NYU Tandon School of Engineering

CS-GY9163: Application Security

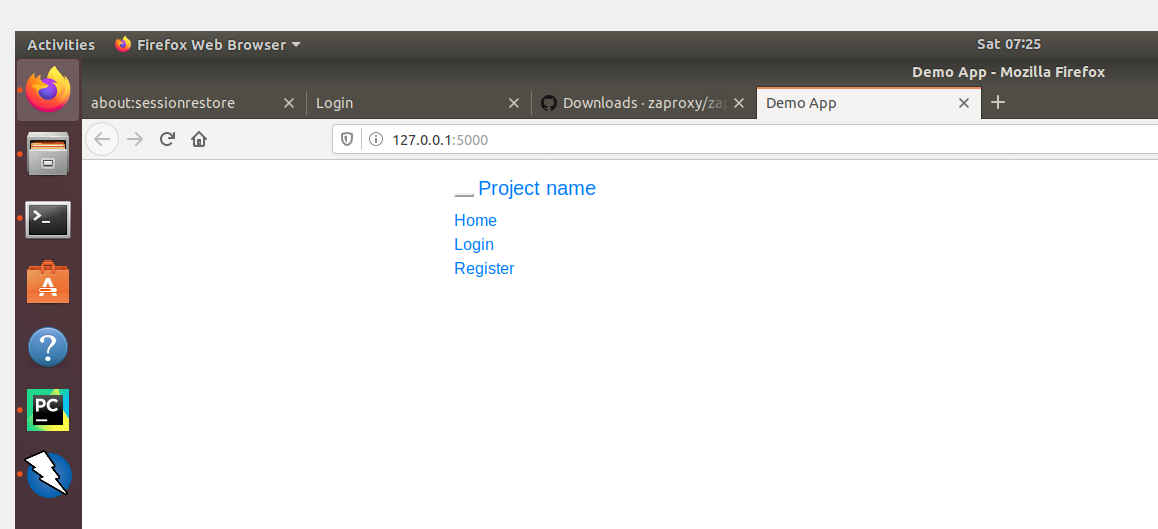
Alfredo Flores

**Introduction:**

The purpose of this assignment is turning a spell checking system into a Web service using Flask, while focusing on the security of the Web service implementation. After the development of the secure Web service, tests are required to ensure it is not vulnerable to common attacks. The link for GitHub repository for the code is https://github.com/alfredofloresf/assignment2

**Procedure:**

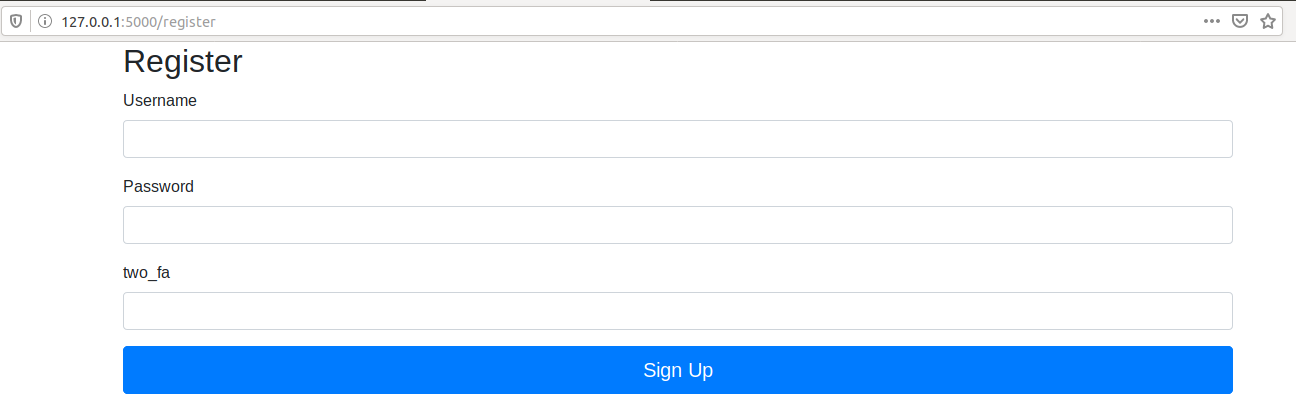
The source code is submitted through GitHub integration. The application to launch with the command flask run so the submission must contain an app.py file in its roof directory.

Program is divided in the following parts: 

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 register in order to use the service. The registration page is required to have the following forms for the user to fill in:

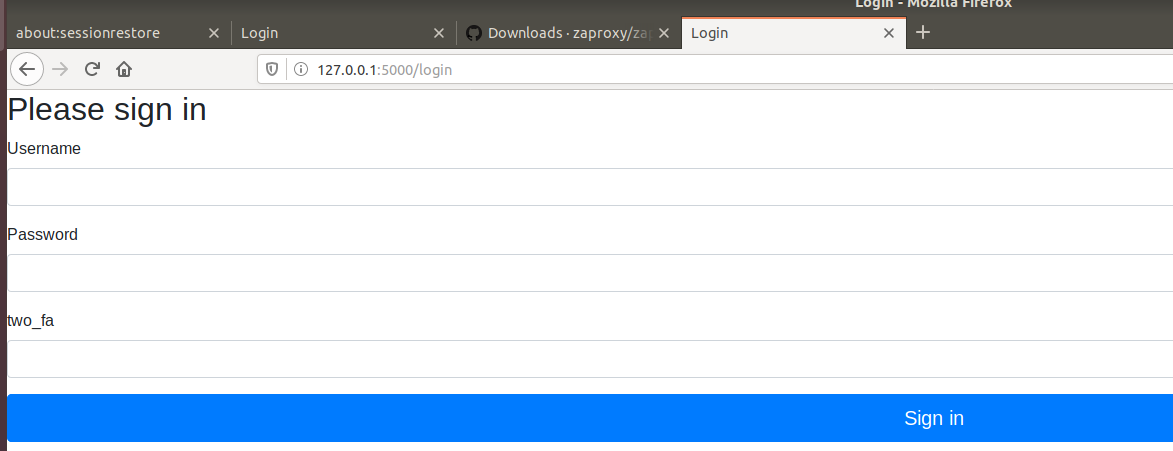


* Enter a username, with id=uname
* Enter a password, with id=pword
* Enter a two-factor authentication, with id=2fa

Usernames must be unique, and users must supply the username, password and two factor-authentication in order to be able to register for the service. If the registration is correct, 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.

**User Login**

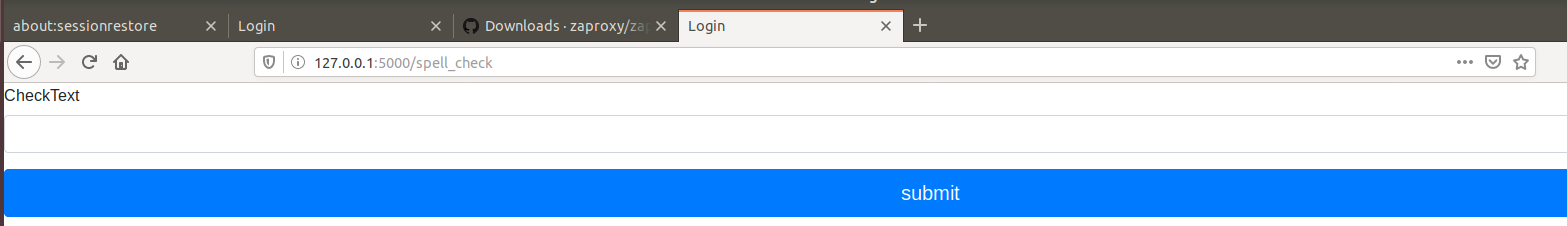
After registration, the user should be able to login to the Web service. The login page is required to have the following forms for the user to fill in:



* Enter a username, with id=uname
* Enter a password, with id=pword
* Enter a two-factor authentication, with id=2fa

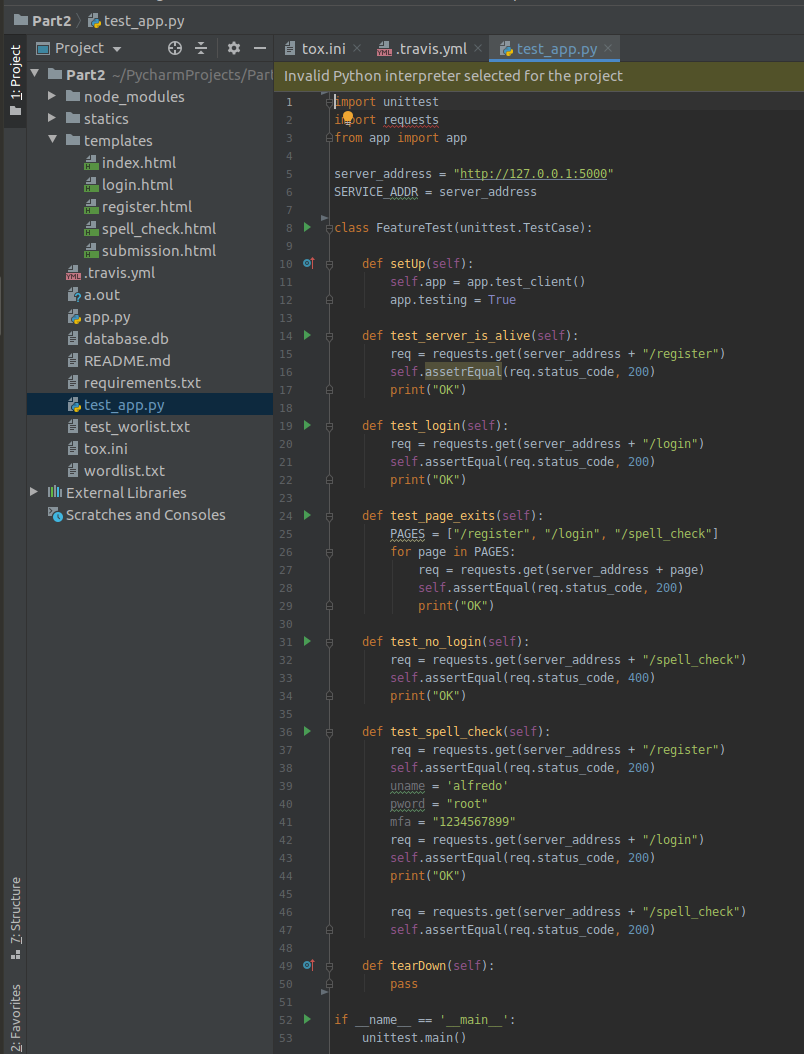
**Text Submission**

After login, the user has the ability to submit bodies of text to check the spelling of the words. The input of the text should be submitted with id=inputtext.

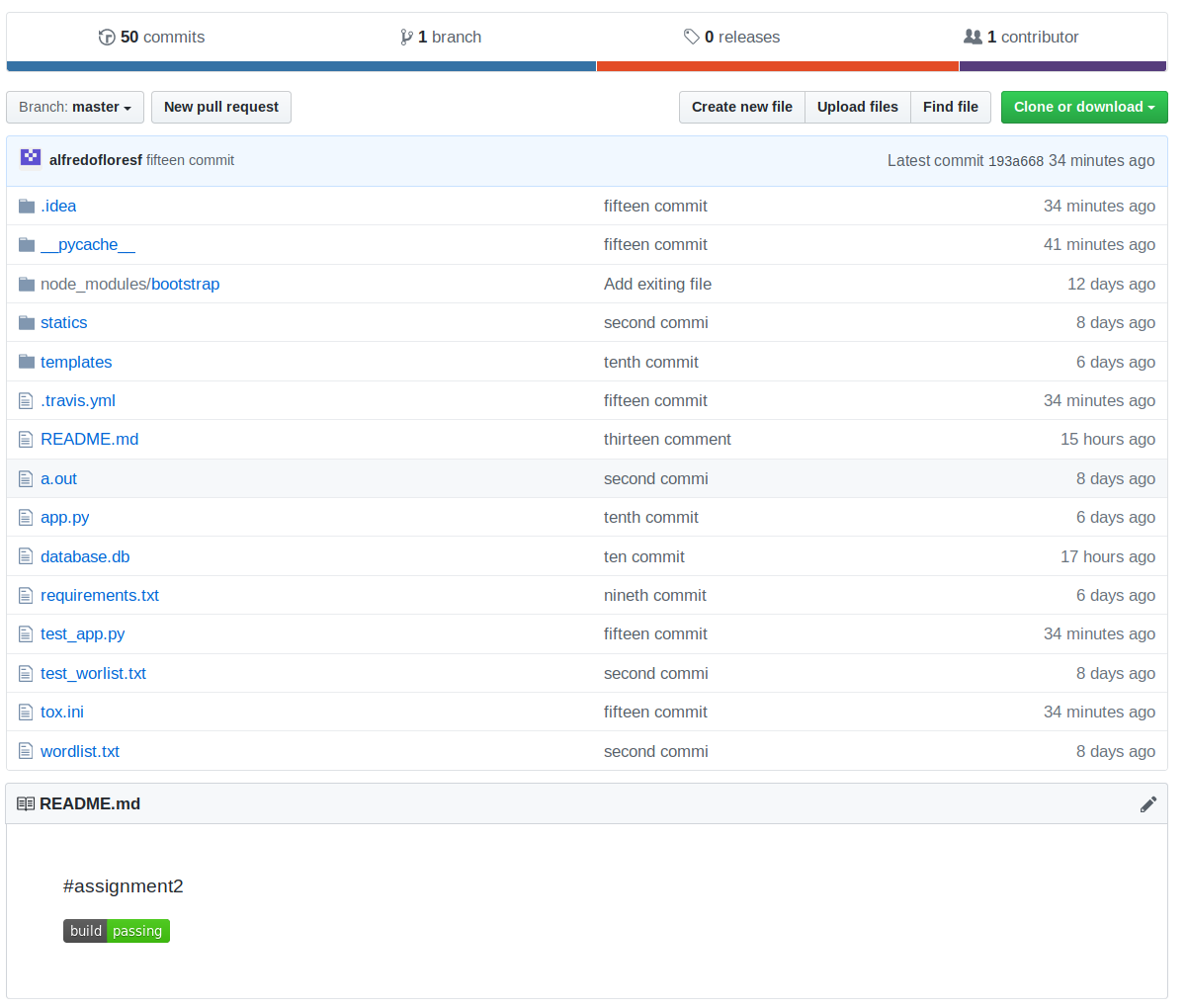


**Tests**

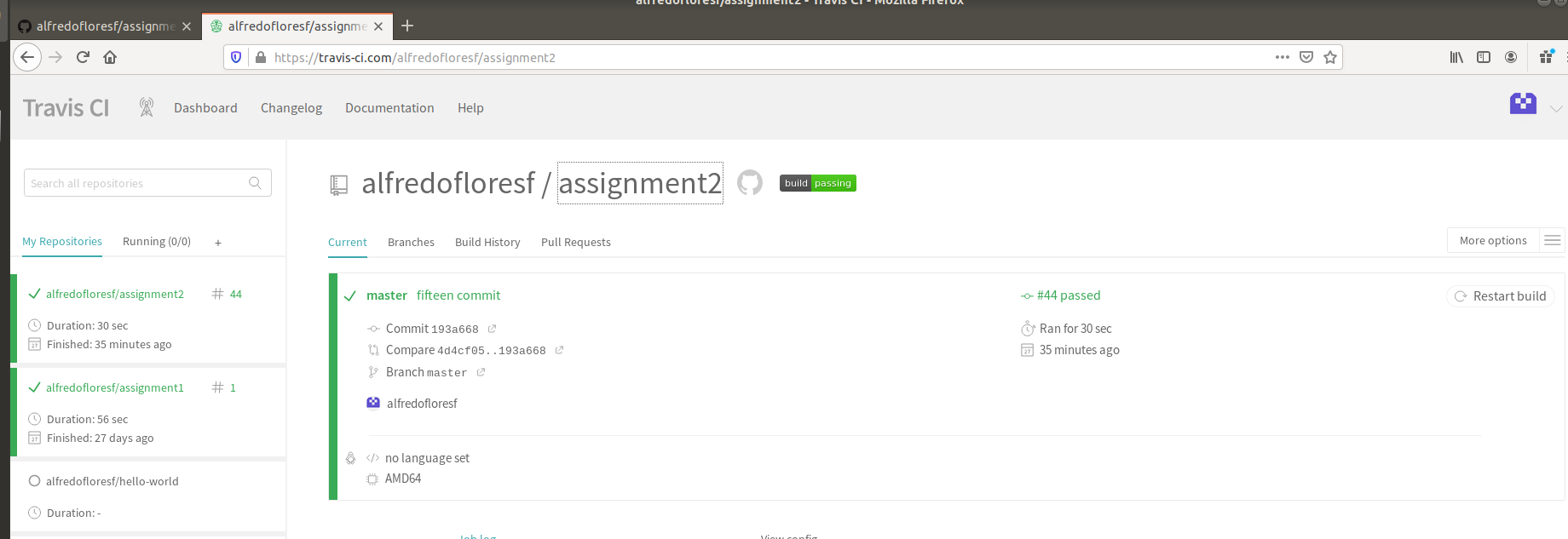
Test cases were accomplished in test\_app.py



**Results in Github**

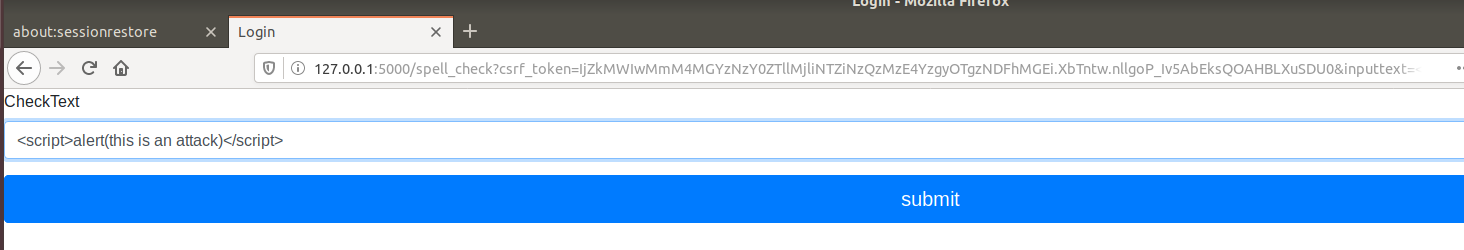


**Results in Travis**



**Defending Against Common Attacks**

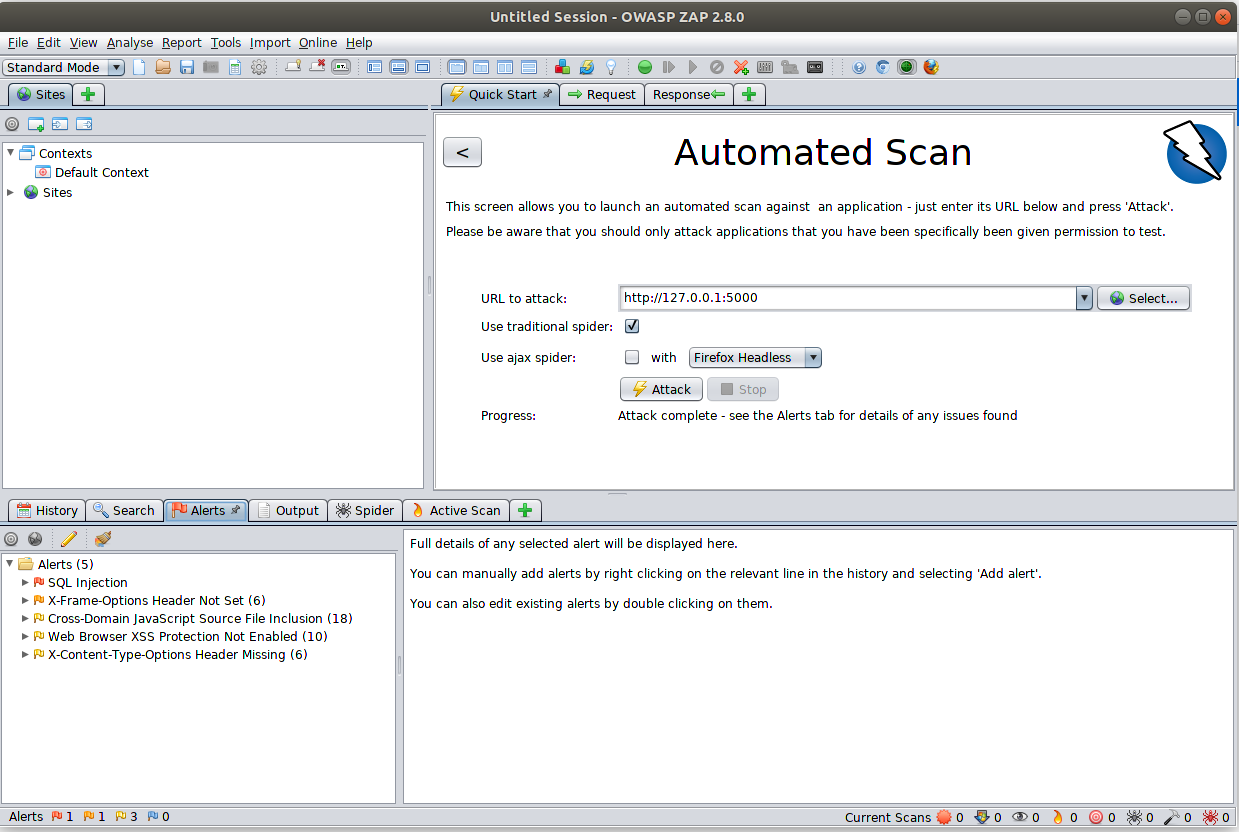
* XSS attack: This kind of attack is a type of injection, in which malicious scripts are injected in to trusted websites. I did a script injection in the inputtext file and I was expecting to have an alert pop up when the user enter the following script <script>alert(this is an attack)</script> but I did not have an pop up after I clicked the submit button. We decided to validate user input and ensure that data rendered by templates is escaped.



* CSRF attack: A CSRF attack occurs when a malicious actor tricks a victim into clicking on a link, or running some code, that triggers a forged request. In this program is possible to prevent this attack, if a csfr\_token is created. For this reason, we decided to add the following extension app.config[‘WTF\_CSRF\_ENABLED’] = True

**Detecting Vulnerabilities**

OWASP ZAP 2.8.0 was the tool used in order to find common Web service vulnerabilities. This tool is an open-source web application security scanner. After the installation of OWASP ZAP 2.8.0, select automatic scan, enter the URL to attack and click Attack and results could be found in the Alerts tab



OWASP ZAP found 5 vulnerabilities.

* SQL Injection: The page results were successfully manipulated using Boolean conditions. The vulnerability was detected by successfully restricting the data originally returned by manipulated the parameter. I didn’t catch it during the first part of the assignment because I was not thinking about malicious code in my sqlite3. In order to protect the website I need to use SQL parameters.
* X-Frame-Options Header Not Set (6): X-Frame -Options header is not included in the HTTP response to protect against ‘ClickJacking’ attacks. I didn’t catch this attack because I was not aware of iframing the content of my site. In order to improve the security of the site against ClickJacking, it is recommended to add the following header to the site. X-Frame-Options: SAMEORIGIN.
* Cross-Domain JavaScript Source File Inclusion (18): The page includes one or more script files from a third-party domain. I didn’t catch this potential attack because I did not know I was using third-party domain code. The solution is ensure JavaScript source files are loaded from only trusted sources.
* Web Browser XSS Protection Not Enabled (10): Web browser XSS protection is not enabled. I didn’t catch this potential attack because Firefox does not implement X-XSS-Protection. An alternative to fix this possible attack is ensure web browser’s XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to ‘1’.
* X-Content-Type-Option Header Missing (6): The Anti-MIME-Sniffing header X-Content-Type-Option was not set to ‘nosniff’. . I didn’t catch this potential attack because I did not know most browsers are now respecting this header, including Chrome/Chromium, Edge, IE >=8.0, Firefox >=50 and Opera >=13. In order to avoid this attack, ensure the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to ‘nosniff’ for all web pages