# THE MEGA HACKING GROUP PROJECT OF DEATH

Total Report

By: Joelle Waugh

& Elicia Ramitt

Instructor: Adam “Abe” Aberthney

Course: COMP 357 Advanced Pentesting

Date: December 9, 2025

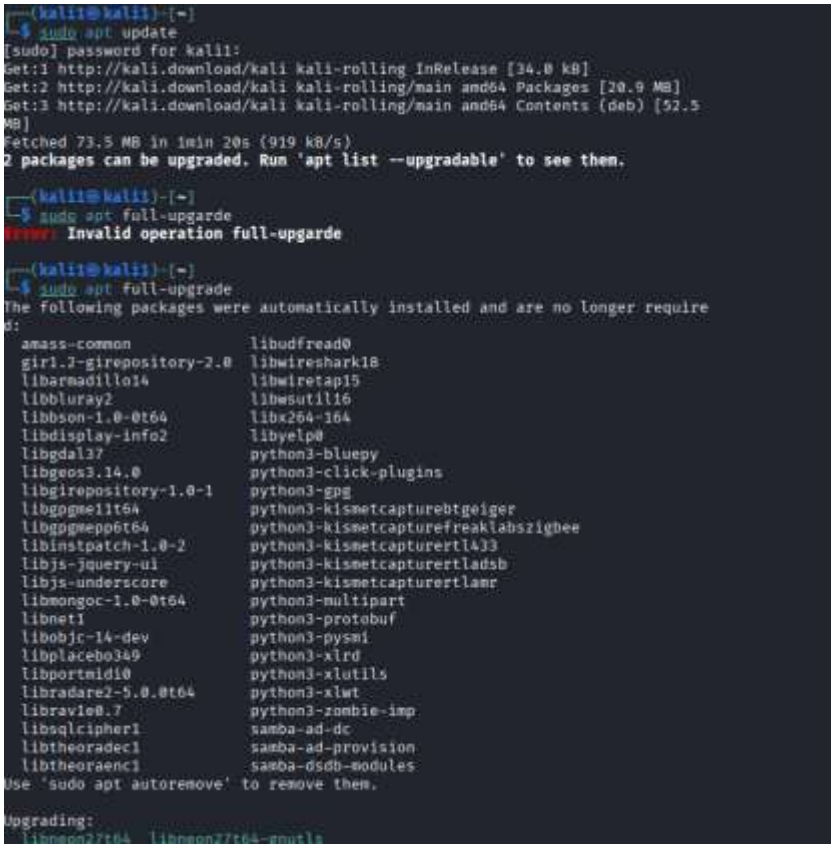
# Table of Contents

# Lab Creation-Browser Exploitation Framework

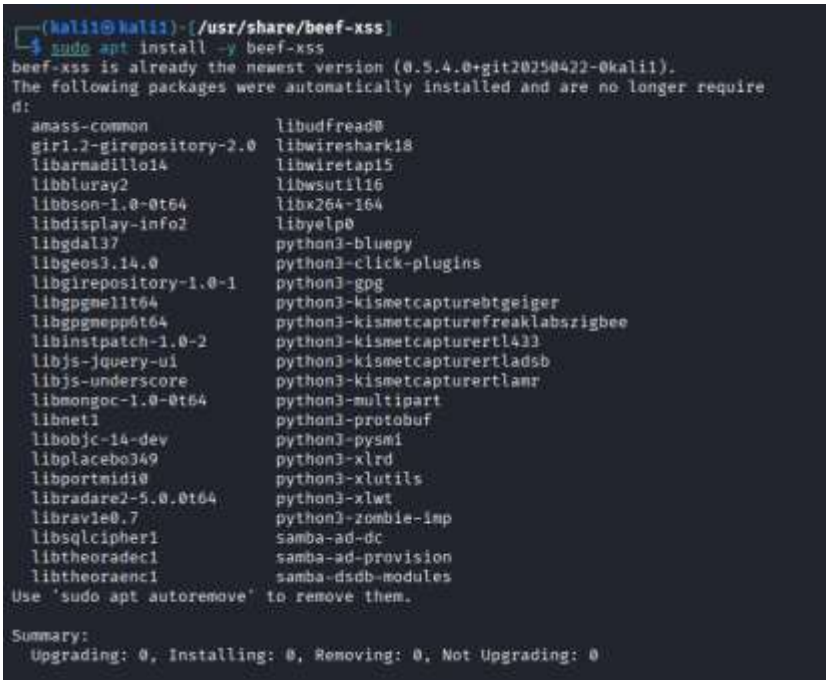
This lab focuses on using BeEF for Browser Exploitation Framework and the Gophish simulation was optional. There is somewhat of a phish simulation using the BeEF control panel command later shown in this lab.

To begin, install the latest version of Kali VM, 2025.4.  For this lab, you only require one VM.  As shown below:

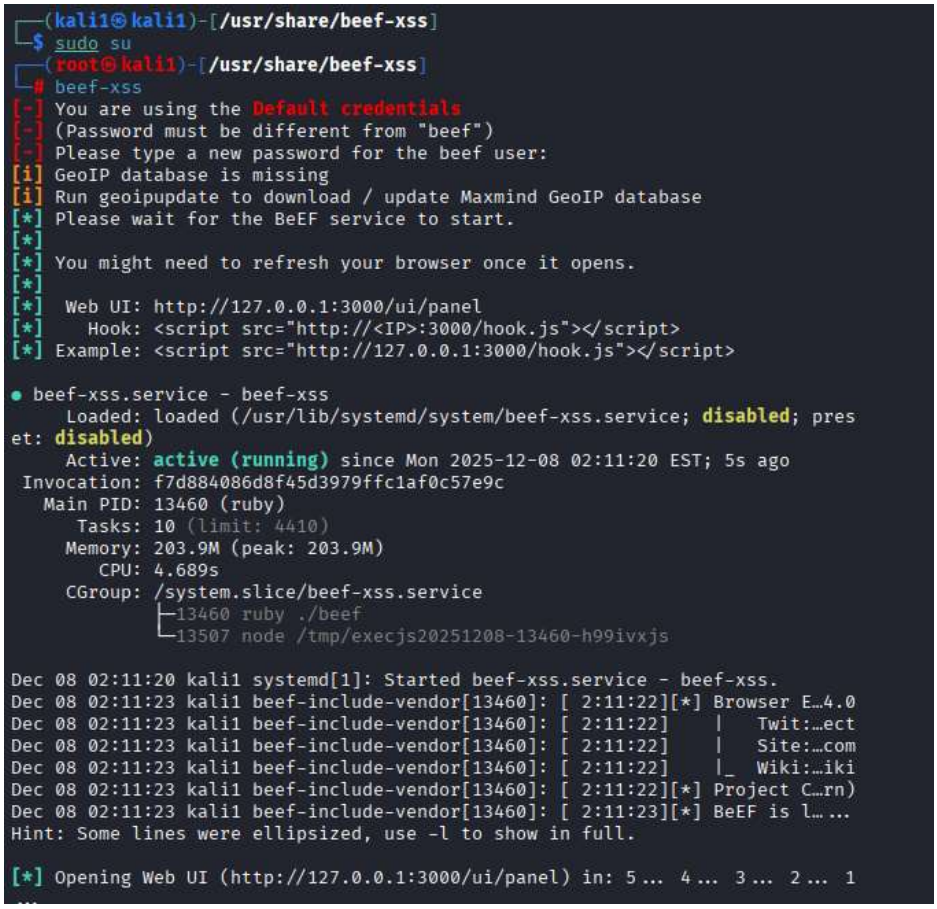
Be sure to run *sudo apt update and sudo apt full-upgrade* before starting, as shown below.



For this lab you will need to install beef on kali. As shown below. Use the command *sudo apt install -y beef-xss*. I have already installed it. It will also make sure you go into */usr/share/ beef-xss* folder on kali.

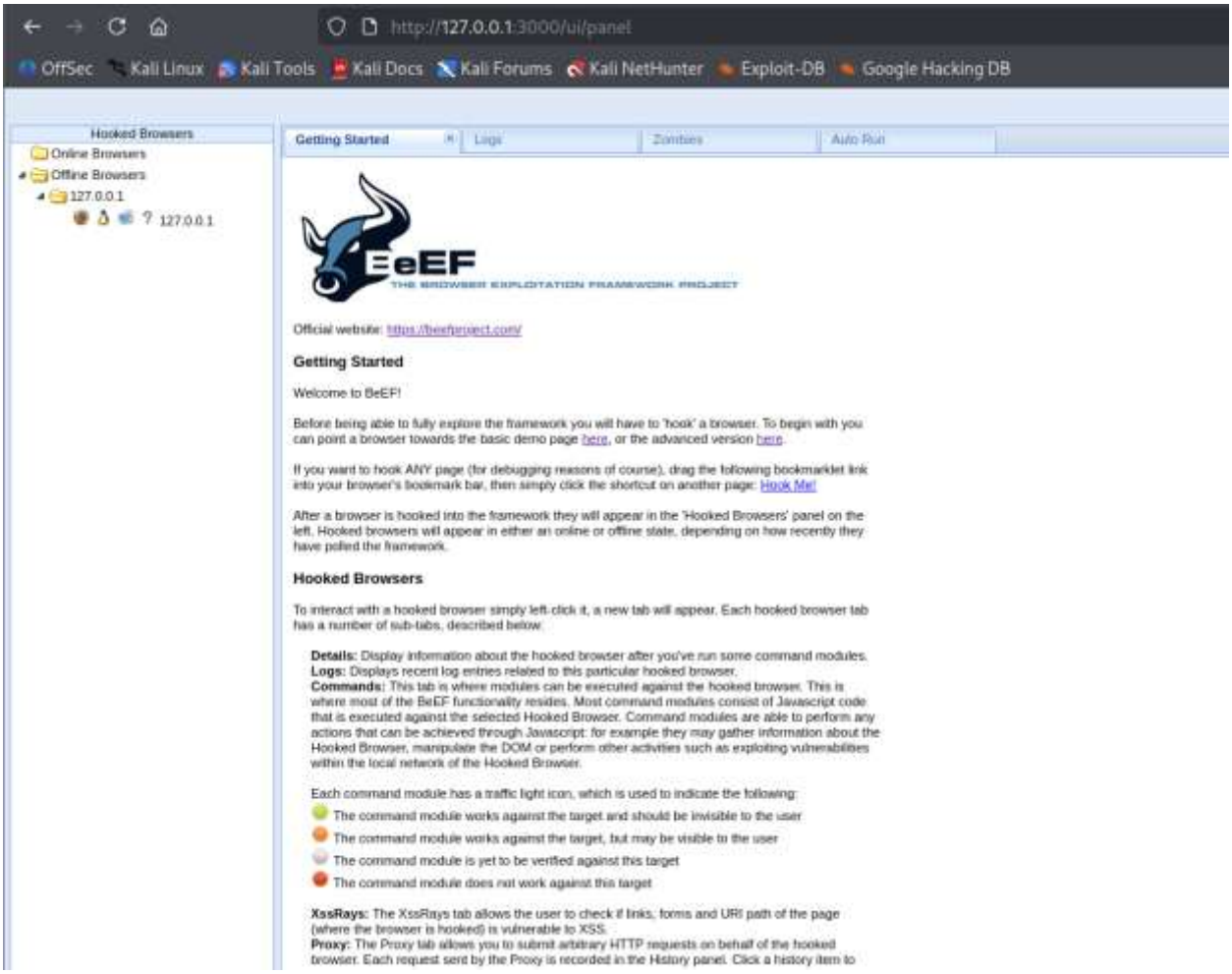


To run use the command *beef-xss* shows below show that is active and running and a password of your choice.



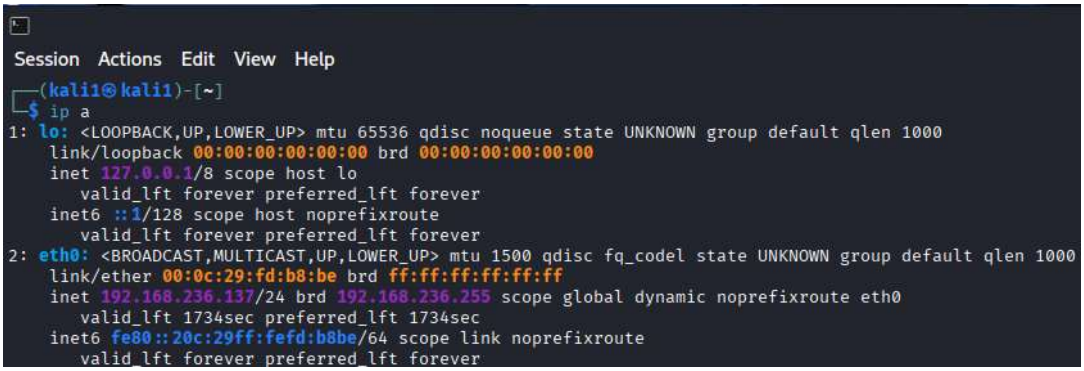
Once you start, you'll be prompted to create a password.  It will then activate the beef and automatically open the graphical interface, where you can enter your username and password. As shown below.

Once you log in, you will see this page as shown below. It is an information page that directs you to all the practice hooks. The version of beef used is 0.5.4.0.

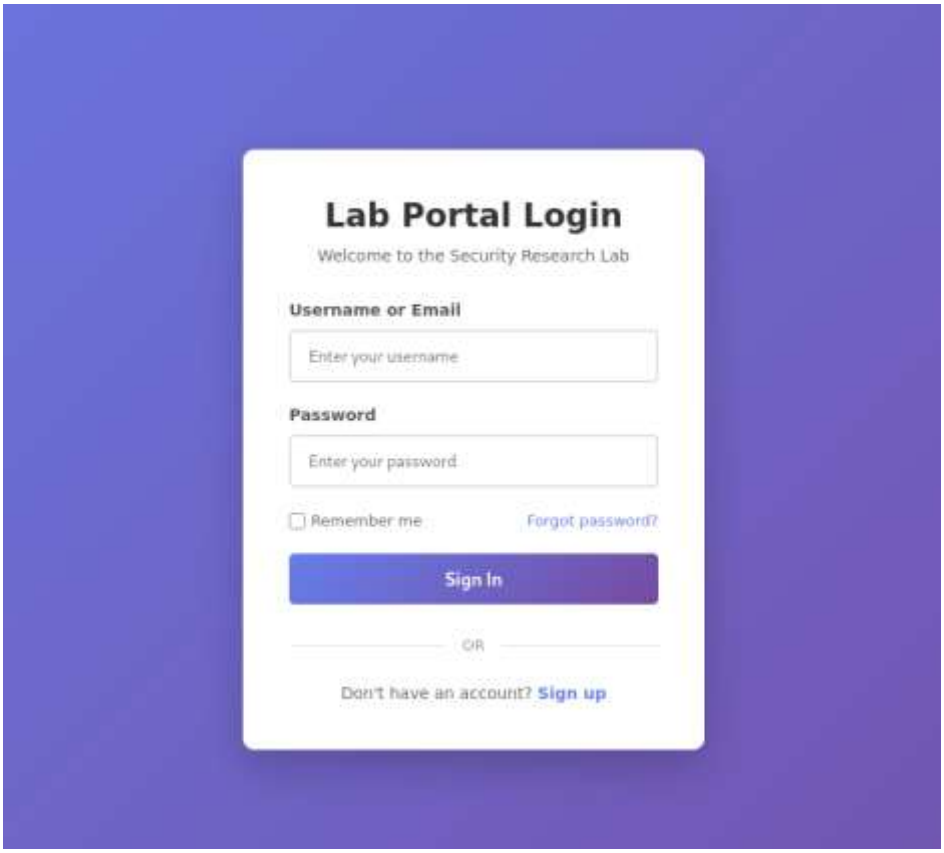



For this lab, we will use the website's demo page to run some of the commands. In addition to the two lab websites, create a sample page where users can enter information. For example, a copy of the code is available on GitHub.  One thing to note in the code is that a script is used to hook the webpage.

The BeEF control panel using the loopback IP 127.0.0.1:3000. For your webpage to work use the IP of the Kali virtual machine as shown below:

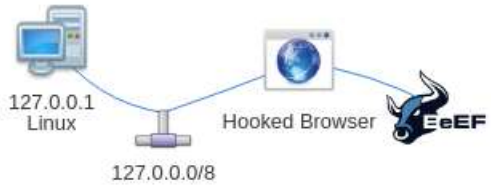



The webpage should appear as below.




Once you have your code and webpage, let's perform some safe attacks on a secure test environment from BeEF and on our own.

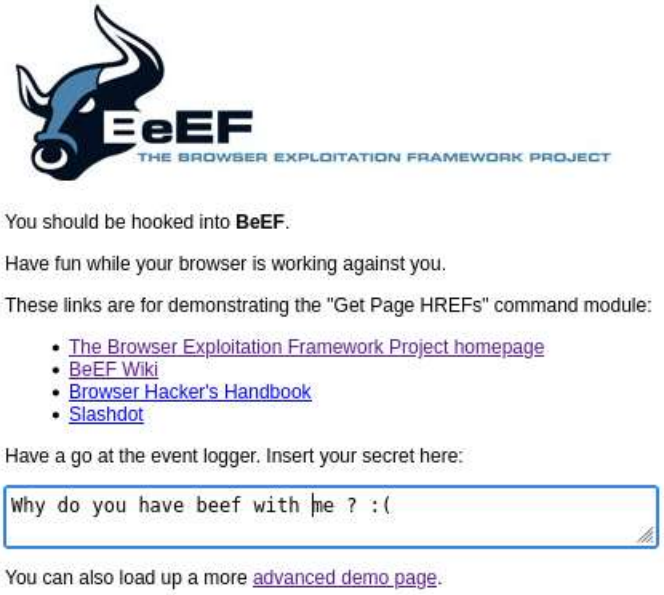
There are two infrastructures one is when connecting to the Linux command using the local host and the other the VM's IP address, as shown below:



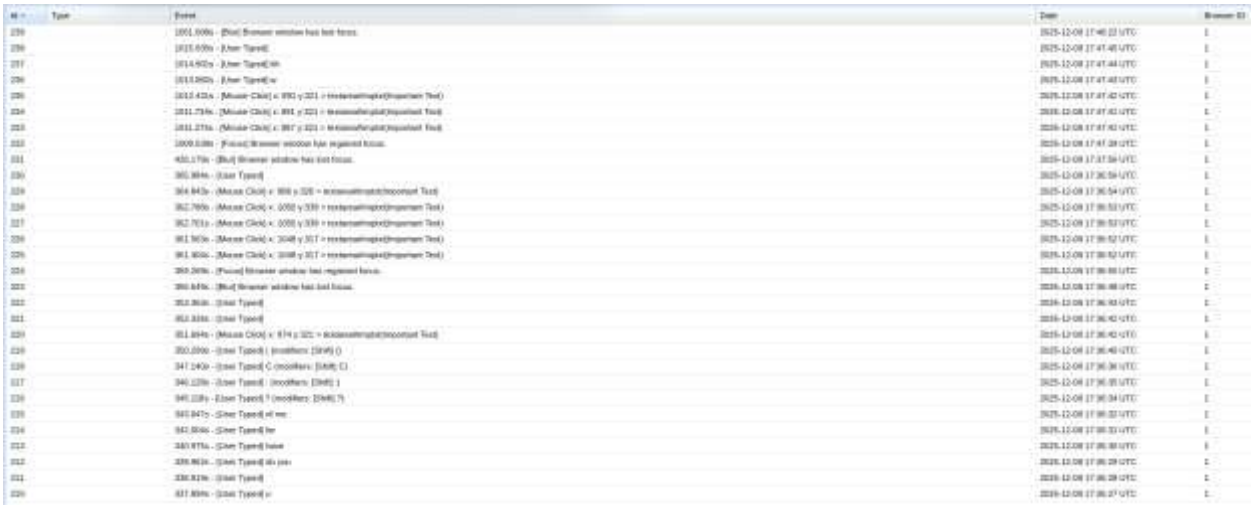

# Attack Report: Browser Exploitation Framework

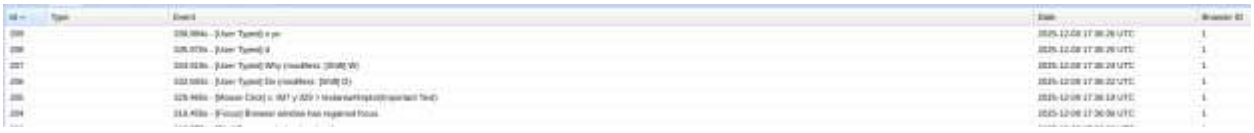
Here is how to go through the attacks performed using the BeEF control panel, demos and create a webpage.

- On the get-start page, two links say here. Click on the first one and a webpage should open as shown below. Follow the steps on the page and check the logs.

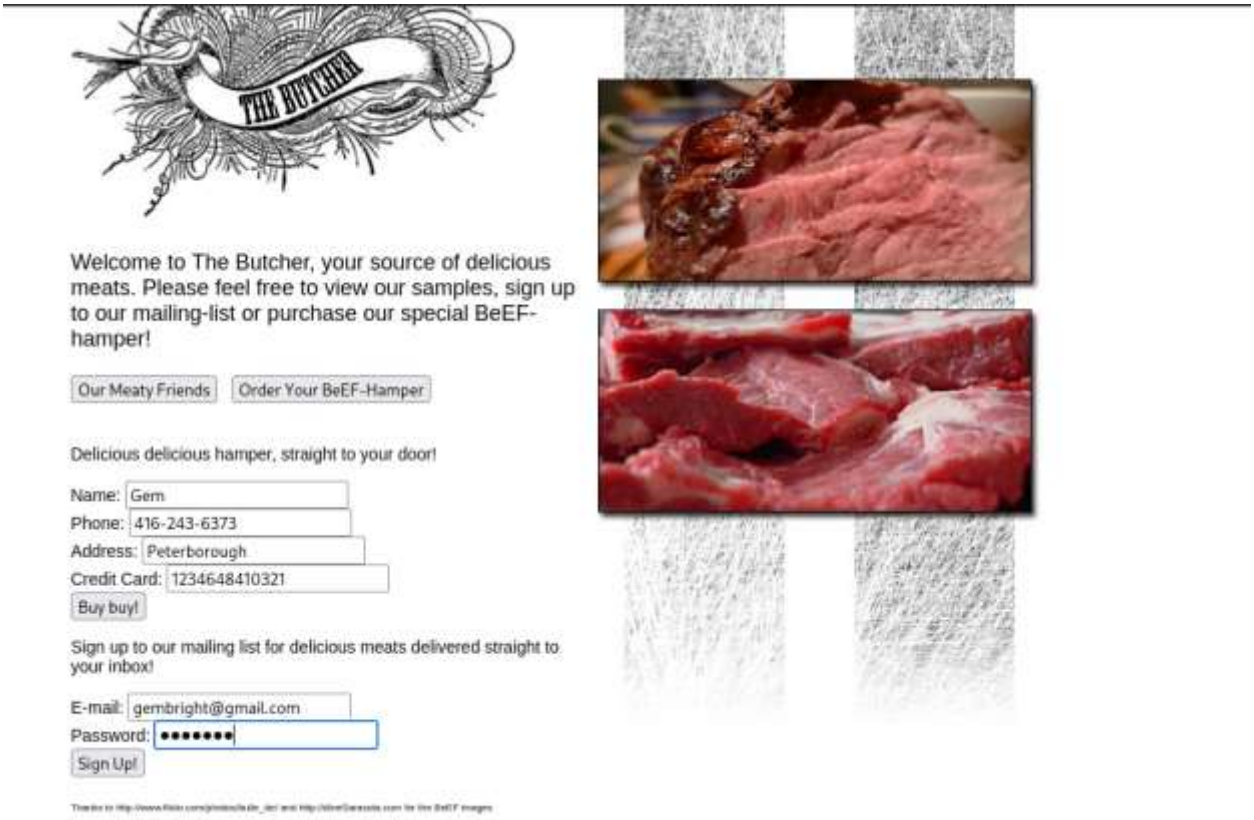



- In the log, you can see if we can find the secret, as shown below. In the log below you we see how detailed it as it logs how I typed in the phrase. It also shows what keys I used.

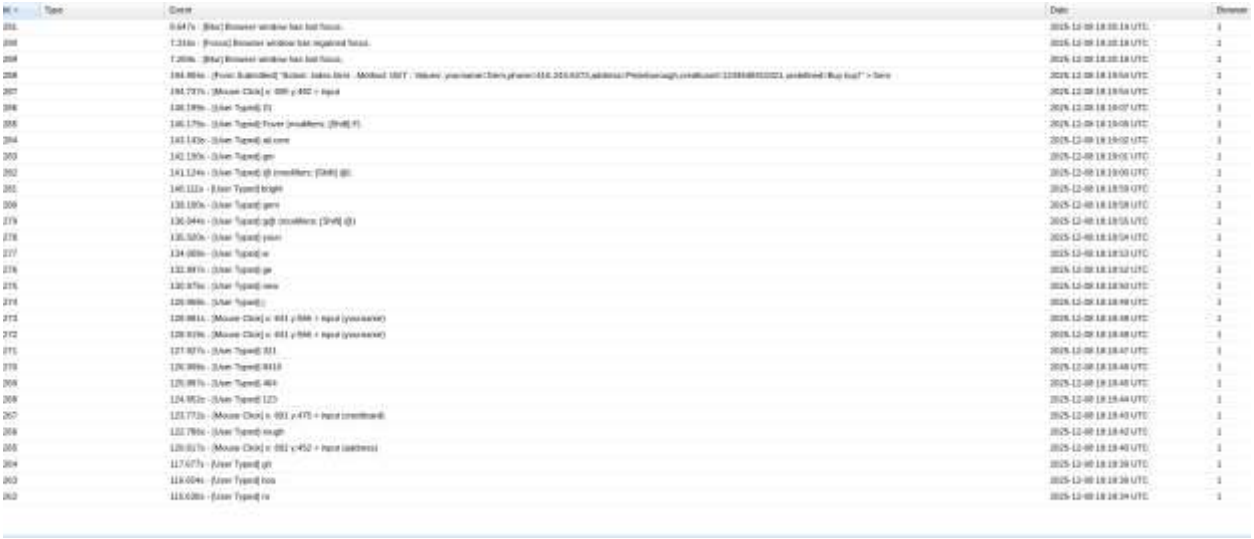




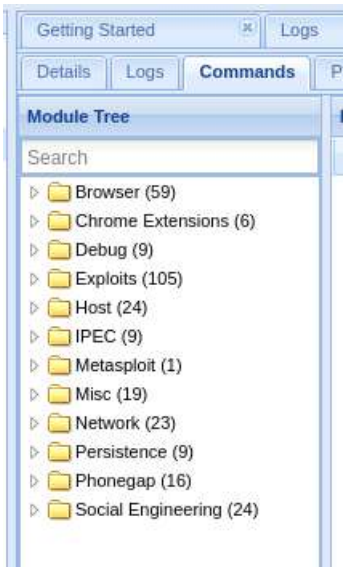
- Now we will try the advanced version. Return to getting started page. Click on the link and the page should look like as shown below and click on the order “Your BeEf-Hamper button and order beef”.




- Click *buy buy!* And sign up and return to the log page to see the information that was encrypt and not encrypted.



- Select current browser and select commands.

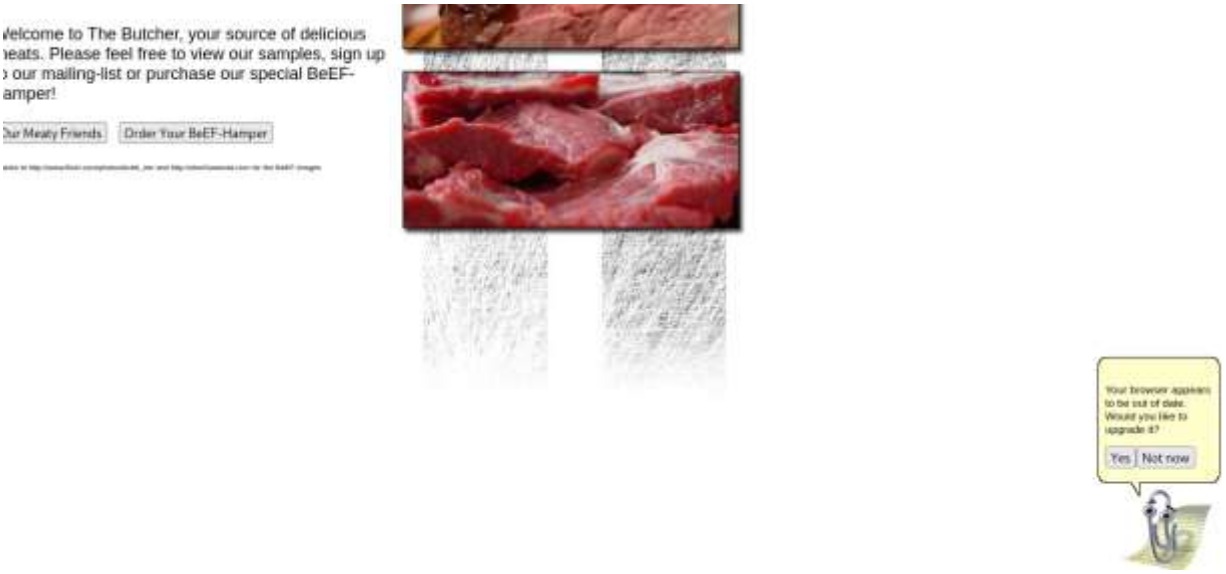



- There are several options, to execute. *Under browser – Hooked Domain – Get Cookie to get the cookie* go back to the webpage and type in the information. Then return to the BeEF control Panel and execute the cookie command. Got to the logs make sure to refresh the page and results should be shown.


Below are the logs:

- We are going to use some other commands to see how it works. Under social engineering folder select *clippy*. You will see a page as shown below. Leave the IP Address. Click on execute and go back to the demo webpage.

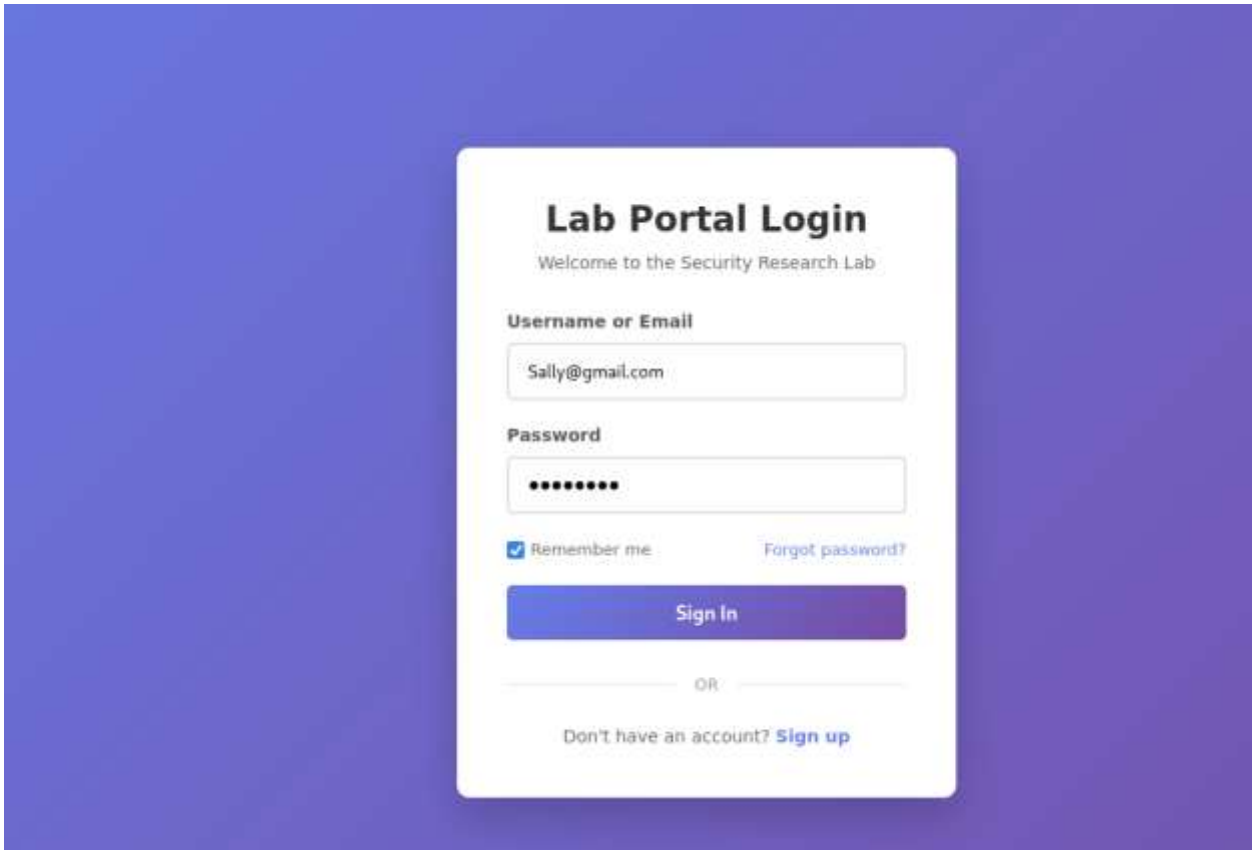
- In the bottom right corner, you will see *clippy* from Microsoft word asking you to upgrade the browser now.



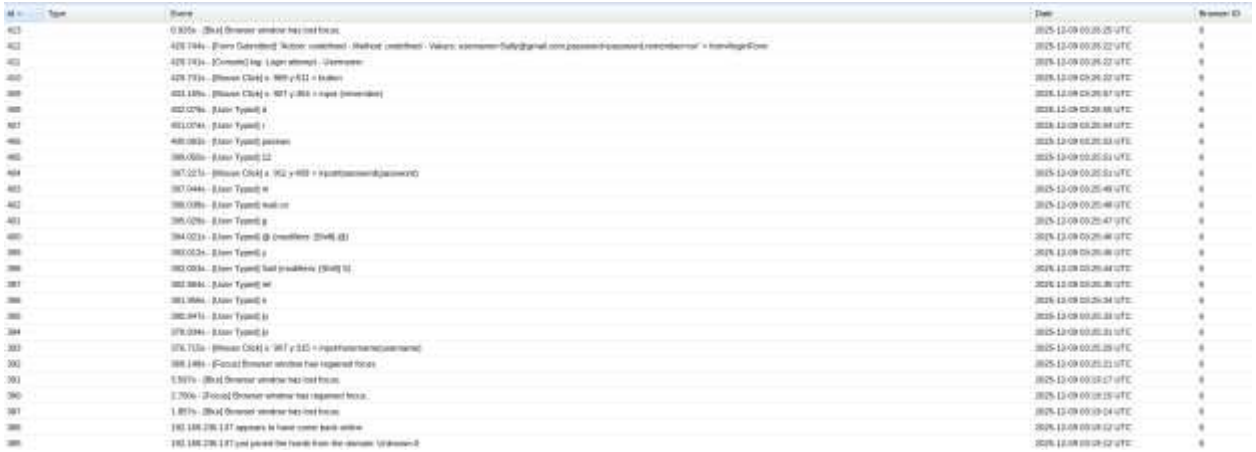

- Return to the logs and you can see that *clippy* was executed.

| Id | Type | Event |
|---|---|---|
| 384 | | 860.135s - [Blur] Browser window has lost focus. |
| 383 | | 844.175s - [Mouse Click] x: 1751 y:667 > button |
| 382 | | Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: UNKNOWN) from command module [cid:15, mod: 23, name:'Clippy'] |
| 381 | | 749.762s - [Focus] Browser window has regained focus. |
| 380 | | 2.585s - [Blur] Browser window has lost focus. |
| 379 | | 127.0.0.1 appears to have come back online |
| 364 | | 890.357s - [Focus] Browser window has regained focus. |
| 363 | | 193.368s - [Blur] Browser window has lost focus. |
| 362 | | 193.000s - [Focus] Browser window has regained focus. |
| 361 | | 179.679s - [Blur] Browser window has lost focus. |
| 360 | | 177.304s - [Focus] Browser window has regained focus. |
| 359 | | 171.238s - [Blur] Browser window has lost focus. |
| 358 | | 155.561s - [Mouse Click] x: 1761 y:653 > div#pipes |
| 357 | | 155.016s - [Mouse Click] x: 1804 y:660 > div#pipes |
| 356 | | 154.353s - [Mouse Click] x: 1805 y:664 > div#pipes |
| 355 | | 154.314s - [Focus] Browser window has regained focus. |
| 354 | | Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: UNKNOWN) from command module [cid:14, mod: 18, name:'Fake Notification Bar'] |
| 353 | | 133.886s - [Blur] Browser window has lost focus. |
| 352 | | 130.966s - [Mouse Click] x: 1670 y:497 > div#pipes |
| 351 | | 129.038s - [Mouse Click] x: 1756 y:656 > button |
| 350 | | Hooked browser [id:1, ip:127.0.0.1] has executed instructions (status: UNKNOWN) from command module [cid:12, mod: 23, name:'Clippy'] |
| 349 | | 127.884s - [Mouse Click] x: 1756 y:656 > div#pipes |
| 348 | | 125.190s - [Focus] Browser window has regained focus. |
| 347 | | 1.859s - [Blur] Browser window has lost focus. |
| 346 | | 19.425s - [Form Submitted] "Action: index.html - Method: GET - Values: yourname=hekh,phone=ghdsa,address=3456,creditcard=1234567890,undefined=Buy buy!" > form |

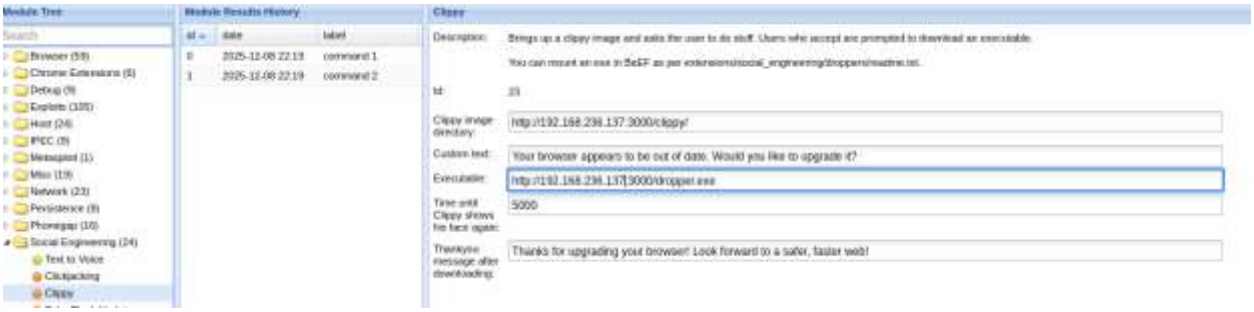
- For this part launch the created website in the browser. We will execute similar commands on this webpage but more to see how it reacts. First will login in than enter the credentials and take a look at the logs.




- Beef captured all of my action included the delete information. It captures the username, keystrokes, and my password.

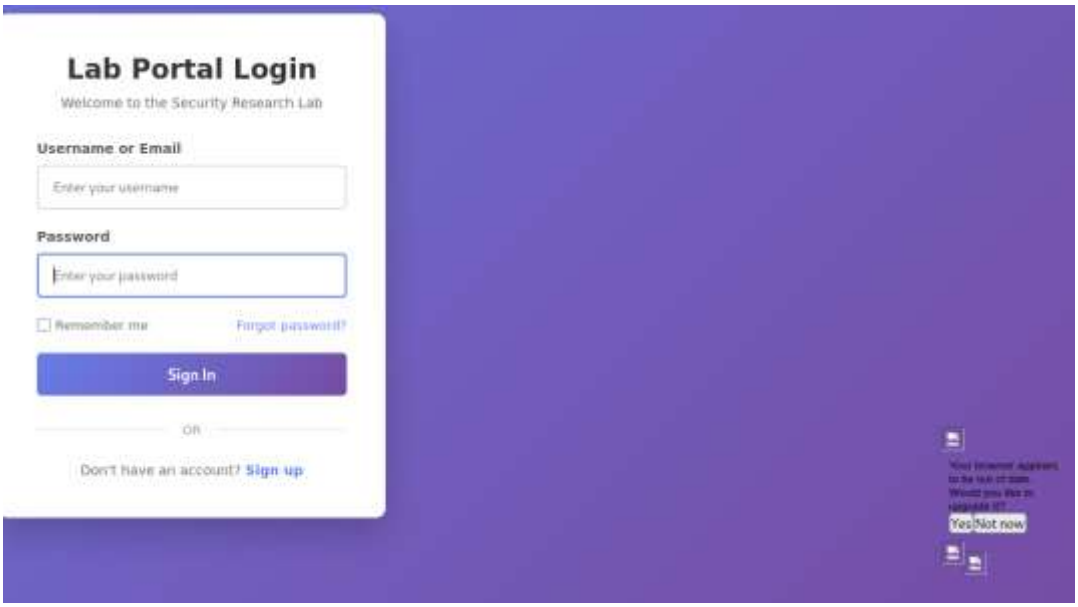


- For the next commands, we will execute the *clippy* again however, it going to be directed to the IP of the website where it is begin hooked. Make sure to click execute.



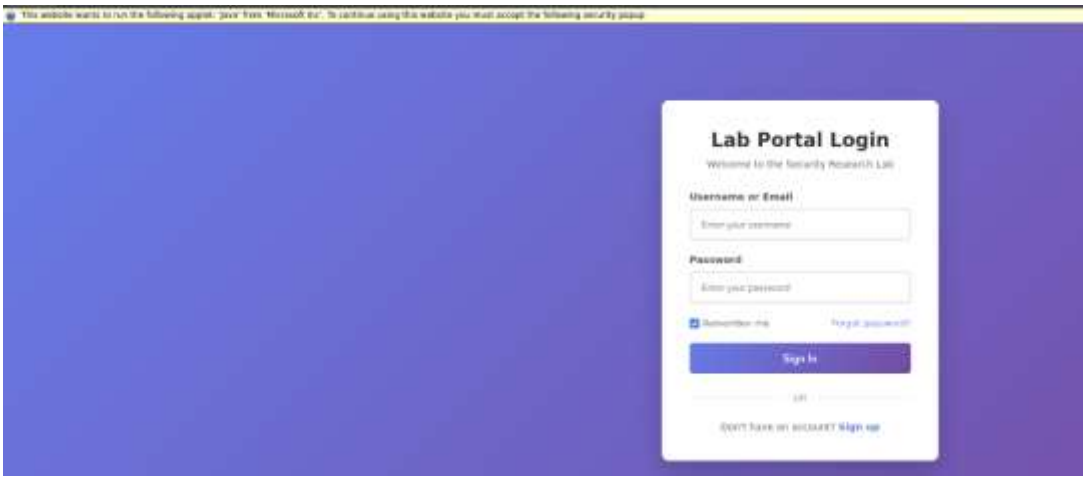
- Below are the logs show the execution.

- Below is an example of how the *clippy* showed up on the created website.

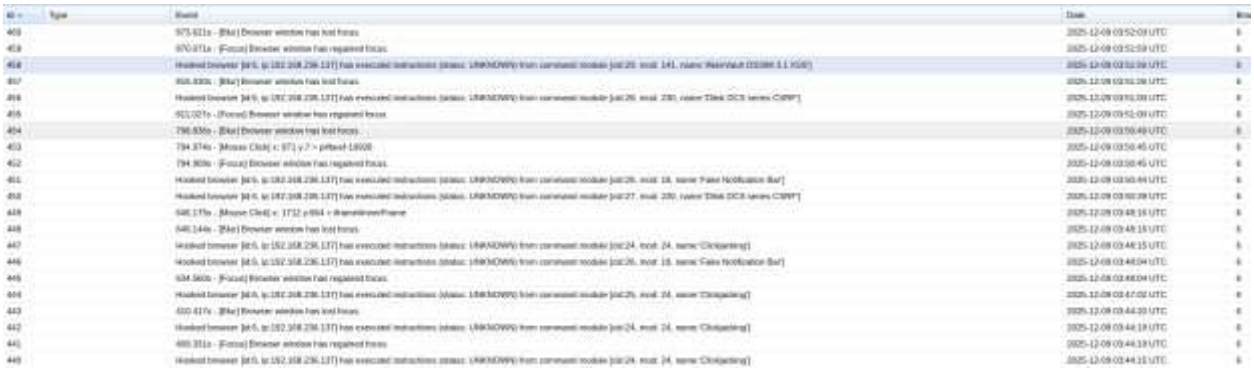



- The next social engineering attack is Fake notification Bar.

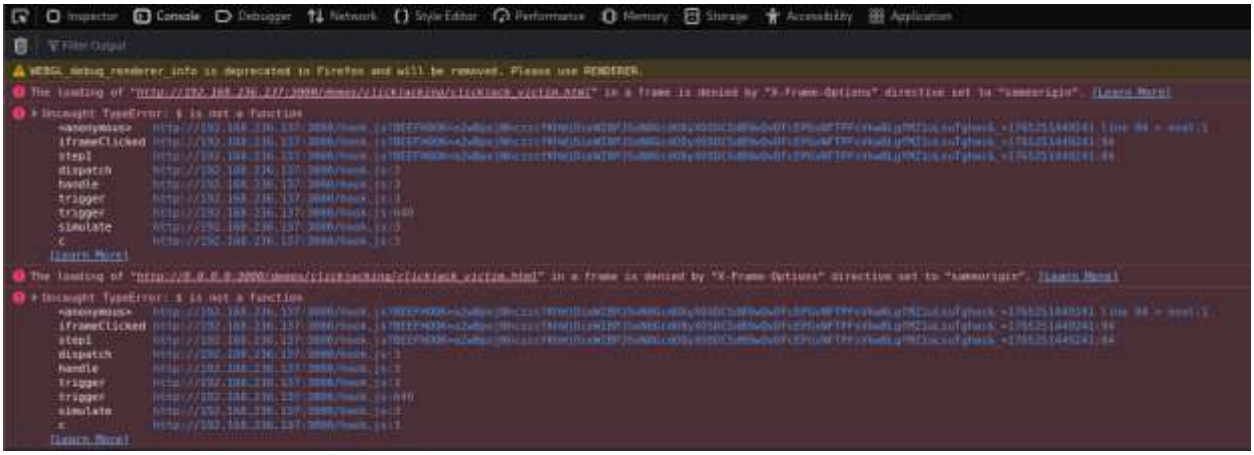
- The Results it also changed the orientation of the page.



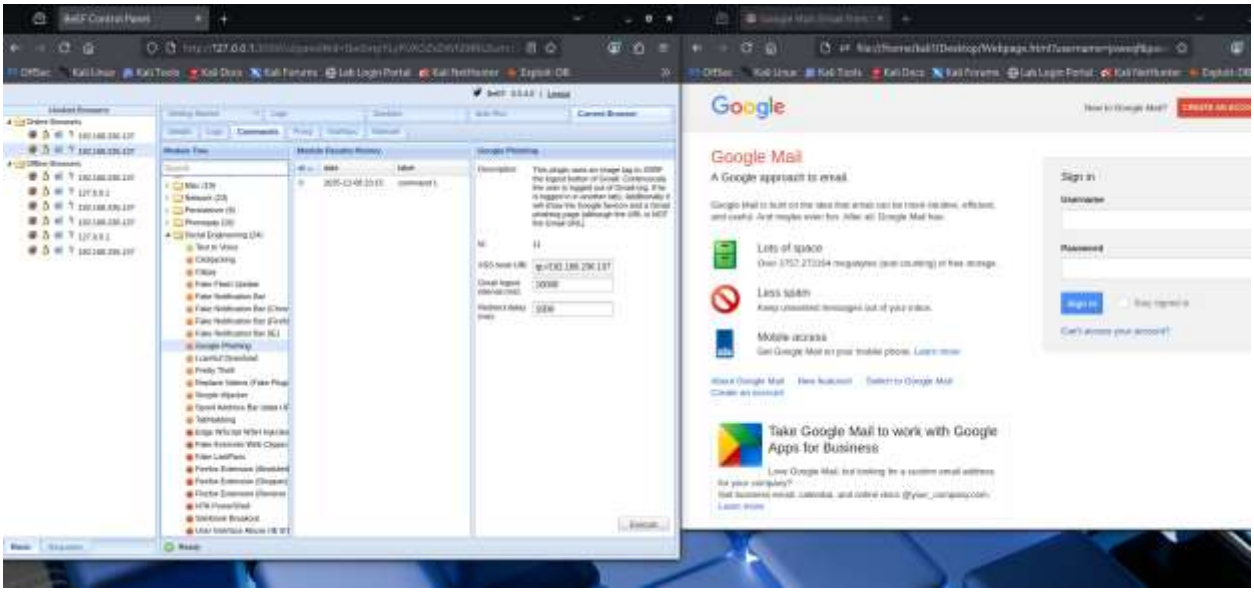

- In the log below you can see all the commands that were executed and tested.



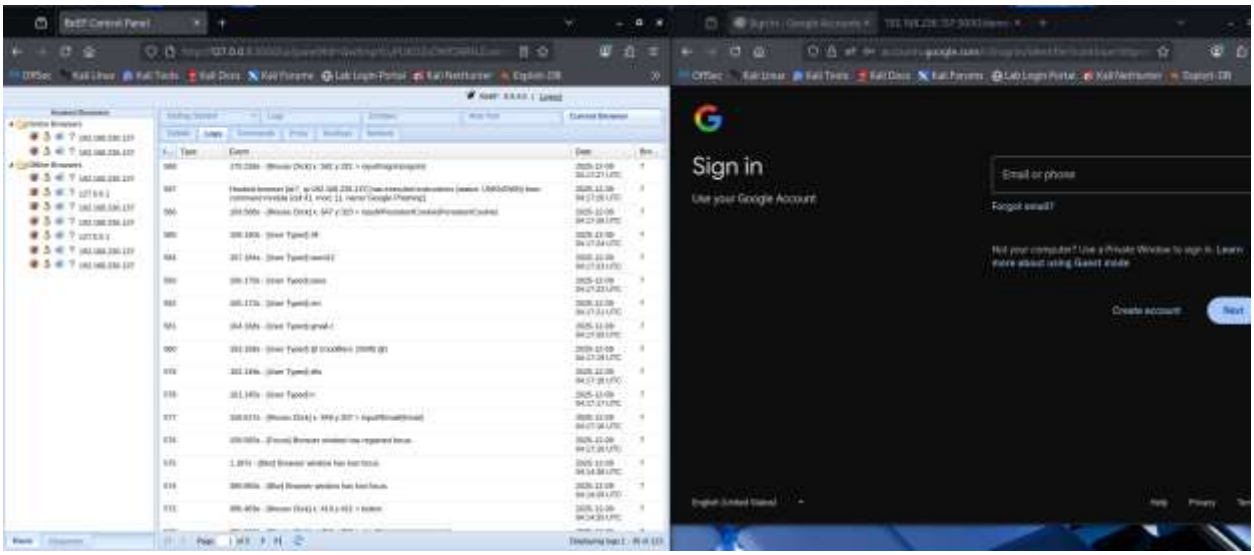
- For example, I used to *clickjack,* this highjacked my clicking ability. In an attempted to see my other execution I inspected the page and found that clickjack worked. I had to exit the page and restart.




- The last test is using google phishing webserver. In this case once you execute the command it replaces the lab Login portal. As shown below:

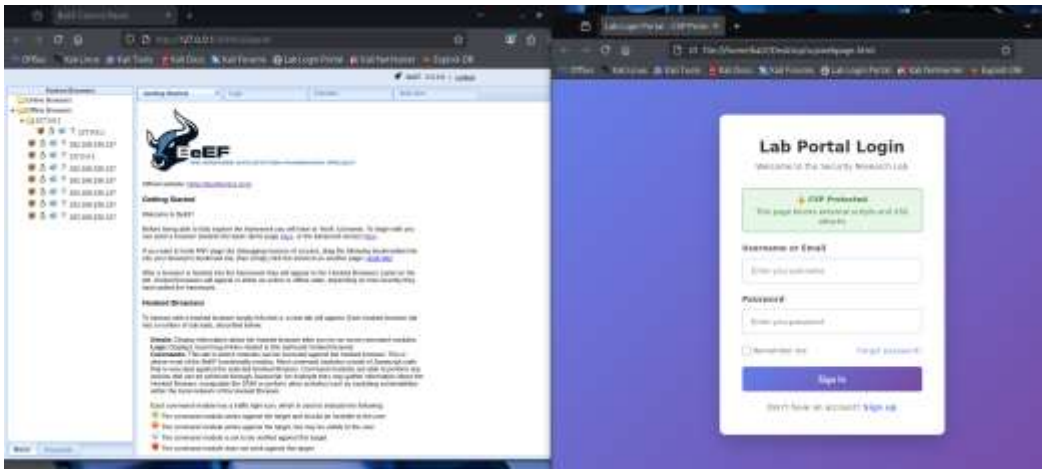



- You can enter your credentials, and it will capture and take you to the correct webpage. It also attempted to access another webpage using the Kali IP address.




# Purple Team Mitigation Report

In the Purple Team exercise, we saw how a website with no security controls can be compromised. There is another HTML file, cspwebpage, on GitHub if you would like to use it. Execute the webpage and see what happens. After CSP-protected code is in place, BeEF cannot hook the website to collect logs or execute malicious code. This is a good way to defend your website, along with other methods such as a WAF. Even with the *Hook Me* option enabled, the website appears in the BeEF control panel as shown below.


Lab Portal Login
Username or Email
Password
Sign In

# Lab Creation-Juice Shop

I was running this through docker on a Windows 11 Machine. Docker was already set up on my machine.

## Application Deployment

I pulled a docker instance from bkimminich/juice-shop.

```
C:\Users\Administrator>docker pull bkimminich/juice-shop
Using default tag: latest
latest: Pulling from bkimminich/juice-shop
Digest: sha256:1c55debeaf4fd5678019b17818a539e1e06ef93d29b268a21f53f0773a9fff5d
Status: Image is up to date for bkimminich/juice-shop:latest
docker.io/bkimminich/juice-shop:latest
```

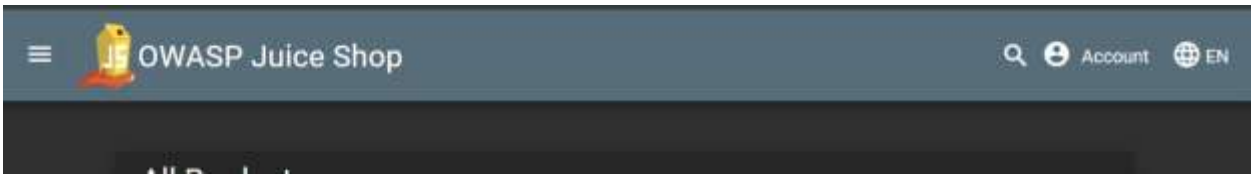
I ran the docker on the localhost using port 3000.

```
C:\Users\Administrator>docker run --rm -p 127.0.0.1:3000:3000 bkimminich/juice-shop
info: Detected Node.js version v22.21.1 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 20 of 20 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file index.html is present (OK)
info: Required file main.js is present (OK)
info: Required file vendor.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file styles.css is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```

# Attack Report

## SQL Injection

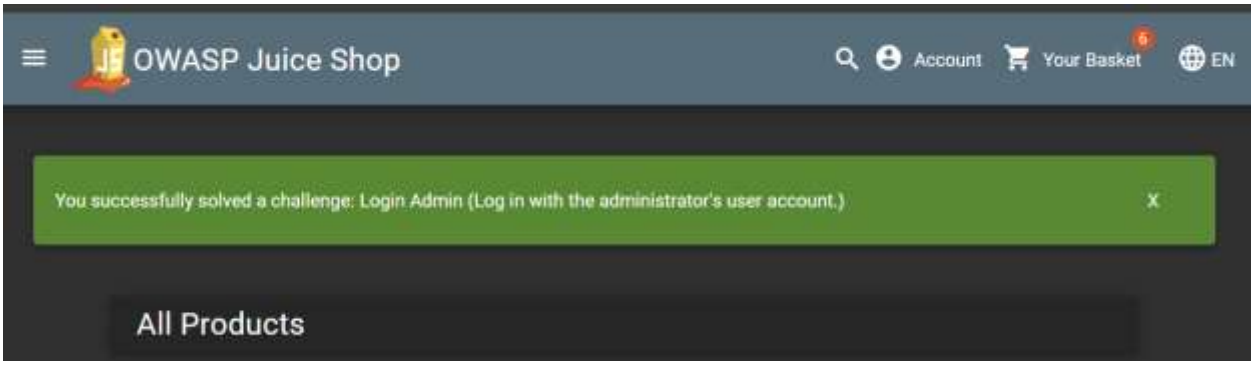
### Target




The first thing that stuck out to me was the account log in. I wanted to try to get into the Administrator's account with a SQL injection.

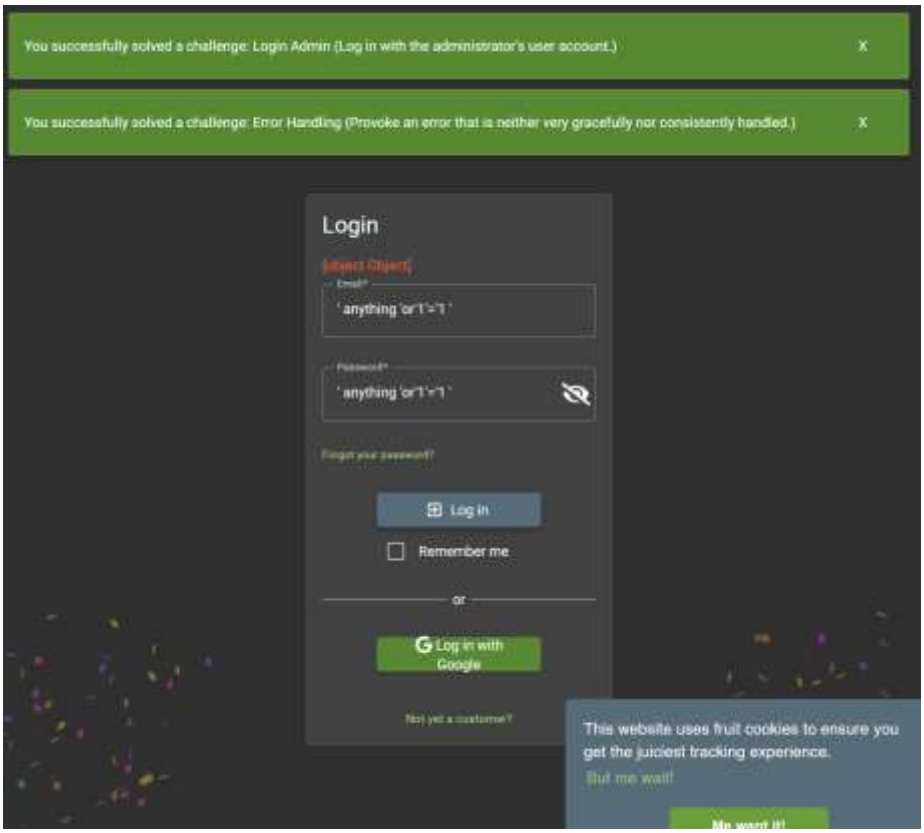
### Exploit


I tried this SQL injection. It is ending the username selection and is asking if 1 is equal to 1. It checks for the correct username or if 1=1.




I was able to successfully log in. However, I was curious about other SQL injections that were available.

I logged back into the administrator using the previous injection.




I tried entering the SQL injection above. This SQL injection looks for “anything” or if 1 is equal to 1 in both the username and password field. While it did not allow me to log in, it did present me with another exploit. I triggered an error that the application could not handle.

## Purple Team Mitigation Report

Sanitization is the most popular way to mitigate against SQL injections.

In /juice-shop/routes/login.ts in the Docker container juice-shop-waf-juice-shop-1, I found some of the insecure code.

The database was being searched with the following line:

```
models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}' AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, { model: UserModel, plain: true }) // vuln-code-snippet vuln-line loginAdminChallenge loginBenderChallenge loginJimChallenge
```

I adjusted it to handle replacements rather than plaintext.

models.sequelize.query(`SELECT * FROM Users WHERE email = :email AND password = :password AND deletedAt IS NULL`, {replacements: { email: email, password: password }, model: UserModel, plain: true }); // vuln-code-snippet vuln-line loginAdminChallenge loginBenderChallenge loginJimChallenge

Unfortunately, the exploit was still successful.

## GitHub Links:

https://github.com/JoelleWa2025/The-Mega-hacking-group-project-of-death

https://github.com/4FTSQRL/OWASP-Juice-Shop-COMP357