ANGULAR AND THE OWASP TOP 10

The OWASP top 10 is one of the most influential security documents of all time. But how do these top 10 vulnerabilities resonate in a frontend JavaScript application?

This cheat sheet offers practical advice on handling the most relevant OWASP top 10 vulnerabilities in Angular applications.

DISCLAIMER This is an opinionated interpretation of the OWASP top 10 (2017), applied to frontend Angular applications. Many backend-related issues apply to the API-side of an Angular application (e.g., SQL injection), but are out of scope for this cheat sheet. Hence, they are omitted.

1 USING *DEPENDENCIES* WITH KNOWN VULNERABILITIES

OWASP #9

- [ ] Plan for a periodical release schedule
- ⚙ Use `npm audit` to scan for known vulnerabilities
- 🍃 Setup automated dependency checking to receive alerts
  Github offers automatic dependency checking as a free service
- 🍃 Integrate dependency checking into your build pipeline

2 BROKEN AUTHENTICATION

OWASP #2

From an Angular perspective, the most important aspect of broken authentication is maintaining state after authentication. Many alternatives exist, each with their specific security considerations.

- ⚙ Decide if a stateless backend is a requirement
  Server-side state is more secure, and works well in most cases

SERVER-SIDE SESSION STATE

- [ ] Use long and random session identifiers with high entropy
  OWASP has a great cheat sheet offering practical advice [1]

CLIENT-SIDE SESSION STATE

- [ ] Use signatures to protect the integrity of the session state
- ⚙ Adopt the proper signature scheme for your deployment
  HMAC-based signatures only work within a single application
  Public/private key signatures work well in distributed scenarios
- [ ] Verify the integrity of inbound state data on the backend
  Explicitly avoid the use of "decode-only" functions in libraries
- [ ] Setup key management / key rotation for your signing keys
- 🍃 Ensure you can handle session expiration and revocation

COOKIE-BASED SESSION STATE TRANSPORT

- [ ] Enable the proper cookie security properties
  *Set the **HttpOnly** and **Secure** cookie attributes*
  *Add the **__Secure** or **__Host-** prefix on the cookie name*
- [ ] Protect the backend against Cross-Site Request Forgery
  *Same-origin APIs should use a **double submit cookie***
  *Cross-Origin APIs should force the use of CORS preflights by only accepting a non-form-based content type (e.g. **application/json**)*

AUTHORIZATION HEADER-BASED SESSION STATE TRANSPORT

- [ ] Only send the authorization header to pre-approved hosts
  *Many custom interceptors send the header to **every** host*

[1] https://bit.ly/2U8kJWc

3 CROSS-SITE SCRIPTING

OWASP #7

PREVENTING HTML/SCRIPT INJECTION IN ANGULAR

- [ ] Use interpolation with `{{}}` to automatically apply escaping
- [ ] Use safe property binding such as [href], [src], [style.color]
- [ ] Use binding to [innerHTML] to safely insert HTML data
- [ ] Do not use `bypassSecurityTrust*()` on untrusted data

PREVENTING CODE INJECTION OUTSIDE OF ANGULAR

- [ ] Avoid direct DOM manipulation
  *E.g. through **ElementRef** or other client-side libraries*
- [ ] Do not combine Angular with server-side dynamic pages
- [ ] Use Ahead-Of-Time compilation (AOT)

4 BROKEN ACCESS CONTROL

OWASP #5

AUTHORIZATION CHECKS

- [ ] Implement proper authorization checks on API endpoints
  Check if the user is authenticated
  Check if the user is allowed to access the specific resources
- [ ] Do not rely on client-side authorization checks for security

CROSS-ORIGIN RESOURCE SHARING (CORS)

- [ ] Prevent unauthorized cross-origin access with a strict policy
- [ ] Avoid accepting the ***null*** origin in your policy
- [ ] Avoid blindly reflecting back the value of the origin header
- [ ] Avoid custom CORS implementations
  Origin-matching code is error-prone, so prefer the use of libraries

5 SENSITIVE DATA EXPOSURE

OWASP #3

DATA IN TRANSIT

- [ ] Serve everything over HTTPS
- [ ] Ensure that all traffic is sent to the HTTPS endpoint
  Redirect HTTP to HTTPS on endpoints dealing with page loads
  Disable HTTP on endpoints that only provide an API
- [ ] Enable ***Strict Transport Security*** on all HTTPS endpoints

DATA AT REST IN THE BROWSER

- [ ] Encrypt sensitive data before persisting it in the browser
- [ ] Encrypt sensitive data in JWTs using JSON Web Encryption