Assignment #2

Release Date: Beginning of Week 5

Due Date: End of Week 7

TA emails: kevin.gallagher@nyu.edu, rma460@nyu.edu

Office Hours: Monday from 4 PM to 6 PM, Wednesday from 5 PM to 7 PM, and Saturday from 12 PM to 2 PM

Introduction

This assignment focuses on secure Web development. For this assignment, you are tasked with turning your spell checking system from Assignment 1 into a Web service using Flask, while focusing on the security of the Web service you are implementing. After you develop the secure Web service, you will test it to ensure it is not vulnerable to common attacks. Though it is not part of the grade, you should continue to use the secure development practices that you established in Assignment 1.

Setup

Completion time – 2 – 5 hours

If you do not already have pip installed on your machine, install pip using the following command:

```
$ sudo apt-get install python3-pip
```

For this assignment you are required to use the Flask Web framework. The Flask web framework can be installed using pip.

```
$ sudo pip install flask
```

For python projects, tox is a good framekwork for running tests. For more information on Tox, see https://tox.readthedocs.io/en/latest/. Tox can be installed using pip.

```
$ sudo pip install tox
```

However, in order to get tox to work well with Travis CI, you may need the tox-travis plugin for Travis. For more information on this plugin, see https://tox-travis.readthedocs.io/en/stable/. This can also be installed using pip.

```
$ sudo pip install tox-travis
```

Deliverables & Grading

The source code is submitted through Gradescope, and the report is to be submitted through NYU Classes. Your report should have a link to the GitHub repository for the code, which should be readable by myself and the TAs or set to public.

1. Web service code - 30 pts.
2. Tests – 20 pts.
3. Write-ups – 50 pts.

Total 100 pts.

Source code can be submitted through GitHub integration or as a ZIP file to the gradescope. All necessary files should be present in your submission, regardless of the method. We expect the application to launch with the command `flask run` so your submission must contain an app.py file in its root directory.

Flask extensions are allowed, but must be declared in a `requirements.txt` file and be able to be installed using:

```
pip install -r requirements.txt
```

Week 4

Completion time – 9 to 11 hours, depending on experience

In the first part of this program, you are tasked with creating a Web service to provide spell checking capability (using the program you created in Assignment 1) to registered users. In order to achieve this, you Web service must provide at least the following functionality, at the following locations:

1. User registration: /your/webroot/register
2. User login: /your/webroot/login
3. Mock Two-factor authentication: /your/webroot/login
4. Text submission: /your/webroot/spell_check
5. Result retrieval: /your/webroot/spell_check

User Registration

Users must be registered in order to use your service. Your registration page is required to have the following forms for the user to fill in:

1. A form for the user to enter a username, with id=uname.
2. A form for the user to enter a password, with id=pword
3. A form for the user to enter a two-factor authentication device, with id=2fa

The standard rules of registration apply. Usernames must be unique, and users must supply at least a username and password to be able to register for the service. When registration is complete, users should be shown a success message in an element with id=success. If the registration failed, the user should be shown a failure message in the element with id=success. The words “success” or “failure” (capitalization irrelevant) must be present in the message.

User Login

After a user registers, he or she should be able to login to your Web service. Your login page is required to have the following forms for the user to fill in:

1. A form for a user to enter a username, with id=uname.
2. A form for a user to enter a password, with id=pword.
3. A form for the user to enter a two-factor authentication code, with id=2fa
    1. Instead of actual 2-factor, this field will just accept the phone number the user signed up with. Unfortunately, setting up a 2fa system requires more effort than can be alloted to this assignment.

In the following cases, your login page should return an error in an element with id=result with the assocaited string in it:

1. Incorrect username or password: “Incorrect”
2. Two-factor authentication failure: “Two-factor” and “failure”

If a user logs in successfully, your login page should return a success message in an element with id=result. The message must contain “success.” Your Web service should do all of the necessary steps for session management.

Text Submission

After a user is logged in, he or she has the ability to submit bodies of text to check the spelling of the words in the text. The text should be submitted through a form with id=inputtext. Your Web service should then take this text and use the spell checker you wrote in Assignment 1 to determine which words are misspelled. The binary will exist with the name a.out. You do not need to submit this binary. It will already be on the gradescope autograder.

Result Retrieval

After the results of the C program are returned to your Web service, your Web service must output the results to the user. The supplied text should be output in an element with an id=textout and the misspelled words, separated by commas, should be output in an element with id=misspelled.

Tying it all together

On a high level, the functional requirement of this assignment its to create a Web site with the following workflow:

1. A user registers to the service. This registration **must persist until the application is closed, but does not need to persist after the Web service is exited.**
2. A user logs into the Web service.
3. A user submits text to be checked by the spell checker. **The binary for the spell checker is supplied on NYU Classes.**
4. The Web service code calls the binary spell checker program.
5. The Web service code received the output of the spell checker program deals with it according to section **result retrieval** above.

Week 5

Completion time – 9 to 11 hours, depending on experience

Security Requirements

In this week you must implement defenses for common Web attacks for your Web service. These defenses should take the form of whitelisting, encoding and decoding, browser headers, and other defenses.

When writing your Web service, you must defend against common attacks against Web services such as

- XSS
- CSRF
- session hijacking
- Command injection
- etc.

Information on how to avoid these attacks can be found at https://owasp.org and their affiliated GitHub repositories. For example, the Cheat Sheet repository holds a page of Cross Site Scripting prevention, found here https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md.

For this assignment, it is sufficient to keep the data in memory only. In other words, using a database is not required, and therefore you do not need to worry about SQLi attacks.

Report

In addition to the Web service code, you are required to submit a write-up that details the following:

- design decisions for the Web service
- how you mitigated different categories of common Web vulnerabilities

The following example is insufficient for protecting against XSS, but illustrates the types of explanations we are looking for in the report.

> *For this program we were required to take text input from the user through a text input field. However, unsanitized input that could later be shown to the user could lead to potential cross-site scripting vulnerabilities. For this reason, we decided to strip all of the '<' and '>' characters from user inputted strings.*

Week 6

Completion time – 5 to 9 hours, depending on experience

After you design and implement your Web service using Flask, you must test your Web serice to detemine if it is vulnerable to common Web service vulnerabilities such as session hijacking, XSS, CSRF, etc. To do this you should attempt to perform such attacks on your Web service. Like Assignment 1, when you find a vulnerability you should patch it and create a new test that would ensure that such a vulnerability does not occur in the future.

Information for detecting these vulnerabilities can be found at https://owasp.org. For example, the page https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002) can be very useful for learning how to detect stored XSS.

In addition to submitting all new tests and patched vulnrerabilities, you must also submit a write-up that describes the vulnerabilities you found, why they occurred, why you didn't catch them during the first part of the assignment, and how you patched them.

Extra Credit

Extra credit may be provided to submissions that go above and beyond the requirements of the assignment. There is no definitive formula or assignment for extra credit.

Hints

You may wish to include your spell checker code as a git submodule. You can read more about git submodules at https://git-scm.com/book/en/v2/Git-Tools-Submodules.
Are your passwords stored securely? Are cookies able to be read by JavaScript? If so, how can you ensure that the session will not be hijacked? There are two types of XSS attacks specifically mentioned in the lectures. Is your Web service protecting against both of them? Can users execute Linux commands through your service? What http response headers are you setting?

Late Policy

Late assignments will not be accepted.