Static Analysis of Java source code

Laboratory for the class "Security Verification and Testing" (01TYASM/01TYAOV)
Politecnico di Torino – AY 2023/24
Prof. Riccardo Sisto

prepared by:
Riccardo Sisto (riccardo.sisto@polito.it)

v. 1.0 (19/12/2023)

Contents

1 Purpose of this laboratory

The purpose of this lab is to make experience with static source code analysis tools for the Java language. More specifically, the lab focuses on the SpotBugs tool, which is integrated in the Eclipse IDE as a plugin, extended with another plugin, called FindSecBugs, which is specific for finding security vulnerabilities. Spotbugs employs several static analysis techniques, and its analysis effort and accuracy can be configured, as well as the minimum level of severity of the reported issues. For the installation of the tools, please refer to the 00Lab_GettingStarted_SVT_v104.pdf guide. In the physical machines at Labinf the Spotbugs plugin is already installed, but it may be necessary to configure it.

All the material necessary for this lab can be found in the course web pages on didattica.polito.it, in the 06lab_SJ folder, and the Dropbox folder, both in the Materiale Didattico section.

2 Getting started with SpotBugs

First of all, let us configure SpotBugs to find only security-related vulnerabilities. In Eclipse, open the Preferences item from the Window menu, and select Java - SpotBugs. Here, configure the plugin by checking only the security box. Then, make sure FindSecBugs is correctly set in the plugins tab. Then, as a first test of the tool, import the eclipse project available in the lab material zip archive, in the 'examples' folder (use File - Open Projects from File System and select the 'examples' folder). Once the project has been created, the maven utility inside Eclipse should download the project dependencies and compile the project automatically. When the project setup and compilation are finished, run SpotBugs on the project (right click on the project name

and select SpotBugs - Find Bugs) and check that SpotBugs reports, as expected, some possible vulnerabilities. You can look at the details of the problems found (description of the problem, and possible ways to fix it) by opening the Spotbugs view or, in the Java view, by opening the Java files for which a number (indicating the number of problems found) is shown on the right of the filename, and clicking on the bug symbols.

3 Static Analysis with SpotBugs

3.1 Analysis and fix of the examples project

The examples project contains simple examples of Java classes, some of which affected by security vulnerabilities. The package com.okta.jettyembedded is taken from the github project

https://github.com/oktadeveloper/okta-spring-boot-jetty-example.

Use SpotBugs to analyse the project code, to look for security vulnerabilities, and create a report with your findings: classification of the reported problems into TP and FP, with explanation for each one. In case of XSS, also specify the type of XSS.

Solution

Package com.okta.jettyembedded, file HikesTodoServlet.java:
line 35: TP: variable hike, which comes from the servlet request, is written into the response, thus enabling an XSS attack (reflected XSS) if the response is incorporated into an HTML document. This vulnerability can be fixed by checking or sanitizing variable hike, when it is extracted from the request parameter.
lines 23, 39, 52: TP: the servlet writes the contents of hikes into the response. These contents may come from inputs from the user, stored into variable hikes (see line 38) and not validated nor sanitized. For this reason, the servlet is vulnerable to (stored) XSS. This vulnerability can be fixed by checking or sanitizing variable hike, when it is extracted from the request parameter.
In summary, the tool reported 5 warnings. All of them are TP (2 of them refer to the same vulnerability). The vulnerabilities found are all XSS: one is a reflected XSS while the other ones are stored XSS.

Package it.polito.dsp.echo.v0, file TcpEchoServer0.java:
line 20: TP: the application uses a plain TCP socket instead of using a TLS socket.

Package servlet, file Hello.java:
line 14: TP: the application is vulnerable to (reflected) XSS because it writes name, which comes from the request, into the HTML response, without sanitizing or checking it.

Package servlet, file Test2.java:
line 29: FP: at this location, the strings 'user' and 'pass' cannot contain arbitrary values but only safe values (alphabetic characters).
line 37: TP: at this location the string 'user' can have any value and it comes from the request.

Package payroll, file EmployeeController.java:
lines 27,33,41,48: TP: the problem reported by the tool is the possible leakage of unexpected properties. If we assume that the id, name and role properties of the Employee class are all intended to be exposed to the clients, technically this could be a false positive. However, the exposure of a persistence class is a bad programming practice which could cause real vulnerabilities in the future (e.g., if other sensitive fields which should not be exposed are added in the future to the Employee class). For this reason, it is advisable to address it.
Here, at the same lines, there is also another issue which is not explicitly reported by the tool: the possibility of having a stored XSS vulnerability, made possible because the Employee received in the body of the post and put operations is not validated.

Fix the vulnerabilities found in the project, with the aid of the suggestions given by SpotBugs, then run again the tool and observe how the results of the analysis changed.

Note that, in order to use the OWASP Encoder, you need to add its dependency to the pom.xml file. This can be done by adding the following text to the pom.xml dependencies element:

```
<!-- https://mvnrepository.com/artifact/org.owasp.encoder/encoder -->
<dependency>
    <groupId>org.owasp.encoder</groupId>
    <artifactId>encoder</artifactId>
    <version>1.2.3</version>
</dependency>
```

Solution

See the fixed code in the packages with the 'fixed' suffix.

3.2 Analysis and Fix of vulnerabilities in the OWASP Java VulnerableApp

The OWASP Java VulnerableApp is a purposedly vulnerable Java application used for testing code analyzers.

Analyze the application with Spotbugs and then focus on the problems found in the following classes:
org.sasanlabs.service.vulnerability.sqlInjection.ErrorBasedSQLInjectionVulnerability
org.sasanlabs.service.vulnerability.pathTraversal.PathTraversalVulnerability
org.sasanlabs.service.vulnerability.cmdInjection.CommandInjection

Tell if each problem is a TP or FP and explain why. Then, for each TP, tell how it could be fixed.

Solution

ErrorBasedSQLInjectionVulnerability, line 64 and 109: TP: the variable 'id' comes from an HTTP request and it is not validated nor sanitized before being inserted into the sql query. The vulnerability can be fixed by using a prepared statement with the id as a parameter or by sanitizing the id before using it.
ErrorBasedSQLInjectionVulnerability, line 64 and 208: FP: the variable 'id' comes from an HTTP request but it is sanitized before being inserted into the sql string.
PathTraversalVulnerability, line 53: the parameter 'filename' may come from an untrusted source (an HTTP request). The vulnerability can be fixed by checking the filename string and reject it if it includes dot characters.
CommandInjection, line 44 and 49: TP: the parameter ipAddress may come from an untrusted source (an HTTP request) and it is not sanitized or checked. The vulnerability can be fixed by checking that it is a valid IP address.

3.3 Analysis and Fix of vulnerability CVE-2021-37573

CVE-2021-37573 is a vulnerability of the Tiny Java Web Server and Servlet Container (TJWS, http://tjws.sourceforge.net/). For your convenience, the CVE report is available in the lab material. After having had a look at it, your task is to find the vulnerability in the Java code of the application, with the help of SpotBugs, and then fix it. For your convenience, you can find a relevant portion of the code (taken from version 115, which is the latest one affected by the vulnerability) in the TJWS2 folder, with a pom.xml file that automates the download of its dependencies and its compilation. This code can be imported into Eclipse as a Maven Project, as done for the other projects.

Solution

The vulnerability (a reflected XSS) can be found in file Acme.Serve.FileServlet.java at line 183: the filename that is put into the body of the 404 response comes from the path in the request URL. The vulnerability can be fixed by sanitizing the filename or by checking it before putting it into the response. See Acme.Serve.FileServlet_fixed.java