Politecnico di Milano

Department of Electronics, Information and Bioengineering

Master Degree course in Computer Science Engineering

Code Inspection of

Glassfish

version 4.1.1 revision 64219

Instructor: Prof. Elisabetta Di Nitto

Authors:	*Code:*
Luca Luciano Costanzo	789038
Simone Disabato	852863

Document version 1.0, released on December 26, 2015

Contents

Chapter 1

Code Inspection Checklist

In this chapter the detailed analysis of the assigned code is presented. We have decided to create a section for each main part of the code inspection's check-list and inside it, all the "sub-points"are analysed at the same time.[1]This decision is to simplify the drafting of the text and to make the document more readable and easy to understand.
However, the wrong code is often displayed and, in some particular cases, a possible correction of the code is provided (for instance the correct indentation of the last method checked).
An important clarification is the following. We don't have tried to find out some bugs into the code for several reasons. First of all, we are not expert in security themes, so we are not able to identify problems or we don't know the best ways to implement the security protocols. Second (and last), the code assigned to us is too short to detect some bugs related to exactly that part of code and that not require to read and understand many and many other lines of code into the classes of the same package of the our one.

[1]The code inspection's check-list is available into the appendix.

1.1 Naming Conventions

In the method *"getSubjectFromSecurityCurrent( )"*at line 963, the two local variables have a meaningless name. A better one can be exactly the same of the belonging class.

```
965         com.sun.enterprise.security.SecurityContext sc = null;
966         sc = com.sun.enterprise.security.SecurityContext.getCurrent();
```

```
977         Subject s = sc.getSubject();
```

To be more precise, since this two variables are used only to memorize a method's return value and then, in the following lines of code, they are returned or used as parameters in other methods, the two names can be accepted.
In the method *"useMechanism(...)"*at line 1019, the local variable as a meaningless name. A meaningful name is *toReturn*.

```
1020        boolean val = true;
```

See the section 1.19 for further observations about the role of this variable into the method.
In the method *"evaluate_client_conformance_ssl(...)"*at line 1086, we have three not respected conventions. First, the name of the method is wrong. The correct one is *evaluateClientConformanceSsl(...)*. After that, the second parameter is wrong because it contains an underscore to split two words.

```
1088                        boolean  ssl_used,
```

The last one is all the names of the local variables, for the same reason of the parameter.

```
1097        boolean ssl_required  = false;
1098        boolean ssl_supported = false;
1099        int ssl_target_requires = 0;
1100        int ssl_target_supports = 0;
```

Finally, if we consider the entire class, also the following names do not respect the conventions.

```
124     private static final java.util.logging.Logger _logger =
```

```
279             int ssl_port = Utility.shortToInt(sslport);
280             String host_name = ssl.addresses[0].host_name;
```

The same two variables above appear, with the same wrong names, at lines 301-302, 401-402 and 427-428.

```
564         int ident_token = sas.supported_identity_types;
```

```
849                 final byte[] target_name = asContext.target_name;
```

```
856                 final String realm_name = new String(_realm);
```

1.2 Indention

In the method "*getSubjectFromSecurityCurrent( )*"at line 963, the line 969 should be indented and the tab character should be replaced with four spaces (the number of spaces is the same of all the document).

```
968 ␣␣␣␣␣␣␣␣␣␣␣␣if(_logger.isLoggable(Level.FINE))␣{
969 ␣␣␣␣␣␣␣␣⟶_logger.log(Level.FINE,"␣SETTING␣GUEST␣---");
970 ␣␣␣␣␣␣␣␣␣␣␣␣}
```

In the method "*evaluate_client_conformance_ssl(. . .)*"at line 1086, we have several indention's errors.
First of all, in the following lines tabs characters were used.

```
1092 ␣␣␣␣␣␣␣␣if(_logger.isLoggable(Level.FINE))␣{
1093 ⟶␣␣␣␣_logger.log(Level.FINE,
1094 ⟶⟶⟶"SecurityMechanismSelector.evaluate_client_conformance_ssl->:");
1095 ⟶}
```

```
1142 ␣␣␣␣␣␣␣␣if(_logger.isLoggable(Level.FINE))␣{
1143 ⟶␣␣␣␣_logger.log(Level.FINE,
1144 ⟶⟶⟶"SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1145 ⟶⟶⟶+␣"␣"␣+␣isSet(ssl_target_requires,␣Integrity.value)
1146 ⟶⟶⟶+␣"␣"␣+␣isSet(ssl_target_requires,␣Confidentiality.value)
1147 ⟶⟶⟶+␣"␣"␣+␣isSet(ssl_target_requires,␣EstablishTrustInClient.value)
1148 ⟶⟶⟶+␣"␣"␣+␣ssl_required
1149 ⟶⟶⟶+␣"␣"␣+␣ssl_supported
1150 ⟶⟶⟶+␣"␣"␣+␣ssl_used);
1151 ⟶}
```

```
1173 ␣␣␣␣␣␣␣␣if(_logger.isLoggable(Level.FINE))␣{
1174 ⟶␣␣␣␣_logger.log(Level.FINE,
1175 ⟶⟶⟶"SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1176 ⟶⟶⟶+␣"␣"␣+␣isSet(ssl_target_requires,␣EstablishTrustInClient.value)
1177 ⟶⟶⟶+␣"␣"␣+␣isSet(ssl_target_supports,␣EstablishTrustInClient.value));
1178 ⟶}
```

```
1189 ␣␣␣␣␣␣␣␣if(_logger.isLoggable(Level.FINE))␣{
1190 ⟶␣␣␣␣_logger.log(Level.FINE,
1191 ⟶⟶⟶"SecurityMechanismSelector.evaluate_client_conformance_ssl:␣true");
1192 ⟶}
```

Besides, the lines 1143, 1174 and 1190 are not indented while the lines 1151, 1178 and 1192 are not correctly aligned to the corresponding if at the lines, respectively, 1142, 1173 and 1189 (they are one 'tab'left).

```
1195 ␣␣␣␣␣␣}␣finally␣{
1196 ⟶␣␣if(_logger.isLoggable(Level.FINE))␣{
1197 ⟶␣␣␣␣␣␣_logger.log(Level.FINE,
1198 ⟶⟶⟶␣␣"SecurityMechanismSelector.evaluate_client_conformance_ssl<-:");
1199 ⟶␣␣}
```

In the last block of code, the content of the *'finally'*clause is not indented. After that, the line 1183 is not indented correctly with respect to the if blocks in which it is inserted.

```
1181 ␣␣␣␣␣␣␣␣␣␣␣␣if␣(␣!␣(␣isSet(ssl_target_requires,␣EstablishTrustInClient.value)
1182 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣||␣isSet(ssl_target_supports,␣EstablishTrustInClient.value)))
1183 ␣␣␣␣␣␣␣␣␣␣␣␣return␣false;␣//␣security␣mechanism␣did␣not␣match
```

Finally, the line 1194 and 1195 have an incorrect number of spaces.

```
1194 ␣␣␣␣␣␣␣␣return␣true␣;␣//␣mechanism␣matched
1195 ␣␣␣␣␣␣}␣finally␣{
```

1.3 Braces

Reading the code assigned to us, we observe that the author decides to follow the *Kernighan and Ritchie* style to write the parentheses[2]. It exists only one

[2]The same style is recommended by SonarQube's rules.

exception shown below.

```
1086    private boolean evaluate_client_conformance_ssl(
1087                        EjbIORConfigurationDescriptor iordesc,
1088                        boolean  ssl_used,
1089                        X509Certificate[] certchain)
1090    {
```

In the following lines of code are displayed the if or if-else blocks composed by only one instructions to execute not surrounded by braces.

```
1122        if (    isSet(ssl_target_requires, Integrity.value)
1123             || isSet(ssl_target_requires, Confidentiality.value)
1124             || isSet(ssl_target_requires, EstablishTrustInClient.value))
1125            ssl_required = true;
1126        else
1127            ssl_required = false;
```

```
1129        if ( ssl_target_supports != 0)
1130            ssl_supported = true;
1131        else
1132            ssl_supported = false;
```

```
1154            if (! (ssl_required || ssl_supported))
1155                return false;  // security mechanism did not match
```

```
1157            if (ssl_required)
1158                return false;  // security mechanism did not match
```

```
1181            if ( ! ( isSet(ssl_target_requires, EstablishTrustInClient.value)
1182                     || isSet(ssl_target_supports, EstablishTrustInClient.value)))
1183            return false; // security mechanism did not match
```

```
1185            if (isSet(ssl_target_requires, EstablishTrustInClient.value))
1186                return false; // security mechanism did not match
```

1.4 File Organization

In the method "*evaluate_client_conformance_ssl(. . .)*"at line 1086, the line 1182 has a length equal to 82 characters while the maximum allowed length is 80.

In addition the comment's block at lines 1102-1115 has a length between 82 and 85 characters. A solution can be split into two lines the header of the table.

1.5 Wrapping lines

In the method "*getSubjectFromSecurityCurrent( )*"at line 963, the following lines are not aligned with the starting of the string at the line above.

```
974             throw new SecurityMechanismException("Could␣not␣find␣" +
975                         "␣security␣information");
```

```
979             throw new SecurityMechanismException("Could␣not␣find␣" +
980                 "␣subject␣information␣in␣the␣security␣context.");
```

In the method "*selectSecurityMechanism(. . .)*"at line 999, the break-line at the line 999 should occur after the close curly bracket.

```
999     private CompoundSecMech selectSecurityMechanism(
1000                CompoundSecMech[] mechList) throws SecurityMechanismException {
```

Besides, the line 1016 is not correctly aligned to the starting of the string at the line above.

```
1015        throw new SecurityMechanismException("Cannot␣use␣any␣of␣the␣" +
1016            "␣target's␣supported␣mechanisms");
```

In the method "*useMechanism(. . .)*"at line 1019, the break-line at the lines 1023 and 1026 should occur after an operator and the following lines should be aligned to the open curly bracket.

```
1023        if (mech.sas_context_mech.supported_naming_mechanisms.length > 0
1024                && !isMechanismSupported(mech.sas_context_mech)) {
```

```
1026        } else if (mech.as_context_mech.client_authentication_mech.length > 0
1027                && !isMechanismSupportedAS(mech.as_context_mech)) {
```

In the method "*evaluate_client_conformance_ssl(. . .)*"at line 1086, there are many wrapping lines' errors.
First of all, the declaration of the method's parameters seems incorrect, but it

is acceptable and readable. Afterwards, into the following lines are shown the errors on the wrapping lines (they should occur after a comma or an operator) and on the alignment of the second line (it should be aligned at the expression's starting).

```
1093        _logger.log(Level.FINE,
1094        "SecurityMechanismSelector.evaluate_client_conformance_ssl->:");
```

```
1122          if (    isSet(ssl_target_requires, Integrity.value)
1123               || isSet(ssl_target_requires, Confidentiality.value)
1124               || isSet(ssl_target_requires, EstablishTrustInClient.value))
```

```
1143        _logger.log(Level.FINE,
1144        "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1145        + "␣" + isSet(ssl_target_requires, Integrity.value)
1146        + "␣" + isSet(ssl_target_requires, Confidentiality.value)
1147        + "␣" + isSet(ssl_target_requires, EstablishTrustInClient.value)
1148        + "␣" + ssl_required
1149        + "␣" + ssl_supported
1150        + "␣" + ssl_used);
```

```
1173          if(_logger.isLoggable(Level.FINE)) {
1174        _logger.log(Level.FINE,
1175        "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1176        + "␣" + isSet(ssl_target_requires, EstablishTrustInClient.value)
1177        + "␣" + isSet(ssl_target_supports, EstablishTrustInClient.value));
1178      }
```

```
1181              if ( ! ( isSet(ssl_target_requires, EstablishTrustInClient.value)
1182                       || isSet(ssl_target_supports, EstablishTrustInClient.value)))
```

```
1190        _logger.log(Level.FINE,
1191        "SecurityMechanismSelector.evaluate_client_conformance_ssl:␣true");
```

```
1197          _logger.log(Level.FINE,
1198          "SecurityMechanismSelector.evaluate_client_conformance_ssl<-:");
```

1.6 Comments

In the method "*getSubjectFromSecurityCurrent( )*"at line 963 and in the method "*useMechanism(. . .)*"at line 1019 there are no comments.
If we consider the whole class the commented-out code without a reason and a date (when the code can be deleted from the source file) is at the lines 128, 136, 150, 317-325, 396, 397, 423, 486-492, 685-708, 719-778, 792 and 806.

1.7 Java Source Files

TO DO

1.8 Package and Import Statements

TO DO

1.9 Class and Interface Declarations

TO DO

1.10 Initialization and Declarations

In the method "*getSubjectFromSecurityCurrent( )*"at line 963, the variables *sc* has an useless assignment, so the lines 965 and 966 should be merged. Since the variable's type-name and the name of the method used to initialize it are too long, the best way to write it is define the variable at the first line and then initialize it in the following line.

```
965         com.sun.enterprise.security.SecurityContext sc = null;
966         sc = com.sun.enterprise.security.SecurityContext.getCurrent();
```

The declaration of the variable *s* (line 977) should be moved at the beginning of the method, immediately after the declaration of the variable *sc*.

In the method "*selectSecurityMechanism(...)*"at line 999, the initialization of the variable *mech* at line 1007 is useless.
Finally, in the method "*evaluate_client_conformance_ssl(...)*"at line 1086, the local variables (lines 1087-1090) should be declared at the beginning of the method.

1.11 Method Calls

TO DO

1.12 Arrays

An array is used only in method "*selectSecurityMechanism(...)*"at line 999 and no error has been found.

1.13 Object Comparison

The comparisons between objects are all made with "*equals*"method and not with "=="operator. The only exceptions are when the second term of the equality is *null* or if the equality is between integer numbers.

1.14 Output Format

TO DO

1.15 Computation, Comparison and Assignments

TO DO

1.16 Exceptions

TO DO

1.17 Flow of Control

TO DO

1.18 Files

No method (between the four assigned to us) uses a file.

1.19 Other Errors

TO DO

Chapter 2

Other Info

This chapter contains information about the used tools and the hours of work by the members of the working group.

2.1 Tools

For this Code Inspection (CI) assignment the following tools were used:

- LaTeX and TexStudio editor
- Eclipse Mars 4.5 for Java EE
- Sonarqube 5.2

2.2 Working hours

Date	Costanzo's hours	Disabato's hours
2015/12/14	1.30h	1.30h
2015/12/15	1h	-
Total CI	2.30h	1.30h
Global	57.30h	56.30h

Appendix A

Code Inspection Checklist

In this appendix, the list of all points checked in our Code Inspection's analysis is provided split into the same sections of the chapter 1.

- **Naming Conventions**
  - All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
  - If one-character variables are used, they are used only for temporary throwaway variables, such as those used in for loops.
  - Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;
  - Interface names should be capitalized like classes.
  - Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().
  - Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.
  - Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;

- **Indention**
  - Three or four spaces are used for indentation and done so consistently.
  - No tabs are used to indent.
- **Braces**
  - 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).
  - All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example:

    ```
    if ( condition )
        doThis();
    ```

    The correct version is:

    ```
    if ( condition ) {
        doThis();
    }
    ```

- **File Organization**
  - Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
  - Where practical, line length does not exceed 80 characters.
  - When line length must exceed 80 characters, it does NOT exceed 120 characters.
- **Wrapping Lines**
  - Line break occurs after a comma or an operator.
  - Higher-level breaks are used.
  - A new statement is aligned with the beginning of the expression at the same level as the previous line.
- **Comments**
  - Comments are used to adequately explain what the class, interface,

methods, and blocks of code are doing.
  - 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.
- **Java Source Files**
  - Each Java source file contains a single public class or interface.
  - The public class is the first class or interface in the file.
  - Check that the external program interfaces are implemented consistently with what is described in the javadoc.
  - Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).
- **Package and Import Statements**
  - If any package statements are needed, they should be the first non-comment statements. Import statements follow.
- **Class and Interface Declarations**
  - The class or interface declarations shall be in the following order:
    * class/interface documentation comment
    * class or interface statement
    * class/interface implementation comment, if necessary
    * class (static) variables
      · first public class variables
      · next protected class variables
      · next package level (no access modifier)
      · last private class variables
    * instance variables
      · first public instance variables
      · next protected instance variables
      · next package level (no access modifier)
      · last private instance variables
    * constructors
    * methods
  - Methods are grouped by functionality rather than by scope or acces-

sibility.
  - Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.
- **Initialization and Declarations**
  - Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
  - Check that variables are declared in the proper scope.
  - Check that constructors are called when a new object is desired.
  - Check that all object references are initialized before use.
  - Variables are initialized where they are declared, unless dependent upon a computation.
  - 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.
- **Method Calls**
  - Check that parameters are presented in the correct order.
  - Check that the correct method is being called, or should it be a different method with a similar name.
  - Check that method returned values are used properly.
- **Arrays**
  - Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
  - Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
  - Check that constructors are called when a new array item is desired.
- **Object Comparisons**
  - Check that all objects (including Strings) are compared with *equals* and not with '=='.
- **Output Format**
  - Check that displayed output is free of spelling and grammatical errors.

  - Check that error messages are comprehensive and provide guidance as to how to correct the problem.
  - Check that the output is formatted correctly in terms of line stepping and spacing.
- **Computation, Comparisons and Assignments**
  - Check that the implementation avoids *brutish programming*.
  - Check order of computation/evaluation, operator precedence and parenthesizing.
  - Check the liberal use of parenthesis is used to avoid operator precedence problems.
  - Check that all denominators of a division are prevented from being zero.
  - Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
  - Check that the comparison and Boolean operators are correct.
  - Check throw-catch expressions, and check that the error condition is actually legitimate.
  - Check that the code is free of any implicit type conversions.
- **Exceptions**
  - Check that the relevant exceptions are caught.
  - Check that the appropriate action are taken for each catch block.
- **Flow of Control**
  - In a switch statement, check that all cases are addressed by break or return.
  - Check that all switch statements have a default branch.
  - Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.
- **Files**
  - Check that all files are properly declared and opened.
  - Check that all files are closed properly, even in the case of an error
  - Check that EOF conditions are detected and handled correctly
  - Check that all file exceptions are caught and dealt with accordingly

Appendix B

A Corrected Code Version

In this appendix, a possible corrected version of the code is provided for only the assigned methods. Two important clarifications follow. First of all, the number of the lines is not the same for two main reasons: first, here the numeration starts from 1 at the first assigned method (without considering all previous code); second, the numeration cannot be the same due to we have modified several lines of codes.
Then, we have added a new method to implement the logger. In fact, the code used to log some events is often duplicated in the assigned methods with a few differences. Hence, a better way to write that code is to define a new private method that includes the duplicated lines of code. In this way the code is more clear and readable and a change can be perform speedily and easily by a change on the method.

```
1   private Subject getSubjectFromSecurityCurrent()
2           throws SecurityMechanismException {
3       com.sun.enterprise.security.SecurityContext securityContext;
4       securityContext = com.sun.enterprise.security.SecurityContext.getCurrent();
5       if(securityContext == null) {
6           fineLevelLog(" SETTING GUEST ---");
7           securityContext = com.sun.enterprise.security.SecurityContext.init();
8       }
9       if(securityContext == null) {
10          throw new SecurityMechanismException("Could not find " +
11                                               "security information");
12      }
13      Subject subject = securityContext.getSubject();
```

```
14          if(subject == null) {
15              throw new SecurityMechanismException("Could not find " +
16                                                   "subject information in the " +
17                                                   "security context.");
18          }
19          fineLevelLog("Subject in security current:" + subject);
20          return subject;
21      }
22
23      public CompoundSecMech selectSecurityMechanism(IOR ior)
24              throws SecurityMechanismException {
25          CompoundSecMech[] mechList = getCtc().getSecurityMechanisms(ior);
26          CompoundSecMech mech = selectSecurityMechanism(mechList);
27          return mech;
28      }
29
30      /**
31       * Select the security mechanism from the list of compound security
32       * mechanisms.
33       */
34      private CompoundSecMech selectSecurityMechanism(CompoundSecMech[] mechList)
35                                  throws SecurityMechanismException {
36          // We should choose from list of compound security mechanisms
37          // which are in decreasing preference order. Right now we select
38          // the first one.
39          if(mechList == null || mechList.length == 0) {
40              return null;
41          }
42          CompoundSecMech mech;
43          for(int i = 0; i < mechList.length; i++) {
44              mech = mechList[i];
45              boolean useMech = useMechanism(mech);
46              if(useMech) {
47                  return mech;
48              }
49          }
50          throw new SecurityMechanismException("Cannot use any of the " +
51                                               "target's supported mechanisms");
52      }
53
54      private boolean useMechanism(CompoundSecMech mech) {
55          boolean val = true;
56          TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);
57
58          if (mech.sas_context_mech.supported_naming_mechanisms.length > 0 &&
59              !isMechanismSupported(mech.sas_context_mech)) {
60              return false;
61          } else if (mech.as_context_mech.client_authentication_mech.length > 0 &&
62                     !isMechanismSupportedAS(mech.as_context_mech)) {
```

```
63              return false;
64          }
65
66          if(tls == null) {
67              return true;
68          }
69          int targetRequires = tls.target_requires;
70          if(isSet(targetRequires, EstablishTrustInClient.value)) {
71              if(! sslUtils.isKeyAvailable()) {
72                  val = false;
73              }
74          }
75          return val;
76      }
77
78      private boolean evaluateClientConformanceSsl(
79                          EjbIORConfigurationDescriptor iordesc,
80                          boolean  sslUsed,
81                          X509Certificate[] certchain) {
82
83          boolean sslRequired  = false;
84          boolean sslSupported = false;
85          int sslTargetRequires = 0;
86          int sslTargetSupports = 0;
87
88          try {
89              fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
90                           "conformance_ssl->:");
91
92              /*********************************************************************
93               * Conformance Matrix:
94               *
95               * |---------------|-----------------|-----------------|-----------|
96               * | SSLClientAuth | targetrequires. | targetSupports. | Conformant|
97               * |               |       ETIC      |      ETIC       |           |
98               * |---------------|-----------------|-----------------|-----------|
99               * |     Yes       |          0      |      1          |    Yes    |
100              * |     Yes       |          0      |      0          |    No     |
101              * |     Yes       |          1      |      X          |    Yes    |
102              * |     No        |          0      |      X          |    Yes    |
103              * |     No        |          1      |      X          |    No     |
104              * |---------------|-----------------|-----------------|-----------|
105              *
106              *********************************************************************/
107
108             // gather the configured SSL security policies.
109
110             sslTargetRequires = this.getCtc().getTargetRequires(iordesc);
111             sslTargetSupports = this.getCtc().getTargetSupports(iordesc);
```

```
112
113             if (isSet(sslTargetRequires, Integrity.value) ||
114                 isSet(sslTargetRequires, Confidentiality.value) ||
115                 isSet(sslTargetRequires, EstablishTrustInClient.value)) {
116                 sslRequired = true;
117             } else {
118                 sslRequired = false;
119             }
120
121             if ( sslTargetSupports != 0) {
122                 sslSupported = true;
123             } else {
124                 sslSupported = false;
125             }
126
127             /* Check for conformance for using SSL usage.
128              *
129              * a. if SSL was used, then either the target must require or
130              *    support SSL. In the latter case, SSL is used because of client
131              *    policy.
132              * b. if SSL was not used, then the target must not require it
133              *    either. The target may or may not support SSL (it is
134              *    irrelevant).
135              */
136             fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
137                          "conformance_ssl:" +
138                          " " + isSet(sslTargetRequires, Integrity.value) +
139                          " " + isSet(sslTargetRequires, Confidentiality.value) +
140                          " " +
141                          isSet(sslTargetRequires,EstablishTrustInClient.value) +
142                          " " + sslRequired +
143                          " " + sslSupported +
144                          " " + sslUsed);
145
146             if (sslUsed) {
147                 if (! (sslRequired || sslSupported)) {
148                     return false;  // security mechanism did not match
149                 }
150             } else {
151                 if (sslRequired) {
152                     return false;  // security mechanism did not match
153                 }
154             }
155
156             /* Check for conformance for SSL client authentication.
157              *
158              * a. if client performed SSL client authentication, then the target
159              *    must either require or support SSL client authentication. If
160              *    the target only supports, SSL client authentication is used
```

```
161             *    because of client security policy.
162             *
163             * b. if SSL client authentication was not used, then the target must
164             *    not require SSL client authentcation either. The target may or may
165             *    not support SSL client authentication (it is irrelevant).
166             */
167
168            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
169                         "conformance_ssl:" +
170                         " " +
171                         isSet(sslTargetRequires,EstablishTrustInClient.value) +
172                         " " +
173                         isSet(sslTargetSupports,EstablishTrustInClient.value));
174
175            if (certchain != null) {
176                if ( ! ( isSet(sslTargetRequires, EstablishTrustInClient.value) ||
177                     isSet(sslTargetSupports, EstablishTrustInClient.value))) {
178                    return false; // security mechanism did not match
179                }
180            } else {
181                if (isSet(sslTargetRequires, EstablishTrustInClient.value)) {
182                    return false; // security mechanism did not match
183                }
184            }
185
186            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
187                         "conformance_ssl: true");
188
189            return true ; // mechanism matched
190        } finally {
191            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
192                         "conformance_ssl<-:");
193        }
194    }
195
196    //At the end of the class or into a specific class dedicated to the logger
197    private fineLevelLog (String s) {
198        if(_logger.isLoggable(Level.FINE)) {
199            _logger.log(Level.FINE, s);
200        }
201    }
```

Listing B.1: "A corrected version of the code."