Requirements:

-Something that can run a kali Linux vm (laptop, desktop, whatever you want)

-1 x kali Linux VM (other distros might work but I used this one)

-Our GitHub page open! It has all the instructions!
https://github.com/SophieP21/FINAL_EPIC_HACK_PROJECT

How to set up the OWSAP Juice Shop Docker Container!

Step 1: Install Docker

Step 1.1: sudo apt update

Step 1.2: sudo apt install -y ca-certificates curl gnupg

Step 1.3: sudo install -m 0755 -d /etc/apt/keyrings

Step 1.4: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor - o /etc/apt/keyrings/docker.gpg

Step 1.5: sudo chmod a+r /etc/apt/keyrings/docker.gpg

Step 1.6: echo \ "deb [arch=$(dpkg --print-architecture) signed by=/etc/apt/keyrings/docker.gpg] \ https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) stable" | \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Step 1.7: sudo apt update

Step 1.8: sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Step 1.9: sudo docker run hello-world

Step 1.10: Everything should work (yippee!)

Step 2: Set up OWSAP Juice Shop Container (Juice-Shop, n.d.)

Step 2.1: sudo docker pull bkimminich/juice-shop

Step 2.2: sudo docker run --rm -p 127.0.0.1:3000:3000 bkimminich/juice-shop

Step 2.3: Browse to http://localhost:3000

Step 2.4: Incredible work, you have everything set up!

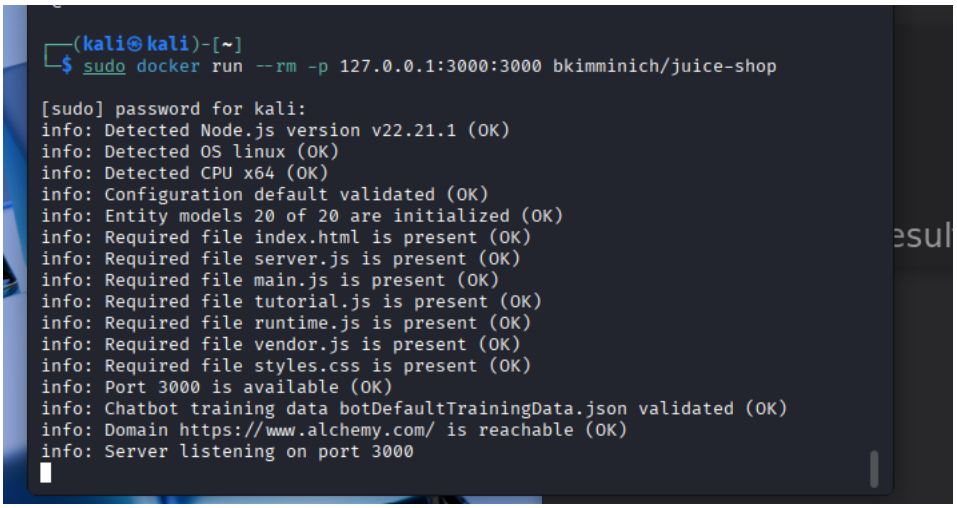
Attack Time!

Here I (your fearless leader) will walk you through the attack steps. For this part of the final, we are doing Cross Site Scripting (aka XSS) because it's almost XMAS (haha get it?)

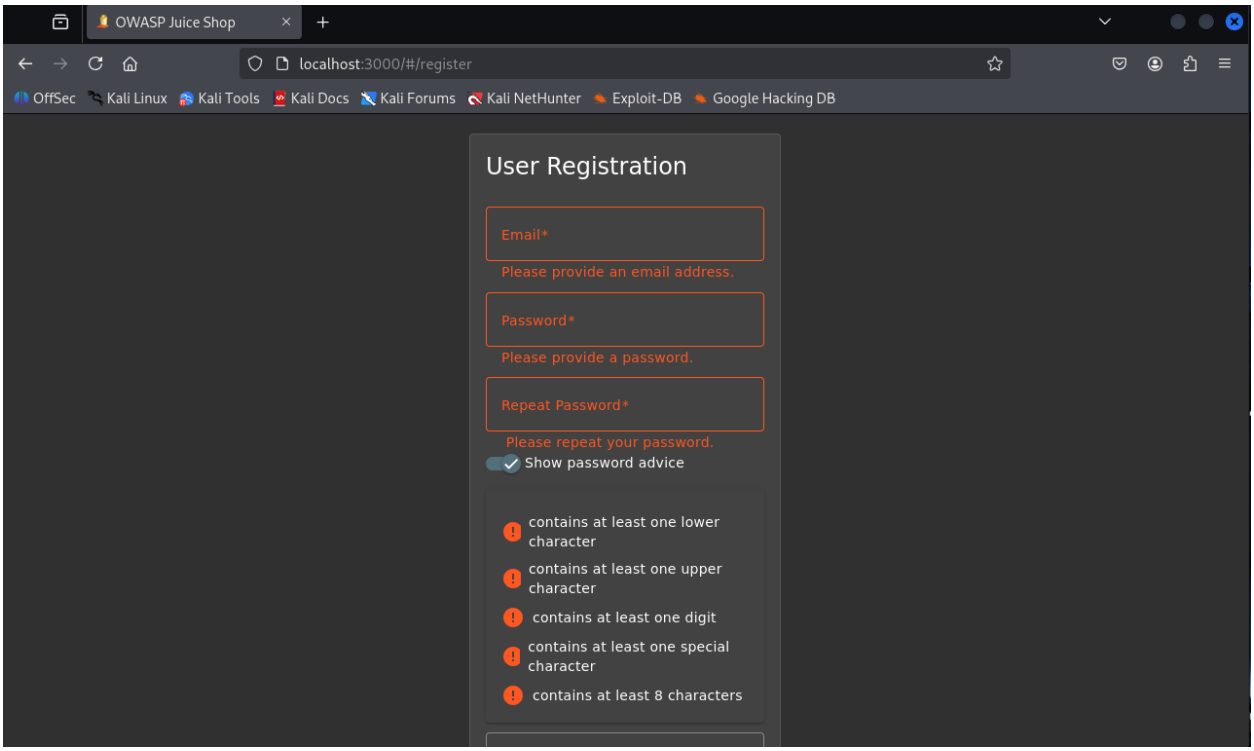
What the heck is Cross Site Scripting?

Well, it is a web security vulnerability that allows attackers to compromise interactable content on a web site such as file uploads, text boxes, etc.

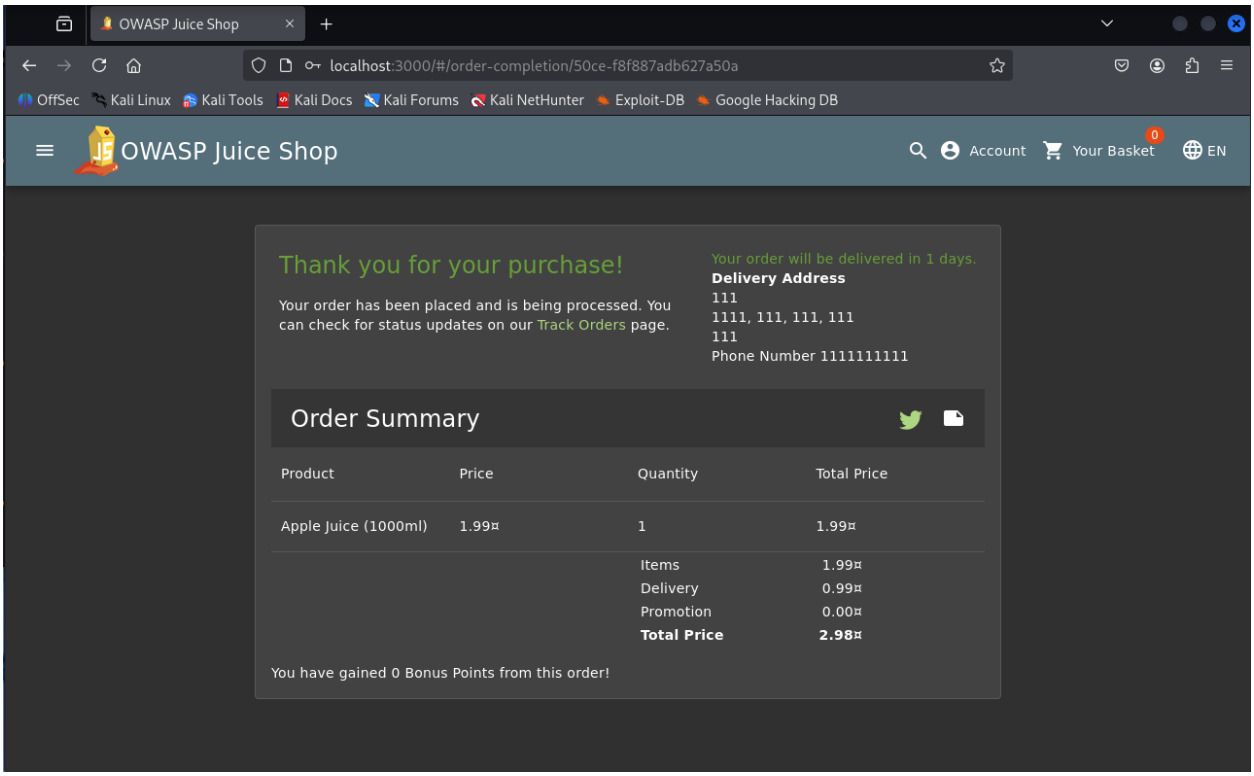
Step 1: Boot up your Docker container and head to http://localhost:3000



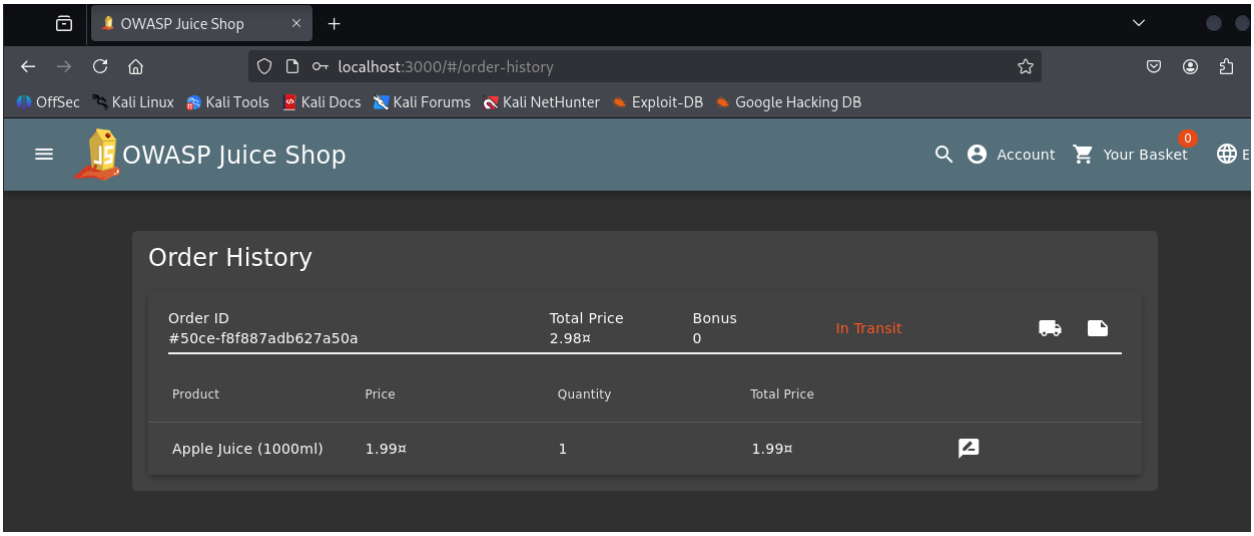
Step 2: Head to the Login In page in the top right corner and then make a new account by clicking the new user button at the bottom.



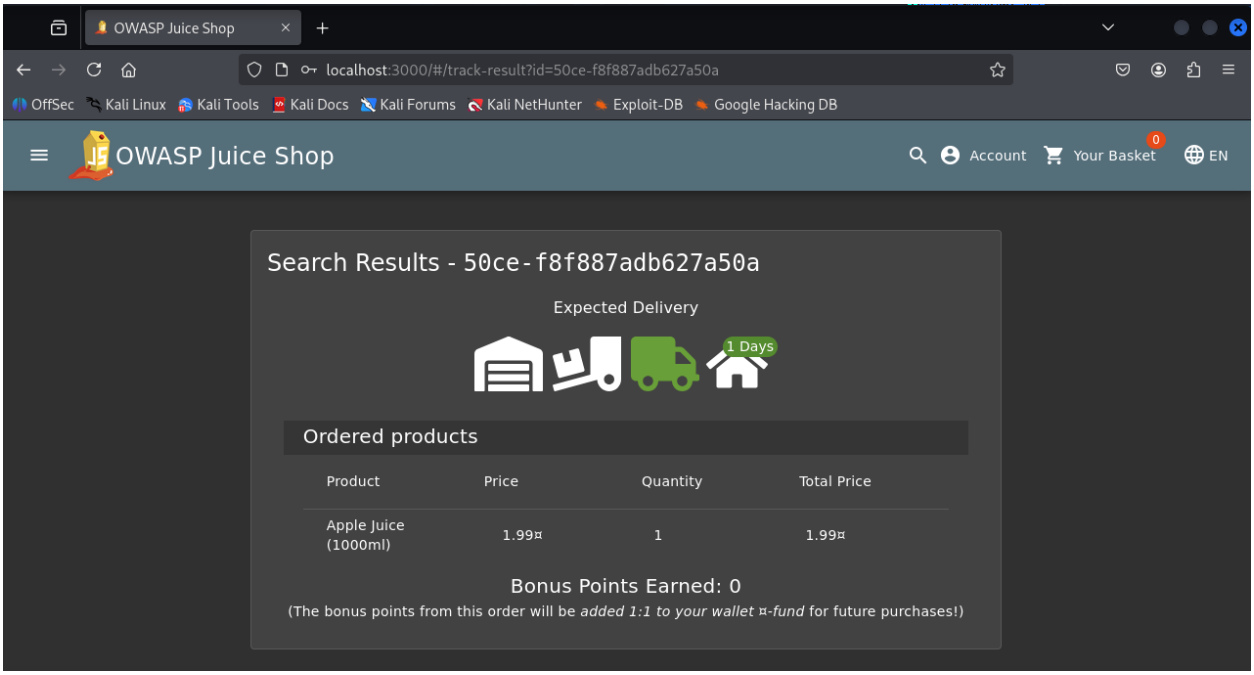
Step 3: Add something to your cart and then go to your cart and check out. Go through all the screens and enter random information until you reach this screen:



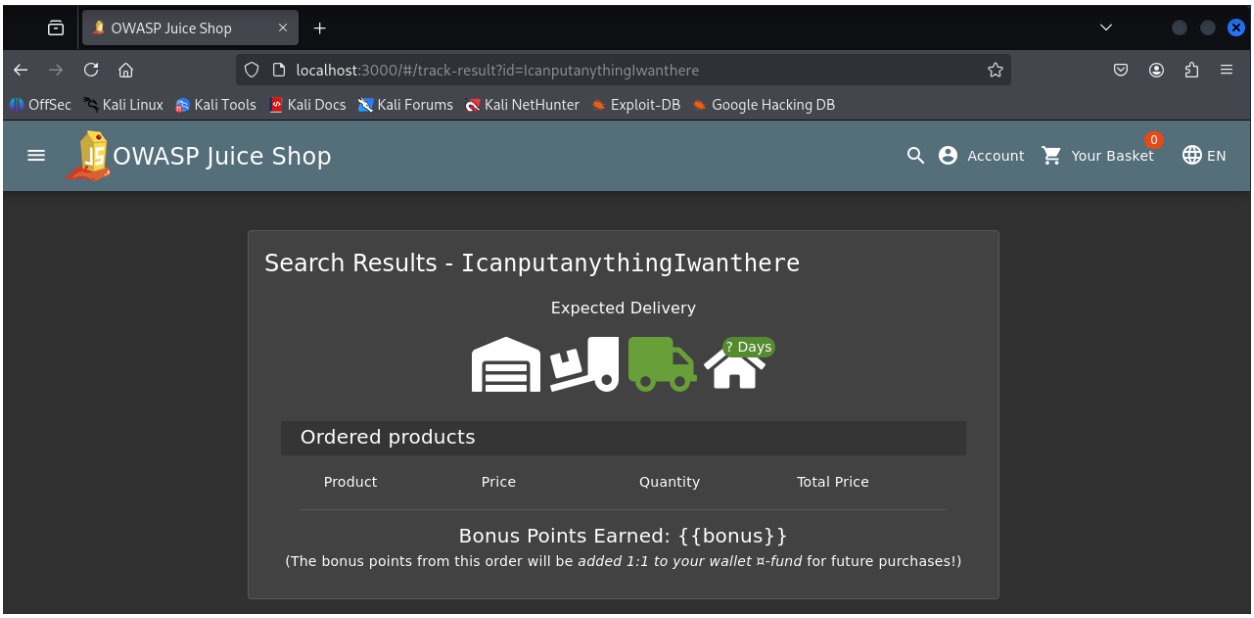
Step 4: After the order confirmed screen go to the top right corner and click on Account -> Orders and Payment -> Order History. You should come to this screen:



Step 5: Click on the little image of the truck in the order information (when you hover over it should show "Track Order") and it should bring you to this screen:



Step 6: In the URL bar replace the current URL with this: http://localhost:3000/#/track-result?id=IcanputanythingIwanthere (F5, n.d.) Then reload the page, it should display 'IcanputanythingIwanthere' in the search results as seen here:



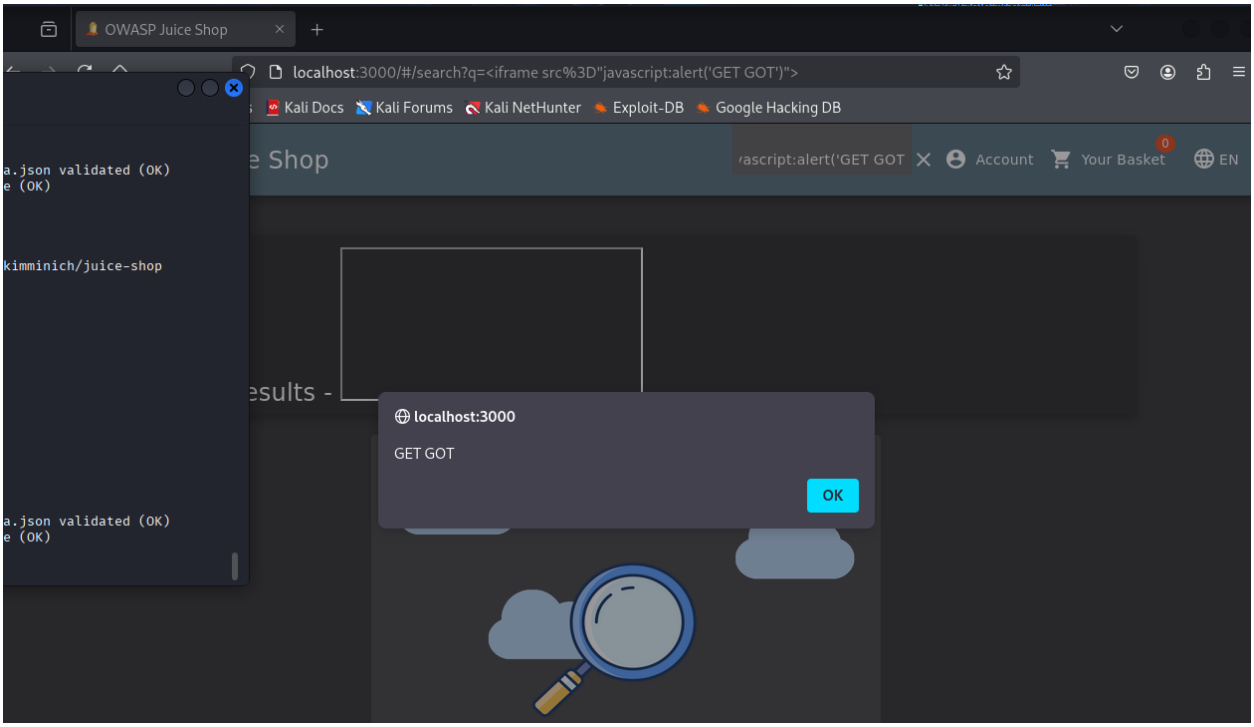
!!!!!BONUS!!!!!

Cross Site Scripting can also be used in another place!

If you go to the search bar up in the top right and type in (or copy paste if we are being realistic) this:

<iframe src="javascript:alert('Get Got!')"> (Automat-IT, 2022)

You should get a popup that says "get got!"



Purple Team

To fix XSS attacks you can do a few things:

Encode data on output

Validate input on arrival

When encoding data on output it prevents the browser from running bad/malicious code, instead of executing anything as a script, the browser displays them safely as text.

Input validation means checking incoming data before the application processes it, this makes sure that the data is safe and properly formatted.

Now that the two ways of defending against this are explained, you might be wondering: “Well, that seems really general, how do I actually fix this?” And your question would be correct, these are categories that all the ways of fixing XSS fall into, so let’s break it down:

 (A little Olympics humor for you)

For the first attack, I changed the text in the URL, and the page showed it. Even though I didn't inject a <script> tag, this shows the page is vulnerable to future XSS. A Content Security Policy would help by stopping the browser from executing any scripts if an attacker tried to turn this into a XSS attack. This would be classified as encoding over validation (PortSwigger, n.d.).

Automat-IT provides a detailed walkthrough of how AWS WAF can be used to prevent XSS attacks, including showing how to configure rules that inspect request bodies, query parameters, cookies, and URI paths. While I do not have access to AWS for testing, this shows a potential defense strategy that could work alongside other defenses such as Content Security Policy. For bonus attack I did, the input is reflected in a URL fragment after the # symbol, which never reaches the server, meaning AWS WAF cannot inspect or block it directly. To mitigate this type of attack, front-end protections such as input validation and output encoding are essential, and converting hash-based routing to path/query-based routing would allow WAF rules to apply. Overall, AWS WAF can serve as a complementary layer, but primary XSS mitigation should rely on secure application code and browser-based safeguards like CSP.

References:

F5. (n.d.). *Module 2 – Discover the OWASP Dashboard* (Lab 2). F5 Web Application Firewall Solutions. https://clouddocs.f5.com/training/community/waf/html/waf2023/module1/lab2.html

PortSwigger. (n.d.). *Preventing cross-site scripting (XSS)*. PortSwigger Web Security Academy. https://portswigger.net/web-security/cross-site-scripting/preventing

Automat-IT. (2022, March 15). *Hack the OWASP Juice Shop application and protect it with AWS WAF — Part 2*. Automat-IT Blog. https://www.automat-it.com/blog/hack-the-owasp-juice-shop-application-and-protect-it-with-aws-waf-part-2/#0e6c

Juice-Shop. (n.d.). *OWASP Juice Shop · GitHub*. GitHub. https://github.com/juice-shop/juice-shop?tab=readme-ov-file#docker-container