

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](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" | \ 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!

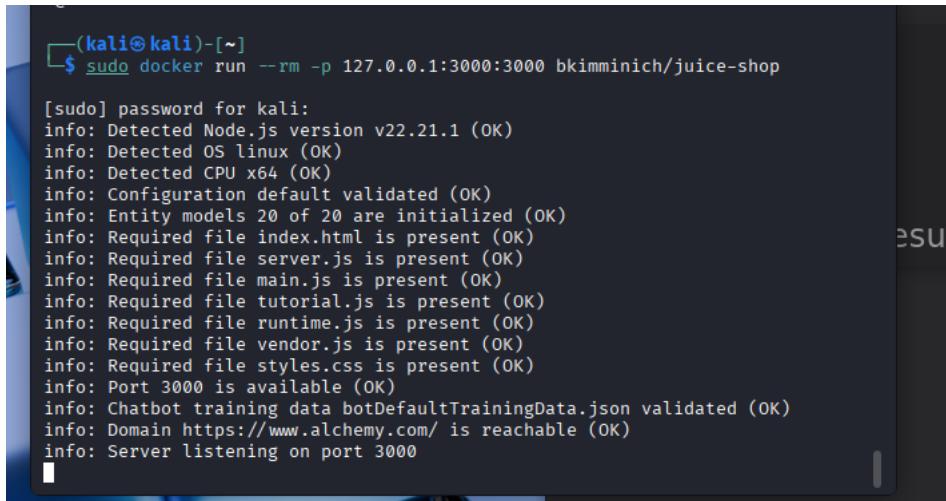
## Attack Time!

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



```
(kali㉿kali)-[~]
$ sudo docker run --rm -p 127.0.0.1:3000:3000 bkimminich/juice-shop

[sudo] password for kali:
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 index.html is present (OK)
info: Required file server.js is present (OK)
info: Required file main.js is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file vendor.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
```

**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.

The screenshot shows a user registration form titled "User Registration". The "Email\*" field has an error message: "Please provide an email address.". The "Password\*" field has an error message: "Please provide a password.". The "Repeat Password\*" field has an error message: "Please repeat your password." Below these fields is a "Show password advice" link. To the right of the form, there is a list of five rules for a strong password:

- ! contains at least one lower character
- ! contains at least one upper character
- ! contains at least one digit
- ! contains at least one special character
- ! contains at least 8 characters

**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:

The screenshot shows an order completion page from the OWASP Juice Shop. At the top, it says "Thank you for your purchase!". It states: "Your order has been placed and is being processed. You can check for status updates on our [Track Orders](#) page." To the right, it says "Your order will be delivered in 1 days." and lists the delivery address: "111 1111, 111, 111, 111" and "Phone Number 1111111111". Below this is an "Order Summary" table:

Product	Price	Quantity	Total Price
Apple Juice (1000ml)	1.99¤	1	1.99¤
		Items	1.99¤
		Delivery	0.99¤
		Promotion	0.00¤
		<b>Total Price</b>	<b>2.98¤</b>

At the bottom, it says "You have gained 0 Bonus Points from this order!"

**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:

The screenshot shows a web browser window for the OWASP Juice Shop. The URL in the address bar is `localhost:3000/#/order-history`. The page title is "Order History". At the top, there is a summary row for an order: Order ID #50ce-f8f887adb627a50a, Total Price 2.98₹, Bonus 0, and status In Transit. Below this is a table showing the details of the single item ordered: Apple Juice (1000ml) at 1.99₹ each, quantity 1, and total price 1.99₹. There is also a small truck icon next to the table.

**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:

The screenshot shows a web browser window for the OWASP Juice Shop. The URL in the address bar is `localhost:3000/#/track-result?id=50ce-f8f887adb627a50a`. The page title is "Search Results - 50ce-f8f887adb627a50a". It displays the "Expected Delivery" status with icons for a building, a delivery truck, and a house, indicating "1 Days". Below this is a section titled "Ordered products" with a table showing the same order details as the previous screen: Apple Juice (1000ml) at 1.99₹ each, quantity 1, and total price 1.99₹. At the bottom, it says "Bonus Points Earned: 0" and provides a note: "(The bonus points from this order will be added 1:1 to your wallet ₹-fund for future purchases!)".

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:

The screenshot shows a web browser window for the OWASP Juice Shop application. The URL in the address bar is `localhost:3000/#/track-result?id=IcanputanythingIwanthere`. The page displays search results for the query "IcanputanythingIwanthere". At the top, there's a section titled "Expected Delivery" with icons for a house, a truck, and a person, followed by a question mark and "Days". Below this is a table header for "Ordered products" with columns: Product, Price, Quantity, and Total Price. A message below the table states "Bonus Points Earned: {{bonus}}" and "(The bonus points from this order will be added 1:1 to your wallet -fund for future purchases!)".

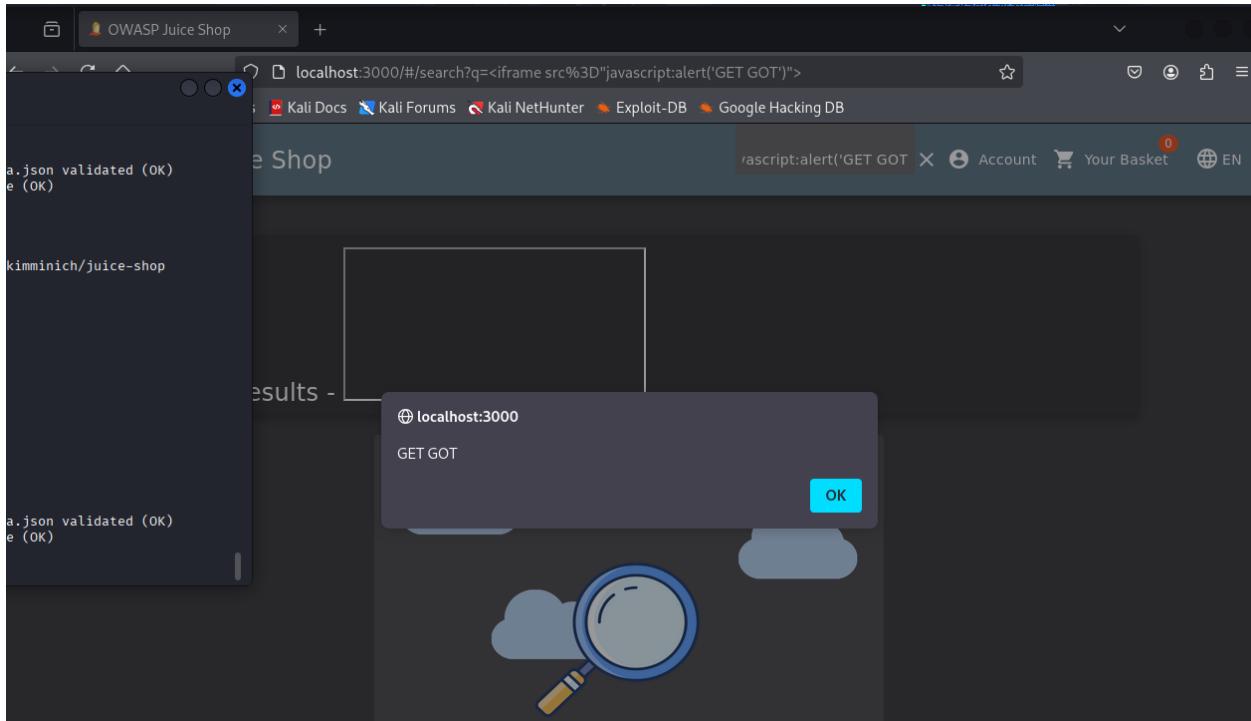
!!!!BONUS!!!!

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>