

WORKSHOP 2 OF 3 • FROM THEORY TO PRACTICE

Today, we hack. Together.

Laptops out. We set up a safe lab and find two real vulnerabilities, by hand.

Mohamed Medjahdi, Cybersecurity Engineer

7-77 Digital Skills Programme • Algérie Télécom SkillsCenter, Saïda

Quick refresh from Sunday.

- › A website is a **conversation**: you send a request, it sends a response.
 - › The attacker's one move: **don't use the page, question it.**
 - › The method is always **recon → exploit → report.**
 - › The vuln we'll start with, **XSS**, lives in OWASP A03: Injection.
- Today we stop talking about it and **do it**, on a lab that's built to be hacked.

Use AI today like a teammate, not an oracle.

UNSTICK YOURSELF

Explain errors

Paste an error or a confusing response, ask it what's happening and what to try next.

GENERATE IDEAS

Suggest payloads

Ask "what could I try in this input, and why?" to learn faster.

FINISH THE JOB

Draft the report

Give it your finding, get a clean first draft to edit.

Always verify. AI is confidently wrong often. Use it to move faster, not to think for you.

Here's our plan for today.

first

Set up the lab

Docker, Juice Shop, Burp.

vuln 1

XSS

Explain, then hunt it together.

vuln 2

SQL injection

Explain, then do it together.

finish

Report it

Write up what we found.

Step 1 • Docker, an app in a sealed box.

Docker lets us run our practice website the exact same way on every laptop, no messy install. Get **Docker Desktop** (Mac & Windows), open it once, and check it works:

```
terminal
```

```
$ docker --version
```

```
Docker version 27.x (any version is fine)
```

Download: docker.com/products/docker-desktop

Step 2 • Run our target, OWASP Juice Shop.

One command downloads and runs a website that's **made to be hacked**.
Completely legal, it's ours.

terminal

```
$ docker run --rm -p 3000:3000 bkimminich/juice-shop  
# then open http://localhost:3000
```

When the shop loads in your browser, you're ready. This is our playground for the rest of the session.

Step 3 · Burp Suite, see and change the conversation.

Burp sits between your browser and the site, so you can read every request and change it before it's sent. Open **Burp Community** (free) and use its built-in browser.

- › **Proxy:** catch a request.
- › **Repeater:** resend it and tweak it.
- › **Target map:** every page the site uses.

We attack **only** our own lab today.

Juice Shop runs on your machine, it's yours, so it's legal. A real website needs written permission. Unauthorized access is a crime in Algeria (**Penal Code 394 bis**). Same skill, the difference is permission.

Vulnerability 1 · why XSS actually happens

Remember Sunday: the site puts your input into the page. Here's the exact line that causes it.

server builds the page

the bug is here

```
html = "<p>Results for: " + userInput + "</p>"
# userInput is glued straight in, as-is

# normal:  userInput = "shoes"    → fine
# attack:  userInput = "<script>...</script>"
                → the browser runs it
```

THE ROOT CAUSE

Your input was mixed into the page's **code**. The browser can't tell your script from the site's own.

THE FIX (blue team)

Clean the output: turn **<** into harmless text so it shows as letters, never runs.

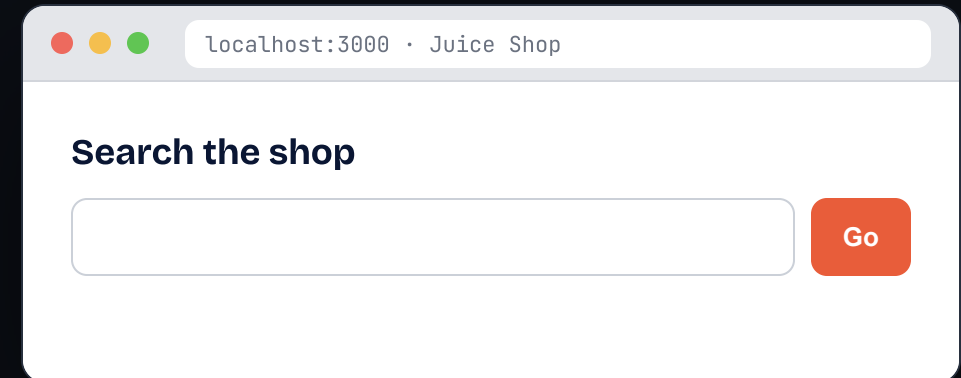
XSS · RECON → EXPLOIT → IMPACT

The search box reflects what we type. So we type code.

- › **Recon:** find an input that echoes back, the search bar.
- › **Exploit:** send a script instead of a word.
- › **Impact:** `document.cookie` is the login session.

the payload

```
<iframe src="javascript:alert(document.cookie)">
```



Vulnerability 2 · why SQL injection happens

Same bug as XSS, different place. To log you in, the site builds a database question by gluing your input into text.

login builds the query

the bug is here

```
query = "...WHERE email='" + input + "' AND pass='"...
```

```
# you type in the email field:
```

```
input = ' OR 1=1--
```

```
# the question becomes:
```

```
WHERE email='' OR 1=1 -- AND pass='...'
```

```
→ always true, password ignored
```

THE ROOT CAUSE

Your input was mixed into the **command**. The quote and -- let you rewrite the question itself.

THE FIX (blue team)

Use

parameterized queries: input is sent as data, never as part of the command.

SQLI • LOGIN BYPASS

We rewrite the question so it's always true.

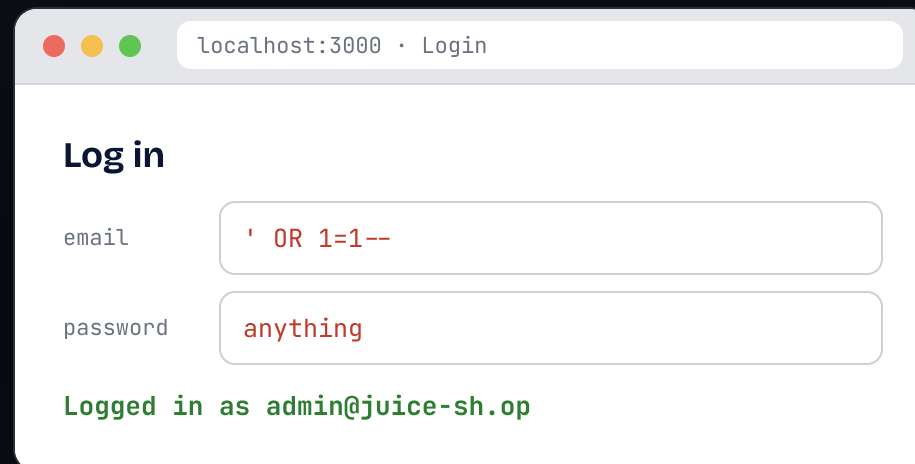
we type this in the email field

' OR 1=1--

the question becomes

WHERE email='' OR 1=1-- ...

always true • password check commented out



A screenshot of a web browser window with the title 'localhost:3000 • Login'. The page has a 'Log in' heading. There are two input fields: 'email' and 'password'. The 'email' field contains the text '' OR 1=1--' and the 'password' field contains 'anything'. Below the fields, a green message reads 'Logged in as admin@juice-sh.op'.

Log in

email ' OR 1=1--

password anything

Logged in as admin@juice-sh.op

We found two bugs. Now we write them up.

A good report says: what you found, how to repeat it, why it matters, how to fix it.
Give your notes to AI for a first draft, then **check every line yourself**.

This is exactly what Thursday's challenge will ask you to do.

XSS in search

HIGH

what	Search runs injected code
steps	1. open search 2. enter payload 3. it runs
impact	Attacker JS runs in the victim's browser → session theft
fix	Clean and encode input before showing it

Keep going tonight, and get ready for Thursday.

PRACTICE TONIGHT (FREE)

PortSwigger Academy

Do the **XSS** and **SQL injection** labs. Same vulns, guided, in your browser. Best practice there is.

THURSDAY

The challenge

Teams compete to find bugs in Juice Shop and write a report. Bring this laptop, keep Juice Shop ready.

Everything on Thursday uses exactly what you did today. Resources: medjahdi.dev/src



tonight's practice
pack
/src/practice-2

THAT'S REAL OFFENSIVE SECURITY

You just found two real bugs, **by hand.**

Questions? Ask me anything.

ALL THREE WORKSHOPS, ONE PLACE

Scan it. It is all yours.

Every slide, the setup guide, the payloads, the practice pack, and the full roadmap. Free, forever.

medjahdi.dev/src

