Security on the web

Recording of presentation (in Swedish):
https://drive.google.com/file/d/1rligXBAj0ywpZ-x6geaar1ijcj-s-rzc/view?usp=sharing

Simon Nielsen 2022-06-14

Why?

- Security is a difficult and important subject
- There are many levels of security and many different solutions
- I want to know what risks I accept with my solution - thinking about it pragmatically

Dr. Philippe De Ryck https://pragmaticwebsecurity.com/

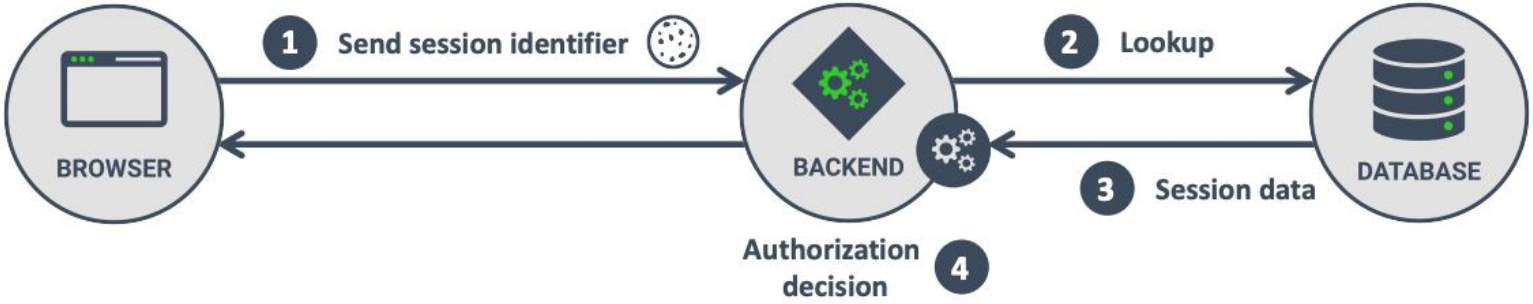
https://pragmaticwebsecurity.com/talks/commonapisecuritypitfalls

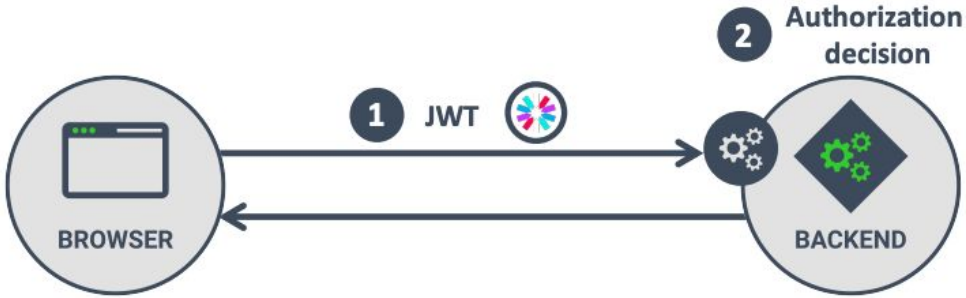
Agenda

- Session vs token-based authentication
- Basics about JWT and Cookies
- Same-origin policy and CORS
- XSS and how we can avoid it

Ask many questions!

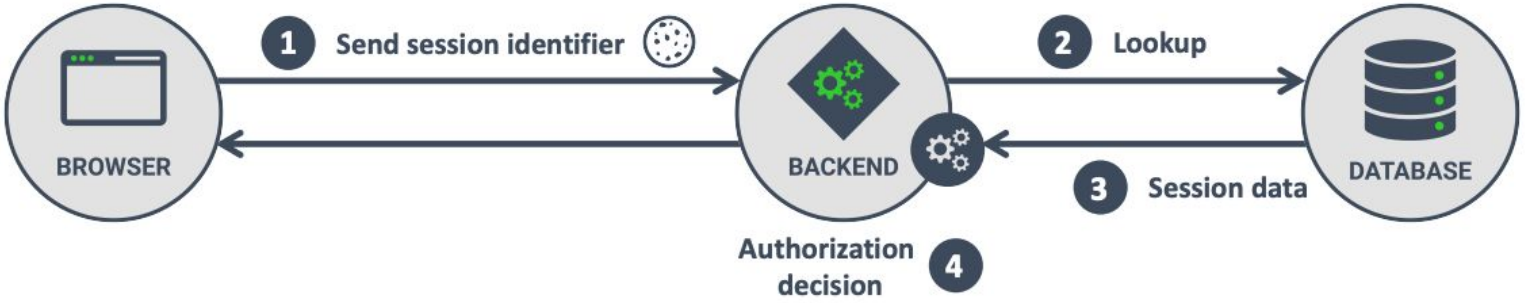
Session vs token-based authentication

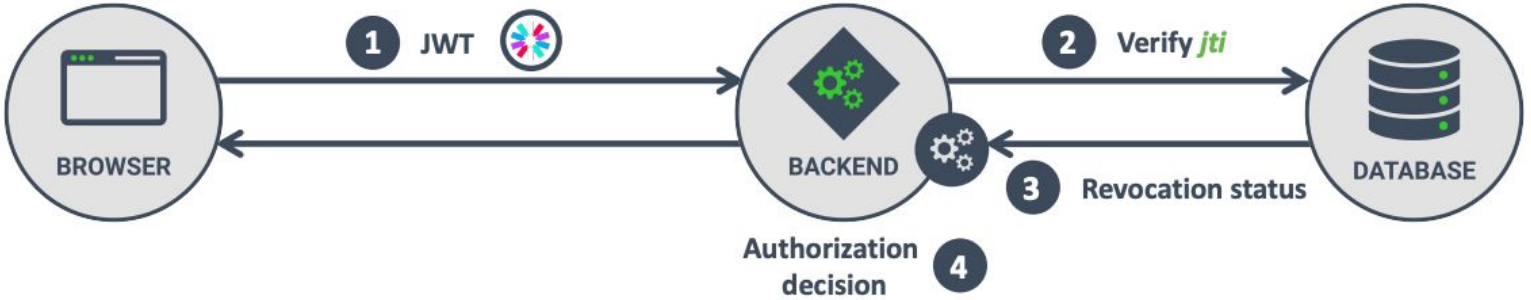

1
Send session identifier
2
Lookup
BROWSER
BACKEND
DATABASE
3
Session data
Authorization
decision
4


2
Authorization
decision
1
JWT
BROWSER
BACKEND

JWT REVOCATION

- A common revocation pattern uses the JWTs unique identifier
  - Keeping a list of invalid identifiers enables the backend to reject revoked JWTs
- Revoking a specific token for a specific device is challenging
  - The backend needs to keep a list of all issued ***jti*** claims
  - These identifiers need to be correlated to users and devices
- Verifying incoming JWTs against a revocation list requires explicit action
  - Depends on a centralized list of invalid identifiers
  - Check needs to happen on each incoming request
  - Adds a form of state to an otherwise stateless backend


1
Send session identifier
2
Lookup
BROWSER
BACKEND
DATABASE
3
Session data
Authorization
decision
4


1
JWT
2
Verify jti
BROWSER
BACKEND
DATABASE
3
Revocation status
Authorization
decision
4

Mistaking JWTs for sessions

JWTs are a way to represent claims, nothing more. Using them for authorization data requires an elaborate support system, such as OAuth 2.0

JWTs

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlBoaWxpcHBlIERlIFJ5Y2siLCJyb2xlcyI6InVzZXIgcmVzdGF1cmFudG93bmVyIiwiaWF0IjoxNTE2MjM5MDIyfQ.KPjhyE9oi83uehgw6Lm_0yAZzRuJhcUqXETD2AIrF2A
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

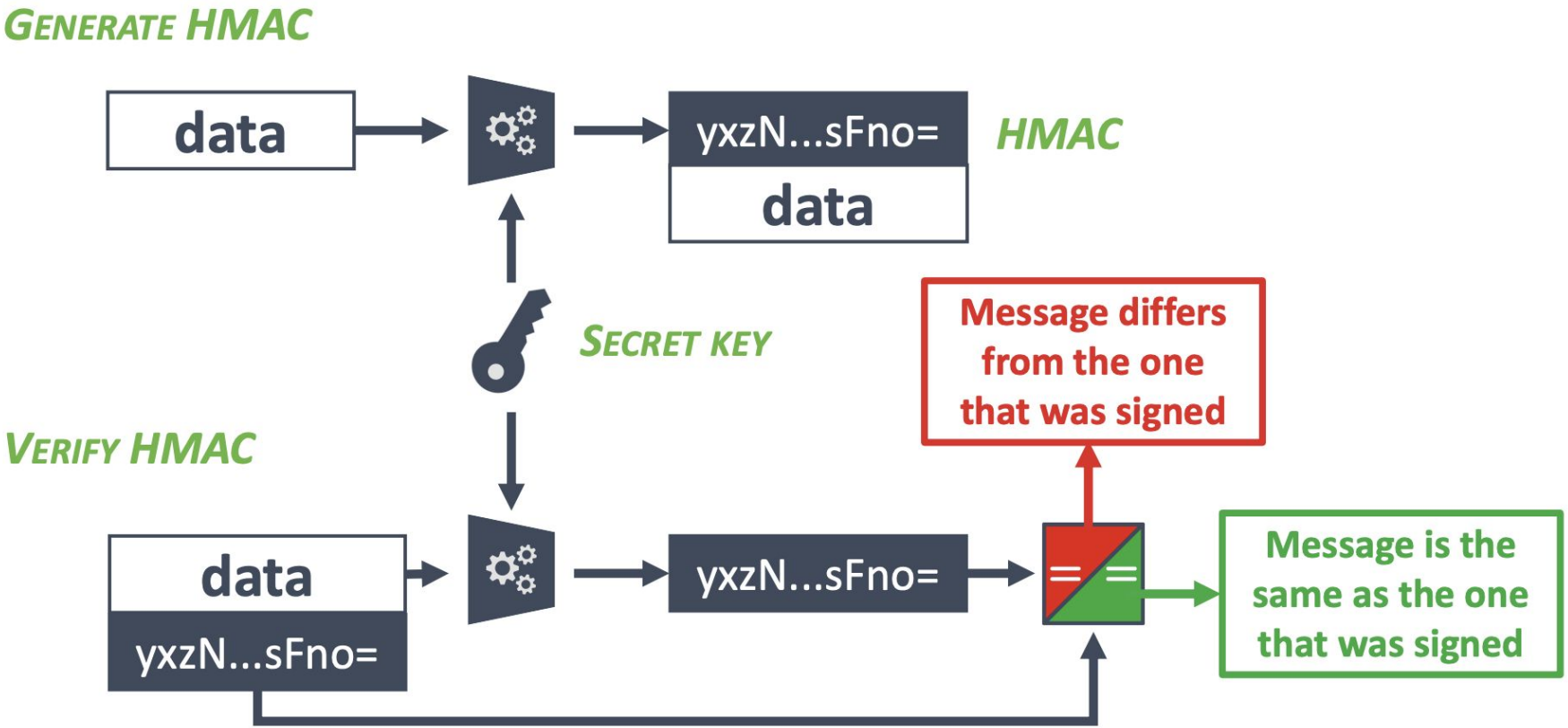
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "Philippe De Ryck",
  "roles": "user restaurantowner",
  "iat": 1516239022
}
```

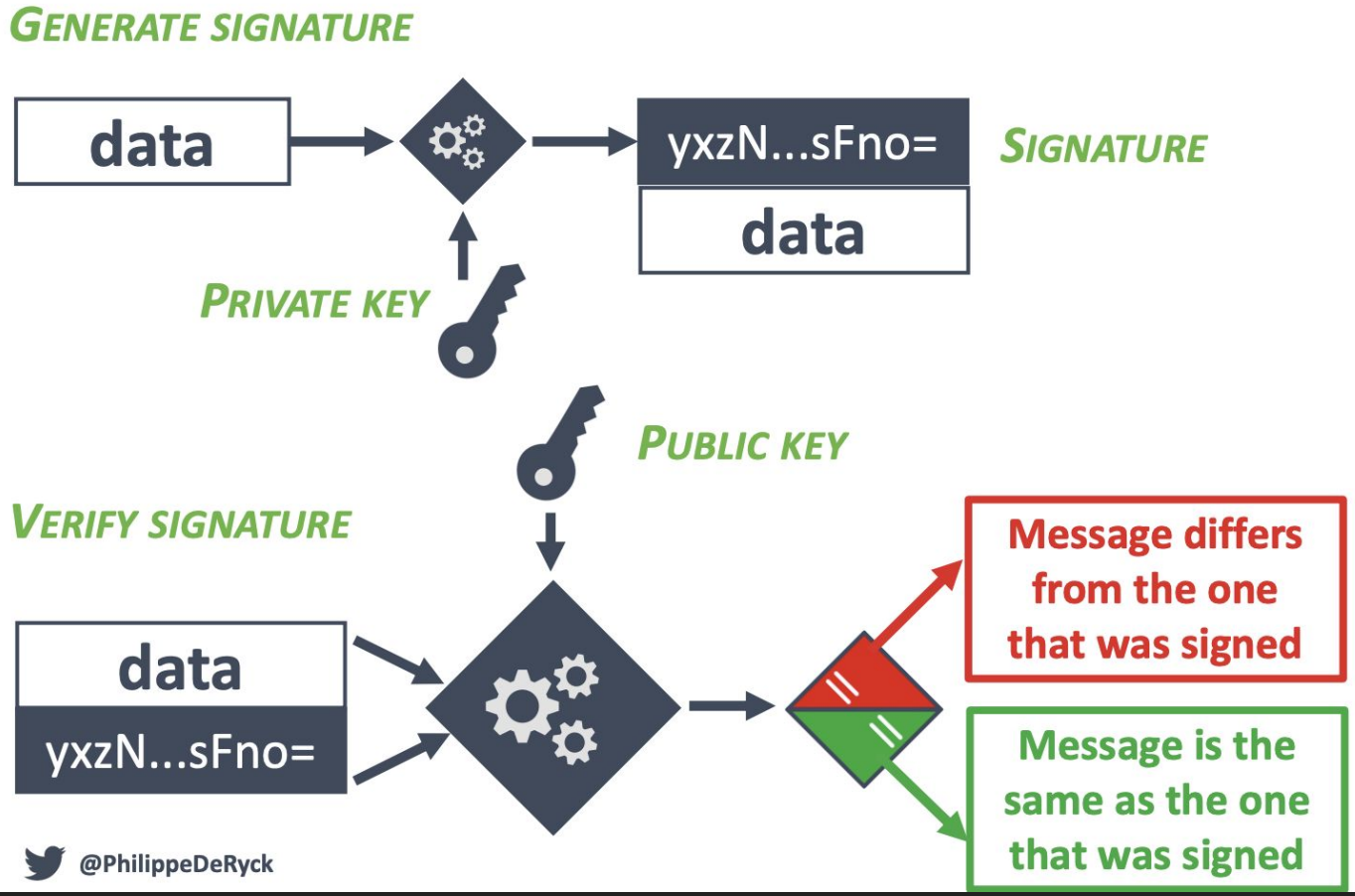
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) ☐ secret base64 encoded
```

HMAC-BASED JWT SIGNATURES



ASYMMETRIC JWT SIGNATURES



HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT",
  "kid": "9d8f0828-89c5-469b-af76-f180701710c5"
}
```

Identify a key known by the receiver

@PhilippeDeRyck

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "jku": "https://restograde.com/jwks.json",
  "kid": "5175cafe-82f0-4eab-8f3f-7bcfb3bf5ee0",
  "alg": "RS256"
}
```

Provide a URL containing a set of keys

@PhilippeDeRyck

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "KjrsfCS8cb9kJFkimgu6FdCqogWXURu-rLTbbyrL7jo",
  "jku": "https://evil.example.com/jwks.json"
}
```

@PhilippeDeRyck


```
builder.Services.AddOptions<JwtBearerOptions>(name: JwtBearerDefaults.AuthenticationScheme)
    .Configure<IssuerSigningKeyResolver, AudienceValidator>((options, signingKeyResolver, audienceValidator) =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            IssuerSigningKeyResolver =
                (_, _, kid:string, validationParameters) =>
                    signingKeyResolver.ResolveIssuerSigningKey(kid, validationParameters),
            ValidIssuer = authConfig["Issuer"],
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateLifetime = true,
            ValidAudiences = authConfig["Audience"].Split(separator: '|'),
            ValidateAudience = true,
            AudienceValidator = audienceValidator.ValidateAudience
        };
    });
```

Cookies

A web server can answer with a header Set-Cookie which then the browser sets on that domain

SameSite - Cookie only sent along if the calling site is on the same domain

Secure - Cookie only sent along if the calling site runs Https (except for localhost now!)

HttpOnly - Can’t get the cookie from JavaScript*

SessionId=42; SameSite=Strict; HttpOnly; Secure; Expires=Thu, 31 Oct 2021 07:28:00 GMT;

Cookie: Id=42

Authorization: Bearer 42

Cookie: JWT=eyJhbGci…

Authorization: Bearer eyJhbGci…

Cookies

Can contain identifiers & session objects

Only works well with a single domain

Automatically handled by the browser

Always present, including on DOM resources and WebSockets

Authorization header

Can contain identifiers & session objects

Freedom to include headers to any domain

Requires custom code to get, store and send session data

Only present on XHR calls, unless you circumvent this with a ServiceWorker

CORS

“Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a *browser* should permit loading of resources.”

CORS asks for approval with a preflight OPTIONS request

- The request tells the server what the browser wants to do
- The server needs to respond with the proper CORS headers to authorize the request

CORS

Access-Control-Allow-Origin

No CORS policy -> Same origin policy -> no cross origin requests are allowed

Access-Control-Allow-Credentials

Access-Control-Allow-Headers

Cookies or bearer token?

“If you use tokens that you store in localStorage you are vulnerable to XSS, if you use cookies you are vulnerable to CSRF”

“Stop storing your JWTs in localStorage!!”

“An HttpOnly cookie is safe from XSS, you can’t get it from JavaScript”

“Instead of spending lots of time figuring out where to store your token (cookie, localstorage, memory), spend that time learning how to prevent XSS - because that’s the real problem.”

- Philippe De Ryck

Källa: Common API Security Pitfalls https://pragmaticwebsecurity.com/talks/commonapisecuritypitfalls

XSS

XSS - First level defense

- Angular, React och Vue sanitizes everything you add in your HTML automatically

```
<div>
  {data}
</div>
```

- Except with *dangerouslySetInnerHtml*. Use DOMPurify to sanitize the input if you have to use it.

```
<div dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(data)}} />
```

- Sanitize also URLs to *href* and *src*

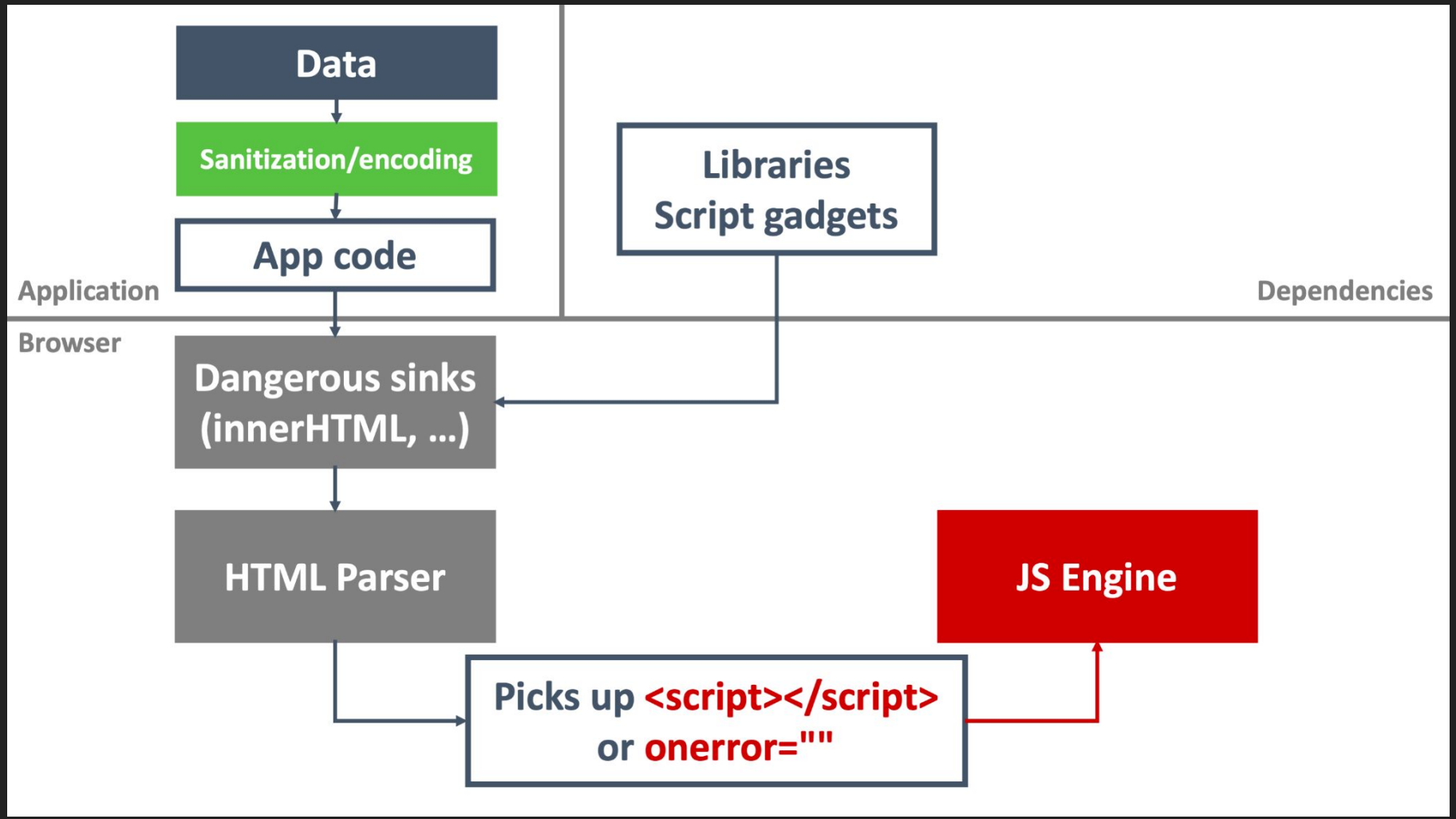
Source: React XSS Cheat sheet https://pragmaticwebsecurity.com/files/cheatsheets/reactxss.pdf

XSS och CSP - Second level of defense

- URL-based: Specify from which origin scripts can be run
  - Issues when trusting a third party CDN where someone hosts malicious code.
- Hashes: For allowing specific script-blocks
  - Recalculate whenever the code changes, not ideal
- Nonces: Generated server-side for each script
  - Needs server side rendering
- Automatic trust propagation with 'strict-dynamic'
  - https://www.npmjs.com/package/strict-csp-html-webpack-plugin

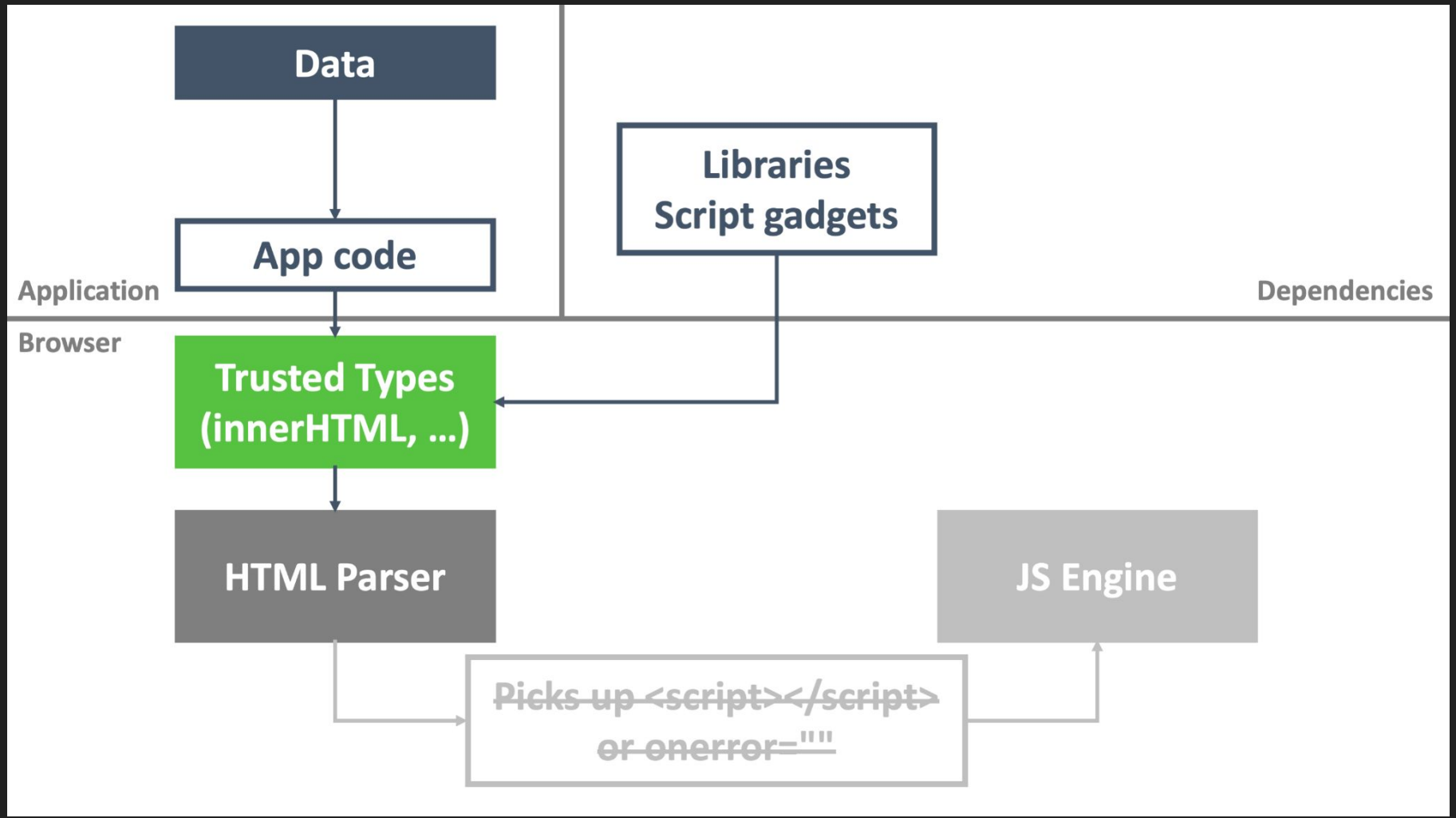
Further reading: https://web.dev/strict-csp/#adopting-a-strict-csp

What about all my NPM packages?


Data
Sanitization/encoding
App code
Application
Libraries
Script gadgets
Dependencies
Browser
Dangerous sinks
(innerHTML, ...)
HTML Parser
JS Engine
Picks up <script></script>
or onerror=""

```
<button class="btn btn-primary"
        data-toggle="tooltip"
        data-html="true"
        title="test<script>alert(1)</script>">
  Do NOT click this button!
</button>
```

```
React.findDOMNode(component).innerHTML = "<script>alert(1)</script>"
```

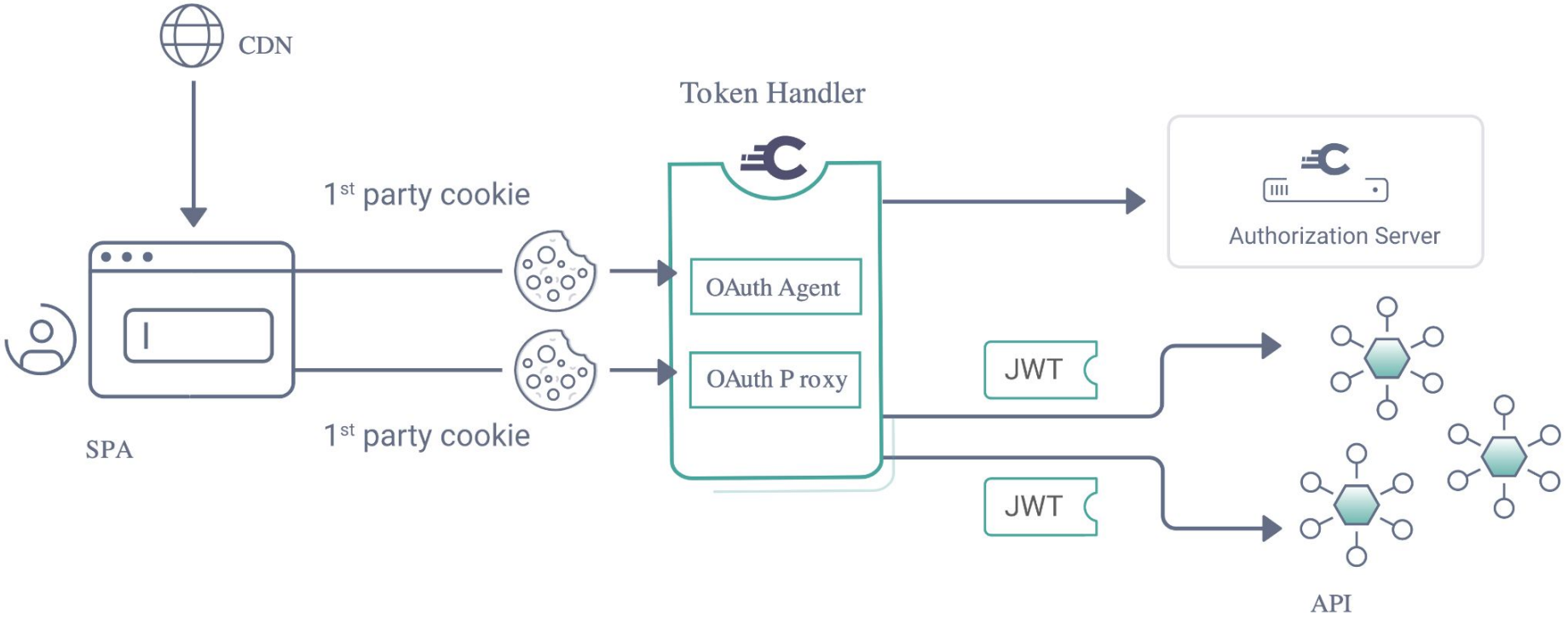

Data
App code
Application
Libraries
Script gadgets
Dependencies
Browser
Trusted Types
(innerHTML, ...)
HTML Parser
JS Engine
Picks up <script></script>
or onerror=""

Trusted Types

```
<meta http-equiv="Content-Security-Policy" content="require-trusted-types-for 'script'">
<script>
   if (window.trustedTypes && trustedTypes.createPolicy) { // Feature testing
       trustedTypes.createPolicy('default', {
         createHTML: string => DOMPurify.sanitize(string, {RETURN_TRUSTED_TYPE: true})
       });
     }
 </script>
```

Further reading:
https://auth0.com/blog/securing-spa-with-trusted-types/#Using-Trusted-Types-in-SPAs
https://github.com/w3c/webappsec-trusted-types#polyfill

Backend for Frontend and the Token Handler Pattern


CDN
Token Handler
1st party cookie
Authorization Server
OAuth Agent
OAuth P roxy
JWT
JWT
SPA
1st party cookie
API

Session *Riding* still remains a problem

Discussion