ADR-004: JWT HTTPOnly Cookies

ADR-004: JWT Storage in httpOnly Cookies

Status: Accepted **Date:** 2026-02-21 **Deciders:** John (AI Director) **Category:** Security

---

Context

Drop is a financial application handling payment initiation, bank account data, and personal information. Secure token storage is critical -- token theft enables full account takeover including payment initiation from the victim's bank account.

The two primary options for JWT storage in browser-based SPAs are:

Option	XSS Risk	CSRF Risk	Implementation Complexity
`localStorage`	**HIGH** -- any XSS payload can read tokens	None	Low
`httpOnly cookie`	**None** -- JavaScript cannot access	Medium -- requires CSRF protection	Medium

Given that Drop processes financial data and operates under PSD2, XSS-based token theft would be catastrophic -- an attacker could initiate payments from a user's bank account. CSRF is a more constrained attack vector with well-understood mitigations.

The mobile app (Expo SDK 54) uses Bearer tokens stored in `AsyncStorage` since cookies are not practical for native apps, but the attack surface is fundamentally different (no XSS in native context).

Decision

Store JWTs in httpOnly cookies for the web application. Use Bearer tokens for the mobile API.

Web cookie configuration (`auth.ts:48-54`):

Property	Value	Rationale
`httpOnly`	`true`	Prevents JavaScript access, eliminates XSS token theft
`secure`	`true` (production)	HTTPS-only transport
`sameSite`	`"Lax"`	CSRF defense (allows BankID redirect back)
`maxAge`	604,800 (7d)	Session lifetime
`path`	`"/"`	Full application scope

> **Implementation note:** The actual implementation uses `maxAge=604800` (7d) and `SameSite=Lax` (changed from the originally specified strict/24h to support BankID OIDC redirect flows).

CSRF protection layers:

1. `sameSite: "Lax"` -- browser refuses to send cookie on cross-origin POST requests
2. Origin header validation against allowed origins whitelist (`app.ts:23-30` CORS middleware)
3. CSRF token generation available (`generateCsrfToken()`) for additional protection

```
graph TB
    subgraph localStorage["localStorage (Rejected)"]
        xss["XSS Attack"] -->|"document.cookie<br/>or localStorage.getItem()"| steal["Token
Stolen"]
        steal --> takeover["Account Takeover<br/>+ Payment Initiation"]
    end

    subgraph httpOnly["httpOnly Cookie (Adopted)"]
        xss2["XSS Attack"] -->|"Cannot access<br/>httpOnly cookie"| blocked["BLOCKED"]
        csrf["CSRF Attack"] -->|"Cross-origin request"| samesite["sameSite: strict<br/>BLOCKED
by browser"]
    end

    classDef danger fill:#FFCDD2,stroke:#C62828
    classDef safe fill:#C8E6C9,stroke:#2E7D32
```

```
    class xss,steal,takeover danger
    class blocked,samesite safe
```

Consequences

Positive

- XSS cannot steal authentication tokens (critical for fintech)
- `sameSite: strict` provides strong CSRF protection with minimal implementation overhead
- React's built-in output escaping + CSP headers provide defense-in-depth
- Aligns with OWASP recommendations for secure session management
- Session revocation via `sessions` table allows server-side token invalidation

Negative

- Slightly more complex CSRF handling compared to Bearer tokens
- Cookie-based auth requires different handling for server-side requests (SSR)
- Cannot share tokens across subdomains without `sameSite` adjustment
- Mobile app requires separate Bearer token flow (dual auth pattern)

Risks

- **CSP bypass:** If CSP includes `unsafe-inline` or `unsafe-eval`, XSS risk increases even with httpOnly cookies (attacker could make API calls from victim's browser). Mitigation: tighten CSP with nonce-based script loading for production.

References

- Security Architecture -- Full security controls documentation
- Authentication System -- Auth flow implementation details
- Middleware Documentation -- CSRF and auth middleware
- ADR-007: BankID OIDC Auth -- Authentication provider decision
- OWASP Session Management Cheat Sheet

Revision #5
Created 2026-02-21 05:58:58 UTC by John
Updated 2026-05-23 10:56:37 UTC by John