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 28, 2015

Contents

Chapter 1

Assignment

In this chapter we will present the class that has been assigned to us. First, the lines of code to be analysed are presented without any comment. Afterwards, a brief description of the class role and function is presented. Obviously this is made by us and it is based only on the code and on the documentation provided by the authors.

1.1 Assigned class

The class assigned to us is called "**SecurityMechanismSelector**" and is located in the following path relative to the root of Glassfish project:
appserver/security/ejb.security/src/main/java/com/sun/enterprise/iiop/security/

The package which the class belongs to is: *com.sun.enterprise.iiop.security*
To present more clearly the class we include the first lines of code containing the documentation of the class and its declaration:

```
/**
 * This class is responsible for making various decisions for selecting
 * security information to be sent in the IIOP message based on target
 * configuration and client policies.
 * Note: This class can be called concurrently by multiple client threads.
 * However, none of its methods need to be synchronized because the methods
 * either do not modify state or are idempotent.
 *
```

```java
 * @author Nithya Subramanian

 */

@Service
@Singleton
public final class SecurityMechanismSelector implements PostConstruct {
```

Listing 1.1: Class documentation and declaration

1.1.1 Functional role

As the documentation states, this class is responsible for selecting the security mechanism based on target configuration and client policies. It reads the data related to a client and to a target from the current running context of the application.

1.2 Assigned methods

The methods assigned to us are 4, that are presented below, with the entire code and with our description of their functional role.

1.2.1 First method: getSubjectFromSecurityCurrent

```java
    private Subject getSubjectFromSecurityCurrent()
            throws SecurityMechanismException {
        com.sun.enterprise.security.SecurityContext sc = null;
        sc = com.sun.enterprise.security.SecurityContext.getCurrent();
        if(sc == null) {
            if(_logger.isLoggable(Level.FINE)) {
          _logger.log(Level.FINE," SETTING GUEST ---");
            }
            sc = com.sun.enterprise.security.SecurityContext.init();
        }
        if(sc == null) {
            throw new SecurityMechanismException("Could not find " +
                        " security information");
        }
        Subject s = sc.getSubject();
        if(s == null) {
            throw new SecurityMechanismException("Could not find " +
                " subject information in the security context.");
        }
        if (_logger.isLoggable(Level.FINE)) {
            _logger.log(Level.FINE, "Subject in security current:" + s);
        }
        return s;
    }
```

Listing 1.2: First assigned method

This method is supposed to return the subject of the current security context. A Subject represents a grouping of related information for a single entity, such as a person. Such information includes the Subject's identities as well as its security-related attributes (passwords and cryptographic keys, for example).

1.2.2 Second method: selectSecurityMechanism

```
    private CompoundSecMech selectSecurityMechanism(
                CompoundSecMech[] mechList) throws SecurityMechanismException {
        // We should choose from list of compound security mechanisms
        // which are in decreasing preference order. Right now we select
        // the first one.
        if(mechList == null || mechList.length == 0) {
            return null;
        }
        CompoundSecMech mech = null;
        for(int i = 0; i < mechList.length; i++) {
            mech = mechList[i];
            boolean useMech = useMechanism(mech);
            if(useMech) {
                return mech;
            }
        }
        throw new SecurityMechanismException("Cannot use any of the " +
            " target's supported mechanisms");
    }
```

Listing 1.3: Second assigned method

As the documentation states, this method selects and returns the first[1] supported compound security mechanism from an array given as parameter. The

[1]It returns only the first supported security mechanism found in the array because the **for** cycle stops with **return** instruction when found

mechanism to use is retrieved by calling the method *useMechanism(CompoundSecMech mech)*. If no security mechanism of the array can be used an exception is thrown.

1.2.3 Third method: useMechanism

```java
    private boolean useMechanism(CompoundSecMech mech) {
        boolean val = true;
        TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);

        if (mech.sas_context_mech.supported_naming_mechanisms.length > 0
                && !isMechanismSupported(mech.sas_context_mech)) {
            return false;
        } else if (mech.as_context_mech.client_authentication_mech.length > 0
                && !isMechanismSupportedAS(mech.as_context_mech)) {
            return false;
        }

        if(tls == null) {
            return true;
        }
        int targetRequires = tls.target_requires;
        if(isSet(targetRequires, EstablishTrustInClient.value)) {
            if(! sslUtils.isKeyAvailable()) {
                val = false;
            }
        }
        return val;
    }
```

Listing 1.4: Third assigned method

This method checks whether a security mechanism (given as parameter) can be used in the communication process between the client and the target or not. The method returns the boolean value **true** if the client request respects the target configuration, i.e. if the security mechanism required is supported by the target system, there are no errors and if the client can use the protocol TLS (Transport Layer Security) when the target system requires it for the desired security mechanism. Otherwise, if any of the conditions above is not satisfied, the method returns the boolean value **false**.

1.2.4 Fourth method: evaluate_client_conformance_ssl

```java
    private boolean evaluate_client_conformance_ssl(
                        EjbIORConfigurationDescriptor iordesc,
                        boolean  ssl_used,
                        X509Certificate[] certchain)
    {
      try {
        if(_logger.isLoggable(Level.FINE)) {
      _logger.log(Level.FINE,
      "SecurityMechanismSelector.evaluate_client_conformance_ssl->:");
  }

        boolean ssl_required  = false;
        boolean ssl_supported = false;
        int ssl_target_requires = 0;
        int ssl_target_supports = 0;

        /*************************************************************************
         * Conformance Matrix:
         *
         * |---------------|---------------------|---------------------|------------|
         * | SSLClientAuth | targetrequires.ETIC | targetSupports.ETIC | Conformant |
         * |---------------|---------------------|---------------------|------------|
         * |     Yes       |          0          |      1              |    Yes     |
         * |     Yes       |          0          |      0              |    No      |
         * |     Yes       |          1          |      X              |    Yes     |
         * |     No        |          0          |      X              |    Yes     |
         * |     No        |          1          |      X              |    No      |
         * |---------------|---------------------|---------------------|------------|
         *
         *************************************************************************/

        // gather the configured SSL security policies.

        ssl_target_requires = this.getCtc().getTargetRequires(iordesc);
        ssl_target_supports = this.getCtc().getTargetSupports(iordesc);

        if (    isSet(ssl_target_requires, Integrity.value)
             || isSet(ssl_target_requires, Confidentiality.value)
             || isSet(ssl_target_requires, EstablishTrustInClient.value))
            ssl_required = true;
        else
            ssl_required = false;

        if ( ssl_target_supports != 0)
            ssl_supported = true;
        else
            ssl_supported = false;

        /* Check for conformance for using SSL usage.
         *
         * a. if SSL was used, then either the target must require or support
         *    SSL. In the latter case, SSL is used because of client policy.
         *
         * b. if SSL was not used, then the target must not require it either.
         *    The target may or may not support SSL (it is irrelevant).
         */
        if(_logger.isLoggable(Level.FINE)) {
      _logger.log(Level.FINE,
      "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
      + " " + isSet(ssl_target_requires, Integrity.value)
      + " " + isSet(ssl_target_requires, Confidentiality.value)
      + " " + isSet(ssl_target_requires, EstablishTrustInClient.value)
      + " " + ssl_required
      + " " + ssl_supported
      + " " + ssl_used);
  }

        if (ssl_used) {
            if (! (ssl_required || ssl_supported))
                return false;  // security mechanism did not match
        } else {
            if (ssl_required)
                return false;  // security mechanism did not match
        }

        /* Check for conformance for SSL client authentication.
         *
         * a. if client performed SSL client authentication, then the target
         *    must either require or support SSL client authentication. If
         *    the target only supports, SSL client authentication is used
         *    because of client security policy.
         *
         * b. if SSL client authentication was not used, then the target must
         *    not require SSL client authentcation either. The target may or may
         *    not support SSL client authentication (it is irrelevant).
         */

        if(_logger.isLoggable(Level.FINE)) {
      _logger.log(Level.FINE,
      "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
      + " " + isSet(ssl_target_requires, EstablishTrustInClient.value)
      + " " + isSet(ssl_target_supports, EstablishTrustInClient.value));
  }

        if (certchain != null) {
            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
1187    }
1188
1189    if(_logger.isLoggable(Level.FINE)) {
1190   _logger.log(Level.FINE,
1191   "SecurityMechanismSelector.evaluate_client_conformance_ssl: true");
1192 }
1193
1194    return true ; // mechanism matched
1195   } finally {
1196  if(_logger.isLoggable(Level.FINE)) {
1197    _logger.log(Level.FINE,
1198    "SecurityMechanismSelector.evaluate_client_conformance_ssl<-:");
1199  }
1200   }
1201  }
```

Listing 1.5: Fourth assigned method

This method evaluates the conformance of the use of the protocol SSL (Secure Sockets Layer) in the authentication process between the client and the target. The client can ask the authentication via SSL to the target. The target may support or not the SSL authentication and may strictly require it or not.
There are five possible cases in which the client authentication request is conformant to the target configuration or not, based on the possible conditions described before. This conformance cases as described in a table inside a comment block from line 1102 and 1115, in the method's code above. The first column of the table shows whether the client asks for the SSL authentication or not, the second column shows whether the target requires it or not, the third columns shows whether the target supports SSL or not and finally the fourth column shows whether the client request is conformant to the target configuration or not.
The returned boolean value of the method is **true** if the client request for SSL is conformant, **false** otherwise.

Chapter 2

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.

2.1 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 2.15 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);
```

2.2 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 Appendix B.

2.3 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.

```
    private boolean evaluate_client_conformance_ssl(
                        EjbIORConfigurationDescriptor iordesc,
                        boolean  ssl_used,
                        X509Certificate[] certchain)
    {
```

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

```
        if (    isSet(ssl_target_requires, Integrity.value)
             || isSet(ssl_target_requires, Confidentiality.value)
             || isSet(ssl_target_requires, EstablishTrustInClient.value))
            ssl_required = true;
        else
            ssl_required = false;
```

```
        if ( ssl_target_supports != 0)
            ssl_supported = true;
        else
            ssl_supported = false;
```

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

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

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

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

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

2.4 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.

2.5 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<-:");
```

2.6 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.

2.7 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).

2.8 Package and Import Statements

No errors related to this topic has been found.

2.9 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;

2.10 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.

2.11 Method Calls

No errors related to this topic has been found.

2.12 Arrays

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

2.13 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.

2.14 Output Format

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

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

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

In the method “*selectSecurityMechanism(. . .)*”at line 999, the same error has been detected.

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

2.15 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)

```
            boolean useMech = useMechanism(mech);
            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
```

2.16 Exceptions

No errors related to this topic have been found.

2.17 Flow of Control

No errors related to this topic have been found.

2.18 Files

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

2.19 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 about this choice.

Chapter 3

Other Info

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

3.1 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[1]

[1]Used only for a first general analysis of the code. After that, the inspection has been performed manually point by point.

3.2 Working hours

Date	Costanzo's hours	Disabato's hours
2015/12/14	1.30h	1.30h
2015/12/15	1h	-
2015/12/18	3h	-
2015/12/23	1.30h	1.30h
2015/12/24	-	1h
2015/12/26	-	5h
2015/12/28	2h	-
Total CI	9h	9h
Global	64h	64h

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 2.

- **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. Three 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.
Finally, to make the following code more *nice for the eyes*, the space special characters are not shown, but obviously all tabs have been replaced with four spaces.

```
    private Subject getSubjectFromSecurityCurrent()
            throws SecurityMechanismException {
        com.sun.enterprise.security.SecurityContext securityContext;
        securityContext = com.sun.enterprise.security.SecurityContext.getCurrent();
        if(securityContext == null) {
            fineLevelLog(" SETTING GUEST ---");
            securityContext = com.sun.enterprise.security.SecurityContext.init();
        }
        if(securityContext == null) {
            throw new SecurityMechanismException("Could not find " +
                                                         "security information");
        }
        Subject subject = securityContext.getSubject();
        if(subject == null) {
            throw new SecurityMechanismException("Could not find " +
                                                 "subject information in the " +
                                                 "security context.");
        }
        fineLevelLog("Subject in security current:" + subject);
        return subject;
    }

    public CompoundSecMech selectSecurityMechanism(IOR ior)
            throws SecurityMechanismException {
        CompoundSecMech[] mechList = getCtc().getSecurityMechanisms(ior);
        CompoundSecMech mech = selectSecurityMechanism(mechList);
        return mech;
    }

    /**
     * Select the security mechanism from the list of compound security
     * mechanisms.
     */
    private CompoundSecMech selectSecurityMechanism(CompoundSecMech[] mechList)
                                throws SecurityMechanismException {
        // We should choose from list of compound security mechanisms
        // which are in decreasing preference order. Right now we select
        // the first one.
        if(mechList == null || mechList.length == 0) {
            return null;
        }
        CompoundSecMech mech;
        for(int i = 0; i < mechList.length; i++) {
            mech = mechList[i];
            if( useMechanism(mech) ) {
                return mech;
            }
        }
        throw new SecurityMechanismException("Cannot use any of the " +
                                             "target's supported mechanisms");
    }

    private boolean useMechanism(CompoundSecMech mech) {
        TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);

        if ( (mech.sas_context_mech.supported_naming_mechanisms.length > 0 &&
             !isMechanismSupported(mech.sas_context_mech)) ||
             (mech.as_context_mech.client_authentication_mech.length > 0 &&
             !isMechanismSupportedAS(mech.as_context_mech))) {
```

```
            return false;
        }

        if(tls == null) {
            return true;
        }
        int targetRequires = tls.target_requires;
        return ! (isSet(targetRequires, EstablishTrustInClient.value) && ! sslUtils.
            isKeyAvailable());
    }

    private boolean evaluateClientConformanceSsl(
                        EjbIORConfigurationDescriptor iordesc,
                        boolean  sslUsed,
                        X509Certificate[] certchain) {

        boolean sslRequired = false;
        boolean sslSupported = false;
        int sslTargetRequires = 0;
        int sslTargetSupports = 0;

        try {
            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
                         "conformance_ssl->:");

            /*******************************************************************
             * Conformance Matrix:
             *
             * |---------------|-----------------|-----------------|-----------|
             * | SSLClientAuth | targetrequires. | targetSupports. | Conformant|
             * |               |       ETIC      |      ETIC       |           |
             * |---------------|-----------------|-----------------|-----------|
             * |     Yes       |          0      |      1          |    Yes    |
             * |     Yes       |          0      |      0          |    No     |
             * |     Yes       |          1      |      X          |    Yes    |
             * |     No        |          0      |      X          |    Yes    |
             * |     No        |          1      |      X          |    No     |
             * |---------------|-----------------|-----------------|-----------|
             *
             *******************************************************************/

            // gather the configured SSL security policies.

            sslTargetRequires = this.getCtc().getTargetRequires(iordesc);
            sslTargetSupports = this.getCtc().getTargetSupports(iordesc);

            sslRequired = (isSet(sslTargetRequires, Integrity.value) ||
                           isSet(sslTargetRequires, Confidentiality.value) ||
                           isSet(sslTargetRequires,EstablishTrustInClient.value));

            sslSupported = ( sslTargetSupports != 0);

            /* Check for conformance for using SSL usage.
             *
             * a. if SSL was used, then either the target must require or
             *    support SSL. In the latter case, SSL is used because of client
             *    policy.
             * b. if SSL was not used, then the target must not require it
             *    either. The target may or may not support SSL (it is
             *    irrelevant).
             */
            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
                         "conformance_ssl:" +
                         " " + isSet(sslTargetRequires, Integrity.value) +
                         " " + isSet(sslTargetRequires, Confidentiality.value) +
                         " " +
                         isSet(sslTargetRequires,EstablishTrustInClient.value) +
                         " " + sslRequired +
                         " " + sslSupported +
                         " " + sslUsed);

            if ((sslUsed && !(sslRequired || sslSupported)) || sslRequired) {
                return false;
            }

            /* Check for conformance for SSL client authentication.
             *
             * a. if client performed SSL client authentication, then the target
             *    must either require or support SSL client authentication. If
             *    the target only supports, SSL client authentication is used
             *    because of client security policy.
             *
             * b. if SSL client authentication was not used, then the target must
             *    not require SSL client authentcation either. The target may or may
             *    not support SSL client authentication (it is irrelevant).
             */

            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
                         "conformance_ssl:" +
                         " " +
                         isSet(sslTargetRequires,EstablishTrustInClient.value) +
                         " " +
                         isSet(sslTargetSupports,EstablishTrustInClient.value));

            if ((certchain != null &&
                 !(isSet(sslTargetRequires, EstablishTrustInClient.value) ||
                 isSet(sslTargetSupports, EstablishTrustInClient.value))) ||
                (isSet(sslTargetRequires, EstablishTrustInClient.value))) {
```

```java
                    return false; // security mechanism did not match
                }

                fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
                             "conformance_ssl: true");

                return true ; // mechanism matched
            } finally {
                fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
                             "conformance_ssl<-:");
            }
        }

        //At the end of the class or into a specific class dedicated to the logger
        private fineLevelLog (String s) {
            if(_logger.isLoggable(Level.FINE)) {
                _logger.log(Level.FINE, s);
            }
        }
```

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