# WEEK-3

Asad Muhammad Channar

DHC-56

## TASK-3:

**Web Vulnerability Exploitation (SQL Injection/XSS):**

## Required:

- Select a vulnerable web application (OWASP Juice Shop or similar).
- Exploit an SQL Injection or CrossSite Scripting vulnerability.
- Document the attack, its impact, and how it can be mitigated.

## Deliverables:

- A brief report on your attack simulation, with screenshots of the process.

# Web Vulnerability Exploitation (SQL Injection/XSS)

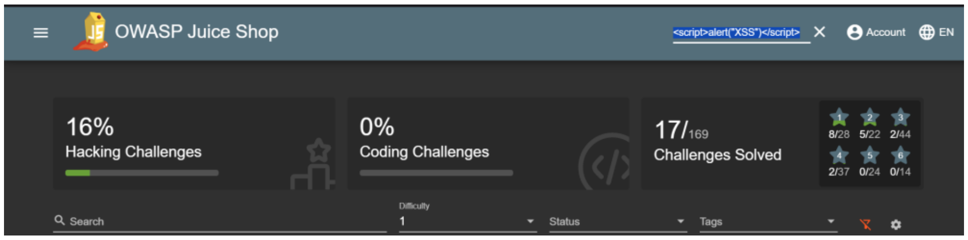
## Attack Simulation: XSS Vulnerability Exploitation in OWASP Juice Shop

### 1. Attack Description

In this simulation, we exploited a Cross-Site Scripting (XSS) vulnerability in the OWASP Juice Shop web application. XSS is a type of vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. In this case, we managed to execute a script that triggers a JavaScript alert displaying the message 'XSS'.

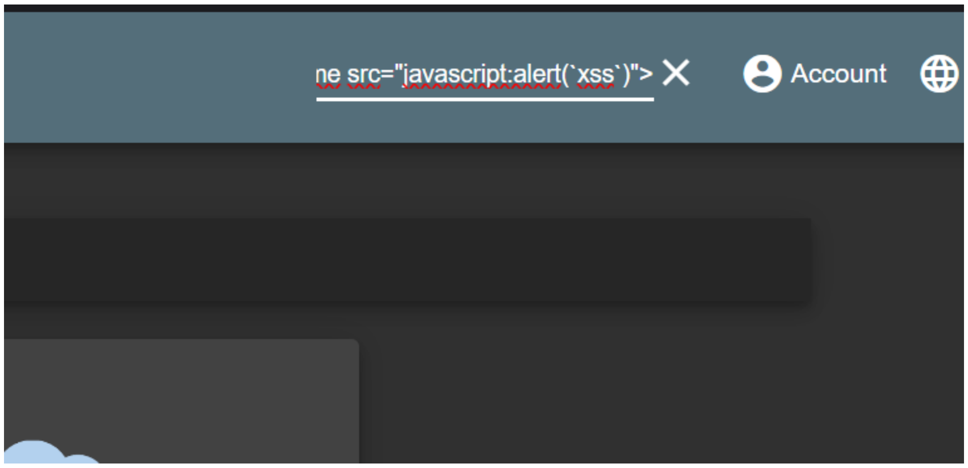
### 2. Screenshots of the Attack

The following screenshots demonstrate the attack:



```
<script>alert("XSS")</script>
```

We use this script in the search bar but nothing happened. Then we process to other similar scripts through which we can achieve our task.

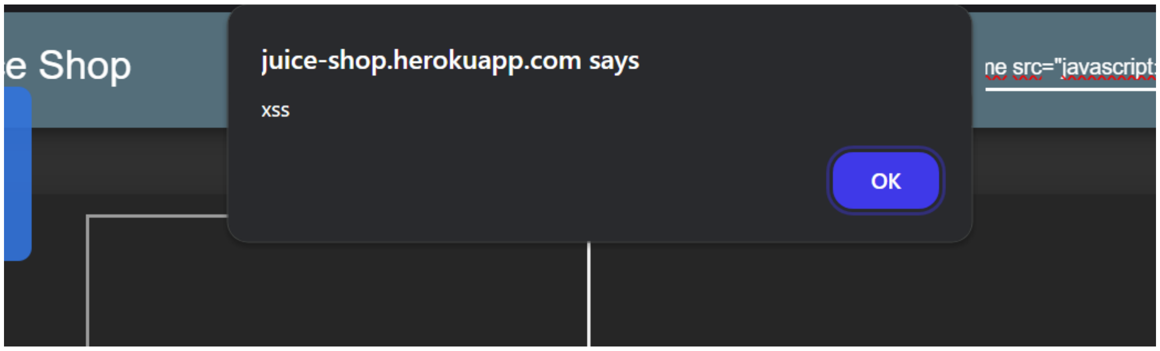


```
ne src="javascript:alert(`xss`)">
```

We try this script but nothing happened again.

```
<iframe src="javascript:alert(`x
```

But in last I use this Script**: <iframe src="javascript:alert(`xss`)">** in the search bar and to my expectations it worked. It executes this script in the browser and generate the alert .



## 3. Impact of the Exploit

The XSS vulnerability allows attackers to execute arbitrary scripts in the context of the user's browser. This can result in session hijacking, defacement of the web page, redirection to malicious sites, or theft of sensitive data such as cookies or login credentials. In our scenario, we demonstrated the potential by triggering a harmless JavaScript alert, but the consequences could be more severe.

## 4. Mitigation Strategies

To mitigate XSS vulnerabilities, the following steps should be implemented:
1. Input validation: Ensure that user input is properly validated and sanitized before it is processed by the server.
2. Output encoding: Encode user input when it is reflected back on the web page, ensuring that it is not executed as code.
3. Content Security Policy (CSP): Implement a strong CSP to limit the execution of scripts to trusted sources only.
4. Use modern web frameworks: Many modern frameworks have built-in protections against XSS, such as automatic encoding of output.