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 made 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 requires to read and understand 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 has a meaningless name. A meaningful name could be *toReturn*.

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

See the section 1.15 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␣{
```

The correct indention of all the methods is available into the **??**.

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 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, **if** or **if-else** blocks composed by only one instruction to execute are 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
```

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

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 to split the header of the table into two lines.

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 there are 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 present 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

The javadoc of the methods *"getSubjectFromSecurityCurrent( )"*, *"useMechanism(...)"* and *"evaluate_client_conformance_ssl(...)"* is missing.
Finally, the javadoc of the method *"selectSecurityMechanism(...)"* is not complete: the only available information are about the method description (parameters, exception and return value description are missing).

1.8 Package and Import Statements

No errors related to this topic has been found.

1.9 Class and Interface Declarations

The convention related to the class attributes' declaration order is not respected. The errors are two. First, the variable used to log (*_logger*) has a class visibility, so it should be declared after **public** and **static** attributes at line 127.

```
124     private static final java.util.logging.Logger _logger =
125        LogDomains.getLogger(SecurityMechanismSelector.class, LogDomains.SECURITY_LOGGER
               );
126
127     public static final String CLIENT_CONNECTION_CONTEXT = "ClientConnContext";
```

The second one is on the variable *localString*. Since it is declared **private** and **static**, it should be written before the non-**static** and **private** variables at the lines 130 and 131.

```
130     private  Set<EjbIORConfigurationDescriptor> corbaIORDescSet = null;
131     private  boolean sslRequired = false;
```

```
138     private static final LocalStringManagerImpl localStrings =
139         new LocalStringManagerImpl(SecServerRequestInterceptor.class);
```

The order of the methods into the class is in general correct because they are grouped by a common role and purpose.
The analysis about the duplicated codes has been perfomed with the use of SonarQube 5.2 tool. It detects two methods that have the same code for the most part of the contents. The methods are *getSSLPort(...)* and *getSSLPorts(...)* with the following headers:

```
221     /**
222      * This method determines if SSL should be used to connect to the
223      * target based on client and target policies. It will return null if
224      * SSL should not be used or an SocketInfo containing the SSL port
225      * if SSL should be used.
226      */
227     public SocketInfo getSSLPort(IOR ior, ConnectionContext ctx)
```

```
344     public java.util.List<SocketInfo> getSSLPorts(IOR ior, ConnectionContext ctx)
```

The duplicated blocks are two: one at the lines 230-249 copied at the lines 346-365 and the other one at the lines 254-276 replaced at the lines 373-395. These blocks of code are too long, so they are not shown here. In our opinion the two methods perform the same actions and they differ for the name (the second one is the plural of the first one) and for the return value.

Finally, we report the complexity analysis of the SonarQube tool. The *Cyclomatic Complexity* of the class is 334 which is greater than 200 authorized. Considering the methods the maximum allowed *Cyclomatic Complexity* is 10 (even if in some cases it can be accepted a violation of this rule). The methods that do not respect this bound are:

- *getSSLPort(...)* at line 227 which has 25;
- *getSSLPorts(...)* at line 344 which has 27;
- *propagateIdentity(...)* at line 614 which has 20;

- *getUsernameAndPassword(. . .)* at line 788 which has 19;
- *getIdentity()* at line 882 which has 16;

1.10 Initialization and Declarations

In the method "*getSubjectFromSecurityCurrent( )*"at line 963, the variable *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 to 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

No errors related to this topic has been found.

1.12 Arrays

An array is used only in method "*selectSecurityMechanism(. . .)*"at line 999 and no errors have 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

In the method "*getSubjectFromSecurityCurrent( )*"at line 963, the following two strings have two spaces between the words split by the break-line.

```
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 same error has been detected.

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

1.15 Computation, Comparison and Assignments

In the method "*selectSecurityMechanism(. . .)*" at line 999, the variable *useMech* is useless, since the boolean return value assumed by it is used only in the following **if** condition. Thus, the called function can be directly write into the **if** condition. (Brutish programming)

```
1010            boolean useMech = useMechanism(mech);
1011            if(useMech) {
```

In the method "*useMechanism(. . .)*" at line 1019, the variable *val* is used only once and it can be replaced by a direct use of the **return** clause. According to the Brutish programming a better way to write the block at the lines 1035-1040 is

return condition, where the condition is obtained by a '*negated*' merge of the two nested **if**'s conditions (the reason for the negation is the value of *val* into the **if**).

```
1035        if(isSet(targetRequires, EstablishTrustInClient.value)) {
1036            if(! sslUtils.isKeyAvailable()) {
1037                val = false;
1038            }
1039        }
1040        return val;
```

The block 1023-1029 contains an **if-else** statement with the same body both into the **if** and into the **else if** part. A better way to write this code is obtained by merging the two conditions into one.

```
1023        if (mech.sas_context_mech.supported_naming_mechanisms.length > 0
1024                && !isMechanismSupported(mech.sas_context_mech)) {
1025            return false;
1026        } else if (mech.as_context_mech.client_authentication_mech.length > 0
1027                && !isMechanismSupportedAS(mech.as_context_mech)) {
1028            return false;
1029        }
```

In the method "*evaluate_client_conformance_ssl(...)*" the initialization of the variables at the lines 1097-1100 are useless (the default value for a boolean variable is false and for an integer variable is zero), but they can be accepted to make the code more readable.

According to Brutish Programming all the *return true/false* are correct because they are not written into an **if-else** clause. Due to the same rule, the assignments to the boolean variable at lines 1122-1127 and 1129-1132 can be written directly in the form *variable = condition*.

```
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;
1128
1129        if ( ssl_target_supports != 0)
1130            ssl_supported = true;
1131        else
1132            ssl_supported = false;
```

Finally, the two blocks at the lines 1153-1159 and 1180-1186 contain nested **if** that can be merged the first one (they have the same body).

```
1153        if (ssl_used) {
1154            if (! (ssl_required || ssl_supported))
1155                return false;  // security mechanism did not match
1156        } else {
1157            if (ssl_required)
1158                return false;  // security mechanism did not match
1159        }
```

```
1180        if (certchain != null) {
1181            if ( ! ( isSet(ssl_target_requires, EstablishTrustInClient.value)
1182                     || isSet(ssl_target_supports, EstablishTrustInClient.value)))
1183            return false; // security mechanism did not match
1184        } else {
1185            if (isSet(ssl_target_requires, EstablishTrustInClient.value))
1186                return false; // security mechanism did not match
```

1.16 Exceptions

No errors related to this topic have been found.

1.17 Flow of Control

No errors related to this topic have been found.

1.18 Files

No methods (between the four assigned to us) use a file.

1.19 Other Errors

In the method *"evaluate_client_conformance_ssl(...)"*there is a big **try** block without any **catch** block and with the **finally** block. Since the use of the **finally** is to ensure a log writing we have many doubts on this choice.