XSS Vulnerable Applications + Fixes

Source code can be found here: https://github.com/c2003-tamu/413/

Environment Setup

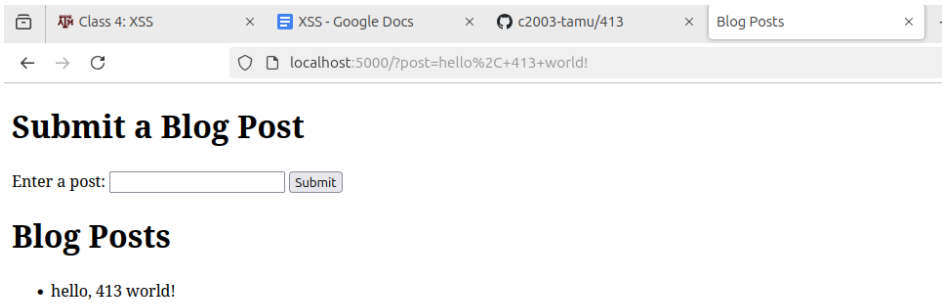
- Clone git repo
- Navigate to 413/xss directory
- Run command: python3 -m venv venv
- Run command: source venv/bin/activate
- Run command: pip install -r requirements.txt

Vulnerabilities

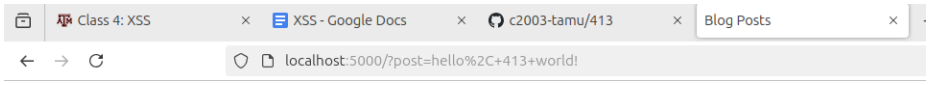
Stored XSS

The stored xss vulnerability can be found in the blog.py file. This is a sample blog application that stores posts simply in memory for simplicity.

- When we run the python file blog.py, we can see that the base page is a simple blog page, where posts can be entered and displayed:



- Without proper sanitization of user inputs, we cannot verify that users are not inputting a malicious payload to this field or not, thus users can utilized a stored xss that will execute code in the browser of each user who visits the site:

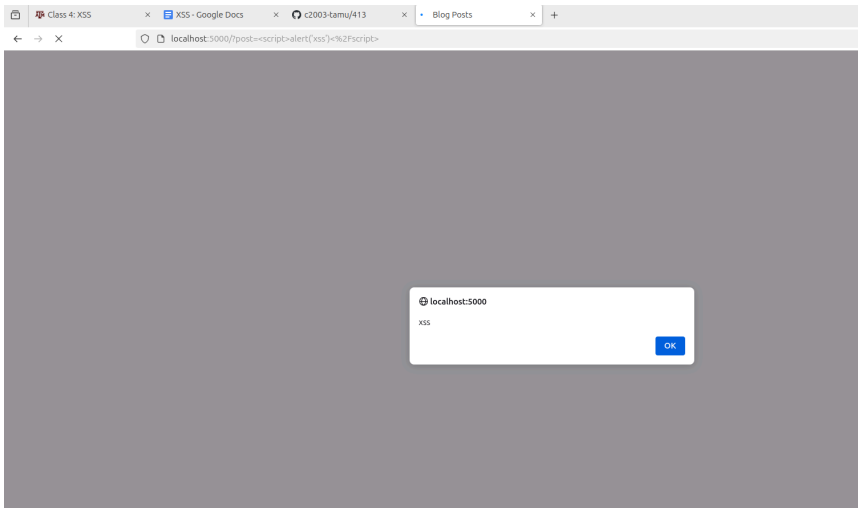


Submit a Blog Post

Enter a post: <script>alert('xss')</script> Submit

Blog Posts

- hello, 413 world!



Reflected XSS

The reflected XSS vulnerability can be found in the search.py file. This is a sample search engine that simply takes in a user input as a search term and displays it. There is no actual "search" being executed.

- When we run search.py file, we can see that the base page is a simple user input with a prompt for the user to input a search term:

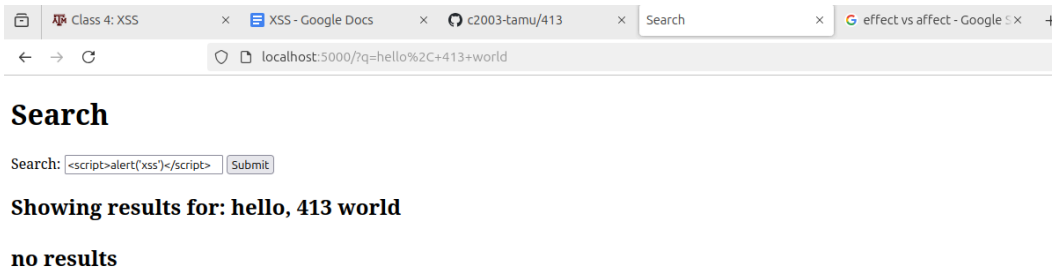
Search

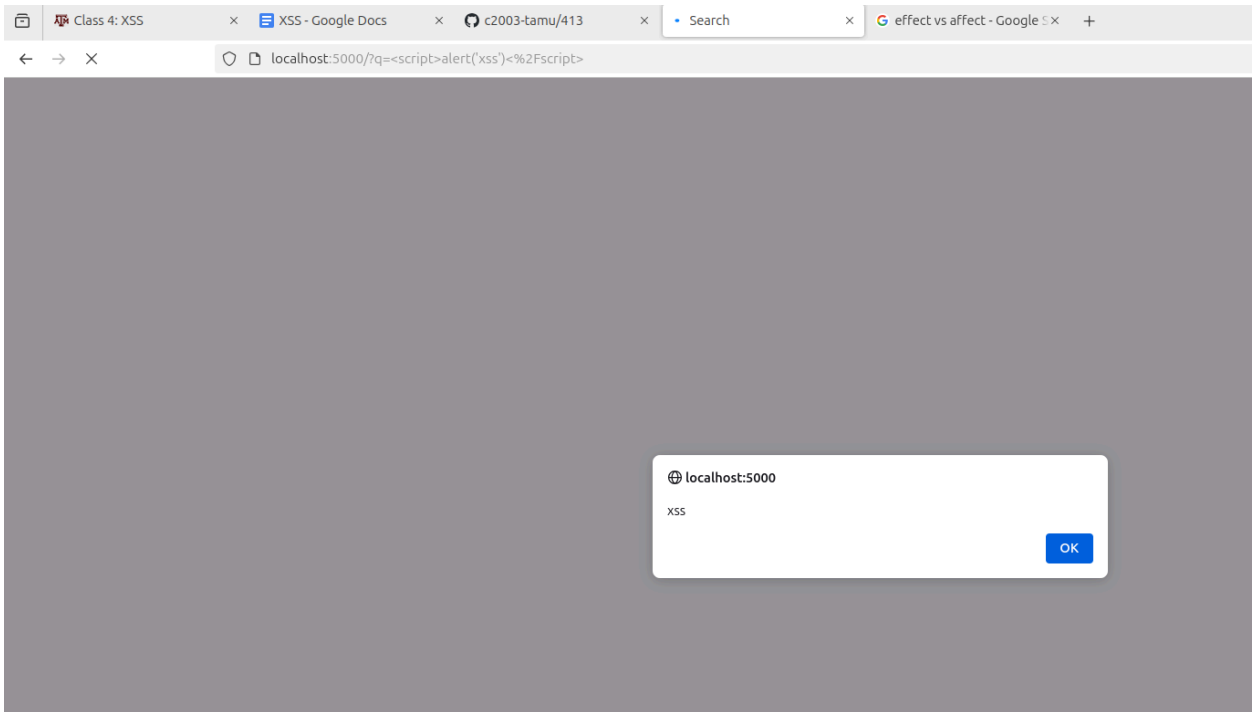
Search: Submit

Showing results for: hello, 413 world

no results

- Again, without proper sanitization, this serves a threat to execute code in the browsers of users. The key difference between reflected xss and stored xss is the fact that stored xss is stored on the server, and serves as a threat to each user who visits the application. On the other hand, reflected xss only affects users who visit a specific page, typically through a maliciously crafted url. Example below:



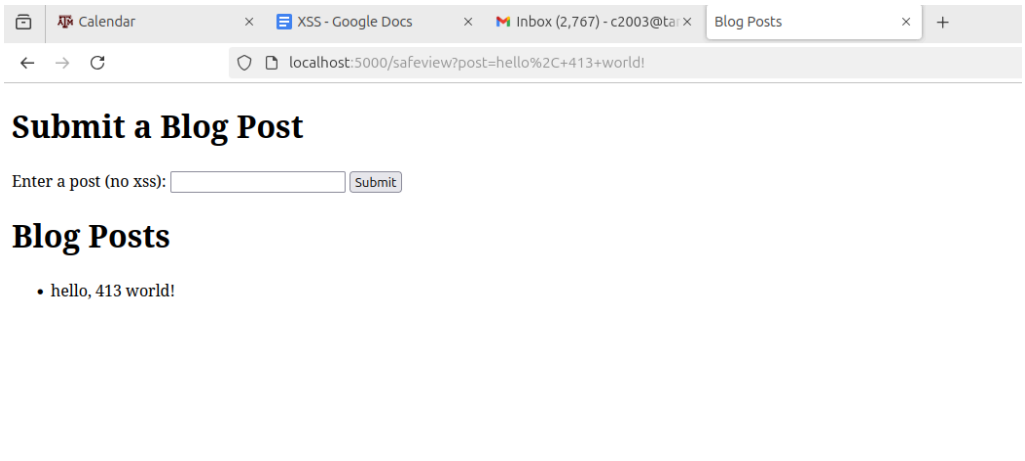


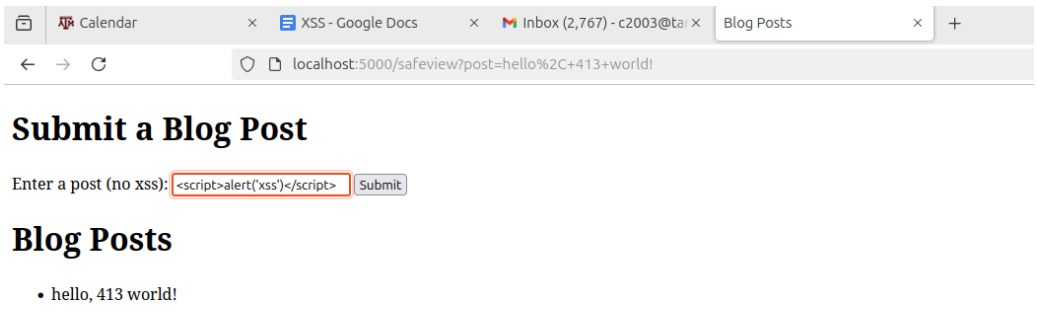
Fixes

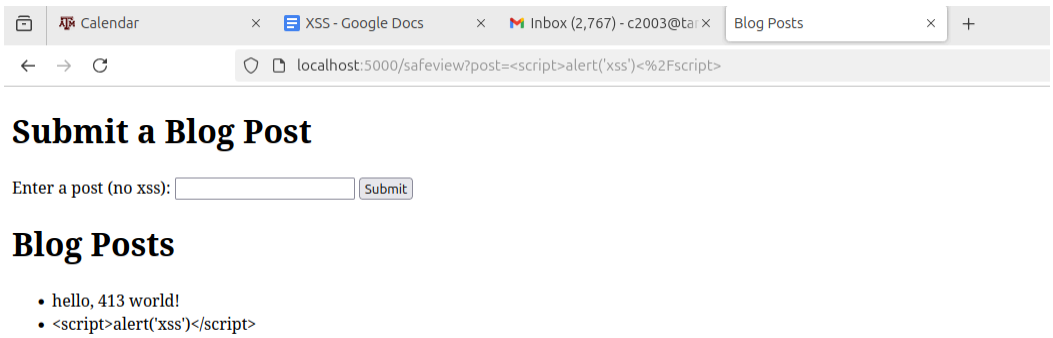
For both of these cases, the solution is sanitization of user inputs. As stated in class, it is best practice to never trust user input. The solution looks slightly different for each case.

Stored XSS

In the case of stored xss, we should ideally sanitize user input at time of ingestion and escape user input at time of display. To see this in action, we can visit the /safeview when running blog.py

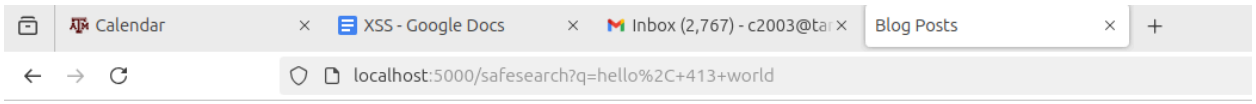






Reflected XSS

In the case of reflected xss, we simply need to escape user input before displaying that same input back to the user. The solution in action can be seen at the /safesearch endpoint when running search.py

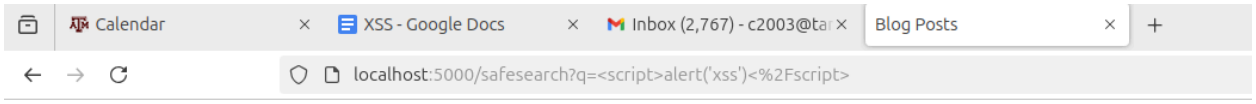


Search

Search: Submit

Showing results for: hello, 413 world

no results



Search

Search: Submit

Showing results for: <script>alert('xss')</script>

no results