Authentication, Sessions & Web Security

Web Technologies — Lecture 11

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Two Words That Sound Alike

- **Authentication (AuthN)** — *Who are you?*
  Verifies identity (username/password, fingerprint, magic link)
- **Authorisation (AuthZ)** — *What may you do?*
  Decides if the (already-authenticated) user can perform an action

Most security bugs come from confusing the two.

NEVER Store Plain Passwords

Rules

- Never store plain text
- Never use MD5 or SHA-1 (too fast = easy to crack)
- Use a **slow, salted** hash: **bcrypt**, **argon2**, **scrypt**
- Each password gets its own random salt
- Library does this for you — never write your own

```
$2b$12$KIXuQ3pK/...   <-- bcrypt hash, includes salt and cost
```

Session Cookies (Server State)

1. User submits username/password
2. Server verifies, creates a session row in the DB
3. Server returns `Set-Cookie: SESSIONID=abc123`
4. Browser sends cookie on every request
5. Server looks up session in DB, finds the user

Pros: easy to revoke (delete the row). **Cons:** state on the server, harder to scale across many servers.

JWT — JSON Web Tokens (Stateless)

```
header.payload.signature
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI0MiIsImV4cCI6...}.HMAC-signature
```

- Server signs a token containing the user's claims
- Client stores it (typically in memory or HttpOnly cookie)
- Sent on every request: `Authorization: Bearer <token>`
- Server verifies signature — **no DB lookup**

Pros: stateless, scales easily. **Cons:** hard to revoke before expiry.

Cookie Best Practices

- `HttpOnly` — JS cannot read it (mitigates XSS)
- `Secure` — only sent over HTTPS
- `SameSite=Lax` (or `Strict`) — mitigates CSRF
- Short `Max-Age`; refresh tokens for long sessions

“Sign in with Google”

1. User clicks “Sign in with Google” on your app
2. Browser is redirected to Google with your client ID
3. User authenticates with Google, approves your app
4. Google redirects back to your app with a **code**
5. Your server exchanges the code for an **access token** + **ID token**
6. Your server creates a session for the user

OAuth 2.0 = authorisation framework. **OpenID Connect** = identity layer on top.

The Big Hitters

1	Broken Access Control
2	Cryptographic Failures
3	Injection (SQL, command, LDAP...)
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable & Outdated Components
7	Identification & Authentication Failures
8	Software & Data Integrity Failures
9	Security Logging & Monitoring Failures
10	Server-Side Request Forgery (SSRF)

XSS — Cross-Site Scripting

Attacker injects `<script>...</script>` that runs in another user's browser. **Defences:**

- **Escape** all user-supplied data when rendering HTML
- Set `Content-Security-Policy` headers
- Use frameworks (React, Vue) that escape by default
- Avoid `innerHTML` with untrusted data
- Mark cookies `HttpOnly`

CSRF — Cross-Site Request Forgery

Malicious site makes the user's browser send a request to your site using their cookies.
Defences:

- `SameSite=Lax` (or `Strict`) cookies
- Anti-CSRF tokens on state-changing forms
- Require custom headers (e.g., `X-Requested-With`) for state changes
- Don't accept GETs that change state

SQL Injection — Quick Reminder

- Attacker supplies SQL fragments through input fields
- **Defence:** parameterised queries / ORMs (Lecture 7)
- Never concatenate strings into SQL

Other Critical Headers

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy:    default-src 'self'
X-Content-Type-Options:     nosniff
X-Frame-Options:            DENY
Referrer-Policy:            strict-origin-when-cross-origin
```

Helmet.js (Node), Spring Security defaults, and Django middleware all set most of these for you.

HTTPS Everywhere

- Use **TLS** (HTTPS) for *all* traffic, including dev
- Free certificates from **Let's Encrypt** (auto-renewed)
- Modern browsers warn or block plain HTTP
- Without HTTPS: passwords, cookies, and tokens are public on the network

Before You Ship

- Passwords hashed with bcrypt/argon2
- All forms parameterised; no string-concat SQL
- HTTPS enforced; HSTS header set
- Security headers via Helmet / Spring Security / Django defaults
- Cookies: `HttpOnly` + `Secure` + `SameSite`
- Dependencies up to date (`npm audit`, OWASP Dependency-Check)
- Rate limiting on login & signup
- Logging & alerting on failed logins, 5xx spikes
- Secrets in environment variables, NOT in the repo

Summary

- AuthN $\neq$ AuthZ
- Hash passwords with bcrypt/argon2 + per-user salt
- Sessions = stateful; JWT = stateless — both are valid
- OAuth/OIDC for "Sign in with X"
- Defend against the OWASP Top 10 with framework defaults + good habits

Next: Deployment & DevOps — Docker, Compose, GitHub Actions.

Questions?