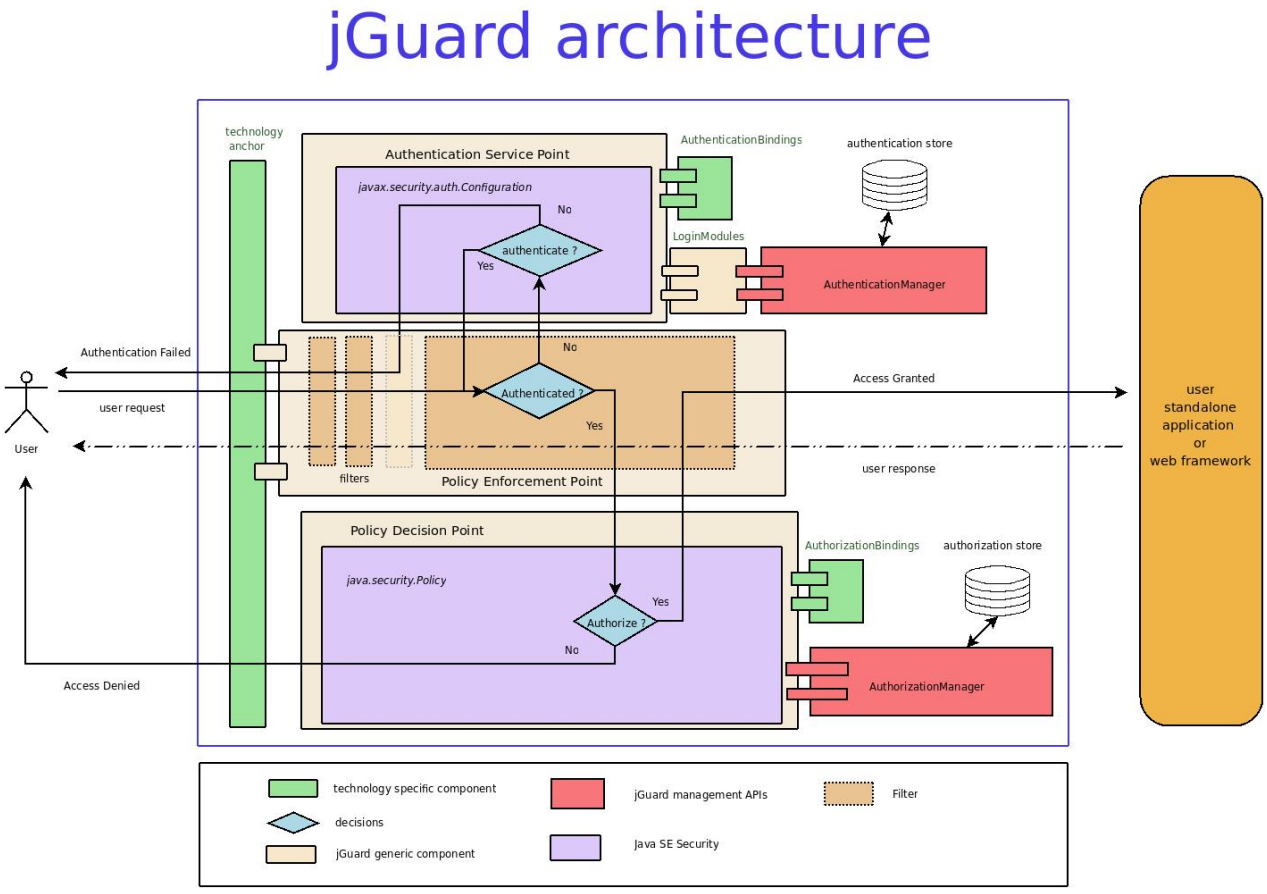
3.1.1. java security architecture

3.1.1.1. java security roots

overall Java security stands on the `java.lang.SecurityManager` implementation in place on theJVM, and on the java.security file located in in`${java.home}/lib/security/`.

3.1.1.2. overall architecture

3.1.1.2.1.



One application need to have an *Authentication* part and an *Authorization* part initialized. It implies for the authentication part, a *javax.security.auth.Configuration* instance defined, and for the Authorization

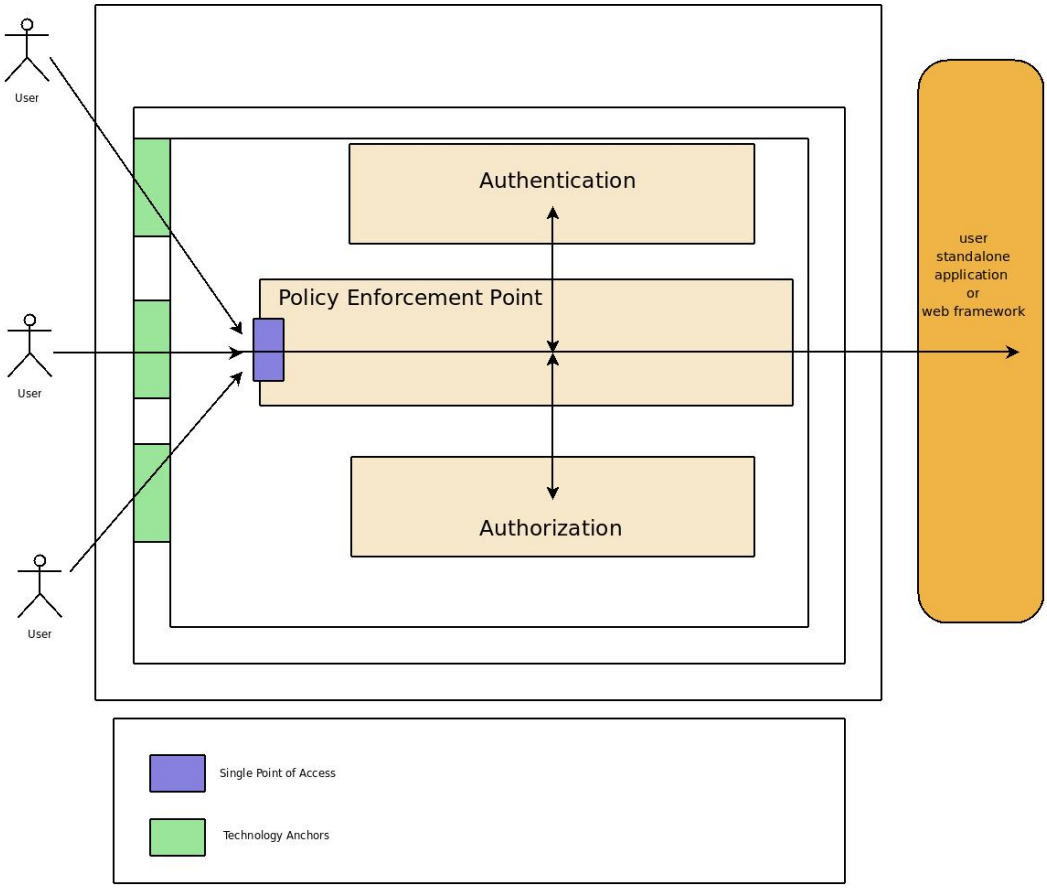
part a *java.security.Policy*instance (or an isolated part of an instance). jGuard provides a single point of access with its PolicyEnforcementPoint.

> **Note**
>
> in a JEE environement, Authentication and Authorization parts are set by the `ContextListener` class from jGuard.

Specific technology parts are minimized. So, integrating a new technology in jGuard implies implementing a *Technology anchor*, an *AuthorizationBindings* implementation, and eventually an*AuthenticationBindings*(not always needed if authentication is done via another technology anchor and scopes).

jGuard provides also some *management* APIs for the Authentication Part (*AuthenticationManager*), and for the Authorization part (*AuthorizationManager*).

3.1.1.2.2. one application to bring them all



jGuard permits for one application, to use different technology anchors simultaneously. It implies that they *share* the same Configuration and Policy (i.e Authentication and Authorization parts). You can see on the above diagram, that all technology anchors provided by jGuard, uses each one a PolicyEnforcementPoint instance.this class acts as a Single Point of Access.

> **Caution**
>
> be aware that to*secure access of your application*, you need to configure technology anchors to force all users to pass through them to access to protected resources. For example, in a webapp, you need to configure the `web.xml` file to enforce user to access to a technology anchor before reach the desired resource.

3.2. Which jGuard security scopes?

3.2.1. jGuard and jee users

jGuard envisions 3 types of "users" in a J2EE environment:

- administrator
- webapp developer
- webapp user

3.2.2. security scopes

jGuard provides two Security scopes, on authentication and authorization. these scopes affect jGuard `javax.security.auth.login.Configuration` and `java.security.policy` implementations.

3.2.2.1. local security scope

> **Note**
>
> this scope permits to have isolate security per classloader; i.e is mainly dedicated to jee applications.

'local' authentication provides a good security level. It allows protection of the webapp resources against webapp users. Each webapp user will be authenticated, and access control will be provided according to his roles. This authentication configuration will not protect webapp developers against webapp developers of others webapps, or administrators.

The authentication configuration is easier, because everything should be configured in the web.xml. There is no need to configure things on the JVM side. Security is present after the first webapp which uses jGuard is loaded by the application server. This security level is reliable for these use cases:

- The webapp is used to test jGuard
- There is only one webapp on the application server
- There are multiple webapps on the same application servers, and there are 'friendly' each others
- One 'friendly' webapp is loaded firstly

3.2.2.2. jvm security scope

'advanced' configuration allows for a more secure environment, but is more difficult to configure: in jee environment, You must install two jars: one for the webapp, and one dedicated to the JVM-side.some bootclasspath tricks are needed too.

This configuration allows for protection of webapp resources against users like the 'usual' configuration; i.e to protect webapp developers against others webapps, and to protect administrator against any webapp developers. The administrator of the machine should also restrict the java rights to protect against the application sever administrator. This configuration is highly secure, and should be used by hosting companies.

This is a`cascading security delegation model`:

- webapp users are controlled by webapps
- webapps are isolated from others webapps (others webapps cannot make tedamages)
- webapps are controlled by the application server administrator which configure the JVM security
- The application server administrator is controlled by the operating system administrator which assign restricted rights to java
- the operating system administrator security relies on BIOS security, which relies on the physical machine security.

  To have this very secured configuration, you must enable the`SecurityManager`.

3.3. debugging

for security reason, jGuard prevent by default, the application to throw to the end-user a `java.lang.Throwable` (i.e a `java.lang.Exception` or a `java.lang.Error`) instance, and its included stack trace: it permits to restrain sensitive information included in the stack trace, like the libraries you use, name of classes and methods and so on....

But, in the development stage, it can be useful to inhibit this default mechanism, for a quicker diagnostic when a problem is present. it can be done by including the parameters of the `technology anchor` (like the `AccessFilter` in servlet-related anchor, or the `AccessListener` for the JSF-based one),a `propagateThrowable` option to `true`.

3.4. configuration files

3.4.1. configuration files used in every context (standalone and web applications)

3.4.1.1. jGuardFilter.xml

goals of this configuration file is to define:

- resources where the user is dispatched depending on the Access Control check result
- authentication schemes used with the specific technology anchor
- specific parameters for the CallbackHandler implementation

3.4.1.1.1. authentication schemes

Authentication schemes are defined as the mechanisms used to transmit credentials from the user (browser for webapps) to the server. These credentials are used on the server to authenticate the user in its backend. They can be configured in the authScheme markup.

> **Caution**
>
> to configure your authentication schemes, you DON'T have to configure your application server to use them (especially, you DON'T have to configure the <login-config> markup in

the `web.xml`, and its related <auth-method> and <realm-name> markup). jGuard replace the specific mechanisms used in your application server, to grab credentials and compute them to authenticate the user.

- `FORM` authentication

  since its inception, jGuard support the `FORM` authentication scheme. Credentials are sent from the browser to the application server through an HTML form.

  some special URIs are involved in this authentication scheme:

  - logonURI

    this URI is used to access to the page which contains the form used to authenticate. This URI is granted to ALL users.

  - logonProcessURI

    this URI is used to send to the server the credentials to authenticate. This URI is granted to ALL users.

  - loginField

    HTML field in the HTML authentication form, used to store your login. This special field is trapped by the accessFilter to grab this credential.

  - passwordField

    HTML field in the HTML authentication form, used to store your password. This special field is trapped by the accessFilter to grab this credential.

Example 3.1. how to configure FORM authentication

```
<authScheme>FORM</authScheme>
```

- BASIC authentication

  jGuard support BASIC authentication. Some special URIs are involved in this authentication scheme:

  - logonProcessURI

    this URI is used to send to the server the credentials to authenticate. This URI is granted to ALL users.

Example 3.2. how to configure BASIC authentication

```
<authScheme>BASIC</authScheme>
```

- `Digest` Authentication

  this authentication scheme is not yet supported.a feature request has been posted on the jGuard bug tracking system hosted on sourceforge.

- `CLIENT_CERT` authentication

jGuard support `CLIENT_CERT` authentication.

Example 3.3. how to configure CLIENT_CERT authentication

```
                                        <authScheme>CLIENT_CERT</authScheme>
```

Note

jGuard use its own mechanisms involved in authentication schemes. But it uses the SSL mechanism provided by the application server, in the case of `CLIENT-CERT` authentication. So, you have to configure your `web.xml` file with this markup:

```
<security-constraint>
                                                <web-resource-collection>
                                                <web-resource-name>all the
 webapp</web-resource-name>
                                                <description></description>
                                                <url-pattern>/*</url-pattern>
                                                </web-resource-collection>
                                                <user-data-constraint>
                                                <description>This part requires
 SSL</description>
                                                <transport-
guarantee>CONFIDENTIAL</transport-guarantee>
                                                </user-data-constraint>
                                                </security-constraint>
```

3.4.1.1.2. example

Example 3.4. jGuardFilter.xml example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
                                <!DOCTYPE configuration SYSTEM "jGuardFilter_1.1.0.dtd">
                                <configuration>
                                <!-- Index uri of your web application. -->
                                <authenticationSucceedURI>/index.jsp</
authenticationSucceedURI>
                                <!-- Uri when the user authentication failed. -->
                                <authenticationFailedURI>/AuthenticationFailed.do</
authenticationFailedURI>
                                <!-- Uri to access to the authentication form -->
                                <logonURI>/Logon.do</logonURI>
                                <!-- uri to be authenticated. The action property of the
 authentication form MUST NOT
                                be set to j_security_check. -->
                                <logonProcessURI>/LogonProcess.do</logonProcessURI>
                                <registerURI>/Registration.do</registerURI>
                                <registerProcessURI>/RegistrationProcess.do</
registerProcessURI>
                                <!-- uri to to be unauthenticated -->
                                <logoffURI>/Logoff.do</logoffURI>
                                <authScheme>FORM</authScheme>
                                <loginField>login</loginField>
                                <!-- Parameter's name of the form's field which holds the
 password. All values are
                                accepted except j_password. -->
                                <passwordField>password</passwordField>
                                <goToLastAccessDeniedUriOnSuccess>true</
goToLastAccessDeniedUriOnSuccess>
                                </configuration>
```

Note

this configuration file specific to a web application is used to define the `URI` used when to follow jGuard Access control decisions. The location of this file is specified in the`web.xml`, especially in the `AccessFilter` declaration in a `configurationLocation` parameter.

Note

the AccessDenied URI is not defined in jGuardFilter.xml file, because it is already handled by the underlying protocol ;HTTP maps it to the status code 401. To use your customized accessDenied page, maps the error code in the web.xml file of your webapp to its path.

3.4.1.2. `jGuardAuthentication.xml`

goals of this configuration file is to define:

- the authentication scope
- the AuthenticationManager implementation
- the loginmodules involved in the authentication process with their options and JAAS keywords (required, optional, and so on...)

3.4.1.3. `jGuardAuthorization.xml`

goals of this configuration file is to define:

- authorization scope
- AuthorizationManager implementation

Chapter 4. java authentication

4.1. Overall Authentication part

Authentication part is composed of the Authentication process, which is involved when user is not authenticated, and AuthenticationManager, which manage users and roles.they both share the same Authentication store.

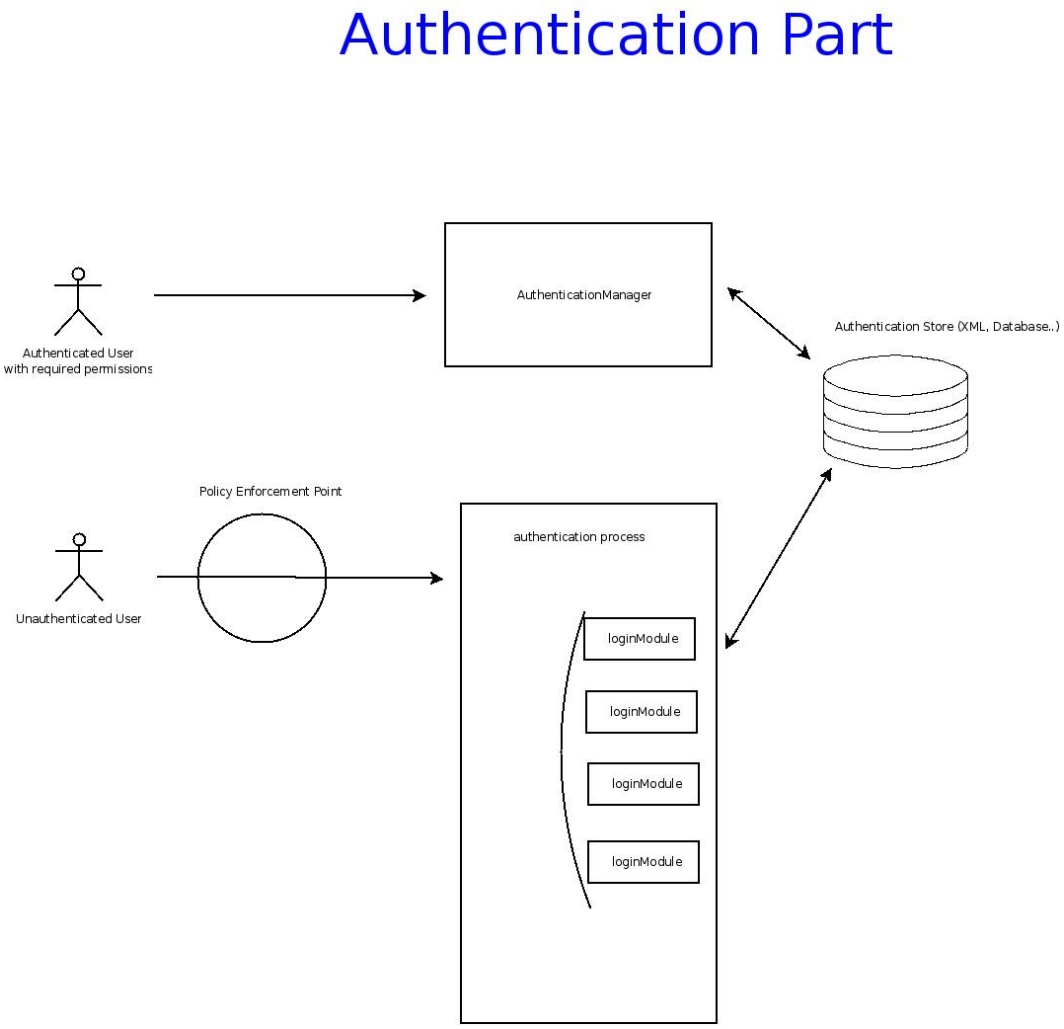


Figure 4.1. Authentication part in jGuard

4.2. AuthenticationManager

4.2.1. description

AuthenticationManager implementations aims to do Create, Read, Update, Delete (CRUD) operations on users and roles of the application. These Users and roles are present in the datasource authentication. This datasource (database, XML and so on..), is also the same one used for the `authentication process` which involve`LoginContext`, `Configuration` and `LoginModules`.

Note

if the user does not tries to authenticate, jGuard automatically authenticates you as the 'guest' user. it's not a security issue, but a design choice. but to fulfills your security

requirements, you can configure that guest (unauthenticated users), hasn't got access to your protected pages. how to do it? => configure the 'guest' role with no permissions. the guest user will only have access to login page and access denied page(access is always grant to these resources).

4.2.2. configuration

Authentication configuration in jGuard, is done via the `jGuardAuthentication.xml` file.

goals of this configuration file is to:

- define the authentication scope
- define the AuthenticationManager implementation
- define the loginmodules involved in the `authentication process` with their options and JAAS keywords (required, optional, and so on...)

4.2.3. implementations

4.2.3.1. XMLAuthenticationManager

4.2.3.1.1. description

This AuthenticationManager implementation permits to persist in a XML file all the auithentication informations of your application.

4.2.3.1.2. parameters

- debug

  This optional parameter, when set to `true`, activate the debug mode (provide more logs to detect easily misconfiguration).

- authenticationXmlFileLocation

  a relative path from the webapp, of the `jGuardUsersPrincipals.xml` file.

4.2.3.1.3. usual configuration in the `jGuardConfiguration.xml` file

```
....
                                ....

 <authenticationManager>net.sf.jguard.authentication.XmlAuthenticationManager</
authenticationManager>
                                <authenticationManagerOptions>
                                <option>
                                <name>authenticationXmlFileLocation</name>
                                <value>WEB-INF/conf/jGuard/jGuardUsersPrincipals.xml</
value>
                                </option>
                                </authenticationManagerOptions>
                                ....
                                ....
```

4.2.3.2. HibernateAuthenticationManager

> **Note**
>
> The `JdbcAuthenticationManager` has been replaced by the `HibernateAuthenticationManager` for a better flexibility. A further `JPAAuthenticationManager`, ORM agnostic `AuthenticationManager` implementation, can be another suitable solution provided in a future release.

4.2.3.2.1. description

This `AuthenticationManager` implementation permits to persist in a database all the authentication informations of your application.HIerbnaetAuthenticationManager needs to use a SessionFactory instance; here are the ways supported to grab the sessionFactory

- hibernate.cfg.xml

  Hibernate can build and use a sessionFactory by reading an Hibernate.cfg.xml config file present on the classPath.

4.2.3.2.2. parameters

- authenticationXmlFileLocation

  WEB-INF/conf/jGuard/jGuardUsersPrincipals.xml for example . This parameter permits to import some data when your database is empty

4.2.3.2.3. configuration

```
....
                                ....

 <authenticationManager>net.sf.jguard.ext.authentication.manager.HibernateAuthenticationManager</
authenticationManager>
                                <authenticationManagerOptions>
                                <option>
                                <name>authenticationXmlFileLocation</name>
                                <value>WEB-INF/conf/jGuard/jGuardUsersPrincipals.xml</
value>
                                </option>
                                </authenticationManagerOptions>
                                ....
                                ....
```

> **Note**
>
> you have to configure the associated HibernateLoginModule.

4.2.3.2.4. ER Diagram

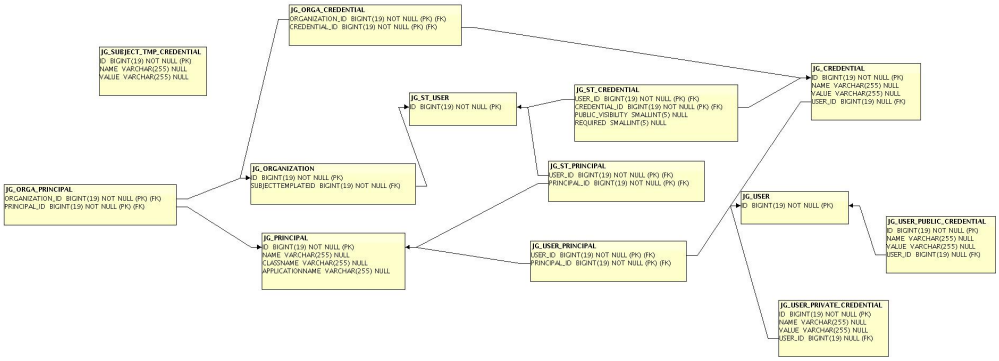


Figure 4.2. authentication ER diagram

4.3. JAAS Authentication process

Authentication process is standardized into java through the JAAS api. It involves the `LoginContext` class, a callbackHandler implementation, a `Configuration` instance, and some loginModules. jGuard provides

4.3.1. javax.security.auth.login.LoginContext

This class is the main entry point to the Authentication Process. it defines :

- which Subject (user) authenticate
- through which way (CallbackHandler)
- for which application
- with which authentication technologies (LoginModules)
- in which mechanism (Configuration)

This class provides multiple constructors which permits to build a convenient LoginContext class. Authentication is done during the `login` method, which return an authenticated`Subject`, or a `LoginException`. when the user quit the application, the `logout` method should to be called.

Note

in webapps, jGuard provides some high-level classes to reduce your work, and simplify the use of JAAS like the AccessFilter servlet filter.

4.3.2. javax.security.auth.callback.CallbackHandler

this class handle the way to grab informations contained into information from a protocol, to fill callbacks (used by loginmodules) to authenticate the user. So, LoginModules can use the same callbacks but with different CallbackHandler depending on the situation. jGuard provides different callbackHandler like `JMXCallbackHandler`, `SwingCallbackHandler` ,and `HttpServletCallbackHandler.`

4.3.3. loginModules

4.3.3.1. description

after the user transmits its credentials through the authentication scheme, jGuard should authenticate the user with them.to authenticate the user, jGuard use its credentials against some security challenges: LoginModules.

loginModules are stackable: you can configure multiple loginModules (in the `jGuardConfiguration.xml` file),which will help you to authenticate a user or not.

each loginModule has got a flag which can be '`REQUIRED`','`OPTIONAL`','`REQUISITE`' ou '`SUFFICIENT`' (JAAS documentation fromSUN™):

- REQUIRED

  The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

- 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).

- 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.

- OPTIONAL

  The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

> **Note**
>
> You should have noticed that the configuration for loginModules involved in validating user identity is very small: configuration already defined in `AuthenticationManager` are reused to establish connections in loginmodules. so, `JdbcLoginmodule` uses `JdbcAuthenticationManager` configuration to establish database connections and so on…

4.3.3.2. jGuard loginModules

4.3.3.2.1. UserLoginModule

This loginModule is an `abstract` class. its subclasses grab informations in various locations (database,LDAP,Xml and so on...) to authenticate users.

Note

stores are configured into AuthenticationManager implementations and are reused by UserLoginModule subclasses.So, you don't have to declare stores into UserLoginModule subclasses.

4.3.3.2.2. XMLLoginModule

description

This Loginmodule inherit from UserLoginmodule and use users and roles located in an XML file called `jGuardUsersPrincipals.xml` to authenticate users.

parameters

- debug

  This optional parameter, when set to `true`, activate the debug mode (provide more logs to detect easily misconfiguration).

XMLLoginModule declaration

Example 4.1. configuration of XMLLoginModule into a fragment of jGuardAuthentication.xml

to reference XMLLoginModule, you have to declare it into `jGuardAuthentication.xml` file.

```
....
                                        ....
                                        <authentication>
                                        <loginModules>
                                        <loginModule>

 <name>net.sf.jguard.ext.authentication.loginmodules.XmlLoginModule</name>
                                        <flag>REQUIRED</flag>
                                        <loginModuleOptions>
                                        <option>
                                        <name>debug</name>
                                        <value>false</value>
                                        </option>
                                        </loginModuleOptions>
                                        </loginModule>
                                        </loginModules>
                                        </authentication>
                                        ....
                                        ....
```

```
                                        <credTemplateId>location</credTemplateId>
                                        </publicRequiredCredentials>
                                        <privateOptionalCredentials>
                                        <credTemplateId>country</credTemplateId>
                                        <credTemplateId>religion</credTemplateId>
                                        </privateOptionalCredentials>
                                        <publicOptionalCredentials>
                                        <credTemplateId>hobbies</credTemplateId>
                                        </publicOptionalCredentials>
                                        <principalsRef>
                                        <principalRef name="admin"
 applicationName="jguard-struts-example"/>
                                        <principalRef name="role3"
 applicationName="anotherApplication"/>
                                        </principalsRef>
                                        </userTemplate>
                                        <credentials>
                                        <credTemplateId>id</credTemplateId>
                                        </credentials>
                                        <principalsRef>
                                        <principalRef name="admin"
 applicationName="jguard-struts-example"/>
                                        <principalRef name="role3"
 applicationName="anotherApplication"/>
                                        </principalsRef>
                                        </organizationTemplate>
                                        <organization>
                                        <userTemplate>
                                        <privateRequiredCredentials>
                                        <credTemplateId>login</credTemplateId>
                                        <credTemplateId digestNeeded="true">password</
credTemplateId>
                                        </privateRequiredCredentials>
                                        <publicRequiredCredentials>
                                        <credTemplateId>firstname</credTemplateId>
                                        <credTemplateId>lastname</credTemplateId>
                                        <credTemplateId>location</credTemplateId>
                                        </publicRequiredCredentials>
                                        <privateOptionalCredentials>
                                        <credTemplateId>country</credTemplateId>
                                        <credTemplateId>religion</credTemplateId>
                                        </privateOptionalCredentials>
                                        <publicOptionalCredentials>
                                        <credTemplateId>hobbies</credTemplateId>
                                        </publicOptionalCredentials>
                                        <principalsRef>
                                        <principalRef name="admin"
 applicationName="jguard-struts-example"/>
                                        <principalRef name="role3"
 applicationName="anotherApplication"/>
                                        </principalsRef>
                                        </userTemplate>
                                        <credentials>
                                        <credential>
                                        <id>id</id>
                                        <value>system</value>
                                        </credential>
                                        <credential>
                                        <id>creation date</id>
                                        <value>1965</value>
                                        </credential>
                                        </credentials>
                                        <principalsRef>
                                        <principalRef applicationName="jguard-struts-
example" name="guest"/>
                                        <principalRef name="admin"
 applicationName="jguard-struts-example" />
                                        </principalsRef>
                                        </organization>
                                        </organizations>
                                        </authentication>
```

4.3.3.2.3. HibernateLoginModule

description

This loginModule inherit from UserLoginmodule and allows a database-based authentication for your application.

parameters

- debug

  This optional parameter, when set to `true`, activate the debug mode (provide more logs to detect easily misconfiguration).

HibernateLoginModule declaration

Example 4.3. configuration of HibernateLoginModule into a fragment of jGuardAuthentication.xml

to reference HibernateLoginModule, you have to declare it into `jGuardAuthentication.xml` file.

```
....
                                        ....
                                        <authentication>
                                        <loginModule>

 <name>net.sf.jguard.ext.authentication.loginmodules.HibernateLoginModule</name>
                                        <!-- flag :'REQUIRED','OPTIONAL','REQUISITE' or
 'SUFFICIENT' -->
                                        <flag>REQUIRED</flag>
                                        <loginModuleOptions>
                                        <option>
                                        <name>debug</name>
                                        <value>false</value>
                                        </option>
                                        </loginModuleOptions>
                                        </loginModule>
                                        ....
                                        ....
```

4.3.3.2.4. JNDILoginModule

description

this UserLoginModule subclass, authenticate users and roles into a JNDI directory like LDAP.

most of the parameters detailed in this page comes from the JNDI Context constants [http://java.sun.com/j2se/1.4.2/docs/api/javax/naming/Context.html] detailed in the related JNDI javadoc [http://java.sun.com/j2se/1.4.2/docs/api/constant-values.html#javax.naming.Context.SECURITY_AUTHENTICATION].

Note

this loginmodule does NOT yet retrieve roles from the directory. only authentication against credentials are supported. roles support will be rpovided in a future release.

connections

connections can be established either through manual configuration, or via application server JNDI lookup. for all parameters, you have to include the prefix ('`preauth.`' or '`auth.`'). these settings apply to `direct authentication` and `pre-authentication` modes.

manual configuration

- initial context factory

```
.....
                                        <option>
                                        <name>preauth.java.naming.factory.initial</
name>
                                        <value>com.sun.jndi.ldap.LdapCtxFactory</
value>
                                        </option>
                                        .....
```

- provider url

```
.....
                                        <option>
                                        <name>preauth.java.naming.provider.url</
name>
                                        <value>ldap://mycompany.com:389</value>
                                        </option>
                                        .....
```

- authentication mode

  you can use `none` , `simple` or a SASL mechanism type defined in theRFC 2195 [http://www.ietf.org/rfc/rfc2195.txt].

```
.....
                                        <option>
                                        <name>java.naming.security.authentication</
name>
                                        <value>none</value>
                                        </option>
                                        .....
```

- activate connection pooling

  you can sometimes activate the connection pooling of the JNDI service providers. if you use the one provided by sun, here is the configuration:

```
.....
                                        <option>

 <name>preauth.com.sun.jndi.ldap.connect.pool</name>
                                        <value>true</value>
                                        </option>
                                        .....
```

- define the preferred number of connections

```
.....
                                        <option>

 <name>preauth.com.sun.jndi.ldap.connect.pool.prefsize</name>
                                        <value>5</value>
                                        </option>
                                        .....
```

- define connection timout

```
.....
                                        <option>

 <name>preauth.com.sun.jndi.ldap.connect.pool.timeout</name>
                                        <value>300000</value>
                                        </option>
                                        .....
```

  this example defines the number of milliseconds (5 minutes in this example) that an idle connection may remain in the pool without being closed and removed from the pool.

- other connection pool settings

  other settings can be reached at the JNDI/LDAP Service provider documentation [http://java.sun.com/j2se/1.4.2/docs/guide/jndi/jndi-ldap.html#POOL] page.

- activate the Fast bind connection mode for Active Directory

  a specific LDAP control can be activated against Active Directory server. more details on it here: Active directory LDAP server fast bind mode documentation

```
.....
                                        <option>
                                        <name>preauth.fastBindConnection</name>
                                        <value>true</value>
                                        </option>
                                        .....
```

application server JNDI lookup

connections can be retrieved via the application servers JNDI system. to specify which name must be used to grab the context, you have to use after the prefix, the jndi name.

```
.....
                                    <option>
                                    <name>preauth.jndi</name>
                                    <value>myDs</value>
                                    </option>
                                    .....
```

description

different use cases are possible, depending on how to find the Distinguished Name(DN), which is the path to the user Entry.this DN will be used to :

- authenticate the user

- grab its associated credentials.

direct authentication (auth mode)

authentication parameters starts with the '`auth.`' prefix. the DN is known, directly from the `userDN` parameter and the user login. for example,

```
....
                                    <option>
                                    <name>auth.userDN</name>
                                    <value>dc=com,dc=mycompany,ou=mysection,cn={0}</
value>
                                    </option>
                                    ....
```

the `{0}` will be replaced by the login provided by the user.

pre-authentication (preauth mode)

when the user DN cannot be known, a first search should be done to know what is the DN. authentication parameters starts with the '`preauth.`' prefix.

- base DN

  this parameter defines from which Distinguished Name (path) starts the search.

```
....
                                                <option>
                                                <name>preauth.search.base.dn</name>
                                                <value>dc=mycompany,dc=com</value>
                                                </option>
                                                ....
```

- search filter

  this parameter define the LDAP filter used to locate the DN of the user.

```
....
                                                <option>
                                                <name>preauth.search.filter</name>
                                                <value>(&(samAccountName={0})(!
(proxyAddresses=*)))</value>
                                                </option>
                                                ....
```

- search scope

  for object scope, use 0. for one level scope, use 1. for subtree scope, use 2.

```
....
                                                <option>
                                                <name>preauth.searchcontrols.searchscope</
name>
                                                <value>2</value>
                                                </option>
                                                ....
```

which connection use to populate the Subject (user)

usually, jGuard reuses the connection used to authenticate the user, to do a lookup on the user entry. sometimes, it can be useful to grab the informations directly from the LDAP entry found to know the user DN. to do it, you have to include this parameter:

```
....
                                    <option>
                                    <name>contextforcommit</name>
                                    <value>true</value>
                                    </option>
                                    ....
```

complete configuration example with `preauth`

in your `jGuardAuthentication.xml` file:

```
.......
                                    <loginModule>

 <name>net.sf.jguard.ext.authentication.loginmodules.JNDILoginModule</name>
                                    <flag>REQUIRED</flag>
                                    <loginModuleOptions>
                                    <option>
                                    <name>preauth.java.naming.factory.initial</name>
                                    <value>com.sun.jndi.ldap.LdapCtxFactory</value>
                                    </option>
                                    <option>
                                    <name>preauth.java.naming.provider.url</name>
                                    <value>ldap://yourcompany.com:389</value>
                                    </option>
                                    <option>
                                    <name>java.naming.security.authentication</name>
                                    <value>none</value>
                                    </option>
                                    <option>
                                    <name>preauth.searchcontrols.searchscope</name>
                                    <value>2</value>
                                    </option>
                                    <option>
                                    <name>preauth.search.base.dn</name>
                                    <value>dc=stuff,dc=com</value>
                                    </option>
                                    <option>
                                    <name>preauth.search.filter</name>
                                    <value>(&(samAccountName={0})(!(proxyAddresses=*)))</
value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.factory.initial</name>
                                    <value>com.sun.jndi.ldap.LdapCtxFactory</value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.provider.url</name>
                                    <value>ldap://yourcompany.com:389</value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.security.authentication</name>
                                    <value>simple</value>
                                    </option>
                                    <option>
                                    <name>contextforcommit</name>
                                    <value>true</value>
                                    </option>
```

```
                                    </loginModuleOptions>
                                    </loginModule>
                                    .......
```

Direct Authentication configuration example

```
.......
                                    <loginModule>

 <name>net.sf.jguard.ext.authentication.loginmodules.JNDILoginModule</name>
                                    <flag>REQUIRED</flag>
                                    <loginModuleOptions>
                                    <option>
                                    <name>preauth.java.naming.factory.initial</name>
                                    <value>com.sun.jndi.ldap.LdapCtxFactory</value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.provider.url</name>
                                    <value>ldap://168.12.45.88:389</value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.security.authentication</name>
                                    <value>simple</value>
                                    </option>
                                    <option>
                                    <name>auth.java.naming.security.authentication</name>
                                    <value>simple</value>
                                    </option>
                                    <option>
                                    <name>com.sun.jndi.ldap.connect.pool</name>
                                    <value>true</value>
                                    </option>
                                    <option>
                                    <name>com.sun.jndi.ldap.connect.pool.prefsize</name>
                                    <value>5</value>
                                    </option>
                                    <option>
                                    <name>contextforcommit</name>
                                    <value>false</value>
                                    </option>
                                    <option>
                                    <name>auth.userDN</name>
                                    <value>{0}</value>
                                    </option>
                                    </loginModuleOptions>
                                    </loginModule>
                                    .......
```

4.3.3.2.5. Certificate-based LoginModules

jGuard looks into the X509 Certificate provided by the user and checks against the Certificate Authority its validity (with the loginMoules described below). When the certicate is valid, jGuard uses the informations present in it to populate the `javax.security.auth.Subject`.

> **Note**
>
> jGuard only handle the first certificate provided by the user (theorically, a user can provide multiple certificates at the same time).

- unique ID

If the certificate field `subjectUniqueID` is present in the certificate, jGuard create a credential called `uniqueID` in the `javax.security.auth.Subject`. this field is OPTIONAL in the certificate. jGuard use the method `getSubjectUniqueID` from the class `java.security.cert.X509Certificate` to grab this information.

- alternative names

  if a `SubjectAltName` extension is present in the certificate, jGuard grab the subject alternatives names with the method `getSubjectAlternativeNames`, and create for each alternative name a credential with the name `alternativeName#aSequenceNumber` . `SubjectAltName` extension in certificate is OPTIONAL.

- X500 principal

  jguard grab a principal object from the field `subject` in the certificate, with the method `getSubjectX500Principal` method,and put it into the `Principals` set of the `Subject.`

  Important

  this field in the certificate is REQUIRED.

  When you use the `JdbcLoginmodule` (or `XmlLoginModule`) in conjunction with a certificate-related loginModule, jGuard will check this value against the value of the credential called '`login`'.

what about certificate informations and other LoginModules?

you can use other loginModules like `XMLLoginModule` and `JdbcLoginmodule` in collaboration with `CRLLoginModule` or `OCSPLoginModule`. in others authentication schemes, i.e`FORM`,`BASIC`, or`DIGEST`, the user actively send its login and password informations. These informations are used by `XMLLoginModule` or `JdbcLoginmodule` to check if the user eixsts and if its password is valid. after this step, it populates the `Subject` with some informations from the Datasource (XML or Database). with `CLIENT-CERT` authentication, the mechanism is in the same way. the only difference is that the user automatically send its login and other information with its certificate. No password are required, because some powerful cryptographic mechanisms check the validity of the certificate. so, when the user transmit its certificate, jGuard populate the Subject with a `X500Principal` object. jGuard use the String returned by the `getName` method of the `X500Principal` stored in the `Subject` as the login information.

what precise informations are required in the X509 certificate?

to summarize, the only information required in the certificate is the '`subject`' field. the value of '`subject`' field should be a '`distinguished name`' (DN), compliant with the RFC 2253 [http://www.ietf.org/rfc/rfc2253.txt]. to have more informations on certificate structure(which fields can be inserted), you can look towards theRFC 2459 [http://www.ietf.org/rfc/rfc2459.txt].

4.3.3.2.6. CRLLoginModule

description

This loginModule permits an authentication based on X509 certificates: it validates their`certPath`, and checks if some of them are revoked against a CRL [http://en.wikipedia.org/wiki/

Certificate_revocation_list] (Certificate Revocation List) which lists certificates that has been revoked by the `Certificate Authority` (CA) before their scheduled expiration date.

> **Note**
>
> Note this certificate validation mechanism is not based on real-time mechanism: the accuracy of this mechanism is based on the `CRL` generation frequency of the Certificate Authority. If you authenticate successfully a user, but the CRL used to check it is too old, you will have a security threat.

parameters

These parameters comes from the `PKIXParameters` [http://java.sun.com/j2se/1.4.2/docs/api/java/security/cert/PKIXParameters.html] java class: parameters description comes from the JDK's javadoc.

- debug

  This optional parameter, when set to `true`, activate the debug mode (provide more logs to detect easily misconfiguration).

- certPathAnyPolicyInhibited

  Sets state to determine if the any policy OID should be processed if it is included in a certificate. can be `true` or `false`, and default value is `false` if not set.

- certPathExplicitPolicyRequired

  If this flag is`true`, an acceptable policy needs to be explicitly identified in every certificate. default value is `false`.

- certPathPolicyMappingInhibited

  If this flag is`true`, policy mapping is inhibited. default value is `false`.

- certPathPolicyQualifiersRejected

  f this flag is`true`, certificates that include policy qualifiers in a certificate policies extension that is marked `critical` are rejected. If the flag is`false`, certificates are not rejected on this basis.Applications that want to use a more sophisticated policy must set this flag to`false`.Default value is `true.`

  > **Note**
  >
  > Note that the PKIX certification path validation algorithm specifies that any policy qualifier in a certificate policies extension that is marked `critical` must be processed and validated. Otherwise the certification path must be rejected. If the `policyQualifiersRejected` flag is set to false, it is up to the application to validate all policy qualifiers in this manner in order to be PKIX compliant.

- certPathRevocationEnabled

  If this flag is`true`, the default revocation checking mechanism of the underlying PKIX service provider will be used. If this flag is`false`, the default revocation checking mechanism will be

disabled (not used).When a PKIXParameters object is created, this flag is set to`true`. This setting reflects the most common strategy for checking revocation, since each service provider must support revocation checking to be PKIX compliant. Sophisticated applications should set this flag to `false` when it is not practical to use a PKIX service provider's default revocation checking mechanism or when an alternative revocation checking mechanism is to be substituted.Default value is`true`.

- certPathSigProvider

  Sets the signature provider's name. The specified provider will be preferred when creating Signature objects. If null or not set, the first provider found supporting the algorithm will be used.

- certPathCrlPath

  if this value is defined, it grabs the CRL from a file and add it to the CRLs collection. the value is a system-dependent fileName.

- certPathUrlCrlPath

  if this value is defined, it grabs the CRL from an HTTP URL and add it to the CRLs collection.

- trustedCaCertsDirPath

  this directory path must contain `Trusted certificates` to build`Trust Anchors`.

- securityProvider

  a security provider class name to use. default value is `org.bouncycastle.jce.provider.BouncyCastleProvider.`

- certPathCertStoreType

  define from which source the certstore will retrieve certificates and CRLs. value can be `LDAP` or `Collection.`

- certPathLdapServerName

  server name used to grab certificates and CRLS for the certstore.default value is `localhost.`

- certPathLdapServerPort

  server port used to grab certificates and CRLS for the `certstore`. default value is `389`.

- javax.net.ssl.trustStore

  file path of the`trustStore`.

- javax.net.ssl.trustStorePassword

  password protecting access to `TrustStore` data present in the file.

- keyStorePath

  file path of the`keyStore`.

- keyStorePassword

password protecting access to `keyStore` data present in the file.

- keyStoreType

  Valid types can be those returned by the `java.security.Security.getAlgorithms("KeyStore")` attribute(JKS,JCEKS,PKCS12, PKCS11 (Java crypto device),CMSKS, JCERACFKS ...)

4.3.3.2.7. OCSPLoginModule

description

This loginModule permits an authentication for your web application based on X509 certificates: it validates their `certPath`, and checks if some of them are revoked against a OCSP mechanism. this mechanism permits real-time certificate revocation check.

parameters

- ocspServerURL

  URL of the server which rpovide OCSP validation service .this parameter is mandatory.

- IssuerCACertLocation

  location of the certificate of the issuer Certificate Authority.

- OcspSignerCertLocation

  location of the certificate of the OCSP signer.

OCSPLoginModule declaration

Example 4.4. configuration of OCSPLoginModule into a fragment of jGuardAuthentication.xml

```
…
                                        …
                                        <loginModule>

 <name>net.sf.jguard.ext.authentication.loginmodules.OCSPLoginModule</name>
                                        <flag>REQUIRED</flag>
                                        <loginModuleOptions>
                                        <option>
                                        <name>debug</name>
                                        <value>true</value>
                                        </option>
                                        <option>
                                        <name>ocspServerURL</name>
                                        <value>http://127.0.0.1:8080/ejbca/publicweb/
status/ocsp</value>
                                        </option>
                                        <option>
                                        <name>IssuerCACertLocation</name>
                                        <value>/home/user/certificates/AdminCA1.der</
value>
                                        </option>
                                        <option>
                                        <name>OcspSignerCertLocation</name>
                                        <value>/home/user/certificates/AdminCA1.der</
value>
                                        </option>
                                        </loginModuleOptions>
                                        </loginModule>
                                        …
                                        ...
```

4.3.3.2.8. JCaptchaLoginModule

description

since jGuard0.80, this loginModule permits to validate a user against a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart), to determine if the user is human or not.

generate the Captcha

to use this loginModule, you need firstly to insert in your login page, this kind of code to generate an image containing the challenge:

```
<img id="captcha" src="<html:rewrite action='/Captcha.do'/>" />
```

note this code is used with the Struts framework, because we use the html:rewrite taglib to generate the url to access to the Captcha Action which will generate the image. but you can use any framework to do the same thing. to permit the user to answer to the challenge, you need also to insert in your login form, the related fields:

```
<div>
                                    <label for="captchaAnswer">captchaAnswer(required)</
label>
```

```
                                    <input id="captchaAnswer" type="text" value=""
size="30" name="captchaAnswer"
                                    tabindex="3" />
                                    </div>
```

validate the Captcha

to use this loginMoule, you need to insert in your JGuardConfiguration.xml file, this declaration:

```
<loginModule>

 <name>net.sf.jguard.authentication.loginmodules.JCaptchaLoginModule</name>
                                    <flag>REQUIRED</flag>
                                    </loginModule>
```

> **Note**
>
> note that you can use other flags,depending on your needs.

Captcha library used

this loginModule use to generate and validate the Captcha, the JCaptcha open source library. Note that the archive shipped with the distribution is not the last release; we use only the JCaptcha 1.0 RC2 release: since the release1.0 RC3, the JCaptcha project has changed its licence from LGPL to GPL, which prevents us to ship it with jGuard (licensed under theLGPL). But we've tested jGuard with JCaptcha 1.0 RC3 or higher successfully, so we advice you to use this JCaptcha release if you accept that your application will be under the GPL umbrella.

> **Note**
>
> JCaptcha team has produced other releases under the LGPL licence.So, you can use these releases with jGuard, but be aware that JCaptcha need a Java 5 or higher JVM to work.

CAPTCHA example

Example 4.5. CAPTCHA example generated with JCaptcha

4.3.3.2.9. implements your own loginModule

you can add your own loginModule on the authentication loginModules list. To do it, you have to implements the `javax.security.auth.spi.LoginModule` interface.

to authenticate a user against a store shipped with an `AuthenticationManager`, you have to inherit from `net.sf.jguard.core.authentication.loginmodules.UserLoginModule`.

4.3.4. javax.security.auth.login.Configuration

this class defines which loginModules are involved in the authentication process, for a defined application (identified by its name). Each application has its own loginModules stack. With some special keywords to customize the mechanism.

4.3.5. javax.security.auth.Subject

Object resulting from a successful authentication. it represents a "real" entity like a user, a machine, and so on.... it contains some *Principals* and some *credentials* present in relative Sets.

- Principals are often considered as Identities (one user can have more than one identity) or roles.
- Credentials are user's attributes specific to him. it can be a first name, a credit card number, a birth date....

  credentials are present either in a public credential Set or a private credential set (access to them is protected by a `PrivateCredentialPermission`).

4.3.6. java.security.Principal

4.3.6.1. RolePrincipal

subject, which represents authenticated user, contains its own credentials, and also some Principals. the main implementation of principal provided in Subject by jGuard is `net.sf.jguard.core.principals.RolePrincipal`; this class represent a*role*, a responsability granted to the user.

4.3.6.2. Organization

Another implementation provided is `net.sf.jguard.core.principals.Organization` ;it represents the entity which grants the user to use the application.Every user has got an Organization and only one. This feature permits to handle in your application multiple organization support;`RolePrincipal`s are owned by organization declared in your application.if no Organization are declared, users are linked with a default organization.

4.3.7. Dynamic role definition

jGuard provides a mechanism that allows to automatically enable or disable a role (RolePrincipal) based on user credentials. This feature is defined in the attribute “definition” of “principalRef” tag. This attribute must evaluate “true” or “false”. For example, we could have in jGuardUsersPrincipals.xml the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
                    <authentication xmlns="http://jguard.sourceforge.net/xsd/
jGuardUsersPrincipals_2.0.0.xsd"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
                    xs:schemaLocation="http://jguard.sourceforge.net/xsd/
jGuardUsersPrincipals_2.0.0.xsd
                    jGuardUsersPrincipals_1.1.0.xsd">
                    <principals>
                    <principal>
                    <name>admin</name>
                    <class>net.sf.jguard.core.principals.RolePrincipal</class>
                    <applicationName>jguard-struts-example</applicationName>
                    <organizationRef>myenterpriseId</organizationRef>
                    </principal>
```

```
                    <principal>
                    <name>guest</name>
                    <class>net.sf.jguard.core.principals.RolePrincipal</class>
                    <applicationName>jguard-struts-example</applicationName>
                    <organizationRef>system</organizationRef>
                    </principal>
                    <principal>
                    <name>role3</name>
                    <class>net.sf.jguard.core.principals.RolePrincipal</class>
                    <applicationName>anotherApplication</applicationName>
                    <organizationRef>system</organizationRef>
                    </principal>
                    </principals>
                    <users>
                    <user>
                    <privateCredentials>
                    <credential>
                    <id>login</id>
                    <value>admin</value>
                    </credential>
                    <credential>
                    <id>password</id>
                    <value>admin</value>
                    </credential>
                    </privateCredentials>

                    <publicCredentials>
                    <credential>
                    <id>firstname</id>
                    <value>Rick</value>
                    </credential>
                    <credential>
                    <id>lastname</id>
                    <value>Dangerous</value>
                    </credential>
                    <credential>
                    <id>location</id>
                    <value>Paris</value>
                    </credential>
                    </publicCredentials>

                    <principalsRef>
                    <principalRef name="admin" applicationName="jguard-struts-example"
                    definition="${subject.publicCredentials.location.contains('Paris')}"
active="true"/>
                    <principalRef name="role3" applicationName="anotherApplication"/>
                    </principalsRef>
                    <organizationRef>system</organizationRef>
                    </user>
                    <user>
                    <privateCredentials>
                    <credential>
                    <id>login</id>
                    <value>guest</value>
                    </credential>
                    <credential>
                    <id>password</id>
                    <value>guest</value>
                    </credential>
                    </privateCredentials>
                    <publicCredentials/>
                    <principalsRef>
                    <principalRef name="guest" applicationName="jguard-struts-example" />
                    </principalsRef>
                    <organizationRef>system</organizationRef>
                    </user>
                    </users>
```

```
                    <organizations>
                    <organizationTemplate>
                    <userTemplate>
                    <privateRequiredCredentials>
                    <credTemplateId>login</credTemplateId>
                    <credTemplateId digestNeeded="true">password</credTemplateId>
                    </privateRequiredCredentials>
                    <publicRequiredCredentials>
                    <credTemplateId>firstname</credTemplateId>
                    <credTemplateId>lastname</credTemplateId>
                    <credTemplateId>location</credTemplateId>
                    </publicRequiredCredentials>
                    <privateOptionalCredentials>
                    <credTemplateId>country</credTemplateId>
                    <credTemplateId>religion</credTemplateId>
                    </privateOptionalCredentials>
                    <publicOptionalCredentials>
                    <credTemplateId>hobbies</credTemplateId>
                    </publicOptionalCredentials>
                    <principalsRef>
                    <principalRef name="admin" applicationName="jguard-struts-example"/>
                    <principalRef name="role3" applicationName="anotherApplication"/>
                    </principalsRef>
                    </userTemplate>
                    <credentials>
                    <credTemplateId>id</credTemplateId>
                    </credentials>
                    <principalsRef>
                    <principalRef name="admin" applicationName="jguard-struts-example"/>
                    <principalRef name="role3" applicationName="anotherApplication"/>
                    </principalsRef>
                    </organizationTemplate>
                    <organization>
                    <userTemplate>
                    <privateRequiredCredentials>
                    <credTemplateId>login</credTemplateId>
                    <credTemplateId digestNeeded="true">password</credTemplateId>
                    </privateRequiredCredentials>
                    <publicRequiredCredentials>
                    <credTemplateId>firstname</credTemplateId>
                    <credTemplateId>lastname</credTemplateId>
                    <credTemplateId>location</credTemplateId>
                    </publicRequiredCredentials>
                    <privateOptionalCredentials>
                    <credTemplateId>country</credTemplateId>
                    <credTemplateId>religion</credTemplateId>
                    </privateOptionalCredentials>
                    <publicOptionalCredentials>
                    <credTemplateId>hobbies</credTemplateId>
                    </publicOptionalCredentials>
                    <principalsRef>
                    <principalRef name="admin" applicationName="jguard-struts-example"/>
                    <principalRef name="role3" applicationName="anotherApplication"/>
                    </principalsRef>
                    </userTemplate>
                    <credentials>
                    <credential>
                    <id>id</id>
                    <value>system</value>
                    </credential>
                    <credential>
                    <id>creation date</id>
                    <value>1965</value>
                    </credential>
                    </credentials>
                    <principalsRef>
                    <principalRef applicationName="jguard-struts-example" name="guest"/>
```

```
                    <principalRef name="admin" applicationName="jguard-struts-example" />
                    </principalsRef>
                    </organization>
                    </organizations>
                    </authentication>
```

The application could provide a way to allow the user to change its credential “loggedProject” between “ProjectA” and “ProjectB”. If the users chooses “ProjectA”, jGuard will automatically enable those roles where attribute “definition” evaluates “true”. You could use more complex expressions using logical operators, for example:

```
(subject.publicCredentials.loggedProject.contains('ProjectA') ||
                    subject.publicCredentials.loggedProject.contains('ProjectB')) &&
                    subject.privateCredentials.login.contains('userA')
```

4.3.7.1. synthax

jguard uses the jakarta commons JEXL project expression library to provide expressiveness on contextual variables. this project has got a syntax page: http://jakarta.apache.org/commons/jexl/reference/syntax.html

4.3.7.2. How to get/set role definition by code

You can get role definition using:

```
RolePrincipal ppal =

 (RolePrincipal)AuthenticationManagerFactory.getAuthenticationManager().getRole(subject,
 role,
                        applicationName);
                        String roleDefinition = ppal.getDefinition();
```

And you can set role definition using:

```


 AuthenticationManagerFactory.getAuthenticationManager().updateRoleDefinition(subject,
 roleName,
                        applicationName, roleDefinition);
```

4.4. password encryption

4.4.1. principle

jGuard can use a cryptographic hash function [http://en.wikipedia.org/wiki/Cryptographic_hash_function] to avoid password to be stored in clear on the server. This function has the characteristic to produce a unique digest of any string. So, a password '`secret1`' will produce with one of these functions a digest like '`1gE1r`'. a password '`secret2`' will produce a digest like '`sF42`'. For each password, a different digest will be produced. So,each original password will have its unique digest.

another characteristic is that it is very difficult to know which password is the origin of a related digest . So, if we store 'digested' password and not the original, anyone which will read these ones will not

have the original password. But when we receive the password from the end user, we calculate the digest of it, and compare it with the one stored on the webapp using jGuard.

4.4.2. supported algorithms

`Cryptographic hash functions` also called `MessageDigest algorithms`, are available in Java. Different algorithms are supported. You need to refer to the documentation of your JVM vendor, especially the `Java Cryptography Architecture` section. For example, many JVM vendors provides these algorithms:

- MD2
- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512

Other ones are also available. To use one algorithm in jGuard, you have to use the right code related to the algorithm chosen like the ones listed above. To use this algorithm, you have to uncomment the `digestAlgorithm` markup in the `jGuardAuthentication.xml` file and put into your messageDigest algorithm code.

```
<digestAlgorithm>MD5</digestAlgorithm>
```

4.4.3. salted passwords

password saltingis described on wikipedia [http://en.wikipedia.org/wiki/Password_salting]. Password are impossible to know if you have only their digest. But if you store the digest of some popular password and the origin in some big tables, you can lookup easily the origin based on the digest. These tables are called `rainbow tables` [http://en.wikipedia.org/wiki/Rainbow_tables] .

Password salting is used to prevent some attackers to use rainbow tables. These tables are based on a known password origin. But if you modify the origin with a '`salt`' specific to your application, it will be impossible to create rainbow tables with all the possibilities of salt.

For example, if you concatenate '`passord1`' with the salt '`salt1`' before to calculate the digest, the result will be different and very difficult to know. To use salt in conjunction of digest algorithm functions (you cannot only use salt without digest algorithm), you have to uncomment in the jGuardAuthentication.xml file the salt markup and put into it the salt chosen. Be aware that sometimes, some rainbow tables also contains some short salts. To be very safe, you should use long salts.

```
<salt>qsd846ss2q6ds4</salt>
```

Chapter 5. Registration

Note

Registration is an optional process in jGuard. to not use it, don't configure it.

5.1. Configure the registration requirements

when you want a user register in your application, you have to define which informations he need to fill in. for example, he need to provide a chosen login, a password, a first name, a last name and so on... but all these informations are not always *required*; some of them can be *optional*.In the same way, these informations can be either *public* or*private*; they can be read from anyone (first name) or kept secret (credit card number). So, a `subjectTemplate` is not a user but a template for defining a user.

these requirements are configured in your datasource (XML or database), and used to build a `SubjectTemplate` object too.

5.2. validate the registration requirements

during the registration, a user submit the required information to your application; these informations are used to build a user candidate. this candidate is also expressed as a `SubjectTemplate.`

A `subjectTemplate` which define global requirements to register a user (must have for example a `login` and `password` and a `city` credential), is also used to validate other `subjectTemplate` which are user candidates.if `credentials` provided by the candidate user (expressed also as a`SubjectTemplate`), are not listed in the `subjectTemplate`, they will be ignored. if the credentials marked as`required`, are not provided, *the registration will fails*. The `SubjectTemplate` should also contains an '`Identity`' crendetial, which is a credential to uniquely identify a user(prevents identity stealth).

Note

this autovalidation mechanism is also used with java permissions with their '`implies`' method.

Chapter 6. Java authorization

6.1. Authorization Mechanism

6.1.1. description

Authorization part is involved to determine which `resources` will be accessible from an authenticated user. jGuard uses an Access Control model which involves the use of `roles` `domains` and `permissions` concepts.
Roles are present in authentication and authorization parts, and represent the unique link between these two parts.

To configure authorization of your web application, you have to :

- defines your application resources

  you can divide your application into resources by different ways, depending on your architecture style and environment.

  - you want to prevent a user to execute a `privileged code`

    this privileged code can be in the local JVM or in another one (via RMI [http://en.wikipedia.org/wiki/Java_Remote_Method_Invocation] ,JINI, or Web service with SOAP, for example); your resources will be Java methods, i.e you want to prevent unauthorized users to execute some specific methods. This approach is`Java code centric`, and often used with standalone applications.

    For distributed applications, the RPC [http://en.wikipedia.org/wiki/Remote_procedure_call] architecture style aims to calls remote methods. This style tries to achieve call transparency by hiding the `remote` characteristic of the infrastructure. So, with RPC architecture like Java'sRMI, client code thinks call code locally but remotely.

  - you want to prevent a user to use a resource

    this way is often used in web applications, by representing a resource with anURL. This architecture style is often known as the RESTarchitecture style. So, you have to identify which URLs you want to protect (often all of them).

- map your resources to permissions

  Permissions are a `security view` of your resources: it represents which actions you can do on your resources. So, you can map one resource to a permission, or multiple resources to a permission. You can protect your application with any `java.security.permission` subclasses, either permissions provided by JDK or jGuard, or custom permissions subclasses implemented by you; jGuard will persist and use any of them.

  which `java.security.permission` subclass use for the architecture style I've chosen?

  - to secure a privileged method, you can use any Business permission code by your own. For example, you can define a `com.mycompany.security.AccountPermission` subclass which will have a `name` (of the account) and `actions` (`getTotal`, `balance` and so on...).

    - to secure resource, jGuard provides a permission implementation based on URL: `URLPermission`.

- defines the roles of your web application

  you need to define which roles will use your application. a role is a functional entity used to describe the activity of a set of person which will use your application.

  > **Note**
  >
  > a good practice is to define some roles `local` to your application. Sometimes, users want to define roles with a scope `global` to an enterprise : this work can be time-consuming and unproductive. So, we think you should be aware before trying to achieve this role unification across multiple applications.

- map roles to permissions

java.security.Policy,principal,permission....

the default implementation provided delegates all security check to the current `java.security.Policy` implementation.

6.1.2. configuration

Authorization configuration in jGuard, is done via the `jGuardAuthorization.xml` file.

goals of this configuration file is to:

- define the authorization scope

```
<scope>local</scope>
```

- enable/disable permission resolution caching

```
<permissionResolutionCaching>true</permissionResolutionCaching>
```

  permission resolution expressions (used in contextual permisisons) mechanism can be sometimes time-consuming. to speed-up this mechanism, jGuard proposes to cache resolved expressions, via the cache library ehcache. [http://www.ehcache.sf.net]
  Sometimes, you can have some issues with caching mechanism used by jGuard, if you use other libraries which already use ehcache like Hibernate [http://www.hibernate.org] for example. To confirm this hypothesis, you have to inactivate cache mechanism and see if it solves your problem (performances can be lower, but not in a dramatic way).Ehcache need to be configured only one time. to fix this issue, you can enable caching mechanism in jGuard and include in the ehcache master configuration file this configuration part (cache names need to be followed strictly, but other parameters can be tweaked):

```
<cache name = "unresolvedPermToNeededExpressions"
                                maxElementsInMemory="10000"
                                eternal="false"
                                overflowToDisk="false"
                                timeToIdleSeconds="120"
                                timeToLiveSeconds="120"
                                diskPersistent="false"
                                diskExpiryThreadIntervalSeconds="120"
```

```
                                    />

                                    <cache name = "unresolvedPermAndValuesToResolvedPerm"
                                    maxElementsInMemory="10000"
                                    eternal="false"
                                    overflowToDisk="false"
                                    timeToIdleSeconds="120"
                                    timeToLiveSeconds="120"
                                    diskPersistent="false"
                                    diskExpiryThreadIntervalSeconds="120"
                                    />
```

- define the AuthorizationManager implementation and its options

```
.........

 <authorizationManager>net.sf.jguard.ext.authorization.manager.XmlAuthorizationManager</
authorizationManager>
                                    <authorizationManagerOptions>
                                    .....
                                    </authorizationManagerOptions>
                                    .........
```

6.2. AuthorizationManager

6.2.1. description

AuthorizationManager implementations are dedicated to the web applications developer.

6.2.2. implementations

6.2.2.1. XMLAuthorizationManager

6.2.2.1.1. description

this `AuthorizationManager` implementation permits an XML file authorization method.

6.2.2.1.2. parameters

- fileLocation

  This parameter must be placed in the AccessFilter parameters list, in the web.xml file. The location must begin with the "file://" prefix. You should also use the "`${java.home}`" and the ${/} variables to have a more flexible configuration.

  Example 6.1. fileLocation parameter example

```
file://C:/jGuardPrincipalsPermissions.xml
```

6.2.2.2. JdbcAuthorizationManager

6.2.2.2.1. description

JdbcAuthorizationManager tries to connect to the needed tables. If no one are found, it creates them automatically. Also, it queries the tables to found data. If not data are found, it insert into them data from the XML backend present in the same directory.

Note

since 0.70 release, all database-based `AuthorizationManager` have been removed and replaced by `JdbcAuthorizationManager`.

6.2.2.2.2. parameters

- ApplicationName
- fileLocation

  location of the properties file corresponding to the database. This properties file contains SQL queries related to your database. The Oracle properties one, is valid for oracle 9i and higher.

  ### Example 6.2. fileLocation example

  ```
  file:///pathToMyConfigurationFile.properties
  ```

- authorizationUrl

  a JDBC compliant URL used to connect directly to the database.

  ### Example 6.3. authorizationUrl example

  ```
  jdbc:oracle:thin:@sweetHome.net:1521:dbName
  ```

- authorizationLogin

  login used to connect to the database with the authorizationUrl.

- authorizationPassword

  password used to connect to the database with the authorizationUrl.

- JNDI

  JNDI name used to grab the datasource. This way is an alternative to the authorizationUrl.

  ### Note

  the best way in a web application use case, is to use the Datasource provided by the application server via JNDI, instead of connecting directly with JDBC parameters to the database. If your are in a web application use case, you should use the JNDI parameter instead of the authorizationUrl, authorizationLogin and authorizationPassword parameters.

6.2.2.2.3. configuration with driver parameters

```
<configuration>
                            <authorization>
                            <!-- 'local' or 'jvm' -->
                            <scope>local</scope>
                            <permissionResolutionCaching>true</
permissionResolutionCaching>
```

```
 <authorizationManager>net.sf.jguard.authorization.JdbcAuthorizationManager</
authorizationManager>
                            <authorizationManagerOptions>
                            <option>
                            <name>databaseDriver</name>
                            <value>com.mysql.jdbc.Driver</value>
                            </option>
                            <option>
                            <name>databaseDriverUrl</name>
                            <value>jdbc:mysql://10.0.0.10/jguardexample</value>
                            </option>
                            <option>
                            <name>databaseDriverLogin</name>
                            <value>jguard</value>
                            </option>
                            <option>
                            <name>databaseDriverPassword</name>
                            <value>jguard</value>
                            </option>
                            <option>
                            <name>authorizationXmlFileLocation</name>
                            <value>WEB-INF/conf/jGuard/jGuardUsersPrincipals.xml</value>
                            </option>
                            <option>
                            <name>authorizationDatabaseFileLocation</name>
                            <value>WEB-INF/conf/jGuard/authorization.mysql.properties</
value>
                            </option>
                            </authorizationManagerOptions>
                            </authorization>
                            </configuration>
```

6.2.2.2.4. configuration with JNDI Datasource

```
....
                            ....

 <authorizationManager>net.sf.jguard.authorization.JdbcAuthorizationManager</
authorizationManager>
                            <authorizationManagerOptions>
                            <option>
                            <name>JNDI</name>
                            <value>java:/comp/env/jguard</value>
                            </option>
                            </authorizationManagerOptions>
                            ....
                            ....
```

6.2.2.2.5. DataModel

Tables and Fields

- tables
  - jg_permission
  - jg_domain
  - jg_principal_domain
  - jg_principal_permisison

    - jg_app_principal
    - jg_principal_hierarchy
- Table: jg_permission
    - id

      the permission´s id.
    - class

      java.security.Permission subclass used to build this permission.
    - name

      name of this permission
    - actions

      parameters of this permission
    - domain_id

      domain which owns this permission
- Table: jg_domain
    - id

      the domain´s id.
    - name

      entities in a functional way this group of permissions.
- Table: jg_principal_domain
    - domain_id

      link to the domain by its id
    - principal_id

      link to the principal by its id
- Table: jg_principal_permisison
    - permission_id

      link to the permission by its id
    - principal_id

      link to the principal by its id
- Table: jg_app_principal

- id

  id of principal

- name

  name of the principal

- Table: jg_principal_hierarchy
  - principal_asc_id

    id of one of its principal ascendants (in an RBAC meaning)

  - principal_desc_id

    id of one of its principal descendants (in an RBAC meaning)

6.2.2.2.6. ER Diagram

Figure 6.1. authorization ER diagram

6.3. Permissions

6.3.1. description

6.3.2. URLPermission

> **Note**
>
> This permission is one possible way to protect webapp resources, but is NOT the only one. You can represent webapp resources by other java.security.permission subclasses, and handle them with jGuard too. You can also mix URLPermission with other ones to protect your webapp.

6.3.2.1. Description

this permission represents the right to access through an URL to a resource on a webapp.

6.3.2.2. Build an URLPermission

URLPermission has got two constructors:

- one single argument constructor required by the Abstract Permission class [http://java.sun.com/j2se/1.4.2/docs/api/java/security/Permission.html]

  ```
  public URLPermission(String name)
  ```

  this constructor cannot be used 'alone'. You should use the method setActions too to complete the object.

- one two arguments constructor

this constructor should be preferred, because its constructs a full URLPermission in one shot.

```
public URLPermission(String name,String actions)
```

the name parameter permits to add to the newly created permission, a custom name to remind it easily. The actions parameter is a string which contains a list of actions separated by ','(this constructor is required by the BasicPermission abstract class [http://java.sun.com/j2se/1.4.2/docs/api/java/security/BasicPermission.html]). Here are the corresponding actions:

- uri
- scheme or protocol (optional, but required if description is present)
- HTTP method (optional), among `DELETE`,`GET`,`HEAD`, `OPTIONS`,`POST`,`PUT`, `TRACE`, or`ANY`(all methods are authorized).If not method is defined, `ANY` is automatically set.
- description (optional)

6.3.2.3. Usage

To use an`URLPermission`, basically you deal with two methods:`implies()`and `equals()`

- `implies()`

  When you create an URLPermission, its URI can be, for example, in the form `http://someurl.domain` or `/someurl.do`. However, you probably want to use GET parameters on that URLs, like `http://someurl.domain?param1=value1&param2=value2`. Here is the "trick" of`URLPermission`. When you define a base URL for a permission, any permission derived from it will be implied. If you have access to the base URL`http://someurl.domain`, certainly you must have access to the derived `http://someurl.domain?param1=value1`. The signature of implies() is:

  ```
  boolean implies(Permission p)
  ```

  Let's call basePerm the base URLPermission, and derivedPerm the derived one. Using the URLs presented before, if you execute:

  ```
  basePerm.implies(derivedPerm)
  ```

  It will return true. In another example, if you have an URLPermission called perm1, with the URI http://webapp/someurl.do, and another called perm2, with the URI http://webapp/anotherurl.do:

  ```
  perm1.implies(perm2)
  ```

  Will return false, since perm2 cannot be derived from perm1

- `equals()`

  URLPermission has an own implementation of equals(), that tests if a given URL is equals to the present one. To be equals, the URL must have its name and URL (including parameters) with the same values of the permission being compared. For example, if you define 2 URLs as following:

  ```
  URLPermission perm_1 = new
  ```

```
                                URLPermission("url_1","http://someurl.domain/path1?
param1=a&param2=b");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path1?param1=a&param2=b");
```

Then:

```
perm_1.equals(perm_2)
```

Will return false, because perm_1 has a different name of perm_2 (url_1 != url_2). Note that parameters order doesn't affect the equals mechanism on jGuard.

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path1?
param1=a&param2=b");

                                URLPermission perm_2 = new
 URLPermission("url_1","http://someurl.domain/path1?param2=b&param1=a");
```

Then:

```
perm_1.equals(perm_2) return true
```

- using the star operator

  when you define URLPermissions in your web applications, you can think that this work is tedious: on big webapps, you can have to create many URLPermissions. a trick to reduce the number of URLPermissions is to use the star operator ,which implies all the URI with the same starting sequence and any characters placed after the last character before the star.

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path1*");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path1234");

                                perm1.implies(perm2) return true
```

  and:

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/pat*h1");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path99999h1");

                                perm1.implies(perm2) return true
```

Important

it is important to give a good attention on URL naming.

- URL parameters

If the URL permission is defined with a URI with no query part, the permission implies any permission with parameters.

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path?param1=a&param2=b");

                                perm1.implies(perm2) return true
```

If the URI of the URLPermission contains a query part, the permission implies only the permissions having the exact same parameters

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path?
param1=a");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path?param1=a&param2=b");

                                perm1.implies(perm2) return false
```

In order to allow at least the permission having the good parameter defined but any values for any other parameter, use &* at the end of the query part of the URL

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path?
param1=a&*");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path?param1=a&param2=b&param3=c");

                                perm1.implies(perm2) return true
```

Stars can also be used in parameter names or values definition just as in path

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path?
param1=a&param2=*&pa*3=c");

                                URLPermission perm_2 = new
 URLPermission("url_2","http://someurl.domain/path?param1=a&param2=b&param3=c");

                                perm1.implies(perm2) return true
```

- and what's about star symbol in our URL?

  URL can contains the star , without any signification. So, to include it in your URL, you have to double your star.

  ```
  URLPermission perm_1 = new
                                  URLPermission("url_1","http://someurl.domain/path1*");
  ```

  this URL will be used by jGuard like a regexp character.

```
URLPermission perm_1 = new
                                URLPermission("url_1","http://someurl.domain/path1**");
```

but this URL won't be used by jGuard like a regexp character, and will be evaluated like a URL with only one star symbol.

6.3.2.4. what's about URLPermission and my web application?

the star operator will not have some impact on the web framework you use (i.e Struts or another one). `AccessFilter` handle all the HTTP user requests, and handle any trick on star characters. So, you can use any star character in your URLs without problems outside jGuard configuration.

6.3.3. Contextual permissions

since1.0 release, jGuard supports '`contextual`' permissions. It implies that you can refers in any permissions (subclass of `java.security.Permission`), to some context variables like user credentials and roles. Like RBAC design impose that user and permissions mustn't bound statically,these variables are resolved dynamically. When the user will try to enforce a permission, jGuard will grab all permissions of the user,including contextual permissions, and will resolve variables of these permissions. Thus, jGuard will compare the permission enforced with the resolved ones.

6.3.3.1. syntax

jGuard uses the Jakarta commons JEXL project [http://jakarta.apache.org/commons/jexl/] expression library to provide expressiveness on contextual variables. This project has got a syntax page: http://jakarta.apache.org/commons/jexl/reference/syntax.html [http://jakarta.apache.org/commons/jexl/reference/syntax.html]

6.3.3.2. examples

- permission refers to credentials

  public credentials are referenced with

  ```
                                  ${subject.publicCredentials}
  ```

  private credentials are referenced with

  ```
  ${subject.privateCredentials}
  ```

  ```
  Permission p1 = new
                                  URLPermission("index","http://www.website.com/
  index.html?name=${subject.publicCredentials.name}");

                                  Permission p2 = new
                                  URLPermission("index","http://www.website.com/
  index.html?name=${subject.privateCredentials.country}");

                                  Permission p3 = new
                                  FilePermission("file://home/user/
  ${subject.publicCredentials.company}", "read");
  ```

- permission refers to roles

private credentials are referenced with

```
${subject.roles}
```

- permission refers to organization

  organization credentials are referenced with

  ```
  ${subject.organization.credentials}
  ```

  organization principals are referenced with

  ```
  ${subject.organization.principals}
  ```

6.3.4. how to create its own permission

6.3.4.1. what is a permission?

a Permission is a class which represents an access to one or multiple resources. This class is used by access Control mechanism to protect your resources. Access control can be done in multiple points. But you should control access homogeneously.

6.3.4.2. do i need to create a custom permission?

jGuard can handle in its authorization system, any Permission subclasses. It provides also a convenient Permission to represents access to URL: `URLPermission` .

so, if your resources can only be accessed by HTTP(through a navigator for example), you should use the jGuard `URLPermission`. Otherwise,you have to create your own Permission implementation, or use some permissions implementations provided by the Java j2se platform like :

- Permissions available on Java 1.4.2 and higher
  - AllPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/security/AllPermission.html]
  - AudioPermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/sound/sampled/AudioPermission.html]
  - AuthPermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/AuthPermission.html]
  - AWTPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/awt/AWTPermission.html]
  - BasicPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/security/BasicPermission.html]
  - DelegationPermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/kerberos/DelegationPermission.html]
  - LoggingPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/LoggingPermission.html]
  - FilePermission [http://java.sun.com/j2se/1.4.2/docs/api/java/io/FilePermission.html]
  - NetPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/net/NetPermission.html]

- PrivateCredentialPermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/PrivateCredentialPermission.html]
- PropertyPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/util/PropertyPermission.html]
- ReflectPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/lang/reflect/ReflectPermission.html]
- RuntimePermission [http://java.sun.com/j2se/1.4.2/docs/api/java/lang/RuntimePermission.html]
- SQLPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/sql/SQLPermission.html]
- SSLPermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/net/ssl/SSLPermission.html]
- SecurityPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/security/SecurityPermission.html]
- SerializablePermission [http://java.sun.com/j2se/1.4.2/docs/api/java/io/SerializablePermission.html]
- ServicePermission [http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/kerberos/ServicePermission.html]
- SocketPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/net/SocketPermission.html]
- UnresolvedPermission [http://java.sun.com/j2se/1.4.2/docs/api/java/security/UnresolvedPermission.html]

- Permissions available on Java 1.5.0 and higher
  - ManagementPermission [http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/ManagementPermission.html]
  - MBeanPermission [http://java.sun.com/j2se/1.5.0/docs/api/index.html]
  - MBeanServerPermission [http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanServerPermission.html]
  - MBeanTrustPermission [http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanTrustPermission.html]
  - SubjectDelegationPermission [http://java.sun.com/j2se/1.5.0/docs/api/javax/management/remote/SubjectDelegationPermission.html]
- Permissions available on Java 1.6.0 and higher
  - WebServicePermission [http://java.sun.com/javase/6/docs/api/javax/xml/ws/WebServicePermission.html]

6.3.4.3. Resources and Permissions relationships

In most cases,Access/use of resources, are "translated" in Java by instantiation of objects, or execution of methods : indirectly,it's these constructors/methods that you want to protect. In Java (i.e in the Java

platform or in custom code),Resource protect "themselves" against misuse: the corresponding method involved, create the related permission which will decide if access should be granted. If your resource is designed by the Class Resource, and like no provided permissions are suitable to protect your resource, you will have too create a `ResourcePermission`. Here is an example of the Resource class:

```
public class Ressource{

                        private String ressourceName = null;

                        public Ressource(String name)throws SecurityException{
                        this.ressourceName = name;
                        AccessController.checkPermission(new
 RessourcePermission(this.ressourceName,"create"));
                        }

                        public void inactivateRessource(Collection coll)throws
 SecurityException{
                        AccessController.checkPermission(new
 RessourcePermission(this.ressourceName,"inactivate"));
                        .....
                        //inactivation code executed only if the user which calls this
 code has got the rights
                        permissions
                        .....
                        .....
                        }

                        public void updateRessource(Collection coll)throws
 SecurityException{
                        AccessController.checkPermission(new
 RessourcePermission(this.ressourceName,"update"));
                        .....
                        //update code executed only if the user which calls this code has
 got the rights permissions
                        .....
                        .....
                        }

                        }
```

the 'magic' code

```
AccessController.checkPermission(new RessourcePermission(...));
```

will invoke a JAAS mechanism to automatically check if the user has got the right permission; otherwise, it will throw a SecurityException. Access Control is delegated to the ResourcePermission. A protected resource can be any kind of resource, like a physical resource (a file), but although a LOGICAL resource like a Debt, a pizzaService(??).... Note that `AccessController.checkPermission(....)`will always check access, also when the securityManager is not set. When the default securityManager is set, it delegates all the permission checks to AccessController. But if a custom SecurityManager is set, if you call the `checkPermission` from the SecurityManager, you are not sure that the AccessController class will be called(it's the choice of the SecurityManager custom implementation). So,for backward compatibility, and to enable access control only when the securityManager is set, you can replace

```
AccessController.checkPermission
```

by:

```
SecurityManager sManager= Security.getSecurityManager();
                        if(sManager != null)
                        sManager.checkPermission(new Ressourcepermission(...));
                        }
```

Note

if you don't need an overall global Security, jGuard provides a `local` mode. So, the security is not at the JVM level (the `JVM` mode), but only at the classLoader level. i.e, only code loaded by the classloader is handled by another AccessController implementation from jGuard (formerly `localAccessController`). This `local` mode securize your webapp if your application server is configured in the official jee way (`parent last` classloader loading strategy). So, to be sure to handle a security check from the right accessController in use, you have to use the `AccessCOntrollerUtils` class.

```
AccessControllerUtils.checkPermission(Subject
                              yourSubject,Permission yourPermission);
```

6.3.4.4. implement your own permission: some rules to follow

if you need to protect a resource(not mapped by any Permission implementations described on top of this page), you have to create your own permission. To do it, you have to extends Permission. Permissions are often created by the resources to protect. So, this resource will give to the Permission the needed informations to 'qualify' the permission, to know what's the caller thread want to do. Permission qualifies actions needed on the Resource, and decides if the needed permission is compatible with the Permissions owned by the calling thread.

Caution

one good security practice is to declare `final` your new Permission class, to avoid anyone to subclass it and change its mechanism.

6.3.4.4.1. Permission abstract class

here are the methods you need to implements.

- `Permission(String name)`

  Permission [http://java.sun.com/j2se/1.4.2/docs/api/java/security/Permission.html] abstract class to extend,implies a Constructor with a name to identify the Permission.

- `boolean equals(Object obj)`

  this method must be implemented in your subclass. Its goal is to test equality of Permissions of the same type, and NOT implies another permission; a dedicated implies method do that.

- `String getActions()`

  this method must be implemented in your subclass. If you use an additional 'actions' parameter, you will return it; otherwise, return an empty string("").In our example, the class Resource give to the RessourcePermission a name and one action String which can be "create","inactivate" or "update".

- `int hashCode()`

this method must be implemented in your subclass.

- `boolean implies(Permission permission)`

  this method is used when JAAS checks that the calling thread (in most cases the user),has got one permission or more which implies this one. So, all your custom code to know if the permission of the caller implies this one should be placed here. This method is the main difference between multiple Permission implementations.

6.3.4.4.2. BasicPermission

it exists a subclass of Permission calledBasicPermission [???], which provides a convenient mechanism to guards some simple Resources. It provides also one more constructor with two String(name and actions).

6.3.5. Negative permissions

6.3.5.1. positive permissions mechanism

When a user or a library tries to access to a resource, Java enforces a check against a permission specified by the resource : the resource calls the `checkPermission` method of the `AccessController` class with the chosen permission.

```
Permission myPermission = new
                        MyPermission(permissionName,permissionActions);
                        AccessController.checkPermission(myPermission);
```

Note that the resource can call the checkPermission method of the SecurityManager , which will delegate check to AccessController if it is set. So, the only difference is that AccessController.checkPermission will always do the check, although if the SecurityManager flag is not set. The SecurityManager will do the check only if it is set, and will delegate it to AccessController.

```
Permission myPermission = new
                        MyPermission(permissionName,permissionActions);
                        SecurityManager securityManager = System.getSecurityManager();
                        if (securityManager != null) {
                        securityManager.checkPermission(myPermission);
                        }
```

this call verifies that the user / library contains one or more permissions implied by this permission. If that's true, access is granted. One permission is necessary to grant access, although if all others are not implied by the permission which guard the resource.

> **Note**
>
> by default, jGuard use positive permissions

6.3.5.2. negative permissions mechanism

negative permissions is set in the web application, by including in you web.xml file this parameter:

```
<context-param>
                        <param-name>negativePermissions</param-name>
```

```
                        <param-value>true</param-value>
                        </context-param>
```

this mechanism grant access if no permission implies the checked permission . If one or more permission imply the requested permission, access will NOT be granted. Access will be blocked. So, if one permission(or more) of the user (or library) implies the checked permission, it acts as aveto [http://en.wikipedia.org/wiki/Veto].

6.3.5.3. mixing positive and negative permissions

this mechanism is not yet available. This mechanism can be useful in some cases, but maybe implies a bigger complexity in managing your application. This mechanism will be added if some users ask it.

Chapter 7. Which Access Control model is the best solution to manage security?

jGuard uses an ABAC access control model.

Here is a list of the different Access Control Models:

7.1. Discretionary Access Control (DAC)

the main disadvantages of this model are:

- loss of flexibility
- security is discretionary, not central

according to the Computer Security resource center of the NIST(US),~~ DAC is used to control access by restricting a subject's access to an object. It is generally used to limit a user's access to a file.

n this type of access control it is the owner of the file who controls other users' accesses to the file.~~ This type of Access Control is generally used in UNIX systems. This mechanism can be used with the help of Access Control Lists (ACL), which allows assignment of permissions to users which are part of the group of the owner, or other users. This access control is not very flexible for medium or large organization which have many users. This mechanism has the following issues: If the user quits the organization, many security operations are needed, to assign the same authorization to the new user; and this system assumes that the user is not malicious, because the user owns the resource, and can do whatever he wants with it, including granting access to to everyone. It can be a big security hole. This mechanism is not suitable for an advanced security system. To reflect these conclusions,and to prove that UNIX systems are not all bound to this old mechanism,you can point to:

- the Solaris RBAC system [http://www.samag.com/documents/s=7667/sam0213c/0213c.htm] introduced since Solaris 8 (Solaris is a UNIX system shipped by SUN™)
- the gr-security [http://www.grsecurity.net/] security system for Linux which also add RBAC features to the Linux kernel

7.2. Mandatory Access Control (MAC)

In Mandatory Access Control models,Subjects(users) receive a clearance label and objects receive a classification label, also referred to as security levels. No users can do operations on objects that are not permitted by the administrator which has configured the system. This system remove the discretionary aspect of the DAC model, to centrally control operations on objects made by users. But this system has got the disadvantage to not be flexible: access rights are defined for each users;this mechanism implies many administrative operations, when by example a user replace another one to a function in the organization.

7.3. Role Based Access Control (RBAC)

many informations comes from this RBAC draft [http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf] provided by the NIST.

RBAC features are divided into 4 components:

- Core RBAC

  this part is sub-divided into 6 parts:

  - relations between users, roles, permissions

    the main principle of RBAC, is that users are assigned to roles(principals),permissions are assigned to roles, and acquires permissions by being members of roles. RBAC has go a great flexibility: user-role and permission-role relations are many-to-many, i.e a user can have many roles, and a role can be assigned to many users; and a permission an be assigned to many roles, and a role can have many permissions.

  - administrative functions

    RBAC implies for administration tasks, to provide the way to know which users has got a specific role, and which roles has got a user. The same administrative function can be done between roles and permissions.

  - user sessions

    core RBAC includes the concept of user sessions, which permit activation and deactivation of roles owned by users.

  - user and multiple roles

    a user (which owns multiples roles) can be able to simultaneously exercise permissions of multiple roles. Some security products handle multiples roles or groups in their mechanism, but cannot exercise permissions of multiples roles at the same time!

  - centrally administering security

  - principle of least privilege

- hierarchical RBAC

  RBAC recognises two types of hierarchies:

  - general hierarchy with support of multiple inheritance

  - limited hierarchy without support of multiple inheritance

- Static separation of duty (SSOD)

  This principle permits to avoid conflict of interest. It consists of constraints added to the user-role assignment, which prevents some roles to be added to users which have got some others roles.The standard only specifies constraints on roles, but it can be useful to put constraint on permissions, or operations on protected resources.

- Dynamic separation of duty (DSOD)

  Dynamic separation of Duty is an extension because it implies a separation of duty across the user's session.

7.4. Attribute Based Access Control (ABAC)

ABAC permits to resolve access control decisions on role, and permissions like in the RBAC model, but also on user attributes. Since jGuard 1.0.0, we permit to include in permissions (see contextual permissions) and roles(see dynamic role definition) some variables referencing user attributes. So, ABAC model stands on RBAC model, and enhance its flexibility.

Chapter 8. Glossary

permission	permission is the right to execute an action on a resource, like access to a page, update an account...
principal	role
subject	user
organization	entity which grnats the user to operate on the application
domain	permission's set used to regroup some features of your application (CRUD operations for Invoice and so on...)
RBAC	Role Based Access Control
ABAC	Attribute Based Access Control
DAC	Discretionary Access Control
MAC	Mandatory Access Control
credential	security attribute owned by the user: personal information like name, password, social security number

Chapter 9. jGuard audit

9.1.

Chapter 10. jGuard and standalone applications

jguard-swing-example™ has to be run with the `SecurityManager` activated, unlike jguard-struts-example™ that can run with or without it. jguard-swing-example is a very basic demonstration of what jGuard™ can do to secure standalone applications. It simply tries to read a file with the rights granted to the connected user. To use jGuard, we must configure the authentication part and the authorization part. The authentication part do not use full jGuard implementation. We keep the Sun™ Configuration implementation to define the loginModules. We will use a jGuard loginModule to set the principals through a XML file. On the other hand, the authorization part is full jGuard.

10.1. Java SecurityManager

to activate the securityManager, you have either:

- to launch your standalone application with this parameter:

```
-Djava.security.manager
```

- or set a java.lang.SecurityManager programmatically:

```
//we test if there is a SecurityManager in place
                        SecurityManager sm = System.getSecurityManager();
                        if (sm == null){
                        System.setSecurityManager(new SecurityManager());
                        }
```

10.2. configure Java security

10.2.1. `java.security`

`java.security` file is the master security configuration file; all security settings (authentication and authorization) stands on it. It is located in `${java.home}/lib/security/` . To install jGuard, you have to:

- verify that `login.configuration.provider` (the `javax.security.auth.login.Configuration` implementation in place) is set to `com.sun.security.auth.login.ConfigFile`, i.e the default implementation shipped with the JVM.
- reference the relatedSUN™'s authentication configuration file :

  modify the property `login.config.url.1` and set it to `login.config.url.1=file:///jguard-swing-example/conf/java.login.config`
- reference jGuard Authorization mechanism:

  modify the property `policy.provider` and set it to `policy.provider=net.sf.jguard.ext.authorization.policy.classic.SingleAppPo`

10.2.2. configure theSUN™'s authentication configuration file

- create a java.login.config file containing the following entries :

```
your_application_name{
                              net.sf.jguard.authentication.loginmodules.XmlLoginModule
 REQUIRED
                              fileLocation="path_to/jGuardUsersPrincipals.xml"
 debug=true ;
                              };
```

Caution

Pay attention to the position of the semicolons. Go to `javax.security.auth.login.Configuration` Javadoc [http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/login/Configuration.html] for more information about configuration file format.

- create the file `JGuardUsersPrincipals.xml` you've defined in fileLocation options in `java.login.config`

Example 10.1. JGuardUsersPrincipals.xml example

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
                                    <!DOCTYPE usersPrincipals SYSTEM
 "jGuardUsersPrincipals_1.00.dtd">
                                    <usersPrincipals>
                                    <principals>
                                    <principal>
                                    <name>admin</name>

 <class>net.sf.jguard.core.principals.RolePrincipal</class>
                                    <applicationName>jGuardSwingExample</
applicationName>
                                    </principal>
                                    <principal>
                                    <name>guest</name>

 <class>net.sf.jguard.core.principals.RolePrincipal</class>
                                    <applicationName>jGuardSwingExample</
applicationName>
                                    </principal>
                                    </principals>
                                    <users>
                                    <userTemplate>
                                    <name>default</name>
                                    <privateRequiredCredentials>
                                    <credTemplateId>login</credTemplateId>
                                    <credTemplateId>password</credTemplateId>
                                    </privateRequiredCredentials>
                                    <publicRequiredCredentials>
                                    <credTemplateId>firstname</credTemplateId>
                                    <credTemplateId>lastname</credTemplateId>
                                    <credTemplateId>location</credTemplateId>
                                    </publicRequiredCredentials>
                                    <privateOptionalCredentials>
                                    <credTemplateId>country</credTemplateId>
                                    <credTemplateId>religion</credTemplateId>
                                    </privateOptionalCredentials>
                                    <publicOptionalCredentials>
                                    <credTemplateId>hobbies</credTemplateId>
                                    </publicOptionalCredentials>
                                    <genericPrincipals>
                                    </genericPrincipals>
                                    <specificPrincipalFactories>
                                    </specificPrincipalFactories>
                                    </userTemplate>
                                    <user>
                                    <privateCredentials>
                                    <credential>
                                    <id>login</id>
                                    <value>admin</value>
                                    </credential>
                                    <credential>
                                    <id>password</id>
                                    <value>admin</value>
                                    </credential>
                                    </privateCredentials>

                                    <publicCredentials>
                                    <credential>
                                    <id>firstname</id>
                                    <value>Rick</value>
                                    </credential>
                                    <credential>
                                    <id>lastname</id>
                                    <value>Dangerous</value>
                                    </credential>
                                    <credential>
                                    <id>location</id>
                                    <value>Paris</value>
                                    </credential>
```

10.2.3. configure Java authorization

this configuration file is specific to the default Policy mechanism provided bySUN™, and followed by all JVM vendors. java.policy is located in `${java.home}/lib/security/` . If you use the default Policy, you have to include for the jguard-swing-example the following permissions to the `java.policy` file:

```
grant codeBase "file:/<eclipse-workspace>/jguard-swing-example/eclipse-bin/-" {

                    permission java.util.PropertyPermission "log4j.defaultInitOverride",
 "read";
                    permission java.util.PropertyPermission "log4j.configuration",
 "read";
                    permission java.util.PropertyPermission "log4j.configuratorClass",
 "read";
                    permission java.util.PropertyPermission "log4j.ignoreTCL", "read";
                    permission java.util.PropertyPermission "log4j.debug", "read";
                    permission java.util.PropertyPermission "log4j.configDebug", "read";
                    permission java.util.PropertyPermission
 "javax.xml.parsers.DocumentBuilderFactory", "read";
                    permission java.util.PropertyPermission "user.dir", "read";

                    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
                    permission java.awt.AWTPermission "accessEventQueue";
                    permission java.awt.AWTPermission "accessClipboard";

                    permission java.io.FilePermission "<M2_REPO>/log4j/log4j/1.2.12/
log4j-1.2.12.jar", "read";

                    permission javax.security.auth.AuthPermission
 "createLoginContext.jGuardSwingExample";
                    permission javax.security.auth.PrivateCredentialPermission
                    "net.sf.jguard.core.authentication.credentials.JGuardCredential
                    net.sf.jguard.core.principals.RolePrincipal "*"","read";
                    permission javax.security.auth.AuthPermission "modifyPrincipals";
                    permission javax.security.auth.AuthPermission
 "modifyPrivateCredentials";
                    permission javax.security.auth.AuthPermission
 "modifyPublicCredentials";

                    permission java.lang.RuntimePermission "modifyThreadGroup";
                    };

                    grant codeBase "file:/<eclipse-workspace>/jguard-ext/eclipse-bin/-" {

                    permission java.util.PropertyPermission
 "net.sf.jguard.application.name", "read";
                    permission java.util.PropertyPermission
 "com.sun.management.jmxremote.login.config", "read";
                    permission java.util.PropertyPermission "org.dom4j.factory", "read";
                    permission java.util.PropertyPermission
 "org.dom4j.DocumentFactory.singleton.strategy", "read";
                    permission java.util.PropertyPermission "org.saxpath.driver", "read";
                    permission java.util.PropertyPermission
 "org.dom4j.QName.singleton.strategy", "read";
                    permission java.util.PropertyPermission
 "org.dom4j.QName.singleton.strategy", "read";

                    permission java.io.FilePermission "\\confjGuardUsersPrincipals.xml",
 "read";
                    permission java.io.FilePermission "<eclipse-workspace>/jguard-swing-
example/conf/jGuardUsersPrincipals.xml",
                    "read";
                    permission java.io.FilePermission "<eclipse-workspace>/jguard-swing-
example/conf/jGuardUsersPrincipals_1.00.dtd",
```

```
                    "read";

                    permission javax.security.auth.PrivateCredentialPermission
                    "net.sf.jguard.core.authentication.credentials.JGuardCredential
                    net.sf.jguard.core.principals.RolePrincipal "*"","read";
                    permission javax.security.auth.AuthPermission "modifyPrincipals";
                    permission javax.security.auth.AuthPermission
 "modifyPrivateCredentials";
                    permission javax.security.auth.AuthPermission
 "modifyPublicCredentials";
                    };

                    grant codeBase "
                    file:/<eclipse-workspace>/jguard-core/eclipse-bin/-" {

                    permission javax.security.auth.PrivateCredentialPermission
                    "net.sf.jguard.core.authentication.credentials.JGuardCredential
                    net.sf.jguard.core.principals.RolePrincipal "*"","read";
                    permission javax.security.auth.AuthPermission "modifyPrincipals";
                    permission javax.security.auth.AuthPermission
 "modifyPrivateCredentials";
                    permission javax.security.auth.AuthPermission
 "modifyPublicCredentials";
                    };

                    grant codebase "file:/<M2_REPO>/-" {

                    permission java.util.PropertyPermission
 "org.apache.xerces.xni.parser.XMLParserConfiguration",
                    "read";
                    permission java.util.PropertyPermission "java.home", "read";
                    permission java.util.PropertyPermission
 "org.dom4j.QName.singleton.strategy", "read";

                    permission java.io.FilePermission "<JDK_HOME>/jre/lib/
xerces.properties", "read";
                    permission java.io.FilePermission "<eclipse-workspace>/jguard-swing-
example/eclipse-bin/log4j.xml",
                    "read";
                    permission java.io.FilePermission "<JDK_HOME>/jre/lib/
xerces.properties", "read";
                    permission java.io.FilePermission "<M2_REPO>/log4j/log4j/1.2.12/
log4j-1.2.12.jar", "read";
                    permission java.io.FilePermission "\\confjGuardUsersPrincipals.xml",
 "read";
                    };
```

replace <eclipse-workspace>,<M2_REPO> , <JDK_HOME> by your own values.

10.2.4. Configure `java.login.config`

- open`java.login.config`located in `/jguard-swing-example/conf/`
- modify `authenticationXmlFileLocation` property as `//jguard-swing-example/conf/jGuardUsersPrincipals.xml`

10.2.5. Configure `jGuardPrincipalsPermissions.xml`

- open jGuardPrincipalsPermissions.xml located in /jguard-swing-exampleconf/
- modify the FilePermission in the full domain to point to an existing file

10.2.6. Create a new run configuration on jguard-swing-example

Add as a JVM argument the following (replace M2_REPO and eclipse-worskpace)

```
Xbootclasspath/a:<M2_REPO>/jguard/jguard-core/1.0.3/jguard-core-1.0.3.jar
                    -Xbootclasspath/a:<M2_REPO>/jguard/jguard-ext/1.0.3/jguard-
ext-1.0.3.jar
                    -Xbootclasspath/a:<M2_REPO>/log4j/log4j/1.2.12/log4j-1.2.12.jar
                    -Xbootclasspath/a:<M2_REPO>/dom4j/dom4j/1.6.1/dom4j-1.6.1.jar
                    -Xbootclasspath/a:<M2_REPO>/jaxen/jaxen/1.1-beta-6/jaxen-1.1-
beta-6.jar
                    -Xbootclasspath/a:<M2_REPO>/commons-lang/commons-lang/2.1/commons-
lang-2.1.jar
                    -Xbootclasspath/a:<M2_REPO>/ehcache/ehcache/1.1/ehcache-1.1.jar
                    -Xbootclasspath/a:<M2_REPO>/commons-jexl/commons-jexl/1.0/commons-
jexl-1.0.jar
                    -Djava.security.manager
                    -Dnet.sf.jguard.policy.configuration.file=<eclipse-workspace>/jguard-
swing-example/conf/jGuardPolicy.xml
                    -Djava.security.debug=access:failure
```

10.2.7. Run

- log with`admin/admin`, try to read the file you set in`jGuardPrincipalsPermissions.xml`: It succeeds.
- log with`guest/guest`, try to read the file you set in`jGuardPrincipalsPermissions.xml`: It fails.

Chapter 11. jGuard on JEE applications

11.1. required libraries

- Java 5 (also known as 5.O) or higher
- JEE 4(also known as j2ee 1.4) or higher
- jguard-core.jar
- jguard-ext.jar
- jguard-jee.jar

11.2. integrate jGuard in your Struts™ web application

11.2.1. configuration files

11.2.1.1. configure the *technology anchor*

the technology anchor used in a Struts web aplication is a `javax.servlet.Filter` implementation called `net.sf.jguard.jee.authentication.http.AccessFilter.`

To restrict all protected resources, declaration must be located at the top of the servlet filters list in `web.xml`. The related filter-mapping should map all of the Struts protected resources. You do *NOT* have to include the following in the protected resources set: images, CSS, javascript files, and so on.....

```
<filter>
                        <filter-name>AccessFilter</filter-name>
                        <description>access filter</description>
                        <filter-
class>net.sf.jguard.jee.authentication.http.AccessFilter</filter-class>
                        <init-param>
                        <param-name>configurationLocation</param-name>
                        <param-value>/WEB-INF/conf/jGuard/jGuardFilter.xml</param-value>
                        </init-param>
                        </filter>
                        .....
                        .....
                        <filter-mapping>
                        <filter-name>AccessFilter</filter-name>
                        <url-pattern>*.do</url-pattern>
                        </filter-mapping>
```

`AccessFilter` configuration in the `web.xml` file requires to specify the location of the `jGuardFilter.xml` file like in any technology anchor.

11.2.1.2. configure authentication and authorization parts

in a web environment,very frequently the java security is not set via a SecurityManager.so, jGuard needs to be installed smoothly and configured at webapp startup. this is done with a dedicated `javax.servlet.ServletContextListener` implmeentation called `net.sf.jguard.listeners.ContextListener`. This implementation will reference the technology-agnostic configuration files called `jGuardAuthentication.xml` and `jGuardAuthorization.xml`.

Example 11.1. configure jGuard initialization at web application startup

```
........
                            ........
                            <context-param>
                            <param-name>authenticationConfigurationLocation</param-name>
                            <param-value>/WEB-INF/conf/jGuard/jGuardAuthentication.xml</
param-value>
                            </context-param>
                            <context-param>
                            <param-name>authorizationConfigurationLocation</param-name>
                            <param-value>/WEB-INF/conf/jGuard/jGuardAuthorization.xml</
param-value>
                            </context-param>
                            ........
                            ........
                            <listener>
                            <listener-class>net.sf.jguard.listeners.ContextListener</
listener-class>
                            </listener>
                            ........
                            ........
```

11.2.1.3. configure `authenticationBindingsFactory` and `AuthorizationBindings` implementations

If no one are defined, `net.sf.jguard.jee.authentication.http.HttpServletauthenticationBindingsFact` and `net.sf.jguard.jee.authorization.HttpServletAuthorizationBindings` are used. Those are specific to any HttpServlet-based framexork like Struts.

to define the implementations used, you have to declare them in the init parameters of the technology ancors. for Struts, you can do it in the web.xml like this:

```
........
                        ........
                        <context-param>
                        <param-name>authenticationBindingsFactory</param-name>
                        <param-
value>net.sf.jguard.jee.authentication.http.HttpServletauthenticationBindingsFactory</
param-value>
                        </context-param>
                        <context-param>
                        <param-name>authorizationBindings</param-name>
                        <param-
value>net.sf.jguard.jee.authorization.HttpServletAuthorizationBindings</param-value>
                        </context-param>
                        ........
                        ........
                        ........
```

11.2.1.4. configure specific technology settings

to cleanup session related informations when session expires, we configure an HttpSessionListener implementation called net.sf.jguard.listeners.SessionListener.

Example 11.2. configure the SessionListener

```
........
                            ........
                            <listener>
                            <listener-class>net.sf.jguard.listeners.SessionListener</
listener-class>
                            </listener>
                            ........
                            ........
```

11.2.2. example provided with the jGuard distribution

11.3. integrate jGuard in your JSF web application

11.3.1. configuration files

11.3.1.1. configure the *technology anchor*

to use JSF, you have to decalre some settings in the web.xml file, including declaring the `javax.faces.webapp.FacesServlet` servlet like this:

```
.....
                          .....
                          <servlet>
                          <servlet-name>Faces Servlet</servlet-name>
                          <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
                          <load-on-startup>1</load-on-startup>
                          </servlet>
                          .....
                          .....
```

the technology anchor used in a JSF web aplication is a `javax.faces.event.PhaseListener` implementation called `net.sf.jguard.jsf.AccessListener.`

JSF configuration is located in a`faces-config.xml` file. In this file, you can declare the `AccessListener` like this:

```
<faces-config>
                          <lifecycle>
                          <phase-listener>net.sf.jguard.jsf.AccessListener</phase-listener>
                          </lifecycle>
                          .....
                          .....
```

11.3.1.2. prevent access to internal JSF resources

a direct access to resource likejsp, xhtml files (if you use facelets™ [https://facelets.dev.java.net/]) or any other view rendering technology for your sensible resources need to be prevented, and pass through the `AccessListener`. To prevent, this, jGuard™ provide a special servlet filter which redirect requests against somes unprotected extensions like *.jsp* , to a protected extension like *.faces*. You can declare it in your web.xml file like this:

```
 <filter>
```

```
                        <!-- prevent direct access to *.jsp by redirecting to .jsf -->
                        <filter-name>redirectFilter</filter-name>
                        <filter-class>net.sf.jguard.jsf.RedirectFilter</filter-class>
                        <init-param>
                        <param-name>targetExtension</param-name>
                        <param-value>.jsf</param-value>
                        </init-param>
                        </filter>
                        <filter-mapping>
                        <filter-name>redirectFilter</filter-name>
                        <url-pattern>*.jsp</url-pattern>
                        </filter-mapping>
                        ........
                        ........
```

11.3.1.3. configure authentication and authorization parts

cf section 2.1.2

11.3.1.4. configure `authenticationBindingsFactory` and `AuthorizationBindings` implementations

JSF relies internally on the HttpServlet. it can configure authenticationBindingsFactory and AuthorizationBindings implementations in a web.xml like this:

```
........
                        ........
                        <context-param>
                        <param-name>authenticationBindingsFactory</param-name>
                        <param-
value>net.sf.jguard.jsf.authentication.JSFauthenticationBindingsFactory</param-value>
                        </context-param>
                        <context-param>
                        <param-name>authorizationBindings</param-name>
                        <param-
value>net.sf.jguard.jsf.authorization.JSFAuthorizationBinding</param-value>
                        </context-param>
                        ........
                        ........
                        ........
```

if no one are defined, `net.sf.jguard.jsf.authentication.JSFauthenticationBindingsFactory` and `net.sf.jguard.jsf.authorization.JSFAuthorizationBinding`s are used by default.

11.3.2. example provided with the jGuard distribution

if no one are defined, `net.sf.jguard.jsf.authentication.JSFauthenticationBindingsFactory` and `net.sf.jguard.jsf.authorization.JSFAuthorizationBinding`s are used by default.

Warning

some special permissions are automatically granted to any users, like the configured *AccessDeniedPermission,AuthenticationFailedPermission*,

authenticationSucceedPermission which maps to a `JSFPermission` (ending in our exampole by *.jsf*). But JSF has got a behavior which implies that a resource (ending for example by jsf), is an abstract name for a concrete resource like a jsp. Mapping between abstract and concrete resource is done in the faces-config.xml configuraiton file. but this mapping can be done:

- by forwarding the call to the jsp (or another view technology) *only at the server-side*
- by redirecting the call to the client pointing to the view resources directly, *with a client-side and a server-side interaction*

  this last option implies to grant access in the guest role to some view resources like the jsp `accessDenied.jsp` if you use this view technology). this requirement has been applied in the jGuardPrincipalPermissions.xml file from the jguard-jsf-example webapp, but is not needed in other examples (automatically granted).

11.4. integrate in your DWR web application

11.5. jGuard can manage multiple webapps

[edit] jGuard can manage multiple webapps in the same servlet container. It isolates authentication and authorization parts of each webapps.

Warning

beware that some application servers, for performance tweaking purpose, do not set by default the classloader mechanism to the one defined in the JEE specification; they Sometimes (like jBoss™ 3.x, Websphere™ 5 and 6.0, but not jBoss™ 4.x) set it to the j2se one, and implies that webapps are not isolated: it can enhance performances, but can also create some security and stability issues.

11.5.1. Authentication isolation

jGuard's AccessFilter registers automatically authentication part of your web application by its name. So, two webapps with the same name in two servlet containers, but backed with the same database, will create a conflict(those two webapps will edit common data). It is planned to permit to tweak this mechanism by providing a chosen identifier for authentication to be unique, or to share authentication part across multiple webapps.

11.5.2. Authorization isolation

in Java, policy is set globally for the JVM usually. In a JEE application server, multiple applications are hosted on it, in the same JVM (we don't talk about application server clustering). So, like policy is very important in the Java architecture, some bad interactions can occurs between multiple web applications.

if you set the `SecurityManager` and prevent some webapps with related permissions to not override the jGuard policy set, all is fine. But frequently, security manager is not set, and multiple applications need to be securized through JAAS .

so, since a long time, jGuard ships a `MultipleAppPolicy`, which can handle multiple applications(each web application, has got an isolated part of the policy). Each web application set this MultipleAppPolicy if not present as the JVM policy, and register the web application to have an isolated part of it. This can be set at runtime. Another 'classic' way is to define the policy as a Java parameter when you launch your applications server, but it is tedious if you don't have a very high security requirement.

jGuard's AccessFilter registers automatically authorization part of your web application by its classloader, which identify uniquely the web application, according to JEE specifications.

it is planned to permit to tweak this mechanism by providing a chosen identifier for authorization to be unique, or to share authorization part across multiple webapps.

Sometimes, some other products relies on different policy implementations and need to replace jGuard policy; also, one requirement to share the policy between multiple web applications (with MultipleAppPolicy), is to put some jars in the 'shared libraries directory' of your application server: some developers can think it is either tedious, or not feasible in their IT environment.

jGuard provides a policy (with the '`local`' scope), specific for each web application, which does not interact with a global policy set on the JVM; other webapps, can use their own policy (in their 'local' mode),or use the 'jvm' scope, and cannot reach this policy: this policy is 'local' to the webapp which has set it. this local mode also , does not require any 'shared library directory' mechanism.

11.6. quick start guide

This procedure allows quick installation of jGuard™ and testing of the `jGuardExample.war` archive provided in the distribution. The `jGuardExample` is configured to work with the XmlLoginModule, `XmlAuthenticationManager` and the`XmlAuthorizationManager`.

11.6.1. requirements

- java 4 (also known as 1.4) or higher
- jee 3(also known as j2ee 1.3)/servlet 2.3 or higher

11.6.2. steps

- Add the jguard-struts-example.war to your application server (under your `${CATALINA_HOME}/webapps` directory for TOMCAT™ or `${JBOSS_HOME}/server/Default/deploy` forjBoss™)
- Start your application server
- test jGuard with the provided example, at `http://127.0.0.1:8080/jguard-struts-example`

Note

There is no requirement by default (the default scope is `'local'`) to put some jGuard™ archives in the "shared libraries" directory of your application server. there is also no requirement to edit any java.policy file. All the security is embedded in your webapp.

11.7. taglibs: Integrate jGuard in your jsp pages

Add`standard.jar`, and `jstl.jar` in the lib directory and add them to the path.

> **Note**
>
> All the jguard tags support Expression Language (EL).

11.7.1. jguard:authorized

The tag is used to protect page fragments:

```
<jguard:authorized uri="/myApplication/SwitchToModule.do?prefix=/forum&page=/
ForumPanorama.do">
                    this text appears only if you are authorized.
                    </jguard:authorized>
```

The text will be rendered by the jsp, only if your user, has at least one role with permission to access the uri `/myApplication/SwitchToModule.do?prefix=/forum&page=/ForumPanorama.do` It is important to note that a fragment is protected with a uri, and not a role. If you update the role permissions, you have not to change the jsp!!!

11.7.2. jguard:hasPermission

The tag , displays content only if the user has the permission defined with this tag. it supports any java.security.Permission subclass. the tag is a specialized version of this tag, dedicated to URLPermission.

```
<jguard:hasPermission className="java.io.FilePermission" name="/home/myDirectory"
                    actions="read">
                    content displayed only if the user has got the specified permission
                    </jguard:hasPermission>
```

11.7.3. jguard:hasPrincipal

The tag , displays content only if the user has a role called like the 'principals' value.

```
<jguard:hasPrincipal principals="admin">
                    hello!! you have got a role called admin
                    </jguard:hasPrincipal>
```

'principals' attribute can have 1 or more roles splitted by a comma. 'operator' attribute permits to set how the tag works with 'roles':

- if `'NONE'` is set, tag will display contents only if authenticated user hasn't got any roles in the specified roleSet.
- if `'ANY'` is set, tag will display contents only if authenticated user has one or more roles in the specified roleSet.
- if `'ALL'` is set, tag will display contents only if authenticated user has all roles in the specified roleSet.
- when the 'operator' attribute is not specified, 'ANY' is used by default.

```
<jguard:hasPrincipal principals="admin,restricted" operator="NONE">
                                hello!! you have got neither a role called admin nor
 restricted
                                </jguard:hasPrincipal>
```

Note

If you change the name of the role, you will update the jsp code, in opposite of the first tag . So, the tag should be used mostly to protect html links, and should be used to protect 'functional' content.

11.7.4. jguard:pubCredential

The tag, display the public credential of a user identified by an "id". If the authenticated user hasn't got this credential, the tag display the default value if the webapp developer use this feature(optional); otherwise, the tag render "".

```
<jguard:pubCredential id="name" default="noName" />
```

11.7.5. jguard:privCredential

The tag, display the private credential of a user identified by an 'id'. If the authenticated user hasn't got this credential, the tag display the default value if the webapp developer use this feature(optional); Otherwise, the tag render "".

```
<jguard:privCredential id="login" default="default" />
```

11.8. Integrate jGuard with Servlets and other "web" classes (Struts Actions, etc.)

11.8.1. Getting the Subject Object

```
AuthenticationUtils auth=

 (AuthenticationUtils)request.getSession(true).getAttribute(CoreConstants.AUTHN_UTILS);
                   Subject subject = auth.getSubject();
```

11.8.2. Getting the 'identity' credential

the 'identity' credential is the credential(i.e user attribute) which identify uniquely the user.

```
//get the Subject
                   AuthenticationUtils authUtils =

 (AuthenticationUtils)request.getSession().getAttribute(HttpConstants.AUTH_UTILS);
                   Subject subject = authUtils.getSubject();

                   //get the SubjectTemplate
                   ServletContext context = request.getSession().getServletContext();
                   AuthenticationManager

 am=(AuthenticationManager).context.getAttribute(CoreConstants.AUTHENTICATION_MANAGER);
                   SubjectTemplate defaultTemplate = am.getDefaultSubjectTemplate();
```

```
                    JGuardCredential identityCredential =
SubjectUtils.getIdentityCredential(subject,defaultTemplate);
```

11.9. smooth integration with j2ee security methods

since the 0.70 release, jGuard integrates smoothly with libraries that use "j2ee security methods" present in the HttpServletRequest class. this integration is realized through the use of a HttpServletRequestWrapper, which is transparent for the jGuard user.

11.9.1. `String getRemoteUser()`

this method return the identifier of the user.jGuard is looking for the 'login' credential, firstly in the public credential set, and if not found, in the private credential set. if there is no credential called 'login', or if the user hasn't got the permission to grab the private credential 'login', this method return null.

```
public class MyDispatchAction extends DispatchAction{

                    public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request,
                    HttpServletResponse response){
                    String remoteUser= request.getRemoteUser(); System.out.println("user
login is = "+remoteUser);
                    }
                    }
```

11.9.2. `Principal getUserPrincipal()`

this method returns a special jGuard™ `Principal` implementation : `SubjectAsPrincipal`. when you use the `getName()` method of this special `Principal`, it returns a String from a credential called`'name'`, either in the public or private credential set. if no credential is found, it returns null. this special Principal permits to grab the Subject object of the user (you can grab it too in the`HttpSession`), with its `getSubject()` method.

```
public class MyDispatchAction extends DispatchAction{

                    public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request,
                    HttpServletResponse response) {
                    Principal principal = request.getUserPrincipal(); String name=
principal.getName();
                    System.out.println("user name is"+name);
                    SubjectAsPrincipal sap = (SubjectAsPrincipal)principal; Subject
subject = sap.getSubject();
                    }
                    }
```

11.9.3. `boolean isUserInRole(String role)`

his methods return `true` if the user has got one principal with the related name; otherwise, it returns `false`.

```
public class MyDispatchAction extends DispatchAction{

                    public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request,
```

```
                    HttpServletResponse response) { boolean admin=
request.isUserInRole("admin");
                    System.out.println("user is an admin = "+admin);
                    }
                    }
```

Chapter 12. securing DWR™ with jGuard™

jGuard 1.0.0 and higher support securization of webapps using DWR1.x. we plan to support also DWR 2.x

12.1. install DWR™ in the webapp

in a classic way,to install DWR, you have to insert in your `web.xml` file, a DWR servlet :

Example 12.1. installing DWR in your webapp without jGuard securization

```
   <servlet>
                    <servlet-name>dwr-invoker</servlet-name>
                    <display-name>DWR Servlet</display-name>
                    <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
                    <init-param>
                    <param-name>debug</param-name>
                    <param-value>true</param-value>
                    </init-param>
                    <servlet-mapping>
                    <servlet-name>dwr-invoker</servlet-name>
                    <url-pattern>/dwr/*</url-pattern>
                    </servlet-mapping>
```

12.2. `DWR.xml`

DWR permits to access directly to beans hosted on the server in the webapp. central configuration file is`DWR.xml`. for example, if you want to permit access to the bean `net.sf.jguard.example.dwr.Dummy`, you have to configure it in `DWR.xml` like this:

Example 12.2. `dwr.xml` configuration example

```
<!DOCTYPE dwr PUBLIC
                    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
                    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">

                    <dwr>
                    <allow>
                    <create creator="new" javascript="Demo">
                    <param name="class" value="net.sf.jguard.example.dwr.Dummy"/>
                    </create>
                    </allow>
                    </dwr>
```

12.3. DWR1Permission : a dedicated Permission

jguard 1.0.0 and higher ships a Permission dedicated to DWR 1.x. this permission has got a name and some parameters, like any subclass of `java.security.BasicPermisison`:

- name

  used to put on the permission functional meaning

- parameters

  - first parameter

class of the Creator used to instantiate the related protected beans.

example: `uk.ltd.getahead.dwr.create.NewCreator`

- second parameter: the class of the bean to protect

  example: `net.sf.jguard.example.dwr.Dummy`

- third parameter : the method to protect. example: `getHello`

Example 12.3. part of `jGuardPrincipalPermission.xml`

```
<permission>
                                        <name>dummy</name>

 <class>net.sf.jguard.jee.extras.dwr1.DWR1Permission</class>
                                        <actions>

 <action>uk.ltd.getahead.dwr.create.NewCreator</action>
                                        <action>net.sf.jguard.example.dwr.Dummy</
action>
                                        <action>getHello</action>
                                        </actions>
                                        </permission>
```

12.4. DWR1AccessControl

now, we need to link access to Dummy bean via DWR with jGuard. to do that, you have to insert one more parameter of the DWR servlet configured previously like this:

Example 12.4. web.xml with a special init parameter `uk.ltd.getahead.dwr.AccessControl`

```
   <servlet>
                    <servlet-name>dwr-invoker</servlet-name>
                    <display-name>DWR Servlet</display-name>
                    <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
                    <init-param>
                    <param-name>debug</param-name>
                    <param-value>true</param-value>
                    </init-param>
                    <init-param>
                        <param-name>uk.ltd.getahead.dwr.AccessControl</param-name>
                        <param-value>net.sf.jguard.jee.extras.dwr1.DWR1AccessControl</
param-value>
                        </init-param>
                    </servlet>
                    <servlet-mapping>
                    <servlet-name>dwr-invoker</servlet-name>
                    <url-pattern>/dwr/*</url-pattern>
                    </servlet-mapping>
```

12.5. what's about jGuard and DWR interactions?

you have to notice that jGuard is linked with the `DWR1AccessControl`. it is used to delegate to jGuard authorization check before the user access via a javascript instruction to the java Bean

declared in the `DWR.xml` file. but you have to configure jGuard to authenticate the user. to do that, accessFilter has to be used. so, `AccessFilter` and its mapped URIs(like all struts actions .do) will be used for Authentication, and authorization checks with your traditional web framework(for example Struts). DWR will be used for ajax interactions, and will delegate authorization check to jGuard. so,in an application hosting Struts and DWR, authentication will be done in a URI ending by *.do, and authorization checks will be done in uri ending by .do and containing the DWR pattern (see servlet mappings configured above).

Chapter 13. JMX security with jGuard™

13.1. requirements

JMX is a great technology shipped with java 5 standard edition and higher.

JMX is also shipped with j2ee 1.4 application servers, to expose some of their components.

13.2. what jGuard propose to enhance JMX security

- unified Security

  jGuard propose to control JMX remote access with the same underlying mechanism than local java access, for a better flexibility and security.

- unified authentication

  JMX authentication with jGuard only support login/pasword authentication challenge. other JMX authentications methods like CLIENT-CERT are planned.

- unified authorization

  Access control is only restricted if you activate the `Security Manager`. This restriction is put by java implementation and not jGuard.if you don't set the Security Manager, authenticated users will have access to ALL `MBeans`. if you set the`Security Manager`, you will have a fine-grained control on `MBeans` exposed. these `MBeans` are protected with `MBeanPermission` . you will have read operation on them, restrict changes and so on... these `MBeanPermissions` need to be registered in roles like any other permissions declared in jGuard. only users with roles containing `MBeanPermissions` will have access to them. this powerful feature is unique, and only provided by jGuard.

13.3. How to start a JMXServer secured with JGuard

here is an example how to protect your newly created MBeanServer:

Example 13.1. MBeanServer securization example

```
//create the MBeanServer
                    MBeanServer mbs =
 MBeanServerFactory.createMBeanServer(applicationName);
                    //create connector's options
                    Map opt=new HashMap();
                    opt.put(JMXConnectorServer.AUTHENTICATOR,new
 JGuardJMXAuthenticator());
                    //create JMXConnector
                    JMXConnectorServer
 connectorServer=JMXConnectorServerFactory.newJMXConnectorServer(url,opt,mbs);
                    connectorServer.start();
```

13.4. How to start your standalone application with an MBeanServer protected with jGuard

The needed property can be separated into 3 parts :bootclasspath properties, JAAS and JMX related properties

- The `-Xbootclasspath` properties define which libraries must be loaded by the bootclasspath. We need to give the JGuard core library and every other libraries used by JGuard during `SingleAppPolicy` and `XmlLoginModule` execution.

```
-Xbootclasspath/a:jguard-core-1.0.3.jar
                        -Xbootclasspath/a:commons-logging_1.1.0.jar
                        -Xbootclasspath/a:commons-lang-2.1.jar
                        -Xbootclasspath/a:dom4j-1.6.1.jar
                        -Xbootclasspath/a:jaxen-1.1-beta-6.jar
```

- JAAS and JMX related properties

```
// activate java security
                        -Djava.security.manager
                        -Dcom.sun.management.jmxremote.ssl=false
                        -Dcom.sun.management.jmxremote.port=9004
                        -Dcom.sun.management.jmxremote.password=false
                        -
Dcom.sun.management.jmxremote.login.config=your_application_name
```

13.5. how to reach the JMX connector Server

you can reach this connector securized by jguard at this url:

```
service:jmx:rmi://localhost/jndi/rmi://rmiRegistryHost:rmiRegistryPort/applicationName
```

13.6. Secure JMX connections with JGuard in standalone applications

- enable the JMX agent for local access

  you have to set this system parameter when you start JVM:

```
-Dcom.sun.management.jmxremote
```

- enable monitoring and management from remote systems

  you have to set this system parameter:

```
-Dcom.sun.management.jmxremote.port=portNum
```

- password protection for JMX access

  password protection is enabled by default. you can define it explicitely with this system parameter:

```
-Dcom.sun.management.jmxremote.authenticate=true
```

- activate JMX on a Windows™ operating system

Caution

JMX will not run on Windows™ operatiing system, on disk partitions formatted in FAT32 format. It will only work if the partition you use is formatted in NTFS format.

13.7. securing JMX remote access in webapps with jGuard

13.7.1. activate JMX security in your webapp

in your web.xml file, you have to insert this code:

```
<context-param>
                    <param-name>enableJMX</param-name>
                    <param-value>true</param-value>
                    </context-param>
```

13.7.2. optional jGuard JMX-related parameters

- MBeanServer for connector

  jguard guard access to MBeans through a connector. you can define which `MBeanServer` will interact with the connector via the key `mbeanServerForConnector`.

  if no one is defined or if the value is `new`, it will create a MBeanServer.

```
<context-param>
                            <param-name>mbeanServerForConnector</param-name>
                            <param-value>new</param-value>
                            </context-param>
```

  if the value is `position#N` ,it will use the `MBeanServer` in the Nth position in the MBeanServer list returned by the `MBeanServerFactory`.

```
<context-param>
                            <param-name>mbeanServerForConnector</param-name>
                            <param-value>position#4</param-value>
                            </context-param>
```

  if the value is MBeanServerName#N,it will use the MbeanServer in the Nth position among the MBeanServer list which has got this name returned by the MBeanServerFactory.

```
<context-param>
                            <param-name>mbeanServerForConnector</param-name>
                            <param-value>myMBeanServerName#0</param-value>
                            </context-param>
```

- RMI Registry Host

  you can define a custom RMI registry host via the key rmiRegistryHost in your web.xml file. if not specified, the default value is localhost.

```
<context-param>
                            <param-name>rmiRegistryHost</param-name>
                            <param-value>192.168.0.5</param-value>
                            </context-param>
```

- RMI registry Port

  you can define a custom RMI registry port via the key rmiRegistryPort in your web.xml file. if not specified, the default value is 9005.

```
<context-param>
                            <param-name>rmiRegistryPort</param-name>
                            <param-value>9016</param-value>
                            </context-param>
```

Chapter 14. jGuard and Object-Relational Mapping tools

14.1. Overview

since several years, the use of Object-Relational Mapping frameworks is growing. one of the more popular one isHibernate [http://www.hibernate.org]. ORMs framework provides many useful features. they act as a layer between the object world, and the Database world. the facade of the Database world is the JDBC Driver. Jguard cannot use hibernate for its purpose, because jGuard persists some JDK security classes which does not fullfills Hibernate requirements (empty constructor is one of them). so, jGuard use the 'classical' way to persist its objects, with direct access to database through JDBC Driver or JNDI Datasource. but you can use jGuard transparently with any ORM including Hibernate!

14.2. Using jGuard with Hibernate transparently

Hibernate and jGuard create connections with Driver parameters, or Datasource grabbed by JNDI. so, they can act on the same database, to manipulate related informations. but how to communicate each others? the common scope of Hibernate and jGuard is the object scope: informations persisted by Hibernate and informations persisted by jGuard should communicate only at the object scope, and not on the SQL scope. jGuard manage users, principals/roles,domains, permissions, which are linked each others. these objects are very "isolated" in a functional way. so, the most common class which will linked to others classes(object navigation), is the user class. remind that CRUD operations on users, roles, permissions are dedicated to AuthenticationManager and AuthorizationManager implementations.

14.3. example

if your webapp persists through Hibernate a class called Enterprise, which contains some users(persisted by jGuard) contained in a Collection, you want to navigate from Enterprise class to users. the Enterprise Class contains getter for these users. this class is mapped with Hibernate through an Enterprise.hbm.xml, which contains fields persisted by Hibernate: don't declare the users field in this configuration file. you will have only to implement the getUsers method like this:

```
public Collection getUsers() throws AuthenticationManagerException{
                return
 AuthenticationManagerFactory.getAuthenticationManager().getUsers();
                }
```

Chapter 15. Import /Export your security data

15.1. import authentication data

we try to import another AuthenticationManager in the current one.this other AuthenticationManager can be backed by an XML file(in this example) or a database.

```
AuthenticationManager otherAuthNManager = new XMLAuthenticationManager();

                Map options = new HashMap();
 options.put(CoreConstants.APPLICATION_NAME,"myAppName");
                options.put(SecurityConstants.AUTHENTICATION_XML_FILE_LOCATION,"/home/
user/myfile.xml");

                otherAuthNManager.init(options);

                //retrieve the current AuthenticationManager
                AuthenticationManager myAuthNManager =
 AuthenticationManagerFactory.getAuthenticationManager();

                //import in the current AuthenticationManager some data
                myAuthNManager.importAuthenticationManager(otherAuthNManager);
```

15.2. export authentication data

several methods exist in net.sf.jguard.ext.authentication.manager.AuthenticationUtils class to export data in XML way.

15.2.1. export your AuthenticationManager as an XMLAuthenticationManager

any AuthenticationManager implementation can be exported as an XMLAuthenticationManager:

```
AuthenticationManager myAuthNManager;
                    XMLAuthenticationManager myXmlAUthNManager =
                    AuthenticationUtils.exportAsXmlAuthorizationManager(myAuthNManager);
```

15.2.2. export your AuthenticationManager as an XML string

```
AuthenticationManager myAuthNManager;
                    String myXmlData =
 AuthenticationUtils.exportAsXMLString(myAuthNManager);
```

15.2.3. export your AuthenticationManager as an XML File

```
AuthenticationManager myAuthNManager;
                    String fileLocation="/home/data/myFile.xml";
                    AuthenticationUtils.exportAsXMLFile(myAuthNManager,fileLocation);
```

15.2.4. export your AuthorizationManager as an HTML form in a stream

```
AuthenticationManager myAuthNManager;
```

```
                    OutputStream myOutputStream =
 yourHttpServletResponse.getOutputStream();
                    AuthenticationUtils.writeAsHTML(myAuthNManager,myOutputStream);
```

15.2.5. export your AuthenticationManager as an XML form in a stream

```
AuthenticationManager myAuthNManager;
                    OutputStream myOutputStream =
 yourHttpServletResponse.getOutputStream();
                    String encodingScheme = yourHttpServletResponse.getEncodingScheme();

 AuthenticationUtils.writeAsXML(myAuthNManager,myOutputStream,encodingScheme);
```

15.3. import authorization data

we try to import another AuthorizationManager in the current one. this other AuthorizationManager can be backed by an XML file(in this example) or a database.

```
AuthorizationManager otherAuthZManager = new XMLAuthorizationManager();

                Map options = new HashMap();
 options.put(CoreConstants.APPLICATION_NAME,"myAppName");
                options.put(SecurityConstants.AUTHORIZATION_XML_FILE_LOCATION,"/home/
user/myfile.xml");

                otherAuthZManager.init(options);

                //retrieve current AuthorizationManager
                AuthorizationManager myAuthNManager =
 AuthorizationManagerFactory.getAuthorizationManager();

                //import in the current AuthenticationManager some data
                myAuthNManager.importAuthenticationManager(otherAuthNManager);
```

15.4. export authorization data

several methods exist in net.sf.jguard.ext.authorization.manager.AuthorizationUtils class to export data in XML way.

15.4.1. export your AuthorizationManager as an XMLAuthorizationManager

```
Authorizationmanager myAuthZManager;
                    XMLAuthorizationManager myXmlAUthZManager =
                    AuthorizationUtils.exportAsXmlAuthorizationManager(myAuthZManager);
```

15.4.2. export your AuthorizationManager as an XML string

```
Authorizationmanager myAuthZManager;
                    String myXmlData =
 AuthorizationUtils.exportAsXMLString(myAuthZManager);
```

15.4.3. export your AuthorizationManager as an XML File

```
Authorizationmanager myAuthZManager;
```

```
                    String fileLocation="/home/data/myFile.xml";
                    AuthorizationUtils.exportAsXMLFile(myAuthZManager,fileLocation);
```

15.4.4. export your AuthorizationManager as an HTML form in a stream

```
Authorizationmanager myAuthZManager;
                    OutputStream myOutputStream =
 yourHttpServletResponse.getOutputStream();
                    AuthorizationUtils.writeAsHTML(myAuthZManager,myOutputStream);
```

15.4.5. export your AuthorizationManager as an XML form in a stream

```
Authorizationmanager myAuthZManager;
                    OutputStream myOutputStream =
 yourHttpServletResponse.getOutputStream();
                    String encodingScheme = yourHttpServletResponse.getEncodingScheme();

 AuthorizationUtils.writeAsXML(myAuthZManager,myOutputStream,encodingScheme);
```

Chapter 16. useful links

16.1. General Security

- general information about RBAC model [http://www.softpanorama.org/Authentication/rbac.shtml]
- Instance-level access control for business-to-business electronic commerce by R. Goodwin, S. F. Goh, and F. Y. Wu on IBM web site [http://www.research.ibm.com/journal/sj/412/goodwin.html]
- Architectural Patterns for Enabling Application Security by Joseph Yoder and Jeffrey Barcalow [http://www.joeyoder.com/papers/patterns/Security/appsec.pdf]
- Access Control (aka Authorization) in Your J2EE Application By Jeff Williams [http://www.aspectsecurity.com/article/access_control.html]
- courses on Cryptography from the University of Washington [http://www.cs.washington.edu/education/courses/csep590/06wi/lectures/]

16.2. Authentication

- RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication [http://www.ietf.org/rfc/rfc2617.txt?number=2617]
- RFC 2829 : authentication methods for LDAP [http://www.ietf.org/rfc/rfc2829.txt?number=2829]
- RFC 2797: Certificate Management Messages over CMS [http://www.zvon.org/tmRFC/RFC2797/Output/index.html]
- Authentication methods overview in http/https (french) by Franck Davy [http://www.hsc.fr/ressources/breves/http_digest.html.fr]
- Linux PAM FAQ [http://www.kernel.org/pub/linux/libs/pam/FAQ]
- PAM modules by Jennifer Vesperman on linux O' Reilly devcenter [http://www.linuxdevcenter.com/pub/a/linux/2001/10/05/PamModules.html]
- writing PAM modules, Part one by Jennifer Vesperman on linux O' Reilly devcenter [http://www.linuxdevcenter.com/pub/a/linux/2002/05/02/pam_modules.html]
- Linux PAM home Page [http://www.kernel.org/pub/linux/libs/pam/]
- Linux-PAM documentation [ttp://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/]
- Linux PAM modules available [http://www.us.kernel.org/pub/linux/libs/pam/modules.html]
- a Java Radius Client [http://jradius-client.sourceforge.net/]
- certificate profile example [http://www.cra-arc.gc.ca/eservices/pki/ext-cp-digsign-f.html#P1227_100009]
- Making Login Services Independent of Authentication Technologies by Vipin Samar, Charlie Lai [http://java.sun.com/security/jaas/doc/pam.html]
- JBoss linux authentication [http://www.mrchucho.net/index.php?p=36]

- RFC 2743:Generic Security Service Application Program Interface Version 2, Update 1 [http://www.faqs.org/rfcs/rfc2743.html]

16.3. Authorization

- Role Engineering and Generic RBAC Theory : explanation of Separation of Duty [http://www.softpanorama.org/Authentication/role_engineering.shtml]

16.4. Security in Java

- security code guidelines [http://java.sun.com/security/seccodeguide.html]
- java security architecture [http://www.skywayradio.com/tech/j2sdk141/security/spec/security-specTOC.fm.html]
- how to verify jar integrity [http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/HowToImplAJCEProvider.html#MutualAuth]
- Propagating Security context over JMS in WebLogic [http://jroller.com/page/maximdim/20050615#propagating_security_context_over_jms]
- JSR 160 Security [http://mx4j.sourceforge.net/docs/ch03s10.html]
- Introduction to Securing Web Applications with JBoss and LDAP [http://www.developer.com/security/article.php/3077421]
- Instance-level access control for business-to-business electronic commerce by R. Goodwin, S. F. Goh, and F. Y. Wu [http://www.research.ibm.com/journal/sj/412/goodwin.html]

16.5. JAAS related information

- JAAS official web site [http://java.sun.com/products/jaas/]
- JAAS FAQ [http://java.sun.com/security/jaas/faq.html]
- "All that JAAS :Scalable Java security with JAAS" article on JavaWorld [http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-jaas.html]
- "All that JAAS :Pluggable authentication and authorization services provide many key security benefits for Java applications" article on JavaPro [http://www.ftponline.com/javapro/2004_06/magazine/features/kjones/]
- "J2EE security: Container versus custom" on JavaWorld [ttp://www.javaworld.com/javaworld/jw-07-2004/jw-0726-security.html]
- "Using JAAS for Authorization and Authentication" by Dan Moore [http://www.mooreds.com/jaas.html]
- Open source login modules implementations by Andy Amstrong [http://free.tagish.net/jaas/doc.html]
- When "java.policy" Just Isn't Good Enough by Ted Neward [http://www.javageeks.com/Papers/JavaPolicy/index.html]

- Extending JAAS by Guosheng Huang [http://www.sys-con.com/story/?storyid=37699.]
- Extend JAAS for class instance-level authorization by Carlos A. Fonseca [http://www.ibm.com/developerworks/java/library/j-jaas/]
- JAAS related informations dedicated to AS/400 systems: configuration, specific JAAS implementation... [http://as400bks.rochester.ibm.com/pubs/html/as400/v4r5/ic2924/info/java/rzaha/jaasbase.htm]
- implementing Security using JAAS and Java GSS-API by Charlie Lai and Seema Malkani [http://java.sun.com/security/javaone/2003/2236-JAASJGSS.pdf]
- white paper intitled 'USER AUTHENTICATION AND AUTHORIZATION IN THE JAVA(TM) PLATFORM" by Charlie Lai, Li Gong, Larry Koved, Anthony Nadalin, and Roland Schemers [http://java.sun.com/security/jaas/doc/acsac.html]
- JAAS Developer's Guide by SUN [http://java.sun.com/security/jaas/doc/api.html]
- JAAS by Bruce A Rich,Java Security Lead IBM/Tivoli Systems [ftp://ftp.oreilly.com/pub/conference/java2001/Rich_Jaas.pdf]
- Http Callback classes in SAP library [http://help.sap.com/saphelp_webas630/helpdata/en/6c/5da94fdd594791bc0f6bfdfad4d36e/content.htm]
- JAAS-compliant authentication-provider using the shadow-password system of Linux. It might also work for other dialects of Unix. [http://www.bablokb.de/jaas/]
- Java & J2EE Conventions, Guidelines and Best Practices document:JAAS rules [https://jjguidelines.dev.java.net/book/html/apbs08.html]
- Using JAAS and SPNEGO/Kerberos to single sign-on from fat java clients [http://bofriis.dk/index.html?http://bofriis.dk/spnego/spnego_jaasclient.html]
- JAAS presentation [http://www.mywelt.net/?q=node/1585]
- all that JAAS from AlBlue's weblog [http://alblue.blogspot.com/2004/10/all-that-jaas.html]
- JAAS example on tomcat [http://www.kopz.org/public/documents/tomcat/jaasintomcat.html]
- Java authorization internals: A guided tour of the Java 2 platform and JAAS authorization architectures by Abhijit Belapurkar [http://www-106.ibm.com/developerworks/library/j-javaauth/?ca=dnt-518]
- Urban code presentations [http://www.urbancode.com/download/presentations/]
- Extending JAAS by Guosheng Huang [http://java.sys-con.com/read/37699.htm]
- Using JAAS in Java EE and SOA Environments by Denis Pilipchuk [http://today.java.net/pub/a/today/2006/09/14/using-jaas-in-ee-and-soa.html]

16.6. java topics which can be interesting for jGuard

- Classworking toolkit: ASM classworking [http://www-128.ibm.com/developerworks/java/library/j-cwt05125/index.html?ca=drs-]

- Programmatically Signing JAR Files by Raffi Krikorian [http://www.onjava.com/pub/a/onjava/2001/04/12/signing_jar.html?page=1]
- instrumentation example (java.lang.instrument package) [http://kumiki.c.u-tokyo.ac.jp/~ichiyama/cgi-bin/hiki/java.lang.instrument.html]

Chapter 17. FAQ

- How does work authentication in jGuard?

  jGuard authenticates users (with the help of JAAS), through a stack of LoginModules.

- How can I configure jGuard to authenticate against an LDAP directory?

  jGuard provides some convenient LoginModules, including a `JNDILoginModule` since jGuard™ 1.0 beta 2. So, the solution is either to use the loginModule provided in the jGuard distribution, or to use a LoginModule provided by sun directly with the Java Runtime Environment(JRE). to do it, you have only to declare in the 'loginmodules' field this one: `com.sun.security.auth.module.JndiLoginModule` note that this loginmodule connect to LDAP through the great abstraction layer calledJNDI. more details can be reached directly at the corresponding page It exists others LoginModule implementations which do the same stuff. the only requirement is only to implements the LoginModule interface.

- How can I configure jGuard to authenticate against a Kerberos system?

  you can configure jGuard to authenticate through a Kerberos system. the loginModule to use is the one provided by sun: `com.sun.security.auth.module.Krb5LoginModule`

- How can I configure jGuard to authenticate against the NT/Unix/Solaris host system?

  jGuard can authenticate with any provided LoginModules implementations. here [http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/package-summary.html] are the one provided by Sun.

- How can I add database support to the authorization system?

  Since the 0.70 release, you have to use the `JdbcAuthorizationManager` and configure it through `jGuardConfiguration.xml` file to set the right driver and jdbc settings in order to and provide authorization. `yourdatabaseName.properties` file contains specific sql queries.

- `AccessFilter` automatically tries to log me in as 'guest'.Why should there be a "default" user in jGuard™? Isn't that a security issue?

  jGuard automatically authenticate you as 'guest' by default. it's not a security issue, but a design choice.But to fulfills your security requirements, you can configure that guest (unauthorized users), hasn't got access to your protected pages. how to do it? => configure the 'guest' role with no permissions. the guest user will only have access to login page and access denied page(access is always grant to these pages).

- Can i create a permission not bound to a Domain?

  “I didn't want to associate a domain to the permission because this permission is alone in a functional point of view. ” All permission must belong to a domain. To solve your problem, it is suitable to create a '*default*' domain which will regroup "*orphan permissions*". but it is not mandatory to assign this domain to a role (this domain has no "*functional meaning*"). You will only assign some permissions of this domain to the role.The reason to always assign a domain to a permission, is to be sure that the sum of permissions of all domains contains all the permissions declared in the application.

- What is the role of `logonProcessURI`?

  `logonProcessURI` is the way jGuard receive credentials through FORM authentication. The html form which contains your login and password will send this information to this special URI interecepted by jGuard.jGuard will evaluate them and authenticate you. It will redirect you to the convenient URI. So, this special URI does not point to a dedicated page.