Secure Programming

Week 4 (Part1)

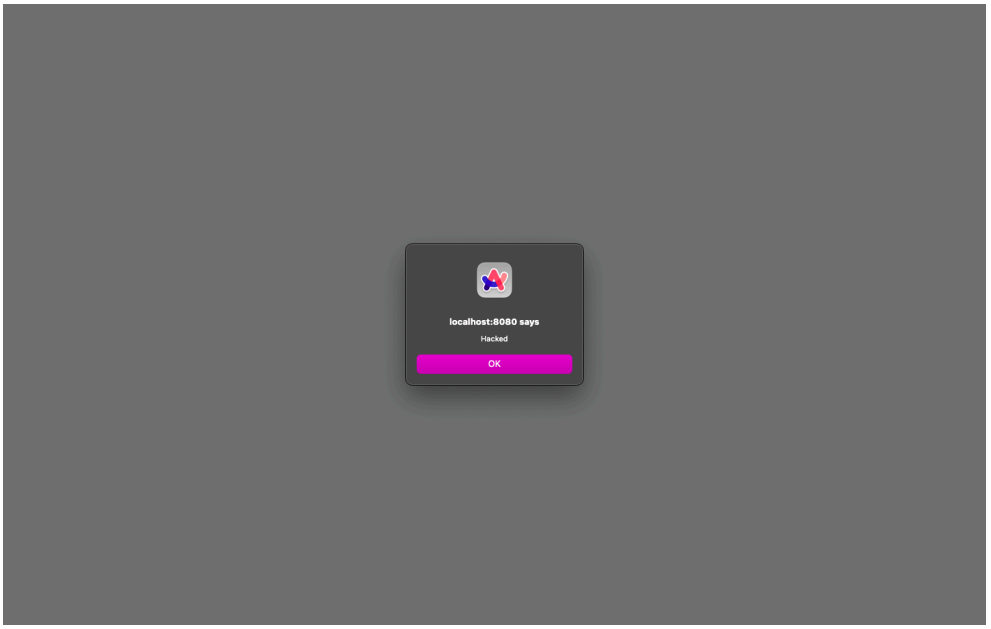
XSS

Exploiting a Reflected XSS Vulnerability

When trying to exploit reflected XSS, we are looking for places in the web application that user input is displayed back to the user, e.g., search boxes, error messages etc. The Coffeeshop application has a vulnerable page where the products are displayed, using the URL: `http://localhost:8080/prod/?id={productid}`

1. What do you see?

   Since the user input is displayed unchanged, we can set a script into the URL and check to see if it works. Try the following:
   `http://localhost:8080/prod/?id=%3Cscript%3Ealert%28%22Hacked%22%29%3C%2Fscript`



   → We see a JavaScript alert pop-up in the browser with the text "Hacked". The page has executed the injected `<script>` tag we placed in the id parameter.

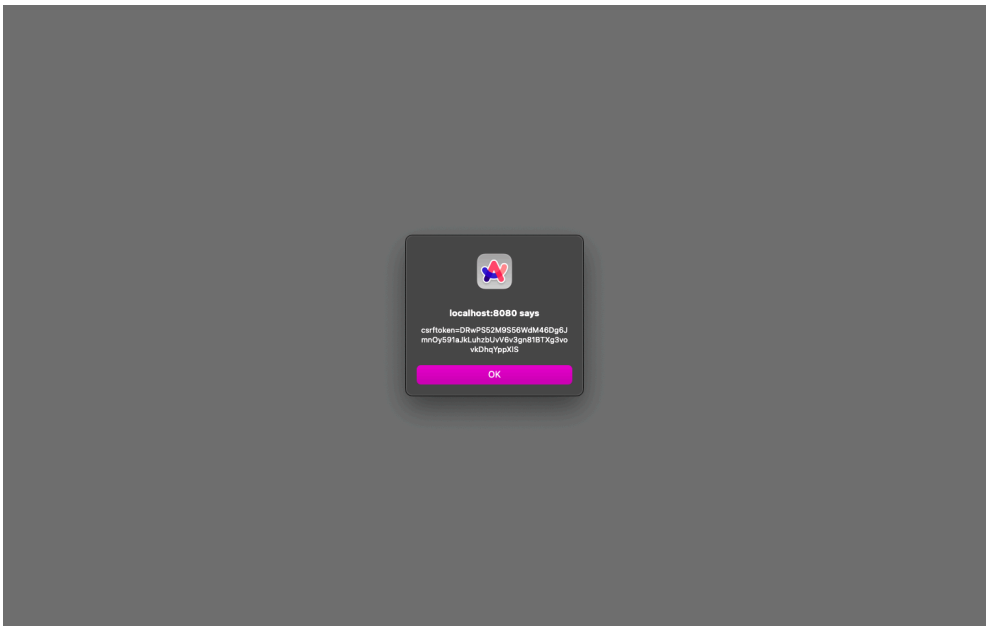
2. What do you think the inserted code means?

   The inserted code is a small JavaScript snippet inside `<script>...</script>` tags. `alert("Hacked")` tells the browser to show a dialog box with the text *"Hacked"*.

If the code is `alert(document.cookie)`, it means "show the current page's cookies" – `document.cookie` returns cookie strings (which often include the session ID/auth token). In other words, the attacker's injected code runs in the victim's browser with the same privileges as the page and can read sensitive data (cookies, DOM), modify the page, or send data to an attacker-controlled server.

3. Now what do you see?

   Change the attack URL to:
   http://localhost:8080/prod/?id=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E



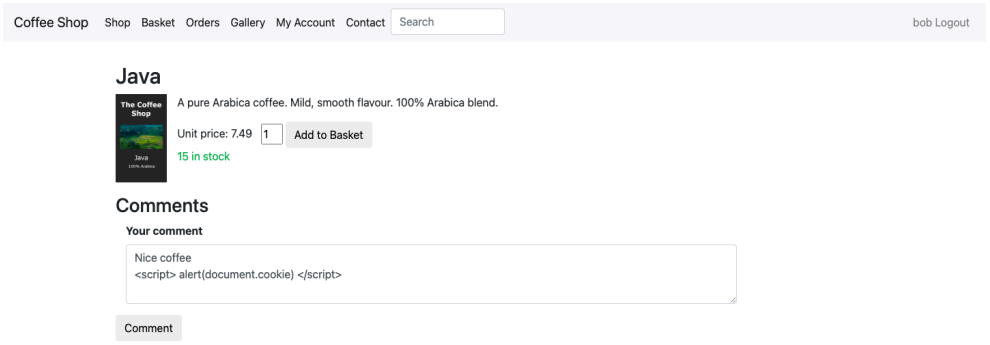
→ We see an alert pop-up that contains the site's cookies (the value of `document.cookie`) – typically including session IDs and possibly other tokens (CSRF token in our case). This demonstrates that an attacker can obtain authentication/session data from a logged-in user and use it to impersonate them or hijack their session.

Exploiting a Stored XSS Vulnerability

Sometimes, user input is persisted in an application's database and then displayed in HTML pages. Examples include social media posts, forum posts, blog posts, product reviews etc. If an attacker can get malicious JavaScript code stored there, it will be executed whenever another user visits the page that displays that input. It will be executed as that user, with that user's cookies.

Enter the following comment and submit:

```
Nice coffee
<script> alert(document.cookie) </script>
```



We are getting a pop-up alert. This alert contains 2 items: the CSRF token and the session ID. The session ID is the authentication token for Bob's current session and this could be used by another user to impersonate Bob.