SECURITY TESTING

PROJECT:
VULNERABILITIES IDENTIFICATION AND EXPLOITATION

Name: Niccolò Parlanti
student id: 239218
submission date :
Name of filled XLS: 239218_NiccolòParlanti_secproject

Department of Information Engineering
and Computer Science
Master's Degree in Computer Science

Academic year 2022/2023

Project Introduction

The main goal of this project is to gain knowledge and understand how to identify, assess and exploit vulnerabilities in web applications. In this case, the applications chosen were Bodgeit store and Vulnerable app, two applications often used for educational purposes as they have simple vulnerabilities that can be worked on. The goal of this report is to show and explain the different phases of the project, in particular the research, exploitation and reporting of vulnerabilities.

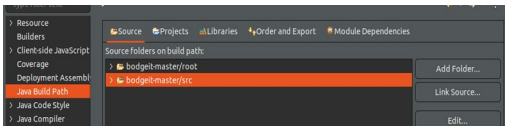
According to the github page, The Bodgeit store is a vulnerable web application which is currently aimed at people who are new to pen testing. The Bodgeit Store include the following significant vulnerabilities: Cross Site Scripting, SQL injection, Hidden (but unprotected) content, Cross Site Request Forgery , Debug code , Insecure Object References , Application logic vulnerabilities.

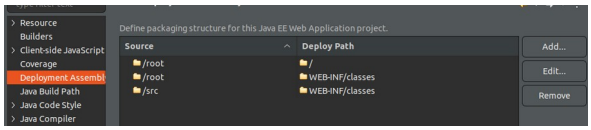
Vulnerable app is a web application that consent to work with tools such as automatic scanners, and to exploit vulnerabilities in a safe way. Vulnerable app include the following significant vulnerabilities: Command injection, File upload, Path traversal, SQL injection, XSS, Open redirect and SSRF.

To set up the two application I used eclipse as environment.

I had some problem with the Bodgeit Store, in particular:

- *In basket.jsp and header.jsp, <%@ page import="java.servlet.http.*" %> needs to be changed in <%@ page import="javax.servlet.http.*" %> .*
- *To solve the problem in basket.jsp on the import it was necessary to change the settings of the project, in Deployement Assembly, adding the path /root with / and src with WEB-INF/classes*
- *In Properties – Java Build Path was necessary to add the folders : bodgeit-master/root and bodgeit-master/src*
- *In web.xml it was necessary to delete the servlet display name field*





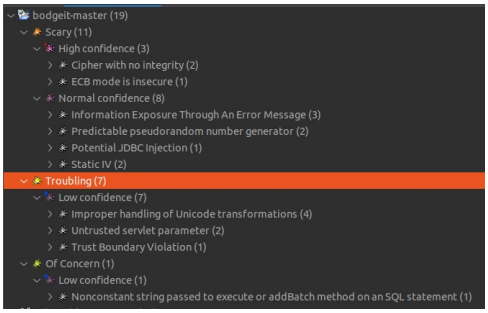
With Vulnerable app I had no problem with the setup.

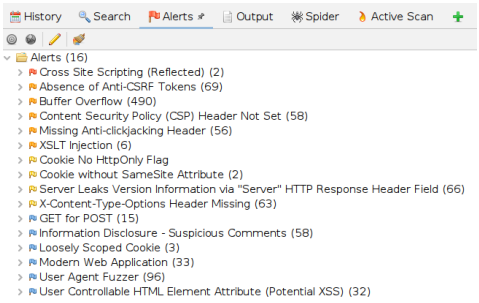
The used tools were Spotbugs with Find-Sec-Bugs plugin (powered in Eclipse IDE for Enterprise Java) that covers the Static Analysis part, and OWASP ZAP 2.11.1 that covered Dynamic Analysis.
I used these two tools while maintaining the configurations recommended by the professor, as I considered them sufficient and functional to complete this project

Explanation of the adopted process

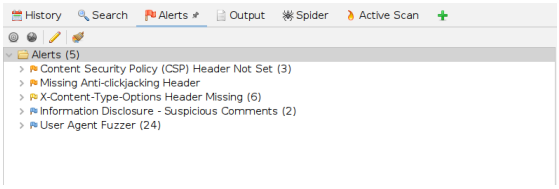
I have the even path, so the process in general consist in running first Bodgeit Store first with Spotbugs (For Static Analysis) and then with OWASP Zap (For Dynamic Analysis), and after Vulnerable App with OWASP Zap and then with Spotbugs.

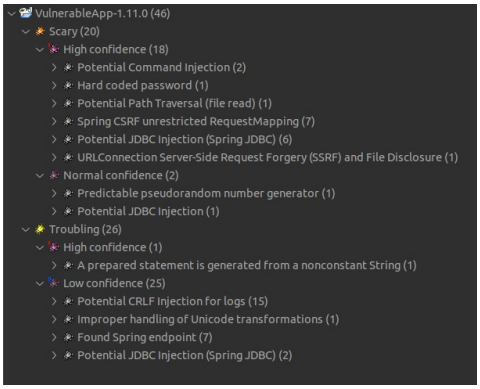
As the execution order says, I started with the Static Analysis of the first application, Bodgeit Store, that gave me as result 19 warnings. Most of this (in particular of the "scary" section) were related to the use of cryptographic protocols that are consider weak or vulnerable in some way. I was not able to exploit any of the vulnerabilities reported. I passed then to the Dynamic Analysis, that gave me 16 class of alerts, with a total of 1050 possible attack vector. I finally performed a Manual Analysis, which led me to exploit just four vulnerabilities, since most of the vulnerabilities present had already been found through dynamic analysis.





I passed then to the second application, Vulnerable App. Following the path, I started with the Dynamic Analysis, which turned out to be less effective than it had been on Bodgeit, giving me 5 classes of alerts, with a total of 36 possible attack vector. In contrast, however, static analysis proved to be much more informative, with 46 warnings, 20 of which are classified as scary. After that, with the Manual Analysis, I was able to exploit the remaining bugs.





Summary of the collected data

The attached XLS file shows all the data that were retrieved and analyzed after a study by Dynamic (ZAP) and Static (Spotbugs) analysis on the Bodgeit Store and Vulnerable App applications, as well as a final manual analysis.
In this section I am going to report and explain some statistics regarding the results I was able to obtain through the previously described methods.

Bodgeit Store

- *Static Analysis:*
  The static analysis reported total of 10 different types of vulnerabilities, arriving at 19 warnings, divided into:
  - *11 "Scary" → 3 High Confidence, 8 Normal Confidence*
  - *7 "Troubling" → 7 Low Confidence*
  - *1 "Of concern" → 1 Low Confidence*

  None of the vulnerabilities reported by Spotbugs were actually found to be exploitable, bringing the reported false positive count to 100%.

- *Dynamic Analysis:*
  Dynamic analysis reported 1050 possible attack vectors for a total of 16 classes of alerts, divided into risk categories as follow:
  - *1 "High" → 1 Medium Confidence → 2 attack vector*
  - *5 "Medium" → 1 High Confidence, 3 Medium Confidence, 1 Low Confidence → 649 attack vector*
  - *4 "Low" → 1 High Confidence, 3 Medium Confidence → 130 attack vector*
  - *6 "Informational" → 1 High Confidence, 3 Medium Confidence, 2 Low Confidence → 248 attack vector*

  It is important to point out that the large number of attack vectors (especially for some types of vulnerabilities) is due to the fact that these are repeated, since they refer to the same page, that was repeatedly accessed by the tool. Therefore, I have reported on the XLS file only the vulnerabilities that refer to different pages (avoiding repetition), for an actual total of 103 attack vectors, of which 31 are exploitable.

- *Manual Analysis*
  Finally, through manual analysis, I was able to exploit 3 other types of vulnerabilities, for a total of 4 attack vectors.

Vulnerable App

- *Static Analysis:*
  The static analysis reported total of 13 different types of vulnerabilities, arriving at 46

warnings, divided into:

- *20 “Scary” → 18 High Confidence, 2 Normal Confidence*
- *26 “Troubling” → 1 High Confidence, 25 Low Confidence*

Of these 46 warnings, I was able to prove 12 to be exploitable, while 34 were false positives.

- *Dynamic Analysis:*
  Dynamic analysis reported 36 possible attack vectors for a total of 5 classes of alerts, divided into risk categories as follow:
  - *2 “Medium” → 1 High Confidence, 1 Medium Confidence → 4 attack vector*
  - *1 “Low” → 1 Medium Confidence → 6 attack vector*
  - *2 “Informational” → 2 Medium Confidence → 26 attack vector*

  As with Bodgeit Store, the actual number of attack vectors is different here, due to some repetition. I therefore worked on a total of 12 different attack vectors, only one of which turned out to be related to an actual vulnerability.

- *Manual Analysis*
  Finally, through manual analysis, I was able to exploit 6 other types of vulnerabilities, for a total of 26 different attack vectors.

Speaking generally, for Bodgeit Store I spent about 224 minutes to analyze the results of the dynamic analysis, 74 minutes for the static analysis, and 14 for the manual analysis (considered effetive time just to realize the exploit of the vulnerabilities found, thus not counting the time to search for them), while for Vulnerable App, I spent about 22 minutes to analyze the results of the dynamic analysis, 162 minutes for the static analysis, and 193 for the manual analysis.

Discussion on Result

Using security testing tools like SpotBugs and OWASP ZAP to identify vulnerabilities in intentionally vulnerable web applications like Bodgeit Store and Vulnerable App is a straightforward and predictable process.
These tools are designed specifically to scan web applications for known security flaws and identify potential areas of weakness. As Bodgeit Store and Vulnerable App are intentionally vulnerable, it is expected that these tools will detect a range of vulnerabilities. Infact, the primary purpose of these applications is to provide a safe and controlled environment for security researchers to practice identifying and exploiting vulnerabilities, and testing them with tools, as we did for this project.

I had no prior experience with software testing, having never actually done any projects of this kind. As a totally novice on it, I found that SpotBugs and OWASP ZAP are both relatively easy to approach for someone who has never done security testing before.
Both tools have user-friendly interfaces and are designed to be straightforward to set up and use. SpotBugs can identify security issues in Java code, which is a widely used programming language in software, and is a great starting point for those with a bit of programming background as me. OWASP ZAP is designed to scan web applications and identify vulnerabilities, making it an excellent tool for test the security of web applications. Both tools provide detailed reports on the vulnerabilities they identify, making it easy for users to understand and remediate the issues.

By analyzing the results obtained on the two applications, we can draw some conclusions about the effectiveness of the tools in this specific context:

Regarding static analysis, Spotbugs proved to be much more effective on analyzing Vulnerable App, compared to Bodgeit Store. A cons I found using SpotBugs is that it identified a significant number of false positives, especially on Bodgeit Store. This can be frustrating and time-consuming, as it requires additional effort to investigate and remediate issues that are not actually exploitable. Despite this issue of false positives, SpotBugs remains a useful and valuable tool for identifying potential security vulnerabilities, and it was very helpfull on the analysis of Vulnerable App.

For what concerns dynamic analysis, the result found was opposite, as I had the best results after Bodgeit Store analysis, compared to Vulnerable App. In particular, I noticed that the spider used by ZAP to search for the pages, during the test analysis on Vulnerable App, has some problems to retrieve all of them, and this may be the cause for the few results obtained.
In spite of all that, I have found ZAP to be very useful, particularly every time a vulnerability was found, it gave fundamental helpful information for the vulnerability exploitation like the parameter to tamper, the type of the vulnerability, a possible attack vector, the possible mitigations and references to the CWE and OWASP Top-10 ranking.

In conclusion, testing these software with security tools has been an experience that formed me a lot, allowing me to identify and remediate potential vulnerabilities while boosting my knowledge in developing secure applications and understanding on possible security issues.