JPass Static Testing
Verificação e Validação de Software 2020–2021

Manuel Tomás 51054
Tiago Varela 51017

2021/01/04

1 Introduction

JPass is a simple tool for encrypting strings. It allows the user to store passwords and usernames, or even any other information contained in short strings, then encrypt it. It encrypts created entries behind one master password, allowing users to protect their information and to use JPass as a password management tool, encrypting their multiple passwords behind a secure master one.

2 Code Structure

- **jpass** is the main package of the program, and similarly contains the main class that launches the GUI.
- **jpass.crypt** is a package that holds classes that implement different types of encryption, such as AES or CBC(using AES).
- **jpass.crypt.io** is a subpackage that holds the classes specifically for reading and writing from encrypted inputs and outputs.
- **jpass.data** is a package that holds classes related to the storage of data. For example, the DataModel class models the documents containing the information input by the user and EntriesRepository writes the entries it receives into files, storing them.
- **jpass.ui** is a package that holds all the classes related to the representation of the user UI, including to subpackages to better organise them for this purpose.
- **jpass.util** is a package that holds classes with utility functions for the various areas of the program. For example, it holds classes for the configuration of the program and utility functions related to cryptography, strings or other important components of the jpass tool.

- **jpass.xml.bind** is a package that holds classes related to the storage of information that represents xml entries. It contains classes which are data models for storing the information in a format similar to xml.
- **jpass.xml.converter** is a package that holds classes for writing and reading XML from a document representation holding this XML content.

3 Static Testing

This Static Testing is being performed for the purposes of increasing the maintainability of the code and detecting faults. The regular use of Static Testing in the code will help ensure that the code is correctly structured and that faults are detected at an earlier stage and without need for heavier testing or execution of the programs. Furthermore, developers that perform Static Testing may learn from the experience, avoiding repeating some of the same mistakes in the future. For this reason, tools for Static Testing have been developed, greatly facilitating and automating areas of the process. The use of these tools will allow the automatic detection of certain patterns in the code that tend to be the source of faults or otherwise bad practices that may lead to future faults.

3.1 Tool Used

To address the static testing of the project JPass we first thought about using SonarQube as it is a tool that we are more familiarized with. We ended up not using this tool because: (1) to produce a file with the reports in text form we would need to make a program that generated it, as SonarQube only offers a url that can be used to get the bugs found in the JSON format or a webGUI where all the bugs are shown graphically; (2) and because we could not find any "tutorial" that demonstrated how to integrate it with Maven directly in the pom.xml file.
As such, we decided to use the SpotBugs tool as it had a simple and direct tutorial of how to integrate it with Maven and it was presented in class so we had an idea of how it worked.

Tool Setup. In order to set up the tool the following lines where added to the plugins list in the "pom.xml" file.

```
<plugin>
    <groupId>com.github.spotbugs</groupId>
    <artifactId>spotbugs-maven-plugin</artifactId>
    <version>4.2.0</version>
    <dependencies>
        <!-- overwrite dependency on spotbugs if you want to specify the version of spotbugs -->
        <dependency>
            <groupId>com.github.spotbugs</groupId>
            <artifactId>spotbugs</artifactId>
            <version>4.2.1</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <configuration>
                <classFilesDirectory></classFilesDirectory>
                <outputDirectory></outputDirectory>
                <spotbugsXmlOutputDirectory>spotbugs/tests</spotbugsXmlOutputDirectory>
                <testClassFilesDirectory></testClassFilesDirectory>
                <xmlOutput></xmlOutput>
                <xmlOutputDirectory>spotbugs/tests</xmlOutputDirectory>
            </configuration>
            <phase>compile</phase>
            <goals>
                <goal>spotbugs</goal>
                <goal>gui</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

These lines add the tool plugin to the project build and in the "execution" tag define what to run, when to run it, and which parameters to use. Respectively, in the "goals" tag, in the "phase" tag and in the "configuration" tag. The only default values we overwrote were the Output Directories in order to choose a safe directory to have the files generated by SpotBugs.

The default configuration for the SpotBugs tool was used for the reports. In order to determine whether any adjustments were necessary for the project we generated an initial report and examined it so as to determine if any of the detected categories were of not important to the project at this stage.

The generated report detected 13 suspicious areas. Upon a preliminary investigation of the categories detected we determined they were all relevant for the project, as seen in section 4. Since all the detected categories were relevant for good maintainability, portability or design of the project, and since the detected patterns were relatively few, we decided to keep the default values which produced these results.

3.2 Bugs Found

SpotBugs found 13 bugs in the first analysis of the code, in this section we present five of them and their solution if any.

BUG 1.

Category	Dodgy Code
Detailed Classification	Useless Code
Package	jpass.util
Class	StringUtils
Method	stripNonValidXMLCharacters
Line	60

Detailed Description:

In this line there is a conditional check which SpotBugs identified as useless code.

The bug in question refers to how this check compares the value of a char to an int value that is beyond the bounds of possible int values for chars in Java. That is, it checks if a char is >= 10000 in hexadecimal, or 65536 in decimal. However, due to the Java encoding of 16-bit Unicode characters, the maximum possible decimal value is 65535.

The documentation reveals the values being checked follow the specifications found in [1], in which it indicates all Unicode characters are found within the ranges being checked here. Therefore, the method serves its purpose of detecting if the characters are all validXMLCharacters and removing any that are not.

As for the whether the check is truly unnecessary, SpotBugs may be incorrect in its assessment as UTF-16 codification of characters in the U+10000 to U+10FFFF range is possible through the use of surrogate pairs: The character is saved as two units of 16-bit code, and then converted as necessary by checking the pair and the corresponding display associated with it [2] [3].
For this reason, we do not consider this a bug, and even if it turns out SpotBugs is correct that the given situation will never occur, the cost is likely worth it, so as to not leave the possibility of a bug later down the line.
Fix: Does not need to be fixed.

BUG 2.

Category	Correctness
Detailed Classification	Null pointer dereference
Package	jpass.ui
Class	SvgImageIcon
Method	update
Line	71

Detailed Description:

In this line SpotBugs identifies that url might be null.

SpotBugs identifies a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed and mentions that the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed and that deciding that is beyond the ability of SpotBugs.

As we saw that this line was written inside a try-catch block, we think that the author thought this could be an error and caught it.
Fix: There is no fault, as it is dealt with by the code.

BUG 3:

Category	Bad Practice
Detailed Classification	Format string
Package	jpass.ui.helper
Class	FileHelper
Method	exportFile
Line	127

Detailed Description:

In this line SpotBugs identifies that the string OPERATION_ERROR_MESSAGE contains a \n and a %n should be used instead.

SpotBugs mentions that in format Strings it's generally preferable to use %n instead of \n, which will produce the platform-specific line separator.
Fix: Replace \n with %n in the given string.

```
private static final String OPERATION_ERROR_MESSAGE =
        "An error occured during the %s operation:\n%s";
```

Figure 1: format string before fix

```
private static final String OPERATION_ERROR_MESSAGE =
        "An error occured during the %s operation:%n%s";
```

Figure 2: format string after fix

BUG 4:

Category	Experimental
Detailed Classification	Unsatisfied obligation to clean uo java.io.InputStream on checked exception
Package	jpass.util
Class	Configuration
Method	constructor
Line	53

Detailed Description:

In this line SpotBugs identifies that an InputStream is created inside a try-catch block.

SpotBugs mentions that if an exception is thrown before reaching line 55 where the close() method is called, the InputStream will not be closed. SpotBugs indicates a solution to this problem using a finally clause in the try-catch block which is always executed so it is secure to close the stream there. Another solution would be to use a try-with-resources which automatically closes the stream. SpotBugs does not show this as a solution to the problem, so it is possible that, when the tool was implemented, try-with-resource was not yet in the java api.
Fix: Introduce a try-catch-finally for the affected InputStream variable.

```
private Configuration() {
    try {
        File filePath = new File(getConfigurationFolderPath(), "jpass.properties");
        if (filePath.exists() && filePath.isFile()) {
            InputStream is = new FileInputStream(filePath);
            properties.load(is);
            is.close();
        }
    } catch (Exception e) {
        LOG.log(Level.WARNING, "An error occurred during loading configuration.", e);
    }
}
```

Figure 3: configuration Constructor before fix

```
private Configuration() {
    try {
        File filePath = new File(getConfigurationFolderPath(), "jpass.properties");
        if (filePath.exists() && filePath.isFile()) {
            InputStream is = null;
            try {
                is = new FileInputStream(filePath);
                properties.load(is);
                is.close();
            }catch(Exception e) {
                //just to close the Stream even if there is an error
            }finally {
                if(is != null) {
                    is.close();
                }
            }
        }
    } catch (Exception e) {
        LOG.log(Level.WARNING, "An error occurred during loading configuration.", e);
    }
}
```

Figure 4: configuration Constructor after fix

BUG 5:

Category	Malicious Code Vulnerability
Detailed Classification	Method returning array may expose internal representation
Package	jpass.data
Class	DataModel
Method	getPassword
Line	123

Detailed Description:

SpotBugs has detected that a mutable variable is returned through this get, which would allow any code making use of this public function to alter the contents of the variables in the DataModel class. A similar issue applies, and was detected by SpotBugs, with setPassword.

The problem is a real one. However, since the application is self-contained and does not interact with any external code, the issue is not critical. We furthermore checked the current usage of the function, and it is never used with the intent of modifying the password, so there is no indication this behaviour is intended. Ultimately, this is not a critical bug, but good practice dictates that it may be fixed now to prevent future issues, such as the class being reused in different projects or contexts.
Fix: A clone of the array should be returned, rather than the reference to the mutable array itself. If the variable is null, the null value should be returned.

```
public byte[] getPassword() {
    return this.password;
}
```

Figure 5: getPassword method before fix

```
public byte[] getPassword() {
    return this.password == null ? null : this.password.clone();
}
```

Figure 6: getPassword method after fix

4 SpotBugs Report

Two test reports were produced: $test - report1.html$ and $test - report2.html$.
The former report was produced initially, using the default configuration for SpotBugs, whereas the latter was produced after we studied 5 bugs selected randomly (that weren't the exact same type of bug).

As for the format of the html reports themselves, they contain a summary with the amount of warnings found, as well as the amount of each individual type.
The summary is followed by a listing of all warnings sorted by type, where clicking each warning provides a more detailed description of the cause of the warning is provided.
Finally, the warnings are followed by a details section in which each category of warning is described, as well as possible suggestions.

We analysed $test - report1$ preliminary to ensure we wanted the SpotBugs configuration to remain as is. $test - report1$ contains the following warnings:

- **EI_EXPOSE_REP & EI_EXPOSE_REP2** - The EXPOSE warnings refer to code vulnerabilities. Internal class parameters than can be accessed or modified externally due to some vulnerability. As this tool is meant to be used for security, we considered relevant to maintain this warning. Two warnings have to do with Strings (**FORMAT** and **USE_STRINGBUFFER**). Since the tool specifically works with text, constantly writing and reading, we considered these warnings relevant.
- **VA_FORMAT_STRING_USES_NEWLINE & SBSC_USE_STRINGBUFFER_CONCATENATION** - Two warnings have to do with Strings (**FORMAT** and **USE_STRINGBUFFER**). Since the tool specifically works with text, constantly writing and reading, we considered these warnings relevant.
- **NP_NULL_ON_SOME_PATH** - The NULL warning is important as there is always the possibility of a fault that crashes or stops the program existing here.
- **OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE** - The **UNSATISFIED_OBLIGATION** warning refers to issues such as input streams never being closed. Since a program such a jpass frequently makes use of streams for dealing with input and output, we considered this type of warning relevant.
- **UC_USELESS_CONDITION_TYPE** - The **USELESS_CONDITION** warning refers to conditions in if statements than can never be true or that can never be false. This type of warning is perhaps the least directly relevant, but since it may detect faults where a condition contains a type or an unintended situation that may break the program, we considered this type of warning relevant.

The list of warnings in the test report contained 13 different detected instances. From that list, 5 bugs of different warning types were randomly selected. We further elaborate

on these bugs in section 3.2, but three of them were fixed. With these new changes we proceeded generate a new report, $test - report2$.

$test-report2$ contains only 6 detections. Considering only three bugs were fixed, the reason the number has reduced so drastically is because the **VA_FORMAT_ STRING_USES_NEWLINE** warning type indicated 5 different locations where an issue was found, but all of them were the result of a single string in a variable. By fixing the string, all 5 instances of the detected bug were fixed. Furthermore, our other two fixes successfully resolved the issues SpotBugs detecting, leading to a total of 13-7 detections.

References

[1] W3C. Extensible markup language (xml) 1.0. [Online]. Available: http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char

[2] Oracle. Supplementary characters in the java platform. [Online]. Available: https://www.oracle.com/technical-resources/articles/javase/supplementary.html

[3] Wikipedia. Utf-16. [Online]. Available: https://en.wikipedia.org/wiki/UTF-16#Code_points_from_U010000_to_U10FFFF