Fix Lab 2: Input Validation Defenses

Date Due: Saturday, September 30, 2023, by 11:59 PM.

Introduction

In this lab assignment, you will fix all the issues you uncovered in the last attack lab. This assumes you have completed both Lab 0: Lab Setup Guide and Attack Lab 1: Input Injection Attacks. For this lab, you will need the ability to run the WebGoatCore application, as well as debug it using Visual Studio 2019 Community Edition. You will also need to have completed the Git setup outlined in Lab 0 and the ability to push code changes to the remote repository on https://code.umd.edu.

For each of the following questions on the WebGoatCore application, you will need to refer to your answer from Attack Lab 1, and fix the weaknesses found in the source code. You will be uploading a writeup along with pushing code changes to **your WebGoatCore** project under branches organized by Lab Number and Phase Number.

Wherever the questions refer to '**implement**', it refers to the following:

IMPORTANT: Implement the fix for the question under a branch in **your WebGoatCore** project that you have shared with the instructor and grading staff, with the following naming convention Lab<Lab #>_Phase<phase #> (for example **Lab2_Phase1**):

1. For example, you can run:
   i. `git checkout main`
   ii. `git checkout -b MyFix # This is a branch where I implement fixes`
2. Now you implement your fix and commit the changes locally:
   i. `git add .`
   ii. `git commit -m "Implemented Lab 2 Phase 1." # Commit with a message`
3. Repeat earlier step as many times as needed to get your fixes working for the given Phase and keep committing them. Now merge your changes in to a named push the newly created branch to the remote repository using:
   i. `git checkout -b Lab2_Phase1 # This is for Lab 2 Phase 1 fix`
   ii. `git merge --squash MyFix # This will squash all commits into a single commit and merge into Lab2_Phase1 branch`
   iii. `git push origin Lab2_Phase1 # This will push your branch up to repo`
4. **Identify** and **submit the commit id** (the long hexadecimal alphanumeric string from `git log --pretty=oneline`) as part of a separate text file named after your UMD email ID + "**_lab2.txt**" (for example **gkini_lab2.txt** all lowercase).

Read https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History for more information on git logs. Wherever the questions below refer to '**discuss**', it refers to providing answers, in your writeup file, to the following questions:

Question Number.

a) **How does the fix work:** For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab or as specified in the question, describe how you fixed the weakness. This can be as detailed as possible to explain how your fix addresses the weakness.
b) **Why does the fix work:** Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.
c) **List all the paths with filenames you are changing** to implement the fix for the weakness.

Phase 1: Prevent SQL Injection (10 points)

Given the techniques discussed in class, try to answer the above given questions for the following:

1. Implement and discuss the fix for ONLY the SQL Injection vulnerability you exercised in the previous lab. Even though the attack took advantage of other vulnerabilities, please only address the SQL Injection.
2. Remember to commit and push your code under the Lab2_Phase1 branch.

Phase 2: Prevent Path Traversal and Prevent Upload of Dangerous Files (10 points)

Given the techniques discussed in class, try to answer the above given questions for the following:

3. Now implement and discuss the fix for ONLY the vulnerability that allows for accessing the stored credit cards file.
4. Now implement and discuss the fix for ONLY the vulnerability on the About page that allows for manipulating the **appsettings.json** file.
5. Now implement and discuss the fix for ONLY the vulnerability in the About page that allows for uploading dangerous files.
6. Finally implement and discuss the fix for ONLY the vulnerability in the Home page that allows for a Readline DoS vulnerability.
7. Remember to commit and push your code under the Lab2_Phase2 branch.

Phase 3: Prevent Regular Expression DoS by implementing Efficient Regex (10 points)

Given the techniques discussed in class, try to answer the above given questions for the following:

8. Now implement and discuss the fix for ONLY the Regex DoS vulnerability in the product search page and answer using the format given at the top of this lab handout.
9. Remember to commit and push your code under the Lab2_Phase3 branch.

Phase 4: Prevent Cross-Site Scripting (XSS) (10 points)

Given the techniques discussed in class, try to answer the above given questions for the following:

10. Now implement and discuss the fix for ONLY the Stored XSS vulnerability in the Blog respond page and answer using the format given at the top of this lab handout.

11. Now implement and discuss the fix for ONLY the Reflected XSS vulnerability in the product detail page and answer using the format given at the top of this lab handout.
12. Remember to commit and push your code under the Lab2_Phase4 branch.

Phase 5: Fuzzing (10 points)

Given the techniques discussed in class, try to answer only question items a), b) and c) for the following:

13. Assuming the WebGoatCore application used the Jil library that you attacked in the previous lab, discuss how you might go about fixing the issue in the application using the format given at the top of this lab handout. **Important:** This question only asks to '**discuss**' and not '**implement**' the fix.

Submission Criteria

Please submit two write ups:

1. Please submit a **Word document** or a **PDF** titled with your UMD Email ID + "**_Lab2Writeup.<pdf|docx>**' (For example, gkini_Lab2Writeup.pdf). Please include Your name, UMD email ID, Phase Number and Question Number you are answering from above.

2. This lab contains code submission with branches for each phase as well. Each of the phases is worth 10 points. Please ensure the commits submitted are accessible by the auto grader. Please upload a **Text file** titled **submission.txt**. This should contain your email id on the first line and then the commit IDs from your WebGoatCore project git log for each Phase from inside each Phase branch. For example:

```
jdoe
a28aefc…
28efa2b…
27920ad…
```

3. Now zip up both the document and the text file as **submission.zip** and upload it on Gradescope.