Jorge Avila

Professor Trey

Secure Programming

23 April 2021

Understanding and Using Static Code Analysis

Task 1: - PART 1

In this task we are going to understand the different types of tools that will aid us in fortifying our code to better secure standards, as we have been reviewing in this course. In order to commence, I want to preface this by the following java code named, "*SimpleWebServer.java*", in which it is used to illustrate security vulnerabilities.

```
/**********************************************************************

SimpleWebServer.java


This toy web server is used to illustrate security vulnerabilities.
This web server only supports extremely simple HTTP GET requests.

This file is also available at http://www.learnsecurity.com/ntk

**********************************************************************/

package com.learnsecurity;

import java.io.*;
import java.net.*;
import java.util.*;

public class SimpleWebServer {
```

```
/* Run the HTTP server on this TCP port. */
private static final int PORT = 8080;

/* The socket used to process incoming connections
   from web clients */
private static ServerSocket dServerSocket;

public SimpleWebServer () throws Exception {
    dServerSocket = new ServerSocket (PORT);
}

public void run() throws Exception {
    while (true) {
        /* wait for a connection from a client */
        Socket s = dServerSocket.accept();

        /* then process the client's request */
        processRequest(s);
    }
}

/* Reads the HTTP request from the client, and
   responds with the file the user requested or
   a HTTP error code. */
public void processRequest(Socket s) throws Exception {
    /* used to read data from the client */
    BufferedReader br =
            new BufferedReader (
                    new InputStreamReader (s.getInputStream()));

    /* used to write data to the client */
    OutputStreamWriter osw =
            new OutputStreamWriter (s.getOutputStream());

    /* read the HTTP request from the client */
    String request = br.readLine();

    String command = null;
    String pathname = null;

    /* parse the HTTP request */
    StringTokenizer st =
            new StringTokenizer (request, " ");

    command = st.nextToken();
    pathname = st.nextToken();
```

```
      if (command.equals("GET")) {
     /* if the request is a GET
        try to respond with the file
        the user is requesting */
          serveFile (osw,pathname);
      }
      else {
     /* if the request is a NOT a GET,
        return an error saying this server
        does not implement the requested command */
          osw.write ("HTTP/1.0 501 Not Implemented\n\n");
      }

      /* close the connection to the client */
      osw.close();
  }

  public void serveFile (OutputStreamWriter osw,
                         String pathname) throws Exception {
      FileReader fr=null;
      int c=-1;
      StringBuffer sb = new StringBuffer();

/* remove the initial slash at the beginning
   of the pathname in the request */
      if (pathname.charAt(0)=='/')
          pathname=pathname.substring(1);

/* if there was no filename specified by the
   client, serve the "index.html" file */
      if (pathname.equals(""))
          pathname="index.html";

      /* try to open file specified by pathname */
      try {
          fr = new FileReader (pathname);
          c = fr.read();
      }
      catch (Exception e) {
     /* if the file is not found,return the
        appropriate HTTP response code  */
          osw.write ("HTTP/1.0 404 Not Found\n\n");
          return;
      }

/* if the requested file can be successfully opened
   and read, then return an OK response code and
```

```
      send the contents of the file */
        osw.write ("HTTP/1.0 200 OK\n\n");
        while (c != -1) {
            sb.append((char)c);
            c = fr.read();
        }
        osw.write (sb.toString());
    }

    /* This method is called when the program is run from
       the command line. */
    public static void main (String argv[]) throws Exception {

        /* Create a SimpleWebServer object, and run it */
        SimpleWebServer sws = new SimpleWebServer();
        sws.run();
    }
}
```

My findings, in which I think are vulnerabilities:

1. There is a problem on how we are throwing and catching errors - we never clean the streams and need to make sure we close regardless if it works or not, since at

the start we opened it.

```
/* try to open file specified by pathname */
try {
    fr = new FileReader (pathname);
    c = fr.read();
}
catch (Exception e) {
    /* if the file is not found,return the
       appropriate HTTP response code  */
    osw.write ("HTTP/1.0 404 Not Found\n\n");
    return;
}

/* if the requested file can be successfully opened
   and read, then return an OK response code and
   send the contents of the file */
osw.write ("HTTP/1.0 200 OK\n\n");
while (c != -1) {
    sb.append((char)c);
    c = fr.read();
}
osw.write (sb.toString());
```

2.

```
public void serveFile (OutputStreamWriter osw,
                       String pathname) throws Exception {
   FileReader fr=null;
   int c=-1;
   StringBuffer sb = new StringBuffer();

   /* remove the initial slash at the beginning
      of the pathname in the request */
   if (pathname.charAt(0)=='/')
       pathname=pathname.substring(1);

   /* if there was no filename specified by the
      client, serve the "index.html" file */
   if (pathname.equals(""))
       pathname="index.html";

   /* try to open file specified by pathname */
   try {
       fr = new FileReader (pathname);
```

We have this above screenshot that has the vulnerability to allow user input and cause some sort of manipulation that may lead to a security breach

The way you need to do the file first is by the following, this is how we are going to use it on the GUI. This will then detect, as in class when you try to do a class without a main, it will look for any vulnerabilities.

```
Jorginhos-Mac:~ jorgejuarez$ cd Desktop/Spring2021/Secure\ Programming/A10-Understanding\ and\ Using
\ Static\ Code\ Analysis\ Tools/
Jorginhos-Mac:A10-Understanding and Using Static Code Analysis Tools jorgejuarez$ javac SimpleWebSer
ver.java
Jorginhos-Mac:A10-Understanding and Using Static Code Analysis Tools jorgejuarez$ ls
SimpleWebServer.class                   Static_Code_Analysis_Assignment.pdf
SimpleWebServer.java
Jorginhos-Mac:A10-Understanding and Using Static Code Analysis Tools jorgejuarez$
```

The plugins: The way I downloaded this is by going to the website listed inside the report which is this: https://find-sec-bugs.github.io/tutorials.htm

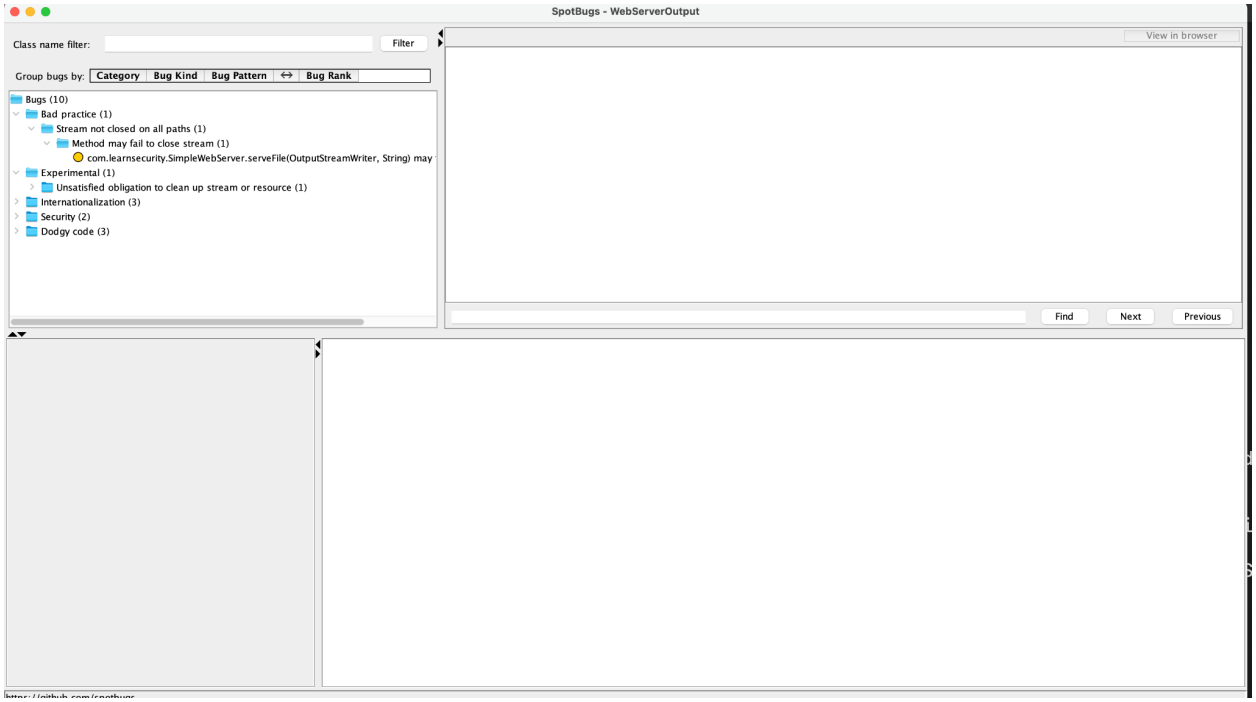
And actually running the process is detailed here:

https://spotbugs.readthedocs.io/en/stable/running.html

But of course I can quickly go over it:

First we are going to download and chmod +x on the spotbug and spotbug2 files that are listed in the lib folder so that it can be executed properly, without it that will not work. Also on the plugin, after you go ahead and download that you are to then put it in the plugin folder that is located in that folder that was downloaded. In order to run this you click on spotbug.jar file that is located in the lib and it would run automatically, keep in mind I have a macOS that executes the following.

The way the GUI looks like is as below:



You can see the output and comments on the left side of the panel.

And then as we try to run them on the GUI below is what we yield:

SpotBugs Report

Project Information

Project: WebServerOutput

SpotBugs version: 4.2.3

Code analyzed:

- /Users/jorgejuarez/Desktop/Spring2021/Secure Programming/A10-Understanding and Using Static Code Analysis Tools/SimpleWebServer.class

Metrics

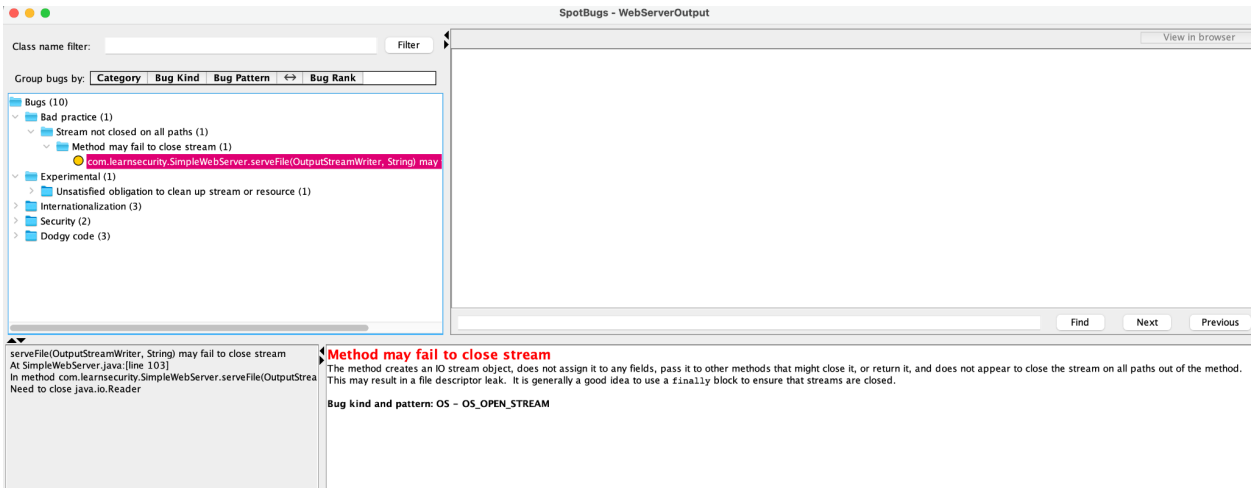
) lines of code analyzed, in 0 classes, in 1 packages.

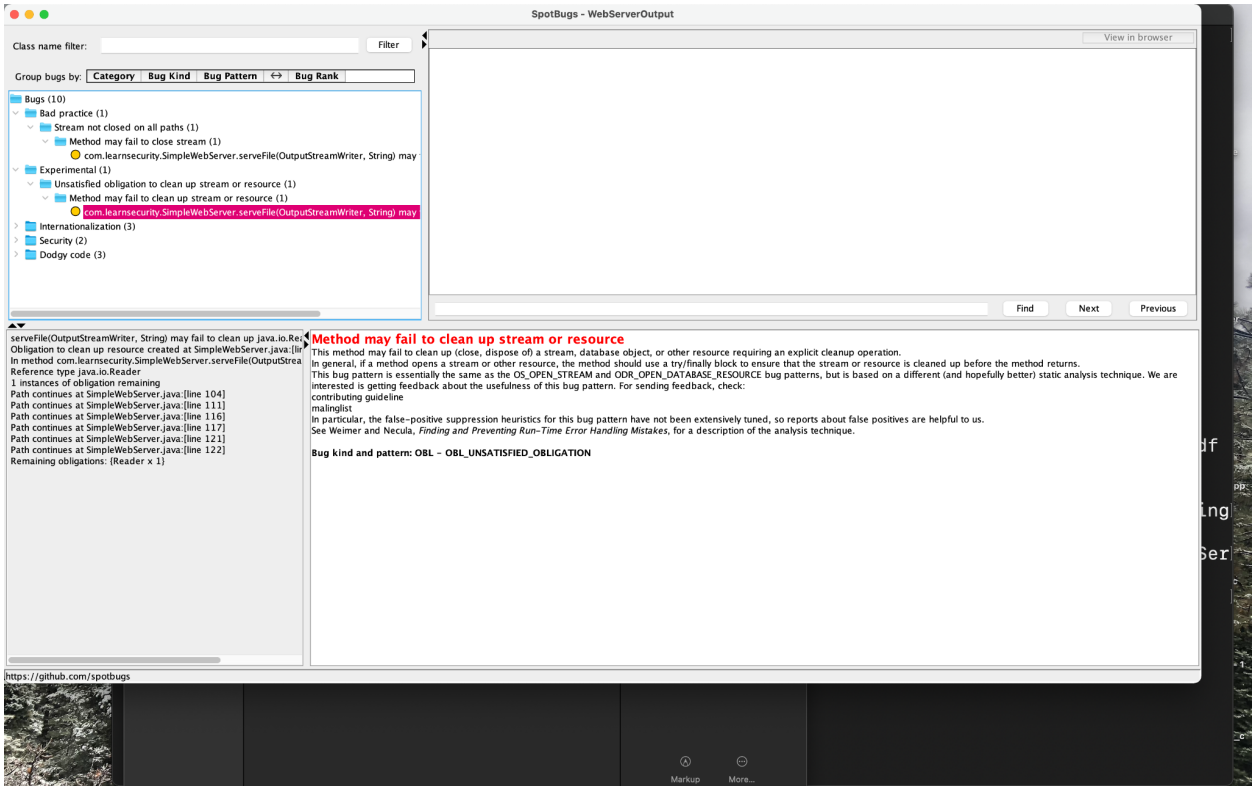
Metric	Total	Density*
High Priority Warnings	5	0.00
Medium Priority Warnings	4	0.00
Low Priority Warnings	1	0.00
Total Warnings	**10**	**0.00**

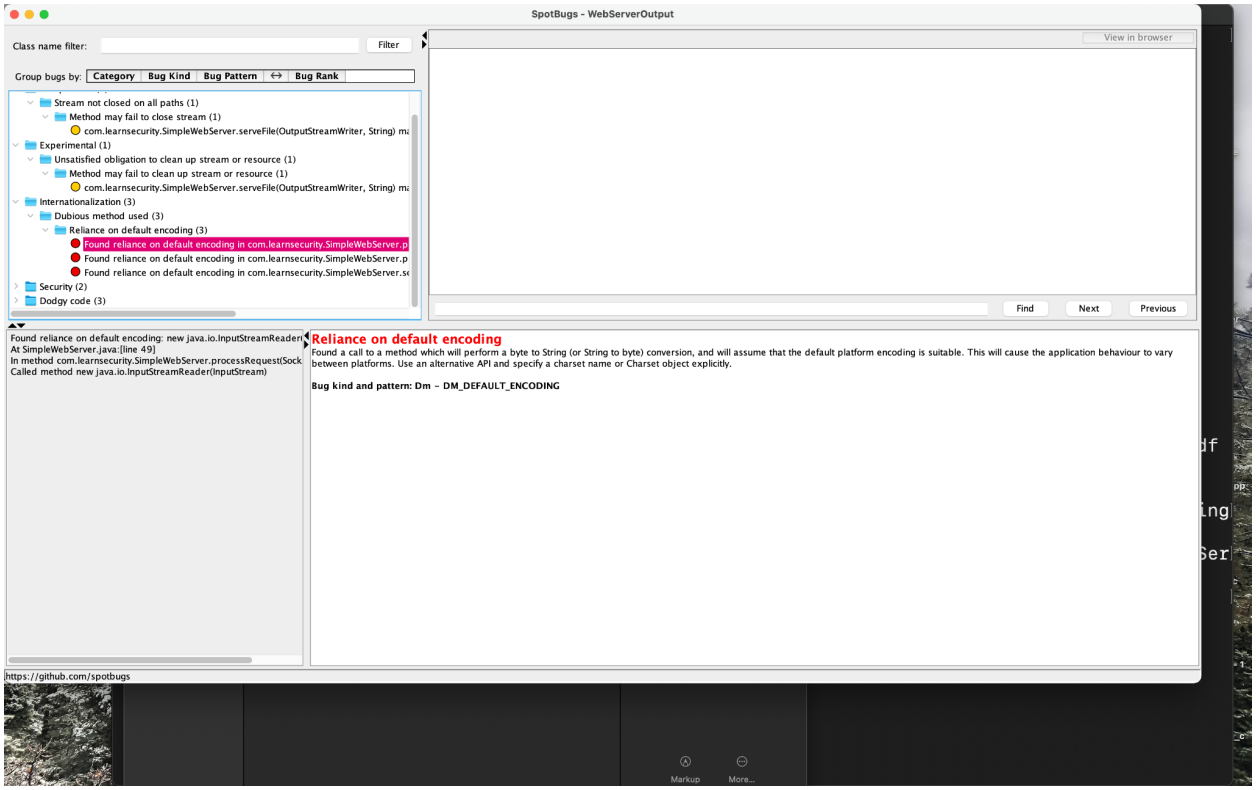
** Defects per Thousand lines of non-commenting source statements)*

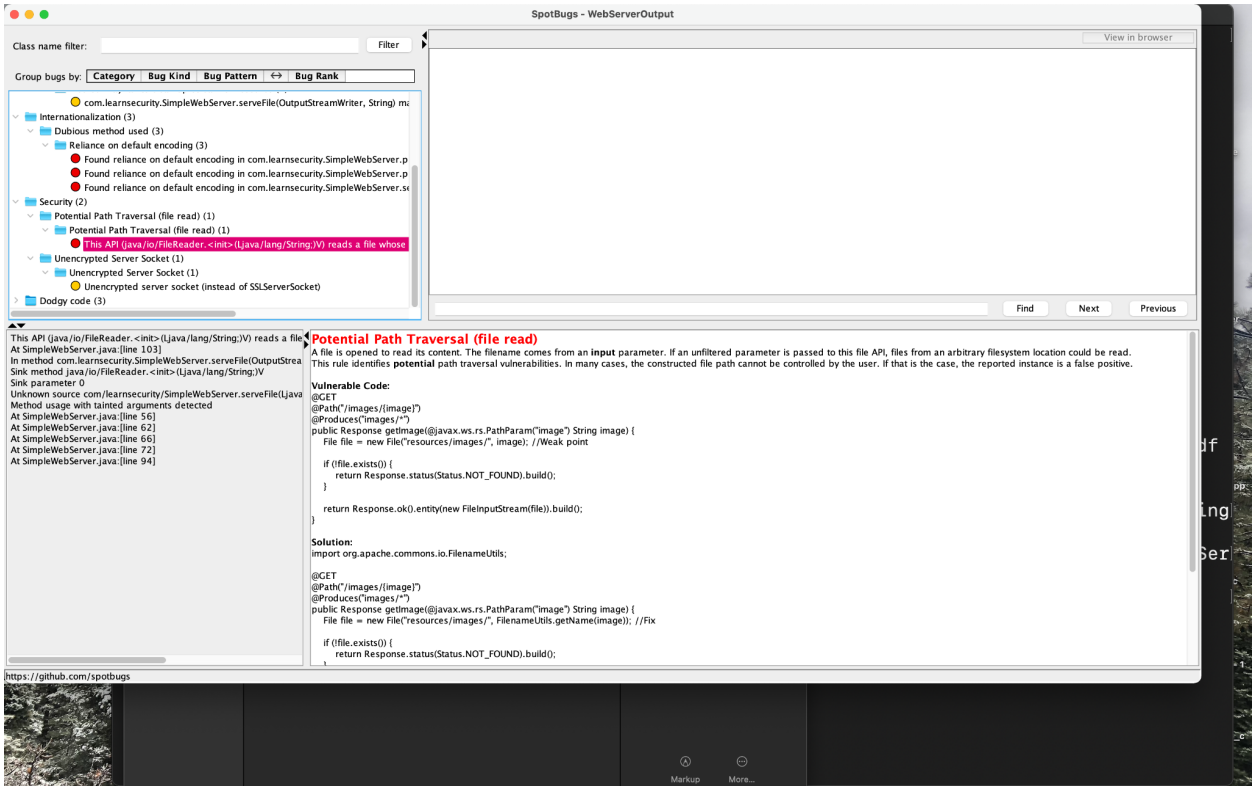
Contents

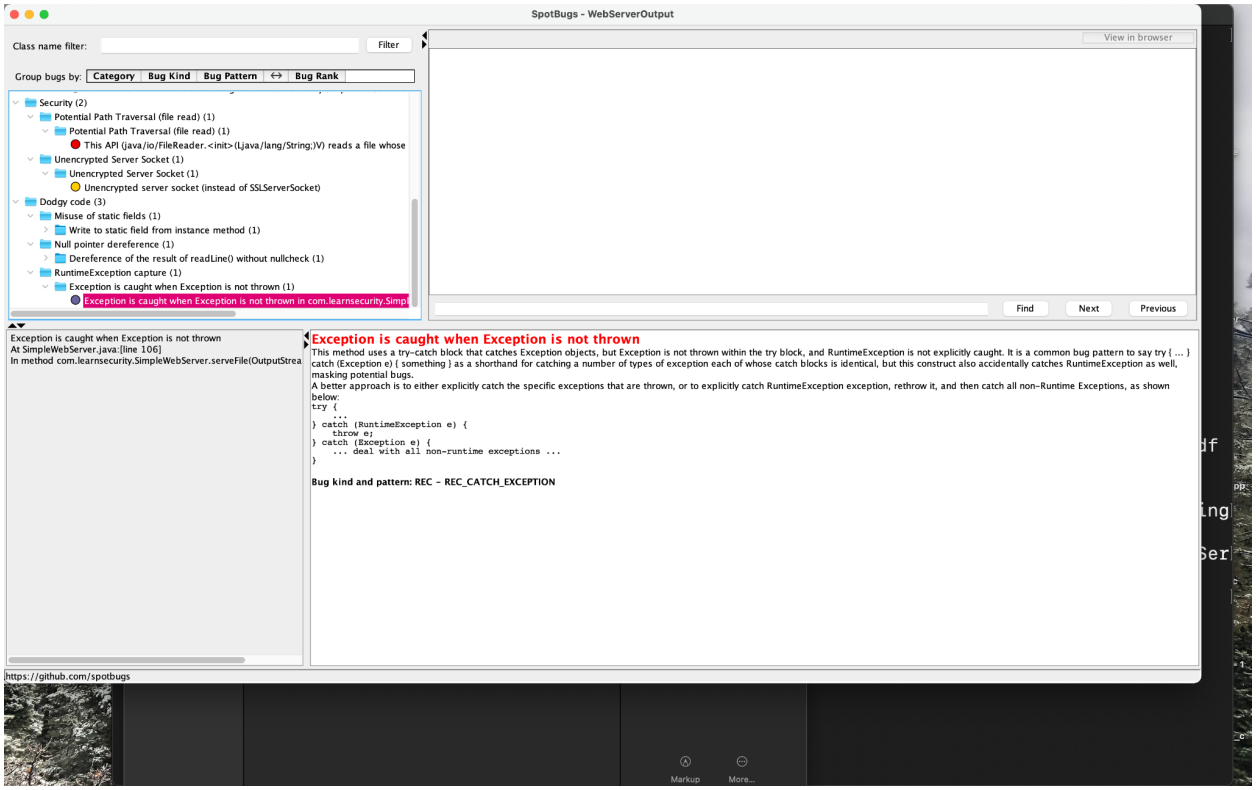
I decided to put it in a HTML format and it is included in this zip folder that this report is in for further examination by yourself the grader.




SpotBugs - WebServerOutput
Class name filter:
Filter
Group bugs by:
Category
Bug Kind
Bug Pattern
Bug Rank
Bugs (10)
Bad practice (1)
Stream not closed on all paths (1)
Method may fail to close stream (1)
com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter, String) may
Experimental (1)
Unsatisfied obligation to clean up stream or resource (1)
Method may fail to clean up stream or resource (1)
com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter, String) may
Internationalization (3)
Security (2)
Dodgy code (3)
View in browser
Find
Next
Previous
serveFile(OutputStreamWriter, String) may fail to clean up java.io.Re
Obligation to clean up resource created at SimpleWebServer.java:[lir
In method com.learnsecurity.SimpleWebServer.serveFile(OutputStrea
Reference type java.io.Reader
1 instances of obligation remaining
Path continues at SimpleWebServer.java:[line 104]
Path continues at SimpleWebServer.java:[line 111]
Path continues at SimpleWebServer.java:[line 116]
Path continues at SimpleWebServer.java:[line 117]
Path continues at SimpleWebServer.java:[line 121]
Path continues at SimpleWebServer.java:[line 122]
Remaining obligations: {Reader x 1}
Method may fail to clean up stream or resource
This method may fail to clean up (close, dispose of) a stream, database object, or other resource requiring an explicit cleanup operation.
In general, if a method opens a stream or other resource, the method should use a try/finally block to ensure that the stream or resource is cleaned up before the method returns.
This bug pattern is essentially the same as the OS_OPEN_STREAM and ODR_OPEN_DATABASE_RESOURCE bug patterns, but is based on a different (and hopefully better) static analysis technique. We are interested is getting feedback about the usefulness of this bug pattern. For sending feedback, check:
contributing guideline
mailinglist
In particular, the false-positive suppression heuristics for this bug pattern have not been extensively tuned, so reports about false positives are helpful to us.
See Weimer and Necula, Finding and Preventing Run-Time Error Handling Mistakes, for a description of the analysis technique.
Bug kind and pattern: OBL - OBL_UNSATISFIED_OBLIGATION
https://github.com/spotbugs
Markup
More...


SpotBugs - WebServerOutput
Class name filter:
Filter
Group bugs by:
Category
Bug Kind
Bug Pattern
Bug Rank
Stream not closed on all paths (1)
Method may fail to close stream (1)
com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter, String) ma
Experimental (1)
Unsatisfied obligation to clean up stream or resource (1)
Method may fail to clean up stream or resource (1)
com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter, String) ma
Internationalization (3)
Dubious method used (3)
Reliance on default encoding (3)
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.p
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.p
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.s
Security (2)
Dodgy code (3)
View in browser
Find
Next
Previous
Found reliance on default encoding: new java.io.InputStreamReader
At SimpleWebServer.java:[line 49]
In method com.learnsecurity.SimpleWebServer.processRequest(Sock
Called method new java.io.InputStreamReader(InputStream)
Reliance on default encoding
Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. Use an alternative API and specify a charset name or Charset object explicitly.
Bug kind and pattern: Dm - DM_DEFAULT_ENCODING
https://github.com/spotbugs
Markup
More...


SpotBugs - WebServerOutput
Class name filter:
Filter
Group bugs by:
Category
Bug Kind
Bug Pattern
Bug Rank
com.learnsecurity.SimpleWebServer.serveFile(OutputStreamWriter, String) ma
Internationalization (3)
Dubious method used (3)
Reliance on default encoding (3)
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.p
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.p
Found reliance on default encoding in com.learnsecurity.SimpleWebServer.se
Security (2)
Potential Path Traversal (file read) (1)
Potential Path Traversal (file read) (1)
This API (java/io/FileReader.<init>(Ljava/lang/String;)V) reads a file whose
Unencrypted Server Socket (1)
Unencrypted Server Socket (1)
Unencrypted server socket (instead of SSLServerSocket)
Dodgy code (3)
View in browser
Find
Next
Previous
This API (java/io/FileReader.<init>(Ljava/lang/String;)V) reads a file
At SimpleWebServer.java:[line 103]
In method com.learnsecurity.SimpleWebServer.serveFile(OutputStrea
Sink method java/io/FileReader.<init>(Ljava/lang/String;)V
Sink parameter 0
Unknown source com/learnsecurity/SimpleWebServer.serveFile(Ljava
Method usage with tainted arguments detected
At SimpleWebServer.java:[line 56]
At SimpleWebServer.java:[line 62]
At SimpleWebServer.java:[line 66]
At SimpleWebServer.java:[line 72]
At SimpleWebServer.java:[line 94]
Potential Path Traversal (file read)
A file is opened to read its content. The filename comes from an input parameter. If an unfiltered parameter is passed to this file API, files from an arbitrary filesystem location could be read.
This rule identifies potential path traversal vulnerabilities. In many cases, the constructed file path cannot be controlled by the user. If that is the case, the reported instance is a false positive.
Vulnerable Code:
@GET
@Path("/images/{image}")
@Produces("images/*")
public Response getImage(@javax.ws.rs.PathParam("image") String image) {
File file = new File("resources/images/", image); //Weak point
if (!file.exists()) {
return Response.status(Status.NOT_FOUND).build();
}
return Response.ok().entity(new FileInputStream(file)).build();
}
Solution:
import org.apache.commons.io.FilenameUtils;
@GET
@Path("/images/{image}")
@Produces("images/*")
public Response getImage(@javax.ws.rs.PathParam("image") String image) {
File file = new File("resources/images/", FilenameUtils.getName(image)); //Fix
if (!file.exists()) {
return Response.status(Status.NOT_FOUND).build();
https://github.com/spotbugs
Markup
More...



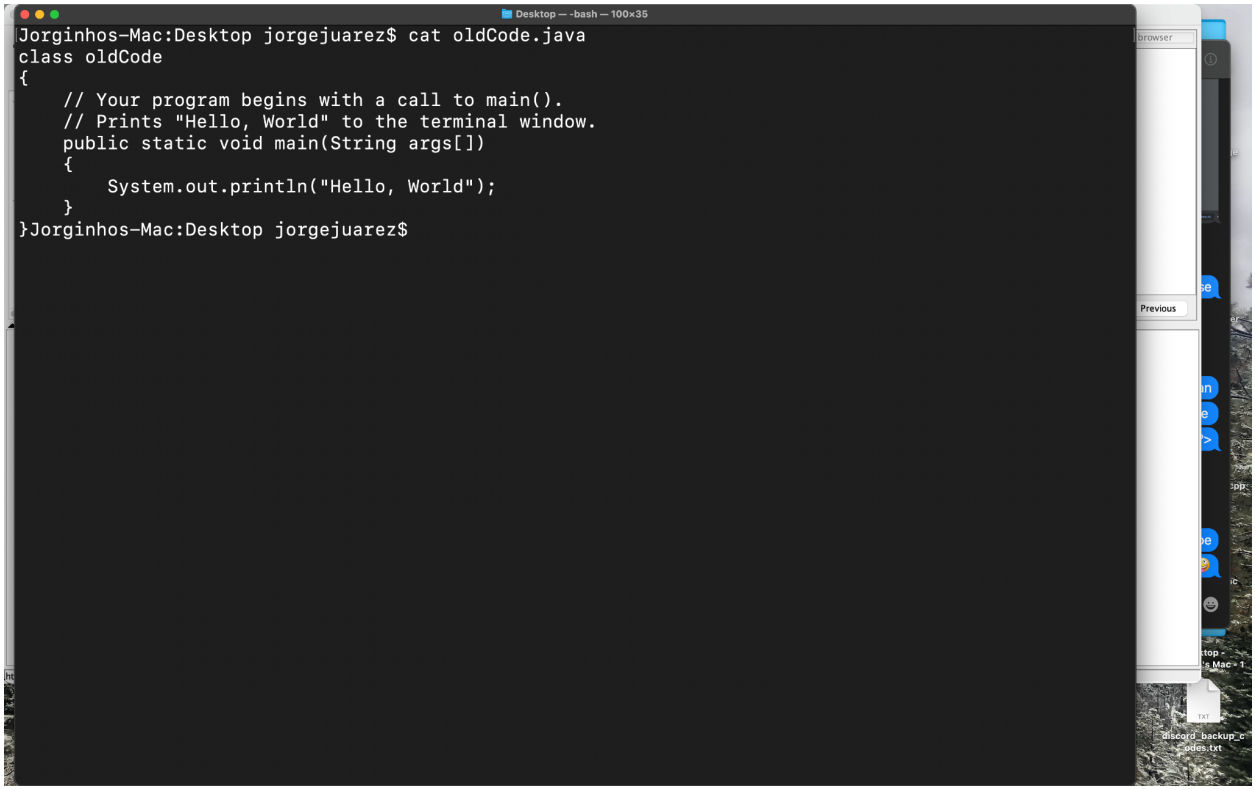
As you can see from the screenshots above, they are all contained in what was wrong. The major themes are:

1) Bad Practice
2) Experimental
3) Internationalization
4) Security
5) Dodgy Code

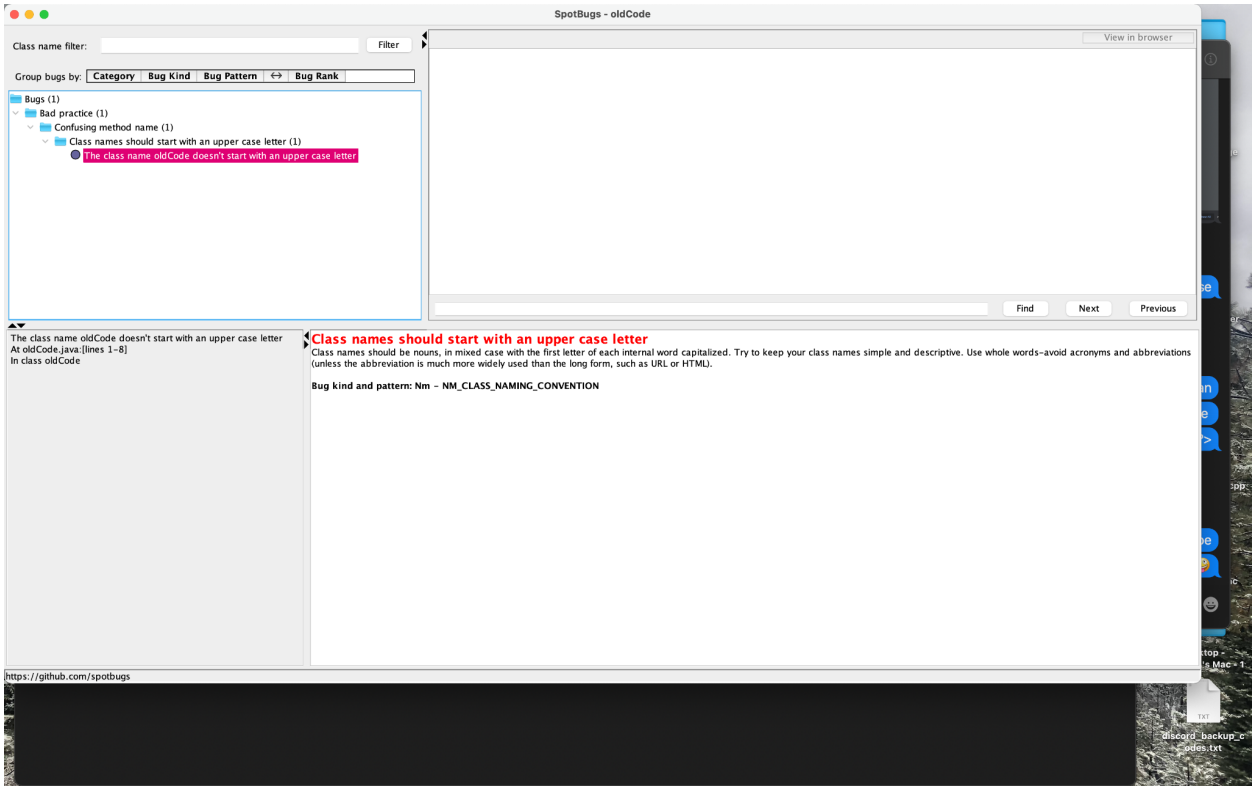
As you can see in the HTML file in this zip folder that Those can be fixed in order to not get as many errors.

Task 2 or Part 2:

I have written this basic C code in the past and I find out no bugs



And then we run it as shown below:



We can see that was the only problem.