OWASP Juice Shop — Kali Lab Guide

Practical, step-by-step exercises for SQLi, XSS, IDOR, CSRF and SSRF (source/npm install)

Introduction & Safety

This lab guide assumes you installed Juice Shop from source (GitHub + npm) on Kali Linux. Only run these exercises against your local instance. Snapshot your VM before heavy testing and reset the app using the provided reset script when finished.

Start / Stop (Quick Commands)

Install deps	cd ~/juice-shop && npm ci
Start (bind localhost)	HOST=127.0.0.1 PORT=3000 npm start
Start dev	npm run start:dev
Background (quick)	nohup HOST=127.0.0.1 PORT=3000 npm start &>/tmp/juice.log &
Stop by PID	pkill -f 'node.*juice-shop' || true

Database & Reset

Find DB file (example):

```
find ~/juice-shop -type f -iname "*.db" -o -iname "*owasp*.db"
```

Reset (remove DB + uploads) and restart:

```
pkill -f 'node.*juice-shop' || true rm -f ~/juice-shop/data/owasp-juice-shop.db rm
-rf ~/juice-shop/data/uploads HOST=127.0.0.1 PORT=3000 npm start
```

Hands-on Exercises

Each exercise includes: Goal, Steps, Commands/Payloads, How to confirm, Mitigation notes.

Exercise 1 — SQL Injection (Authentication bypass & dump)

Goal: Find injectable parameter, bypass authentication or enumerate user table and extract credentials. Steps: 1) Start Juice Shop bound to localhost and open browser (http://127.0.0.1:3000). 2) Use DevTools / Burp to identify API endpoints (login, products, search). 3) Intercept the login request, send to Repeater. Try payload: username: ' OR '1'='1 4) If manual injection suspected, save request to burp_req.txt and run sqlmap: sqlmap -r burp_req.txt --batch --level=2 --risk=1 --threads=1 --dbs 5) If DBs found, enumerate tables and dump users table. Example sqlmap commands: sqlmap -r burp_req.txt --dbs sqlmap -r burp_req.txt -D juice_db --tables sqlmap -r burp_req.txt -D juice_db -T Users --dump --batch Confirm: sqlmap returns DB names / Users table with emails/password hashes. Mitigation: Use parameterized queries / ORM binding; avoid string concatenation for SQL queries.

Exercise 2 — Stored Cross-Site Scripting (Product review)

Goal: Inject a stored XSS payload into a review/comment field and trigger execution when the review is viewed. Steps: 1) Find product review or feedback form; intercept the POST in Burp. 2) Inject payload into comment: <script>alert('xss')</script> 3) Submit and navigate to the page that lists reviews. Inspect DOM / page for alert. Manual test payloads (escaped for PDF): <script>alert(1)</script> "><img src=x onerror=alert(1)> Confirm: Browser shows alert popup or injected JS runs; review content contains the payload. Mitigation: Contextual output encoding, sanitize inputs, CSP.

Exercise 3 — IDOR / Broken Access Control (Order enumeration)

Goal: Access another user's order or resource by modifying an ID in API requests. Steps: 1) Login as a normal user or create an account. 2) Find an endpoint that returns your order, e.g., /api/Orders/ or /api/Orders?user=... 3) Intercept a request that fetches your order and increment/change the ID in the path or parameter. 4) Send modified request and observe response. Automate enumeration: for id in {1..200}; do curl -s -b "session=" "http://127.0.0.1:3000/api/Orders/$id" -o /dev/null -w "%{http_code} %{url_effective}\n"; done Confirm: You receive 200 and data for other user's order. Mitigation: Implement server-side object-level authorization checks; do not rely on obscurity.

Exercise 4 — CSRF (State change via crafted form)

Goal: Demonstrate CSRF by creating a malicious HTML page that performs a state-changing action (e.g., add address). Steps: 1) Identify state-changing POST endpoint (e.g., /api/User/addresses). 2) Create an HTML file on attacker machine: document.getElementById('f').submit() 3) While logged-in user visits the attacker's page, the form auto-submits using user's cookies. Confirm: The address appears in the user's profile. Mitigation: Use anti-CSRF tokens, check Origin/Referer headers, SameSite cookies.

Exercise 5 — SSRF (Fetch internal resource)

Goal: Use an input that triggers server-side fetch to access internal resources (e.g., localhost, metadata). Steps: 1) Identify a URL-fetching endpoint (image preview, URL shortener). 2) Provide a URL targeting internal IP: http://127.0.0.1:3000 or metadata service http://169.254.169.254/latest/meta-data/ (cloud-only). 3) Observe server behavior or exfiltrate via response. Use Burp Collaborator or requestbin to detect outbound call. Example curl: curl -X POST "http://127.0.0.1:3000/api/preview" -d 'url=http://127.0.0.1:3000' Confirm: Server attempts to fetch internal URL or returns error revealing internal access. Mitigation: Whitelist allowed domains, block private IP ranges, validate and parse URLs safely.

Tools & Quick Commands

Burp Suite	Proxy, Repeater, Intruder. Capture requests and save to file for sqlmap.
sqlmap	Automated SQLi detection and extraction. Example: sqlmap -r burp_req.txt --dbs
curl	Quick manual requests. Example: curl -v 'http://127.0.0.1:3000/api/Products'
Browser DevTools	Inspect XHR endpoints and DOM.
Docker	Reset by removing container or DB files if using mounted volume.

Reset Script & Systemd Unit (examples)

Included in /mnt/data: juice_reset.sh and juice_shop.service (edit paths/user before use).

Appendix: Tips & OPSEC

• Always run locally and snapshot before heavy testing.

• Start with manual checks before automated tools.

• Document all commands and outputs for reporting.

• Respect legal and ethical boundaries.