# Code inspection

M. Albanese, M. Bianchi, A. Carlucci

January 4, 2016

# Contents

**1 Description of classes** **3**

**2 Functional roles of the assigned class** **4**
2.1 Authentication . . . . . 4
2.2 Users, groups, roles and principals . . . . . 4

**3 List of issues** **6**
3.1 Common issues . . . . . 6
3.2 authenticate() . . . . . 10
3.3 preSetRunAsIdentity() . . . . . 10
3.4 postSetRunAsIdentity() . . . . . 11
3.5 principalSetContainsOnlyAnonymousPrincipal() . . . . . 11
3.6 invokeWebSecurityManager() . . . . . 12
3.7 hasUserDataPermission() . . . . . 12

**4 Other issues** **14**

**References** **15**

# 1 Description of classes

The `RealmAdapter` class provides authentication and authorization functionalities to access web resources. In particular, its role is to let users login, logout and to check users privileges.

The class is contained in the package `com.sun.web.security` which mainly contains:

**HTTP wrappers**
`HttpRequestWrapper.java` and `HttpResponseWrapper.java`.

**Helper login functions**
`LoginProbeProvider.java` and `LoginStatsProvider.java`

**SSL Factory**
`SSLSocketFactory.java`

## 2 Functional roles of the assigned class

Java EE provides an elaborate security system to provide access to protected resources on the web server. According to official documentation (see [2]):

> Java EE applications consist of components that can contain both protected and unprotected resources. Often, you need to protect resources to ensure that only authorized users have access. *Authorization provides controlled access to protected resources.*
> Authorization is based on identification and authentication.
> **Identification** is a process that enables recognition of an entity by a system.
> **Authentication** is a process that verifies the identity of a user, device, or other entity in a computer system, usually as a prerequisite to allowing access to resources in a system.

The main function for this class is to provide code that guarantees what stated above; in other words, code for *authentication* and *logout*, *authorization*, *role checking*.

### 2.1 Authentication

The whole set of protected resources is partitioned into security spaces called **realms**, characterized by security policies and allowed users/groups. There are two important predefined realms in Java EE:

- `file`: check inserted credentials with the ones contained in a `keyfile`;
- `certificate`: an `X.509` certificate is provided by the client through an `HTTPS` connection, which is verified on the server side.

The class `RealmAdapter` wraps auth functionalities both via credentials and certificates by using several overloaded `authenticate()` functions and logout functionalities through a `logout()` method. In particular, each `authenticate()` delegates to the one in line 676 (described in section 3.2).

### 2.2 Users, groups, roles and principals

The actual recognition of customers on the server is OS-like: the ones defined in the system are called **users** and can either be humans or applications. Multiple users can belong to the same **group**, which is a useful subdivision when several people are involved (e.g. an e-commerce site could contain a `CUSTOMER` group defined). In a similar fashion, **roles** define a division of users; they differ in the *scope*:

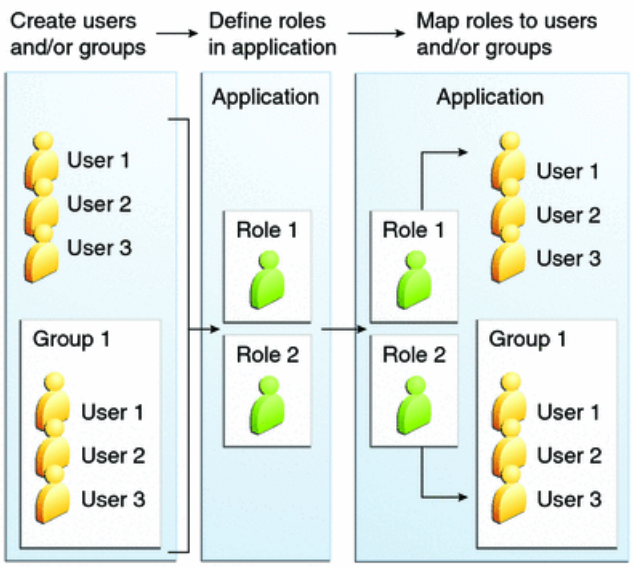



Figure 1: Mapping between users and roles

> A group is designated for the entire GlassFish Server, whereas a role is associated only with a specific application in the GlassFish Server.

This implies however that roles must be mapped to a certain user or to one or more groups in the overall system in order to have consistent results (see Figure 1). Groups and roles have of course different permissions when accessing protected resources (this means that certain elements are accessible if and only if the request comes from a user from a specified group/role). The class provides useful methods for role verification (`hasRole()` methods).

In more general terms, any entity that could be authenticated by an authentication protocol is called **principal**. The class provides some useful methods regarding principals, such as `getPrincipal()` given a user name, the already cited `authenticate()` and `hasRole()` functions, `preSetRunAsIdentity()` and `postSetRunAsIdentity()`.

```
1316 //    Function not required anymore.
1317 //    private String getExtension(String uri) {
1318 //        int index=uri.lastIndexOf(".");
1319 //        if(index >= 0) {
1320 //            return uri.substring(index);
1321 //        } else {
1322 //            return "";
1323 //        }
1324 //    }
1325
```

Figure 2: Commented out code

# 3 List of issues

Here is a comprehensive list of all issues we have found according to the checklist reported in [1]. For the sake of clarity we decided to write down only relevant and meaningful points and to omit the other ones.

## 3.1 Common issues

### [7] Constant declarations

`Name` constant in line 178 should be uppercase.

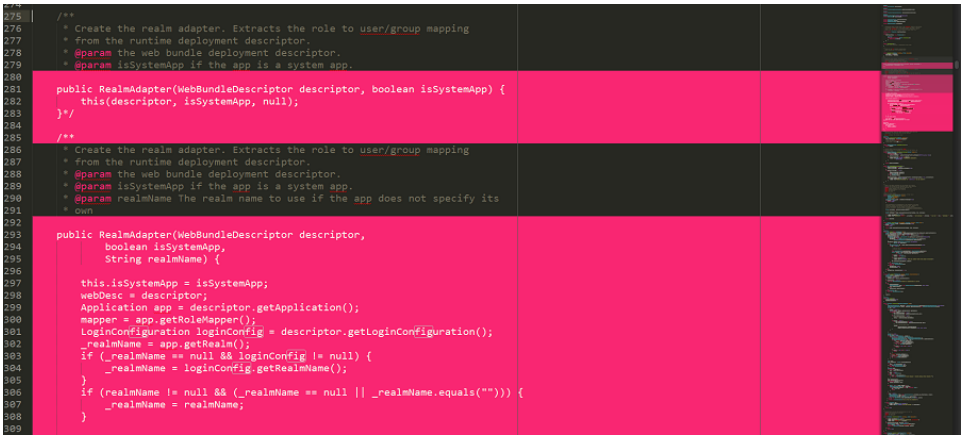
### [18] Comments

The comment style is not uniform; some methods seem to be more explained, both through JavaDoc (see later) and several inline comments, while others do not have any comment for dozens of line:

- `authenticate (lines 518-622)`
- `authenticate (from line 676)`
- `invokeWebSecurityManager (from line 965)`
- `redirect (from line 1250)`
- `preAuthenticateCheck (from line 1415)`
- `validate (from line 1626)`

### [19] Commented out sections

There is only one function commented out: `getExtension` in line 1316 (see figure 2). According to the comment above, the function is no more useful and can be safely removed.
However, some snippets are commented out for reference and must not be erased according to its comment (in lines 1146 and 1691).



```
    /**
     * Create the realm adapter. Extracts the role to user/group mapping
     * from the runtime deployment descriptor.
     * @param the web bundle deployment descriptor.
     * @param isSystemApp if the app is a system app.
     */
    public RealmAdapter(WebBundleDescriptor descriptor, boolean isSystemApp) {
        this(descriptor, isSystemApp, null);
    }*/

    /**
     * Create the realm adapter. Extracts the role to user/group mapping
     * from the runtime deployment descriptor.
     * @param the web bundle deployment descriptor.
     * @param isSystemApp if the app is a system app.
     * @param realmName The realm name to use if the app does not specify its
     * own
     */
    public RealmAdapter(WebBundleDescriptor descriptor,
            boolean isSystemApp,
            String realmName) {

        this.isSystemApp = isSystemApp;
        webDesc = descriptor;
        Application app = descriptor.getApplication();
        mapper = app.getRoleMapper();
        LoginConfiguration loginConfig = descriptor.getLoginConfiguration();
        _realmName = app.getRealm();
        if (_realmName == null && loginConfig != null) {
            _realmName = loginConfig.getRealmName();
        }
        if (realmName != null && (_realmName == null || _realmName.equals(""))) {
            _realmName = realmName;
        }
```

Figure 3: Errors in Javadoc in lines 275–337

**[23] JavaDoc**
There are two wrong JavaDoc comments, shown in fig. 3, involving two constructors (lines 275–337).
There are also a lot of methods without any JavaDoc comment.

- `WebBundleDescriptor`
- `getWebSecurityManager`
- `updateWebSecurityManager`
- `hasRole` (line 438)
- `doLogout` (line 489)
- `authenticate` (line 518)
- `authenticate` (line 646)
- `authenticate` (line 661)
- `SecurityContext` (line 852)
- `SecurityContext` (line 856)
- `getPassword` (line 884)
- `getPrincipal` (line 888)
- `getHostAndPort` (line 1147)
- `redirect` (line 1250)
- `getCanonicalName` (line 1304)

- `getResourceName` (line 1308)
- `setRealmName` (line 1343)
- `validate` (line 1626)
- `shouldRegister` (line 1754)
- `mapEntryToBoolean` (line 1762)
- `resetPolicyContext` (line 1790)
- `getSecurityContextForPrincipal` (line 1959)
- `setCurrentSecurityContextWithWebPrincipal` (line 1977)
- `setCurrentSecurityContext` (line 1983)
- `initConfigHelper` (line 1988)
- `postConstruct` (line 1995)

Also, two inner classes: `AuthenticatorProxy` (line 1798) and `HttpMessageInfo` (1846) have no JavaDoc comment.

**[25] Class declarations**
Points **A**, **B**, **C** are OK;

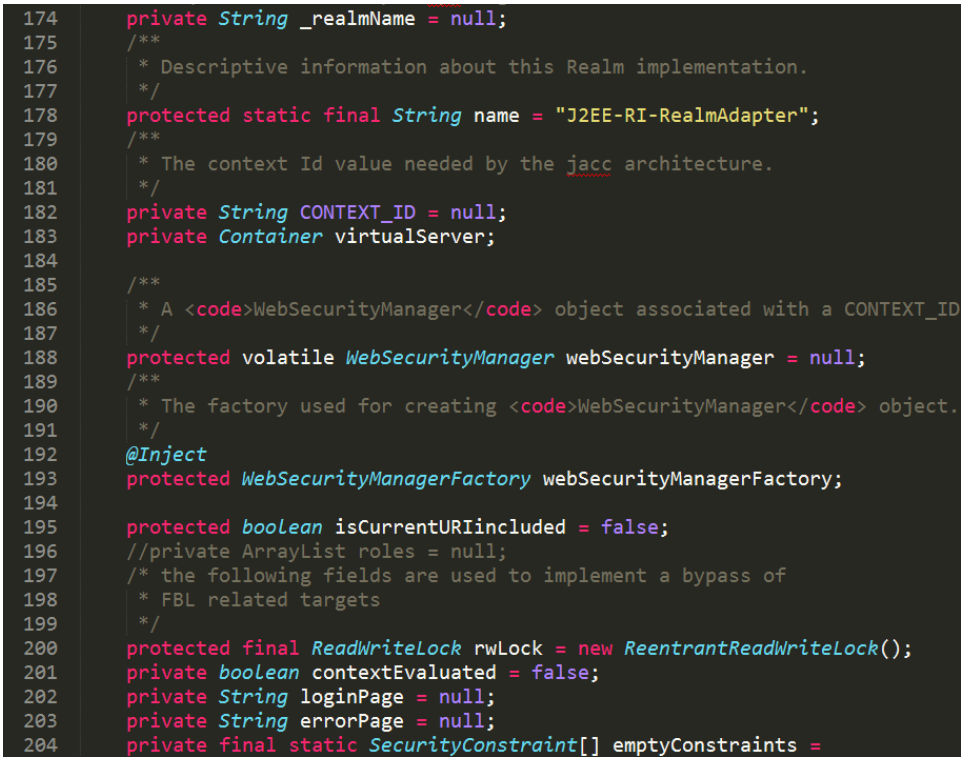
Point **D**: static variables are mainly grouped on top of the class, few are mixed with non-static variables and methods, like `reentrancyStatus` in line 249, `CONF_FILE_NAME` and `HTTP_SERVER_LAYER` in line 1578.

Point **E**: instance variables are not defined in the order shown in [1] (see fig. 4), but are logically related.
This means that some protected variables appear after private ones (for example see lines 188–200).

Points **F** and **G**: As for the methods, constructors correctly appear before any other method.

**[26] Method organization**
Methods seem to be logically grouped, even if some are out of order:

- Setting up WebSecurityManager (lines 368-400)
- Role checking (lines 400–444)
- Logout(lines 448–516) and Login (lines 518–714)



```
174     private String _realmName = null;
175     /**
176      * Descriptive information about this Realm implementation.
177      */
178     protected static final String name = "J2EE-RI-RealmAdapter";
179     /**
180      * The context Id value needed by the jacc architecture.
181      */
182     private String CONTEXT_ID = null;
183     private Container virtualServer;
184
185     /**
186      * A <code>WebSecurityManager</code> object associated with a CONTEXT_ID
187      */
188     protected volatile WebSecurityManager webSecurityManager = null;
189     /**
190      * The factory used for creating <code>WebSecurityManager</code> object.
191      */
192     @Inject
193     protected WebSecurityManagerFactory webSecurityManagerFactory;
194
195     protected boolean isCurrentURIincluded = false;
196     //private ArrayList roles = null;
197     /* the following fields are used to implement a bypass of
198      * FBL related targets
199      */
200     protected final ReadWriteLock rwLock = new ReentrantReadWriteLock();
201     private boolean contextEvaluated = false;
202     private String loginPage = null;
203     private String errorPage = null;
204     private final static SecurityConstraint[] emptyConstraints =
```

Figure 4: Protected and private variables are mixed

- Run-as functions(lines 715–963)
- Permission check functions (lines 965–1144)
- Host and port management (lines 1145–1247)
- Redirecting (lines 1250–1300)
- Getters & setters (lines 1300–1350)
- Security constraints (lines 1356–1394)
- Pre and post auth methods (lines 1415–1575)
- Validation (lines 1626–1750)
- Other getter functions (lines 1754–1793)
- AuthProxy class (lines 1800–2000)

**[27] Long methods**
Here's a list of some quite long methods. They should be split using helper methods.

- `validate` (lines from 1626)
- `getHostAndPort` (lines from 1147)
- `invokeWebSecurityManager` (lines from 965)
- `authenticate` (lines from 518)

## 3.2 authenticate()

**[13] 80 characters limit**
There are two lines that exceed the 80 character limit: the comment in line 687 and the line 688.

**[18] Comments**
The JavaDoc comments explaining the method parameters are not up to date and seem to be misleading. No comments present in method, some may be useful to better explain the inner workings.

**[34] Parameters order**
The parameters are not aligned with what explained in the JavaDoc above; however, the list of parameters is sound and presented in a reasonable order (username, password, certificate).

**[38] IndexOutOfBoundExceptions**
There could be an overflow in line 684 but the exception is caught.

**[42] Error messages**
Error messages are not detailed and do not provide any guidance to how to solve problems (`web.login.failed` and `Exception` at line 701 and 704).

**[52] Exception handling**
All code is wrapped in a try catch block; Exceptions are caught at a high level, which means they are all caught but handled in a very general way (see Point 42).

## 3.3 preSetRunAsIdentity()

**[5] Method names should be verbs**
`Pre` is not a verb; the other words are with the first letter capitalized so that part is ok.

**[18] Comments**
JavaDoc is unclear, since it does not explain exactly when the function is called. The code in line 722 inside is self-explainatory, but something is not working regarding the principals (typed names) for a certain servlet, as the comments in line 752–755 state.

**[33] Declarations at the beginning of blocks**
Only one initialization in line 749, not on top of the block; however, this is due to the fact a check is performed before this instruction.

### 3.4 postSetRunAsIdentity()

**[5] Method names should be verbs**
`Post` is not a verb; the other words are with the first letter capitalized so that part is ok.

**[13] 80 characters limit**
A comment in line 843 starts after the 80th character.

**[18] Comments**
Comment in line 843 (`always null`) does not explain why it is so.

**[33] Declarations at the beginning of blocks**
Variables defined just before use, not on top of block.

**[34] Parameters order**
The parameters are not aligned with what explained in the JavaDoc above; however, the list of parameters is sound and presented in a reasonable order (username, password, certificate).

### 3.5 principalSetContainsOnlyAnonymousPrincipal()

**[1] Meaningful names**
Variable `rvalue` in line 868 has an unmeaningful name.

**[13] 80 characters limit**
Line 867 and line 869 exceed 80 character limit.

**[18] Comments**
No comments, but code is self-explanatory.

**[36] Return values**
As it might not seem, return value is coherently used during validation in line 1684, where the principalSet is analyzed.

### 3.6 invokeWebSecurityManager()

**[13] 80 characters limit**
Lines 989 and 1031 exceeds 80 characters.

**[14] 120 characters limit**
Line 1031 exceeds 120 characters.

**[16] Higher-level breaks**
In lines 965, 966, 967, 1003 and 1026 there are breaks which are not "higher-level" breaks.

**[18] Comments**
Few comments aside JavaDoc are present, but code is well-written and comprehensible.

**[36] Return values**
The return value is used in two points where permission checking is needed: line 932 (`hasResourcePermission`, which wraps this function) and line 1432 (`preAuthenticateCheck`, to check if auth will actually be necessary). It is used correctly in both cases.

**[44] Brutish programming**
There are duplicate checks after line 1000. Lines from 1000 to 1020 could be improved by putting every `_logger.isLoggable(Level.FINE)` inside `_logger.fine()`.

### 3.7 hasUserDataPermission()

**[1] Meaningful names**
Class variable `rvalue` in line 1140 has an unmeaningful name.

**[13] 80 characters limit**
Lines 1095, 1100, 1119, 1139 exceed 80 character limit.

**[14] 120 characters limit**
Line 1095 exceeds 120 characters.

**[16] Higher-level breaks**
In lines 1084, 1085, 1089, 1090 and 1139 there are breaks which are not "higher-level" breaks.
Also, the break in line 1139 is misplaced since a better positioning of it would have also solved the 80 chars exceeding problem.

**[33] Declarations at the beginning of blocks**
In general yes, except for initialization in line 1105, which is however done just before use in the subsequent block.

**[42] Error messages**
Error codes are delegated to HttpServletResponse, with a `BAD_REQUEST` or `FORBIDDEN` argument indicating that the request was not well-formed or not authorized (line 1119 and 1139 respectevely).
A warning is issued too with an error message and an ID in the first case.

**[52] Exception handling**
All IO exceptions are thrown out like the previous method.
An `IllegalArgumentException` indicating a bad request is caught and correctly handled.

# 4 Other issues

After a complete overhaul of the class in which our methods are placed, we have spotted these bugs which can affect the proper behaviour of the class.

According to the text comments in line 354, the method doesn't work properly because its run will result in a classload failure. There is also a known bug (`no.4757733`) which affects the security context loading and saving. The issue can be shown in method `preSetRunAsIdentity` starting in line 734 and in method `postSetRunAsIdentity` starting in line 827.

The object passed in the two functions seems not to be the same, according to the JavaDoc comment in line 813, and this, in practical, forces the second method to always set the `SecurityContext` to NULL.

Also, the comment in line 1124 says that there is a bug (`no.4947698`) in method `hasUserDataPermission` (starting at line 1084), but an analysis of the given portion of the code does not seem to enlighten any issue.

# References


[1] Prof. Di Nitto. *Assignment 3: Code Inspection.*

[2] AA.VV., *Java EE 6 Official Documentation.* `https://docs.oracle.com/javaee/6/tutorial/doc/bnbxj.html`

[3] AA.VV., *GlassFish Appserver Parent Project 4.1.1 API - Javadoc.* `http://glassfish.pompel.me/`


# Hours spent

| **Code Inspection** | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| Data | Michele | Alain | Mattia | What was done? | |
| 15/12/2015 | 2 | 0 | 2 | Tia, Mitch: First method inspection | |
| 16/12/2015 | 2 | 2 | 1,5 | Al: code inspection. Mitch: first 2 methods inspection; Tia: Second and third method inspection | |
| 17/12/2015 | | | 2 | Tia: fourth, fifth and sixth method inspection | |
| 18/12/2015 | 1,5 | | | Mitch: remaning methods | |
| 19/12/2015 | | 0,5 | | Al: trying to change bug format and reverting to old one. | |
| 22/12/2015 | 3 | 3 | | Al & mitch: formatted list + revision | |
| 23/12/2015 | 2 | | | Mitch: added imgs, refs, functional role description | |
| 02/01/2016 | | 1 | 1 | Al & tia: Other issues in the given class. | |
| | **10,5** | **6,5** | **6,5** | | |