

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.

¹The 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 has 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 parentheses². It exists only one

²The 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