POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2015/2016
Software Engineering 2

Assignment 3: Code Inspection

Version 1.0

Alessandro Baldassari (mat. 841561)
Alberto Bendin (mat. 841734)
Francesco Giarola (mat. 840554)

December 28, 2015

Contents

1 Classes that were assigned to the group

We are inspecting a piece of code from Glassfish 4.1.1, revision 64219 of 2015-10-16.
We were assigned the "WebPermissionUtil.java" class, located in the path:

`appserver/security/core-ee/src/main/java/com/sun/enterprise/security/web/integration/`

In particular we had to analyze the methods:

- handleNoAuth(Permissions collection , MapValue m , String name)
- handleConnections(Permissions collection , MapValue m , String name)
- processConstraints(WebBundleDescriptor wbd , PolicyConfiguration pc)
- createWebRoleRefPermission(WebBundleDescriptor wbd , PolicyConfiguration pc)

2 Functional role of assigned set of classes

The class "WebPermissionUtil" generates web permissions and it is part of the set of classes that manages all the security decisions required to allow the access to a resource. In particular it fulfills the role of utility class specialized in parsing and managing the policy configurations related to web-onnection security and permissions.
Evidences of the functional role of the class are already present in the path and name of its own package:
`appserver/security/core-ee/src/main/java/com/sun/enterprise/security/web/integration/`

Moreover after an exhaustive search of all the usages of the class in the call hierarchy, we found out that the only caller is the "WebSecurityManager", and both a code inspection of the latter and its own javadoc (image below) had confirmed the role of "WebPermissionUtil" class.

WebSecurityManager's javadoc

The class implements the JSR 115 - JavaTM Authorization Contract for Containers. This class is a companion class of EJBSecurityManager. All the security decisions required to allow access to a resource are defined in that class.

Author:
Jean-Francois Arcand, Harpreet Singh.

Field Summary

Fields

Modifier and Type	Field and Description
protected CodeSource	codesource
static String	CONSTRAINT_URI Request path.
protected javax.security.jacc.PolicyConfiguration	pc
protected javax.security.jacc.PolicyConfigurationFactory	pcf
protected Policy	policy

Method Summary

Methods

Modifier and Type	Method and Description
protected boolean	checkPermission(Permission perm, Set principalSet)
void	destroy()
static String	getContextID(WebBundleDescriptor wbd)
boolean	hasNoConstrainedResources() returns true to indicate that a policy check was made and there were no constrained resources.
boolean	hasResourcePermission(javax.servlet.http.HttpServletRequest httpsr) Perform access control based on the HttpServletRequest.
boolean	hasRoleRefPermission(String servletName, String role, Principal p)
int	hasUserDataPermission(javax.servlet.http.HttpServletRequest httpsr, String uri, String httpMethod) if uri == null, determine if the connection characteristics of the request satisfy the applicable policy.
void	loadPolicyConfiguration()
boolean	permitAll(javax.servlet.http.HttpServletRequest req)
void	release() Analogous to destroy, except does not remove links from Policy Context, and does not remove context_id from role mapper factory.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

CONSTRAINT_URI

public static final String CONSTRAINT_URI

Request path. Copied from org.apache.catalina.Globals; Required to break dependence on WebTier of Security Module

See Also:
Constant Field Values

3 List of issues found by applying the checklist

Here are reported only the issues found while analyzing the code with the provided Java code inspection checklist.

Naming Conventions

2. **If one-character variables are used, they are used only for temporary "throw-away" variables, such as those used in for loops.**

   At line 379 the variable "`m`" should have been named with a more meaningful name since it is not a throwaway variable:

abstract from method “handleNoAuth”

```
379     static void handleNoAuth(Permissions collection, MapValue m,
380                              String name) {
381     String actions = null;
382     BitSet noAuthMethods = m.getNoAuthMethods();
383     if (!m.otherConstraint.isAuthConstrained()) {
384         BitSet methods = m.getMethodSet();
385         methods.andNot(noAuthMethods);
386         if (!methods.isEmpty()) {
```

Indention

8. **Three or four spaces are used for indentation and done so consistently.**

   In the following example three and four spaces are mixed, in the first line a tab and 4 spaces are used, while in the second line there are 2 tabs and 7 spaces.

abstract from method “processConstraints”

```
490 ····    logger.log(Level.FINE,"JACC: constraint translation:
        CODEBASE = "+
491 ····    ·····pc.getContextID());
```

   Another example is at line 380 where there are 3 tabs and 5 spaces.

abstract from method “handleNoAuth”

```
379 ····static void handleNoAuth(Permissions collection, MapValue m,
380 ····    ·····String name) {
381 ····String actions = null;
```

   The same goes for lines 609, 628, 630.

9. **No tabs are used to indent.**

   The following example shows how tabs are often used, sometimes mixed with spaces too.

abstract from method “processConstraints”

```
488 ····if (logger.isLoggable(Level.FINE)){
489 ····    logger.entering("WebPermissionUtil", "processConstraints");
```

   One should avoid using tabs to indent code also because the interpretation of tabs varies with different IDEs or text editors.
   Another example is at line 387 where 2 tabs are used, while at line 388 1 tab and 4 spaces, and since tabs (in this case) are associated with 4 spaces the two lines appear aligned even though theoretically they are not at the same level of indentation.

abstract from method “handleNoAuth”

```
383 ····if (!m.otherConstraint.isAuthConstrained()) {
384 ····    BitSet methods = m.getMethodSet();
385 ····    methods.andNot(noAuthMethods);
386 ····    if (!methods.isEmpty()) {
387 ····        actions = "!" + MethodValue.getActions(methods);
388 ····    }
389 ····} else if (!noAuthMethods.isEmpty()) {
```

   The whole package uses randomly tabs for indentation.

Braces

10. **Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).**

    In the following example line 487 opens the method using the "Allman" style, all the other blocks follow the "Kernighan and Ritchie" style.

    abstract from method "processConstraints"

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                         PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation:
                CODEBASE = "+
491                 pc.getContextID());
492     }
```

    The same is for the opening of the method at line 583.

11. All `if`, `while`, `do-while`, `try-catch`, and `for` statements that have only one statement to execute are surrounded by curly braces.

File Organization

13. **Where practical, line length does not exceed 80 characters.**

    In the following example lines 503 and 504 could have been broken in three lines instead of two.

    abstract from method "processConstraints"

```
501     boolean deny = wbd.isDenyUncoveredHttpMethods();
502     if (logger.isLoggable(Level.FINE)){
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                    denied" : "permitted"));
505     }
506
507     // for each urlPatternSpec in the map
508     Iterator it = qpMap.values().iterator();
```

    The same can be applied to other lines facing the same problem, for instance lines 586, 608, 609, 615, 620 and many others.

14. **When line length must exceed 80 characters, it does NOT exceed 120 characters.**

    All the lines which (even if arguable) reasonably exceed 80 characters never violate the limit of 120 characters. Other lines that trespass the limit fall within the lines that should be wrapped in the point 13 of this checklist.

Wrapping Lines

15. **Line break occurs after a comma or an operator.**

    In the following example line 503 is written wrong because the line-break precedes the "+" operator.

abstract from method "processConstraints"

```
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                        denied" : "permitted"));
```

    The same goes for lines 629, 659, 667, 671.

17. **A new statement is aligned with the beginning of the expression at the same level as the previous line.**

    The whole method "processConstraints" lacks a level of indentation (is at the same level of the external code); an example of this is the opening of the method itself at line 488 (it is evident when the number of spaces associated to a tab is 4).

abstract from method "processConstraints"

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                         PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation: CODEBASE = "+
491                 pc.getContextID());
492     }
```

    The same goes for method "handleNoAuth" which lacks one level of indentation, from line 381 to 399.
    Other examples are `while` loops or `if` statements at the same level of the upper-level code, like at line 541, where one level of indention is missing.

abstract from method "processConstraints"

```
539         Enumeration e = excluded.elements();
540         while (e.hasMoreElements()) {
541         Permission p = (Permission) e.nextElement();
542         String ptype = (p instanceof WebResourcePermission) ? "WRP  " : "
                WUDP ";
543         logger.log(Level.FINE,"JACC: permission(excluded) type: "+ ptype
                + " name: "+ p.getName() + " actions: "+ p.getActions());
544         }
```

    The same is for line 548, 564, 597, 601, 608, 615, 618, 627.
    Lines 592 and 594 are wrongly aligned with respect to the previous lines:

abstract from method "createWebRoleRefPermission"

```
590     Role anyAuthUserRole = new Role("**");
591     boolean rolesetContainsAnyAuthUserRole = roleset.contains(
            anyAuthUserRole);
592         Set<WebComponentDescriptor> descs = wbd.
                getWebComponentDescriptors();
593     //V3 Commented for(Enumeration e = wbd.getWebComponentDescriptors();
            e.hasMoreElements();){
```

```
594         for (WebComponentDescriptor comp : descs) {
595         //V3 Commented WebComponentDescriptor comp = (
                WebComponentDescriptor) e.nextElement();
```

The `for` statement at line 594 closes at line 640, but it is almost impossible to read unless with the help of parentheses highlight; the closing bracket is wrongly aligned.

Comments

18. **Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.**

    Comments are not adequately used to explain what the code is trying to do, for example in line 484 the method is public and has no comments at all to describe its behavior.

abstract from method "processConstraints"

```
481     pc.removeRole("*");
482     }
483
484     public static void processConstraints(WebBundleDescriptor wbd,
485                         PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
```

19. **Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.**

    Line 724 in the example below has not been commented properly at all; the same is for lines 593 and 595.

abstract from method "processConstraints"

```
723     ignoreRoleList = false;
724     //roleList = new ArrayList<String>();
725     connectSet = 0;
```

The lines 1128, 1129 and 1130 have been commented out with a reason but without any date for safe remove.

abstract from method "processConstraints"

```
1120            for (MethodValue v : values) {
1121                /*
1122                 * NOTE WELL: prior version of this method
1123                 * could not be called during constraint parsing
1124                 * because it finalized the connectSet when its
1125                 * value was 0 (indicating any connection, until
1126                 * some specific bit is set)
1127                 *
1128                if (v.connectSet == 0) {
1129                v.connectSet = MethodValue.connectTypeNone;
1130                }
1131
1132                 */
1133
1134                if (v.isConnectAllowed(cType)) {
```

Java Source Files

20. **Each Java source file contains a single public class or interface.**

    The rule is respected because the only public class is "WebPermissionUtil", the others ("ConstraintValue", "MethodValue", "MapValue") are not public classes.

23. **Check that the javadoc is complete.**

    Javadoc are almost missing for this class, as shown in the picture below.

Constructor Detail

WebPermissionUtil

```
public WebPermissionUtil()
```

Method Detail

parseConstraints

```
public static HashMap parseConstraints(WebBundleDescriptor wbd)
```

removePolicyStatements

```
public static void removePolicyStatements(javax.security.jacc.PolicyConfiguration pc,
                          WebBundleDescriptor wbd)
                                   throws javax.security.jacc.PolicyContextException
```

Remove All Policy Statements from Configuration config must be in open state when this method is called

Parameters:

pc -

wbd -

Throws:

javax.security.jacc.PolicyContextException

processConstraints

```
public static void processConstraints(WebBundleDescriptor wbd,
                      javax.security.jacc.PolicyConfiguration pc)
                               throws javax.security.jacc.PolicyContextException
```

Throws:

javax.security.jacc.PolicyContextException

createWebRoleRefPermission

```
public static void createWebRoleRefPermission(WebBundleDescriptor wbd,
                              javax.security.jacc.PolicyConfiguration pc)
                                       throws javax.security.jacc.PolicyContextException
```

Throws:

javax.security.jacc.PolicyContextException

Class and Interface Declarations

25. **The class or interface declarations shall be in the following order:**

    (a) class/interface documentation comment;

    (b) class or interface statement;

    (c) class/interface implementation comment, if necessary;

    (d) class (static) variables;

        i. first public class variables;

ii. next protected class variables;
iii. next package level (no access modifier);
iv. last private class variables.

(e) instance variables;
i. first public instance variables;
ii. next protected instance variables;
iii. next package level (no access modifier);
iv. last private instance variables.

(f) constructors;

(g) methods.

At line 69, the class constructor is before the list of private static variables, as shown below.

abstract from method "processConstraints"

```
65 public class WebPermissionUtil {
66
67     static Logger logger = Logger.getLogger(LogDomains.SECURITY_LOGGER);
68
69     public WebPermissionUtil() {
70     }
71
72     /* changed to order default pattern / below extension */
73     private static final int PT_DEFAULT      = 0;
74     private static final int PT_EXTENSION    = 1;
75     private static final int PT_PREFIX       = 2;
76     private static final int PT_EXACT        = 3;
```

26. **Methods are grouped by functionality rather than by scope or accessibility.**
Methods are grouped by accessibility rather than by functionality; in order there are package level methods, public methods and finally private methods.

Initialization and Declarations

30. **Check that constructors are called when a new object is desired.**
The following example represents a case in which the declaration (line 494) may be split in declaration and assignment.

abstract from method "processConstraints"

```
494     HashMap qpMap = parseConstraints(wbd);
495     HashMap<String,Permissions> roleMap =
496         new HashMap<String,Permissions>();
```

The same is for lines: 382, 384, 510, 541, 548, 560, 566.

31. Check that all object references are initialized before use.

32. Variables are initialized where they are declared, unless dependent upon a computation.

33. **Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a `for` loop.**
In the following example the `if` statement at line 488 must be postponed till after line 501 and line 508 must be put before the block of line 502.

abstract from method “processConstraints”

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                         PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation:
                CODEBASE = "+
491                 pc.getContextID());
492     }
493
494     HashMap qpMap = parseConstraints(wbd);
495     HashMap<String,Permissions> roleMap =
496         new HashMap<String,Permissions>();
497
498     Permissions excluded = new Permissions();
499     Permissions unchecked = new Permissions();
500
501     boolean deny = wbd.isDenyUncoveredHttpMethods();
502     if (logger.isLoggable(Level.FINE)){
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                    denied" : "permitted"));
505     }
506
507     // for each urlPatternSpec in the map
508     Iterator it = qpMap.values().iterator();
```

The same is for:

- line 539 must be before line 537
- line 564 must be before line 561
- lines 662 and 663 must be before line 657

Method Calls

34. Check that parameters are presented in the correct order.
35. Check that the correct method is being called, or should it be a different method with a similar name.
36. Check that method returned values are used properly.

Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
39. Check that constructors are called when a new array item is desired.

Object Comparison

40. Check that all objects (including Strings) are compared with `equals` and not with `==`.

Output Format

41. Check that displayed output is free of spelling and grammatical errors.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.
43. Check that the output is formatted correctly in terms of line stepping and spacing.

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming": (see `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`).
45. Check order of computation/evaluation, operator precedence and parenthesizing.
46. **Check the liberal use of parenthesis is used to avoid operator precedence problems.**

    At line 637 some parentheses could be used as in the first part of the line.

    abstract from method "createWebRoleRefPermission"

```
637         if ((!role.contains(anyAuthUserRole)) && !
                rolesetContainsAnyAuthUserRole) {
638           addAnyAuthenticatedUserRoleRef(pc, name);
```

47. Check that all denominators of a division are prevented from being zero.
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
49. Check that the comparison and Boolean operators are correct.
50. Check throw-catch expressions, and check that the error condition is actually legitimate.
51. Check that the code is free of any implicit type conversions.

Exceptions

52. Check that the relevant exceptions are caught.
53. Check that the appropriate action are taken for each catch block.

Flow of Control

54. In a `switch` statement, check that all cases are addressed by break or return.
55. Check that all switch statements have a default branch.
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Files

57. Check that all files are properly declared and opened.
58. Check that all files are closed properly, even in the case of an error.
59. Check that EOF conditions are detected and handled correctly.
60. Check that all file exceptions are caught and dealt with accordingly.

4 Any other problem you have highlighted

5 Appendix

5.1 Software and tools used

- TeXstudio 2.10.4 (http://www.texstudio.org/) to redact and format this document.
- NetBeans 8.1 (https://netbeans.org/) to download and inspect the code.
- Sublime Text (http://www.sublimetext.com/) to inspect the code.

5.2 Hours of work

The time spent to redact this document:

- Baldassari Alessandro: 20 hours.
- Bendin Alberto: 20 hours.
- Giarola Francesco: 20 hours.