

**Lab #4**  
Matthew Langlois - 7731813  
Nov. 27

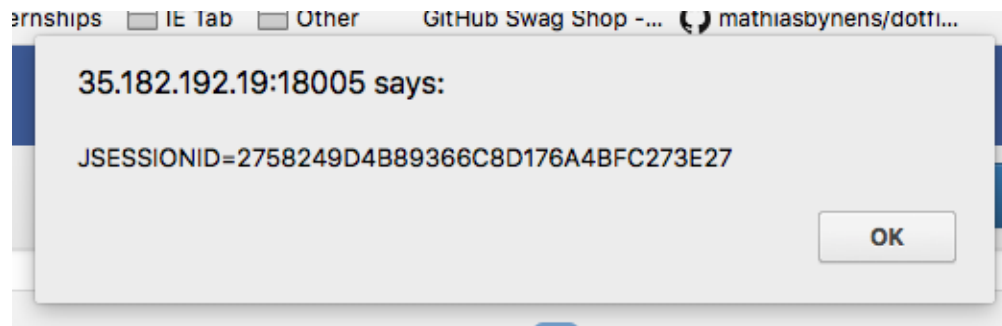
---

1. The first vulnerability was very straight forward. There was a reflected XSS in the search parameter. To exploit this vulnerability the XSS injection just needed to be placed in the query parameter of the URL.

The code used was: `<script>alert(document.cookie)</script>`

The vulnerability could have been prevented by sanitizing the user's input and ensuring that they are not injecting Javascript

Here's an example of the vulnerability being exploited:

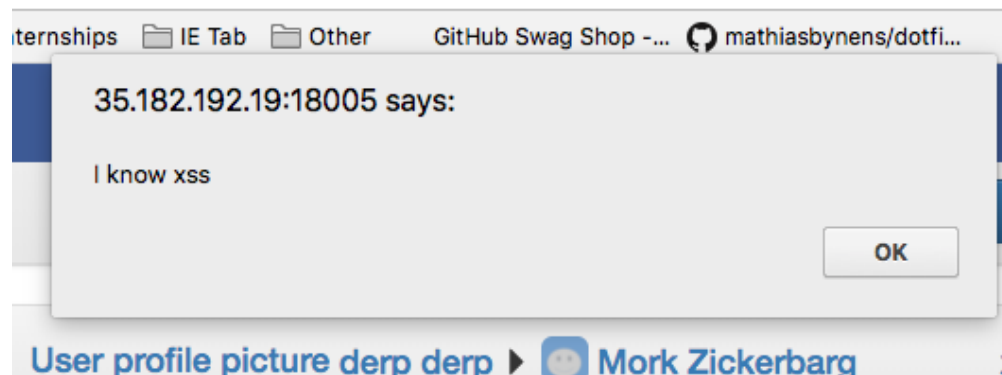


2. The second vulnerability was also quite straight forward. There was a stored XSS in the post section of Facebroke. To exploit this vulnerability the XSS injection just needed to be placed in the contents of the post.

The code used was: `<script>alert('I know xss')</script>`

The vulnerability could have been prevented by sanitizing the user's input and ensuring that they are not injecting Javascript.

Here's an example of the vulnerability being exploited:

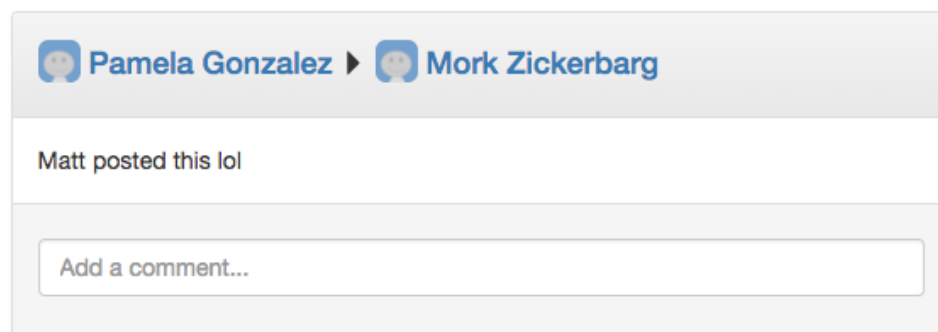


3. The third vulnerability made use of modifying a POST request as it was in transit. To complete this task I setup a simple proxy using Burp to intercept the post message request. Inside the post request there was a variable representing the user's ID. This could be changed to be another user thus posting it on their behalf.

No code required modifications for this vulnerability. Instead the POST request required modification.

The vulnerability could have been prevented by using proper access control at the endpoints. In this case the user posting the comment should have been checked against the user which is currently logged in and sending the request. To do this the ID of the user should be stored in their session which can be identified by the cookie. Then the session can be compared against the id in the request.

Here's an example of the vulnerability being exploited:

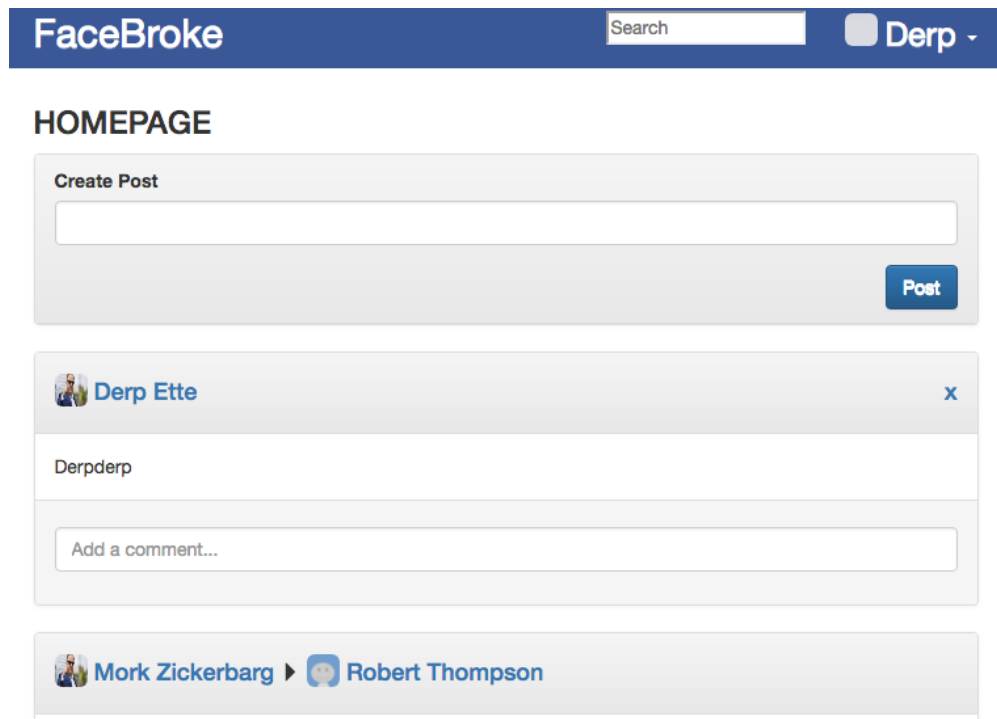


4. The 4th vulnerability made use of spoofing the user's ID again. When uploading an image the superuser ID 103 could be set thus allowing the image to be uploaded on behalf of another user.

Similar to the last vulnerability, the exploit required no code but rather modifying a POST request.

Much like the last exploit this one could have been prevented with proper access control and authorization checks on the endpoint.

Here's an example of the vulnerability being exploited:



5. The 5th vulnerability made use of a very basic SQL injection attack in the query parameter of the get request.

To exploit this vulnerability all that was required was setting the userid parameter to `1 or 1=1 --`.

The vulnerability could have been prevented by sanitizing the user's input. Furthermore prepared statements could have been used to ensure that SQL injection isn't possible.

Here's an example of the vulnerability being exploited:

FaceBroke			
User Info			
Pamela	Gonzalez	pamelagonzalez92@fake.ca	PamelaGonzalez92
Sharon	Reyes	sharonreyes34@fake.ca	SharonReyes34
Melissa	Lopez	melissalopez2@fake.ca	MelissaLopez2
Maria	Miller	mariamiller76@fake.ca	MariaMiller76
Stephanie	Bailey	stephaniebailey33@fake.ca	StephanieBailey33
Cynthia	Ruiz	cynthiaruiz37@fake.ca	CynthiaRuiz37
Kenneth	Moore	kennethmoore71@fake.ca	KennethMoore71
Nicholas	Morales	nicholasmorales93@fake.ca	NicholasMorales93
Carol	Campbell	carolcampbell96@fake.ca	CarolCampbell96
Emily	James	emilyjames12@fake.ca	EmilyJames12
Kathleen	Cox	kathleencox0@fake.ca	KathleenCox0
Matthew	Smith	matthewsmith84@fake.ca	MatthewSmith84
Samuel	Sanders	samuelsanders6@fake.ca	SamuelSanders6
Jennifer	Jimenez	jenniferjimenez71@fake.ca	JenniferJimenez71
Kimberly	Bennett	kimberlybennett94@fake.ca	KimberlyBennett94
Jose	Lopez	joselopez13@fake.ca	JoseLopez13
Susan	Mendoza	susandmendoza51@fake.ca	SusanMendoza51
Amanda	Cook	amandacook82@fake.ca	AmandaCook82
Scott	Kim	scottkim2@fake.ca	ScottKim2
Christina	Campbell	christinacampbell90@fake.ca	ChristinaCampbell90
Deborah	Vasquez	deborahvasquez2@fake.ca	DeborahVasquez2
Matthew	Murphy	matthewmurphy46@fake.ca	MatthewMurphy46
Barbara	Lee	barbaralee23@fake.ca	BarbaraLee23

- The last vulnerability made use of an XSS within an SVG file. The SVG spec allows the use of script tags. When the browser renders the SVG the vm notices the Javascript and executes it. Thus, in this case, popping up the alert box.

The code used was:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full"
  xmlns="http://www.w3.org/2000/svg">
  <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900"
    stroke="#004400"/>
  <script type="text/javascript">
    alert('This app is probably vulnerable to XSS attacks!');
  </script>
</svg>
```

The vulnerability could have been prevented by disallowing SVG files from being uploaded for display pictures. However if SVG is absolutely required then the uploaded file should be sanitized to ensure that there are no script tags.

Here's an example of the vulnerability being exploited:

The screenshot displays a web application interface with a dark blue header. On the left, the text "FaceBroke" is visible. On the right, there is a search bar with the placeholder text "Search" and a dropdown menu showing "Derp". A central modal window is overlaid on the page. The modal has a title bar that reads "35.182.192.19:18006 says:" and a message "This app is probably vulnerable to XSS attacks!" with an "OK" button. Below the modal, the user profile form is visible. It includes fields for "Email address" (Derpette@derp.com), "First Name" (Derp), "Last Name" (Ette), "Date of Birth" (1994-03-23), "Password" (Password), and "Confirm Password" (Confirm Password). There is a "Save Changes" button. Below the form, there is a "Profile Picture" section with a note "PNG or JPG up to 2MB in size", a "Choose File" button, and a status "No file chosen". There is also an "Upload" button. At the bottom of the form, there is a button labeled "Get My DB Representation".