

Lab 13. Advanced session/token security and app hardening

Goal: Configure logout, refresh tokens, security headers, logging, and (optionally) TLS, rate limiting.

In this lab, you will improve the security of a web application by protecting user sessions, securing authentication tokens, adding security-related HTTP headers, and implementing basic defensive techniques used in real-world systems.

Tasks:

1. Session and token Management

Ensure that users cannot keep using old or stolen authentication credentials.

1.1. Implement session invalidation on logout.

When a user logs out, their server-side session must be destroyed. This prevents someone from reusing the same session ID after logout.

Example concept: If Alice logs out on a shared computer, the next person should not be able to access Alice's account by refreshing the page.

1.2. Implement JWT refresh token rotation (REST).

- Access tokens should be short-lived.
- Refresh tokens are used to request new access tokens without logging in again.
- When a refresh token is used: Issue a new refresh token; Invalidate the old one (this is called rotation)

Why this matters: If a refresh token is stolen, rotation limits how long an attacker can reuse it.

1.3. Add expiration handling.

- Tokens and sessions must expire automatically.
- Expired tokens should be rejected by the server.
- Key idea: authentication should never last forever.

2. Security headers (both tracks)

HTTP security headers instruct the browser to enforce extra safety rules.

You must configure the following headers:

2.1. X-Content-Type-Options: nosniff

- Prevents the browser from guessing file types.
- Protects against certain script injection attacks.

2.2. X-Frame-Options: DENY (or SAMEORIGIN)

- Prevents your site from being embedded in an <iframe>.
- Protects against clickjacking attacks.

2.3. Content-Security-Policy (simple form appropriate for your project).

- Controls where scripts, styles, and images can load from.
- Start with a simple policy that works for your app.

2.4. Referrer-Policy: no-referrer (or strict-origin-when-cross-origin)

Controls how much URL information is sent when navigating to other sites.

2.5. Secure cookie attributes

Ensure cookies used for authentication have:

- HttpOnly (not accessible to JavaScript)
- Secure (only sent over HTTPS)
- SameSite (prevents CSRF attacks)

3. Secure logging with SLF4J

Logging helps developers detect attacks and debug issues - but logs must never leak sensitive data.

3.1. Log the following events:

- Failed login attempts
- Suspicious requests (e.g., repeated invalid input)
- Unauthorized access attempts (403 / 401 errors)

3.2. What NOT to log:

- Passwords
- JWTs or refresh tokens
- Personally Identifiable Information (PII)

Rule of thumb: Logs should help defenders, not attackers.

Recommended dependencies:

implementation 'org.slf4j:slf4j-api:xxx'

implementation 'ch.qos.logback:logback-classic:xxx'

4. Rate limiting (optional but recommended)

Prevent attackers from making too many requests in a short time.

4.1. Implement very simple rate limiter (Redis or in-memory).

Real-world example: if someone tries 1000 passwords per minute, your app should stop responding to them.

5. Transport Security (optional but recommended)

Protect data while it travels over the network.

5.1. Configure HTTPS with real certificate (Let's Encrypt) OR self signed certificate.

5.2. Force redirect from HTTP --> HTTPS: All insecure HTTP requests should automatically redirect to HTTPS.

5.3. Add Strict-Transport-Security header: instructs browsers to always use HTTPS for your site.

Checklist

Session and token security

- Session invalidation implemented.
- JWT expiration implemented.
- Refresh token implemented (REST).
- Refresh token rotation (invalidate old refresh token).
- Tokens secured (HTTP-only cookie or secure storage).

Security headers

Configured through SecurityFilterChain or custom filter:

- X-Content-Type-Options
- X-Frame-Options
- Content-Security-Policy (basic)
- Strict-Transport-Security (if HTTPS is implemented)

Logging (SLF4J)

- Log failed logins (without sensitive data).
- Log unauthorized access attempts.
- Log suspicious query patterns.
- No PII or passwords in logs.

Optional: Rate limiting

Implemented brute-force protection (in-memory or Redis).

Optional: Transport Security

- Enforced HTTPS.
- Redirect HTTP → HTTPS.
- Explained HSTS in a short writeup.

Sources:

X-Content-Type-Options header, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Content-Type-Options>

X-Frame-Options header, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options>

Content Security Policy (CSP), <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP>

Content security policy, <https://developers.google.com/web/fundamentals/security/csp>

Google CSP Evaluator, <https://csp-evaluator.withgoogle.com>

Referrer-Policy, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Referrer-Policy>

Spring Boot Logging, <https://docs.spring.io/spring-boot/reference/features/logging.html>

Securing Spring Boot Applications With SSL,
<https://spring.io/blog/2023/06/07/securing-spring-boot-applications-with-ssl>

HTTPS using Self-Signed Certificate in Spring Boot, <https://www.baeldung.com/spring-boot-https-self-signed-certificate>