Angular Security

Agenda

Review of General Security

Using custom browsers for attacks

Angular Security Guide

Security in the Angular API Docs

Sanitizers and element schema

XSRF

Browsers And Hackers

Easy for hacker to download Chromium (the open source Chrome project)

- Make “alterations” and run as a custom browser
- Applications that run in browser and server need to be fully aware of this
- Don’t trust anything that comes from browser
- Validate in browser and definitely again on server

Security Review

Industry is moving to using https always

- TLS 1.2 strongly recommended
- E.g. consider use with Stripe

Seemingly innocuous details passed over http can eventually become a threat

- Where I was born, went on holidays, name of dog

SSL also allows more innovative services (Web Sockets, webRTC) without interference from enterprise perimeters (is this good or bad?)

Other Issues

Two-factor authentication

- So many password hacks that moving to two-factor auth is really a good idea
- End-users are beginning to understand its benefits

SQL Injection attacks

- Mainly an issue on server
- Don’t dynamically create SQL

Angular Security Guide

Every app developer needs to become familiar with:

- https://angular.io/guide/security

e.g. lists these best practices:

- “Keep current with the latest Angular library releases”
- “Don't modify your copy of Angular”
- “Avoid Angular APIs marked in the documentation as ‘Security Risk.’ ”

Additional Best Practice

Enable Content Security Policy (CSP)

Do not use DOM API directly

- (Additional benefit: allows use of different renderers in future, such as web workers)

AOT Compiler

Do not use server-side templates

Security Risk Types

API docs marks types that are security risks as follows:

- https://angular.io/api?status=security-risk

Sample entries are:

- ElementRef
- DomSanitizer

Cross Site Request Forgery (XSRF)

Explanation of the attack

Mitigation involves client and server

Cookie

- token for authentication

Custom request header

Same origin policy

Notes from Security Guide

"XSRF/CSRF tokens should be

- unique per user and session,
- have a large random value generated by a cryptographically secure random number generator,
- and should expire in a day or two."

RNG should be cryptographic, not general purpose

Angular HTTP / XSRFStrategy

Exported Interface:

- export abstract class XSRFStrategy {
  abstract configureRequest(req: Request): void; }

CookieXSRFStrategy

- export class CookieXSRFStrategy implements XSRFStrategy {
- constructor(private _cookieName: string = 'XSRF-TOKEN',
  private _headerName: string = 'X-XSRF-TOKEN') {}
- configureRequest(req: Request) {
- let xsrfToken =
  __platform_browser_private__.getDOM().getCookie(this._cookieName);
- if (xsrfToken && !req.headers.has(this._headerName)) {
- req.headers.set(this._headerName, xsrfToken); } } }

Cross-site Script Inclusion

Prevent by ensuring JSON is non-executable

- ")]}',\n"

Angular HTTP Client does this

Security Context

Identifies part of DOM that needs to be considered from a security viewpoint

- export enum SecurityContext {
- NONE,
- HTML,
- STYLE,
- SCRIPT,
- URL,
- RESOURCE_URL,
- }

Sanitizers

Cleans up potentially dangerous areas

- export abstract class Sanitizer {
- abstract sanitize(
  context: SecurityContext, value: string): string;
- }

DOM Element Schema Registry

Not documented on Angular.io

Look at Angular src under:

- https://github.com/angular/angular/tree/master/packages/compiler/src/schema

Examine registry

Supplied Sanitizers

DomSanitizationService

HtmlSanitizer

StyleSanitizer

UrlSanitizer

- https://github.com/angular/angular/tree/master/packages/platform-browser/src/security