REPORT

• INTRODUCTION

1. Static analysis software used:
   SpotBugs

2. URL:
   https://spotbugs.github.io/

3. Documentation:
   https://spotbugs.readthedocs.io/en/latest/

4. Brief:
   Spotbugs is a static code analysis tool which finds the bugs patterns in Java Code. It is a free software and requires minimum of 512MB of memory. It requires JDK 1.8 or latest, but it can also accept any complied java code from any version.
   (Additional information: It is said to be spiritual successor of FindBugs as FindBugs no longer has the support and it is not updated to the latest bug-patterns trends, while Spotbugs is the has a support team that rolls out new updates of software for newly discovered bug-patterns)

5. "How does Spotbugs work?"
   or
   "How do static code analysis software work?":
   Spotbugs basically finds the bug patterns that are usually considered to be insecure. It does so by looking at the bytecode and creating the abstract model of the entire program/project. It then analyses this against predefined bug patterns to spot the bug.

6. What does it take as an input?
   Spotbugs take bytecode as its input. To be specific, it can take .jar or .class files as its input. The plugin I used in my Eclipse IDE is intelligent enough to find the .jar or .class (even by just specifying directories) of the project that I chose to analyze using Spotbugs.

7. What “bugs” does it find and what bug it does not find?
   While using Spotbugs on different codes I realized it is very useful to find the defects very easily. For example, it was very likely that programmers can use a single “=” while checking conditions instead of “==”, this can be found by these static analysis tools very easily in C or C++. Since the Spotbugs is the static analysis software that just looks at the bytecode, it sometimes struggles to find the very complex runtime exceptions as it just looks at the predefined bug patterns. The best thing about bug-pattern analysis tools is that they can be very fast even with the large amount of the code, or even with the code that is partially complete.

8. How does it find bugs?
   This is typically performed by matching the coding pattern in the program with the expected bug pattern. Spotbugs after getting the bytecode as the input, checks against its Abstract Syntax Tree. This is how static analysis tool that depend on bug-pattern matching find the defects.

- ## SOURCE CODES ANALYZED

These are the bugs that I found in the three different codes that I chose to analyze. First was the test.java provided in the class, in which Spotbugs managed to find 10 bug instances.

I then used a public java code (PublicCode.java) as a practice to learn more from this static analysis tool.

Lastly, when I finally understood everything about it, I used it against a long multithreaded java code (Server.java) that I had written last semester as a part of File-Synchronization application, where I managed to find many bug patterns.

- # BUGS FOUND

(From publicCode.java)

1. **SBSC_USE_STRINGBUFFER_CONCATENATION**
   Description: At line 17 (Screenshot below), in PublicCode.java, the test() method is building a String using concatenation in a loop. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

2. **UUF_UNUSED_FIELD**
   Description: At line 6 (Screenshot below), in PublicCode.java, This is because the field is never used.

(From Server.java)

1. **DM_BOOLEAN_CTOR**
   Description: Creating new instances of java.lang.Boolean wastes memory, since Boolean objects are immutable and there are only two useful values of this type.

```
39
40              File syncDirectory = new File(baseDirectory);
41              Boolean syncDirectoryPresent = syncDirectory.exists();
42
43              if(!syncDirectoryPresent)
44                  syncDirectory.mkdir();
45
46              oos.writeObject(new Boolean(syncDirectoryPresent));
47              oos.flush();
48
```

2. **DM_STRING_CTOR**

   Description: Invoking inefficient String constructor.

```
 99                File baseDirectoryFile = new File(baseDirectory);
100                if(syncDirectoryPresent)
101                    visitAllDirsAndFiles(baseDirectoryFile);
102
103                oos.writeObject(new String("DONE"));
104                oos.flush();
105                System.out.println("Sync Complete!");
106            oos.close();
107            ois.close();
108            sock.close();
```

3. **NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE**

   Description: Null value possible for children.length as a return value from dir.list()

```
198    private static void delete(File dir) {
199        if (dir.isDirectory()) {
200            String[] children = dir.list();
201            for (int i=0; i<children.length; i++) {
202                delete(new File(dir, children[i]));
203            }
204        }
```

4. **OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE**

   It looks for stream to be closed inside try-catch block, where is can help cleaning the stream if exception arises before closing the stream.

```
165    private static void sendFile(File dir) throws Exception {
166        byte[] buff = new byte[sock.getSendBufferSize()];
167        int bytesRead = 0;
168        InputStream in = new FileInputStream(dir);
169        while((bytesRead = in.read(buff))>0) {
170            oos.write(buff,0,bytesRead);
171        }
172        in.close();
173        oos.flush();
174        reConnect();
175    }
```

5. **ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD**
   Description: (Screenshot below on Server.java at line 23)
   This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

(From Test.java)

1. **DMI_EMPTY_DB_PASSWORD**
   Description: (Screenshot below on Test.java at line 28)
   This code creates a database connect using a blank or empty password. This indicates that the database is not protected by a password.

2. **DM_DEFAULT_ENCODING**
   Description: Reliance on default encoding can cause different results on different platforms.
   Line 60:

```
58          File rf=new File("Welcome.txt");
59          Writer w=null;
60              w=new PrintWriter(rf);
61              w.write("Welcome to Secure Programing");
62              w.flush();
63              w.close();
64          String sha1 = null;
65          String tohash="mypassword";
```

3. **NM_CLASS_NAMING_CONVENTION**
   Description: Class names should start with an upper case letter.

```
13  public class test{
14      public static void main(String[] args ) throws IOException {
15          Connection con=null;
16          if(args.length <1) {
17              System.out.println("No arguments given");
18              System.exit(1);
19          }
20          //Returns the Runtime object
21          Runtime rt = Runtime.getRuntime();
22          String[] cmd = new String[3];
23          cmd[0] = "cmd.exe" ;
24              cmd[1] = "/C";
25              cmd[2] = "dir " + args[0];
26              rt.exec(cmd);
```

4. **ODR_OPEN_DATABASE_RESOURCE**
   Description: Method may fail to close database resource
   (Screenshot below on Test.java at line 28,29)

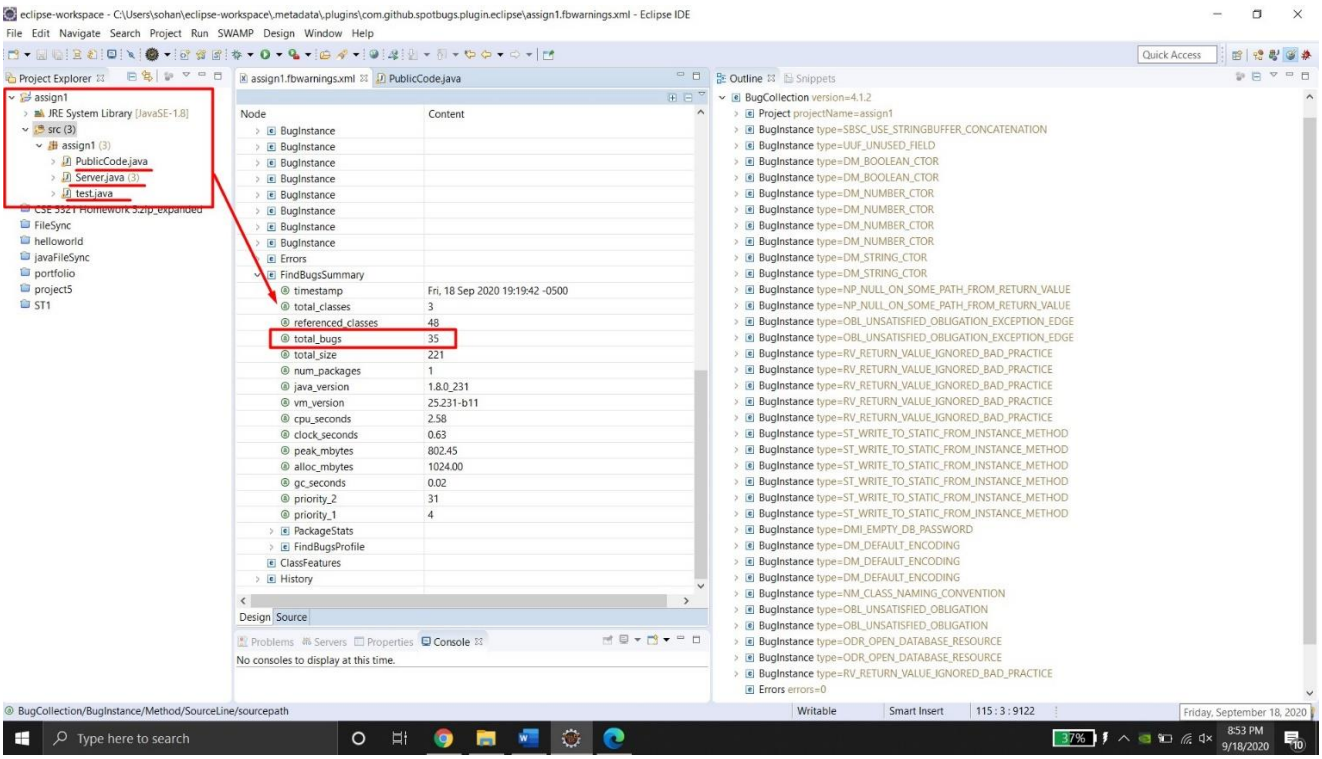
5. **RV_RETURN_VALUE_IGNORED_BAD_PRACTICE**
   Description: Method fails to check the return value. It is a bad practice.
   Line 45.

```
40          Scanner scan=new Scanner(System.in);
41          System.out.print("Enter file path");
42          String fpath=scan.nextLine();
43          if(fpath.startsWith("/archive/"))   {
44              File file=new File(fpath);
45              file.delete();
46          }
```

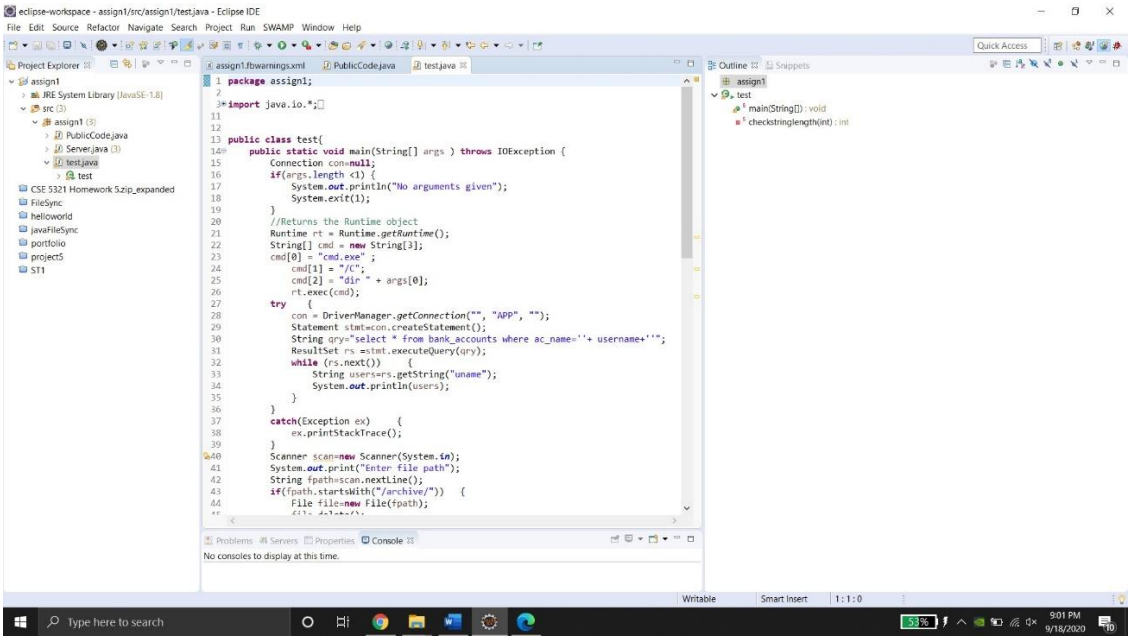
- SCREENSHOTS

I am adding code samples and true positive list of bugs that Spotbugs helped me to analyze.

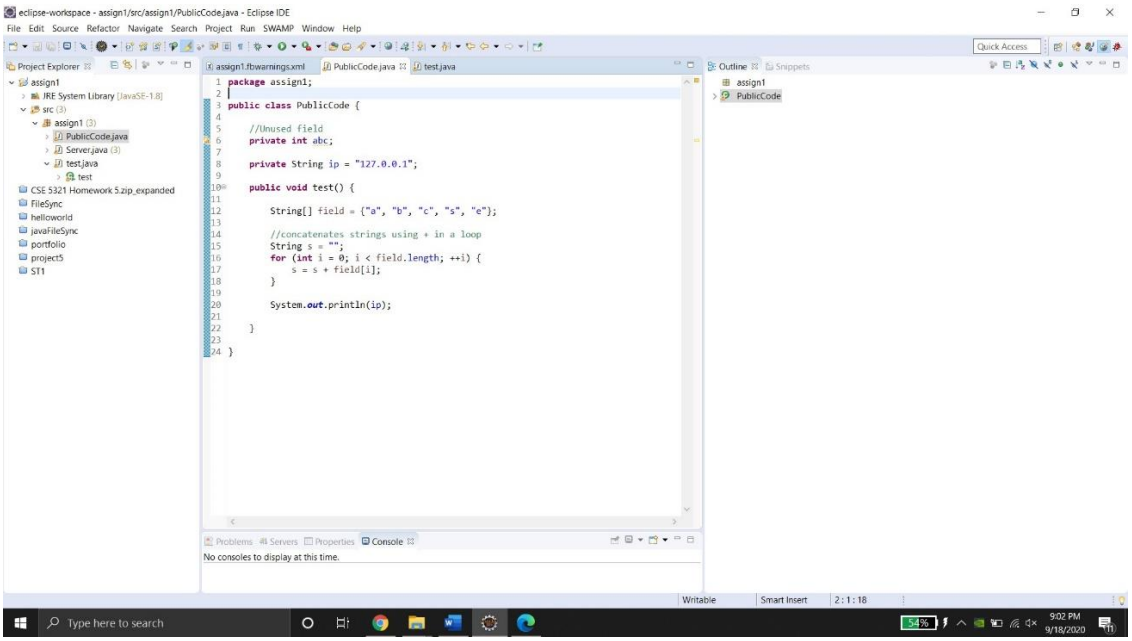
1. Overview Screenshot of the analysis report:



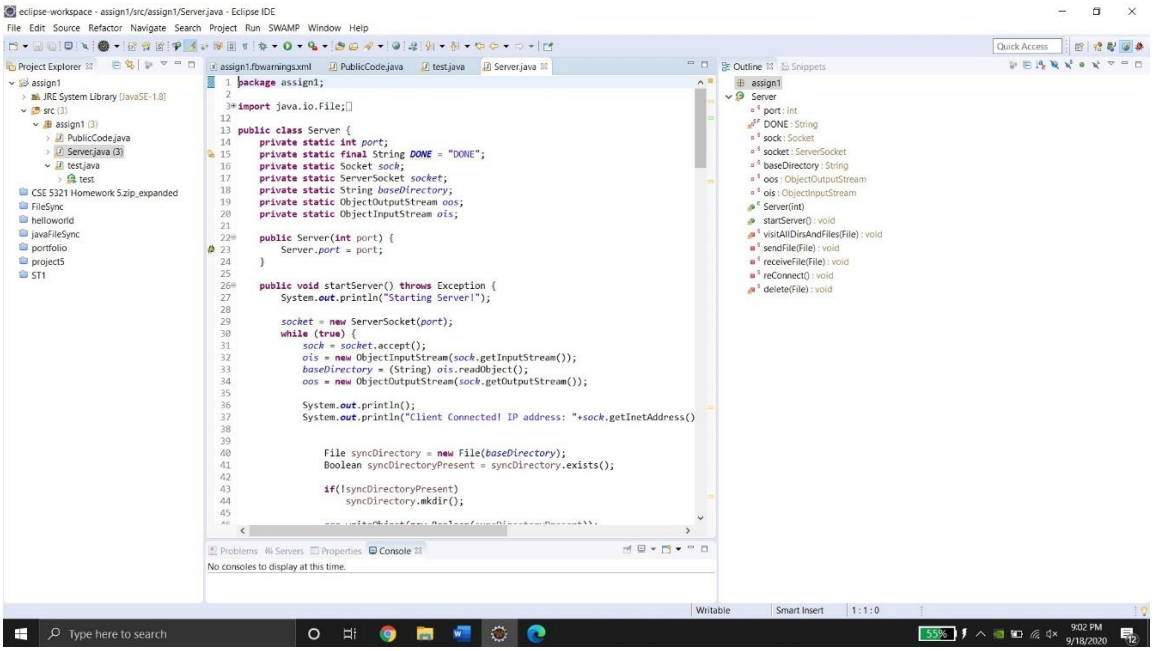
2. Test.java



3. PublicCode.java



4. Server.java



- # REFERENCES

1. http://findbugs.sourceforge.net/bugDescriptions.html
2. https://spotbugs.readthedocs.io/en/latest/introduction.html
3. http://www.opensourceforu.com/2011/09/joy-of-programming-technology-behind-static-analysis-tools/
4. http://fileadmin.cs.lth.se/cs/Education/EDAN70/CompilerProjects/2015/Reports/ErikssonKuhs.pdf
5. https://samate.nist.gov/docs/Evaluating_Bug_Finders_COUFLESS_2015.pdf
6. https://www.perforce.com/blog/qac/what-are-false-positives-and-false-negatives
7. https://www.ibm.com/developerworks/library/j-findbug1/#:~:text=FindBugs%20is%20flexible%20about%20what,or%20a%20list%20of%20directories.&text=The%20optional%20attribute%20output%20specifies,xml%20%2C%20text%20%2C%20or%20emacs%20.
8. https://www.synopsys.com/blogs/software-security/static-analysis-tools-finding-bugs/#:~:text=Static%20analysis%20tools%20do%20not,as%20any%20environment%20related%20issues.

THANK YOU

Name: Sohan Badade
ID: 1001729097