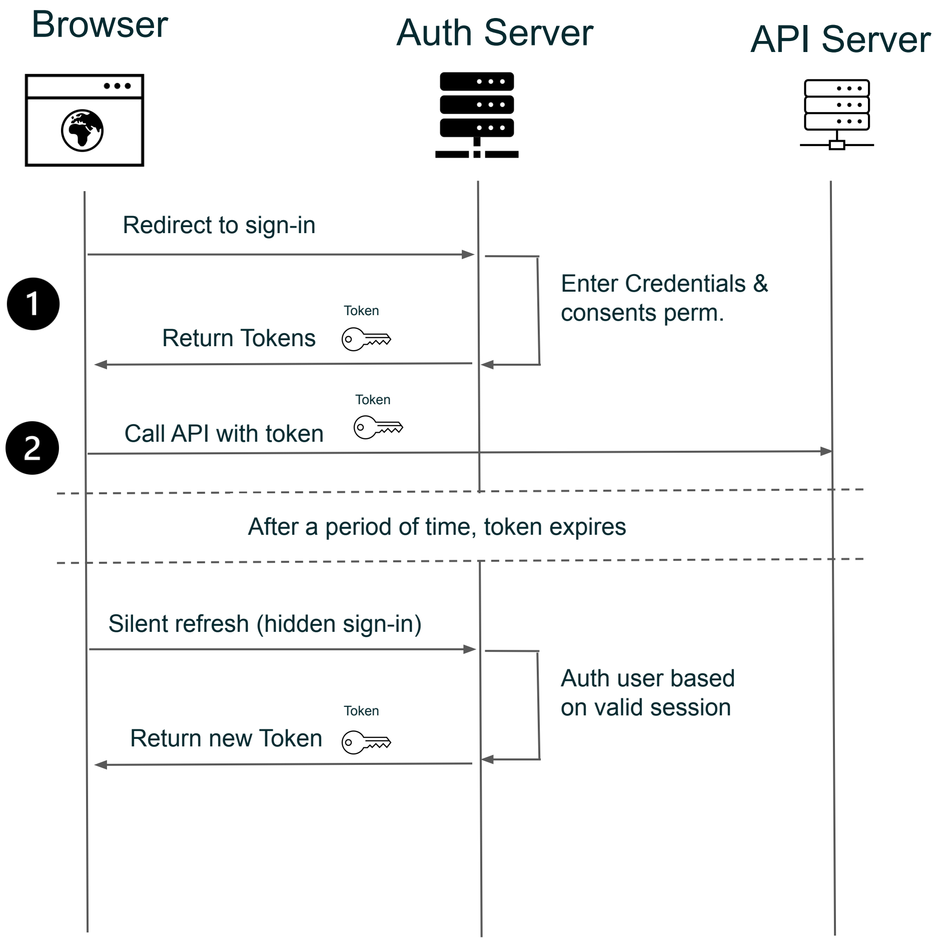
The insecurity of OAuth 2.0 in frontends

Peter Cosemans
Michiel Olijslagers

OAuth 2.0 Flows

OAuth 2.0 has different grant types for various scenarios. Here are a few of them.

- **Implicit flow**: (previous) used for single-page JavaScript apps where secrets cannot be securely stored.
- **Client credentials flow**: used for server to server auth.
- **Authorization Code flow**: for mobile and server based web apps
- **Authorization Code with PKCE**: new standard for mobile and web apps


Browser
Auth Server
API Server
Redirect to sign-in
1
Enter Credentials & consents perm.
Token
Return Tokens
2
Token
Call API with token
After a period of time, token expires
Silent refresh (hidden sign-in)
Auth user based on valid session
Token
Return new Token

Implicit Flow

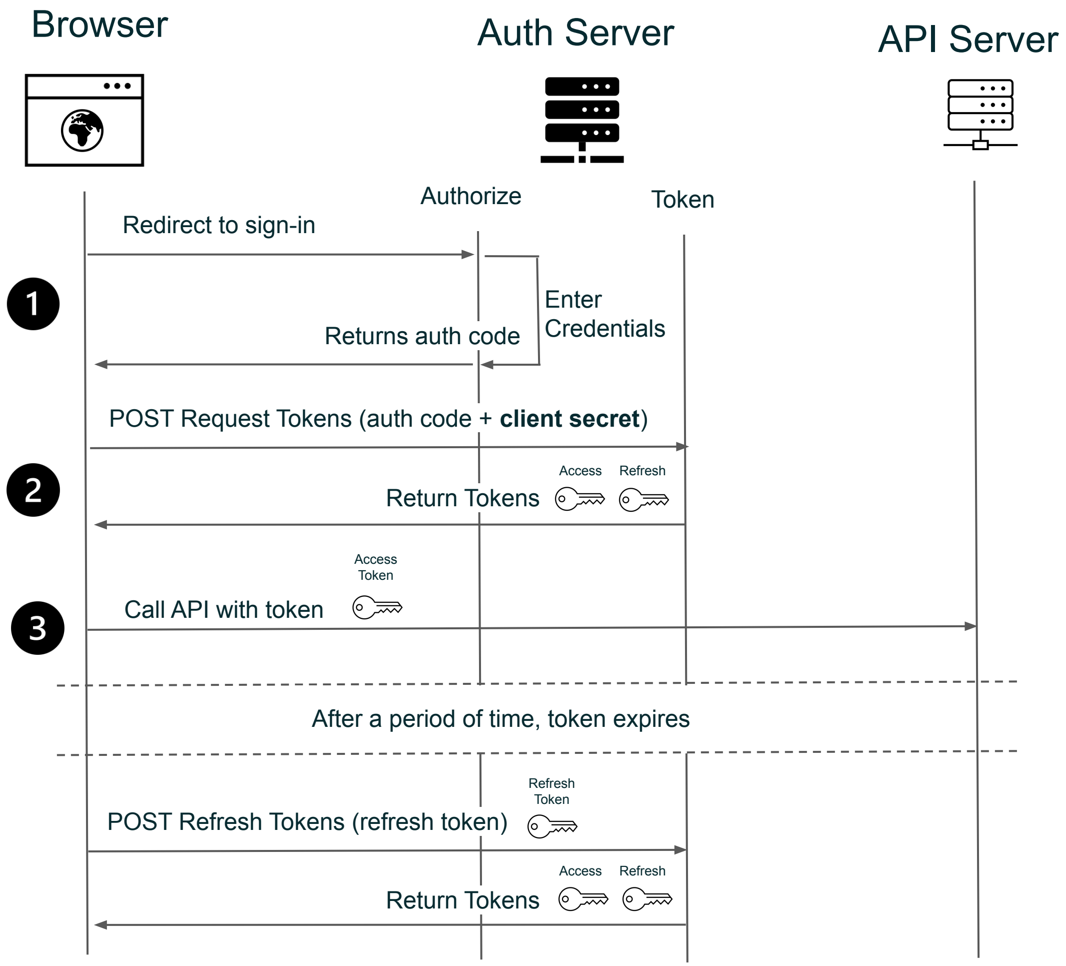
- Silent refresh doesn't work anymore by blocking of 3th party cookies (over multiple domains)
- Is vulnerable to token interception attack 🫣
- Token is received via the URL, cached in history. 🫣

OAuth 2.0 Recommendation: Browser-based clients MUST use the Authorization Code flow and MUST NOT use the Implicit flow to obtain access tokens.

(and some servers prohibit this flow entirely)

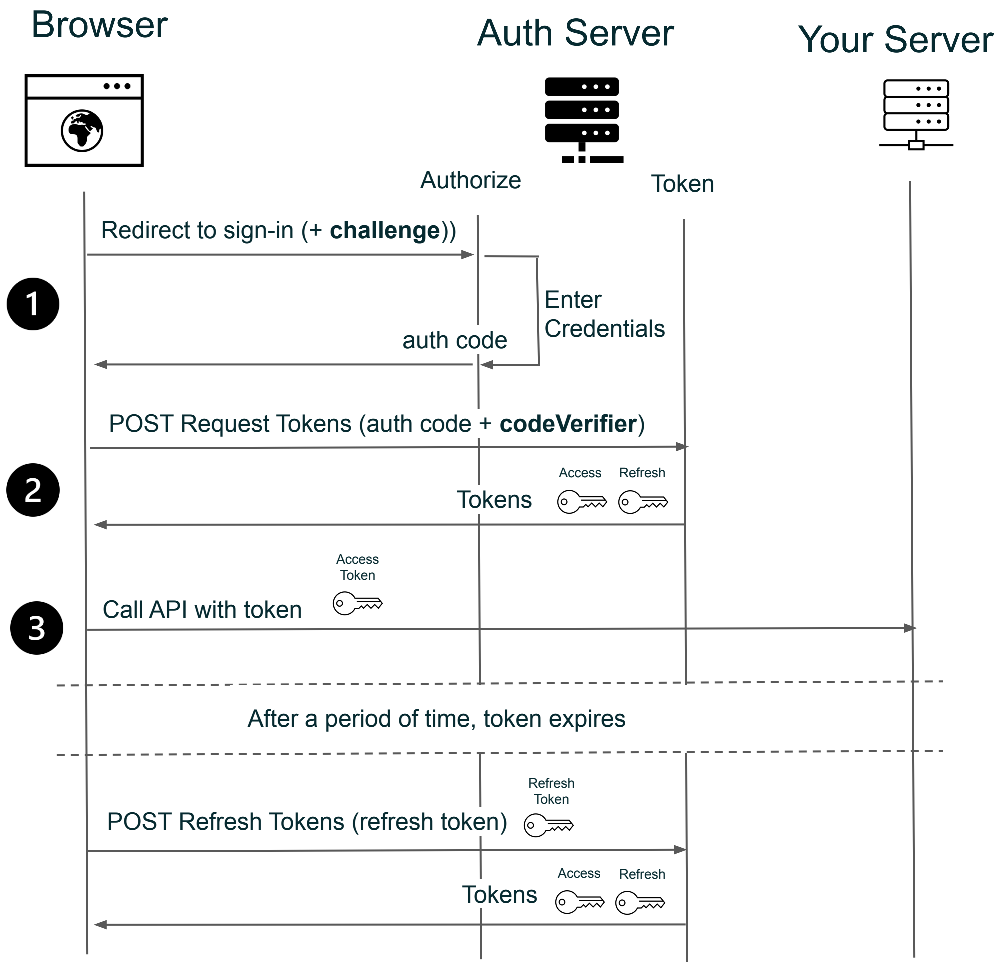
https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-implicit-grant-flow
https://datatracker.ietf.org/doc/html/draft-ietf-oauth-browser-based-apps#name-oauth-implicit-flow

Authorization Code Flow



- Only for confidential clients (servers)
- Is vulnerable to token interception attack 🫣

https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow


Browser
Auth Server
Your Server
Authorize
Token
Redirect to sign-in (+ **challenge**))
1
Enter Credentials
auth code
POST Request Tokens (auth code + **codeVerifier**)
2
Access
Refresh
Tokens
Access Token
Call API with token
3
After a period of time, token expires
Refresh Token
POST Refresh Tokens (refresh token)
Access
Refresh
Tokens

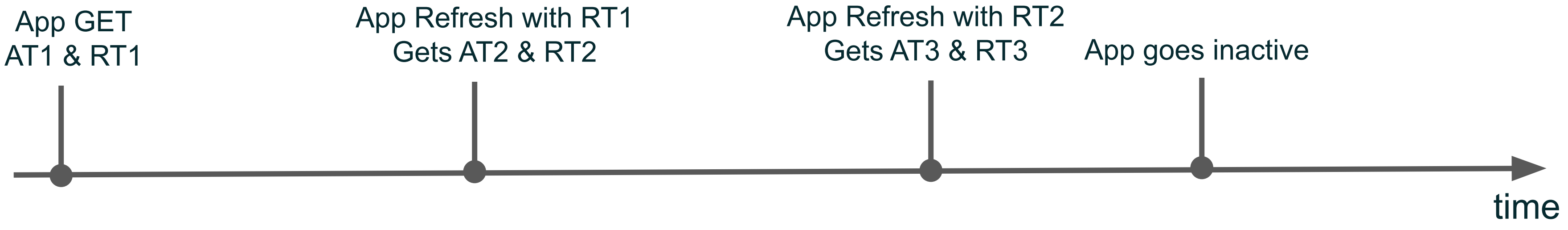
Authorization Code Flow with PKCE

(Proof Key for Code Exchange)

- Improved security by proof of possession
- Can be used by public or confidential clients (web & mobile)

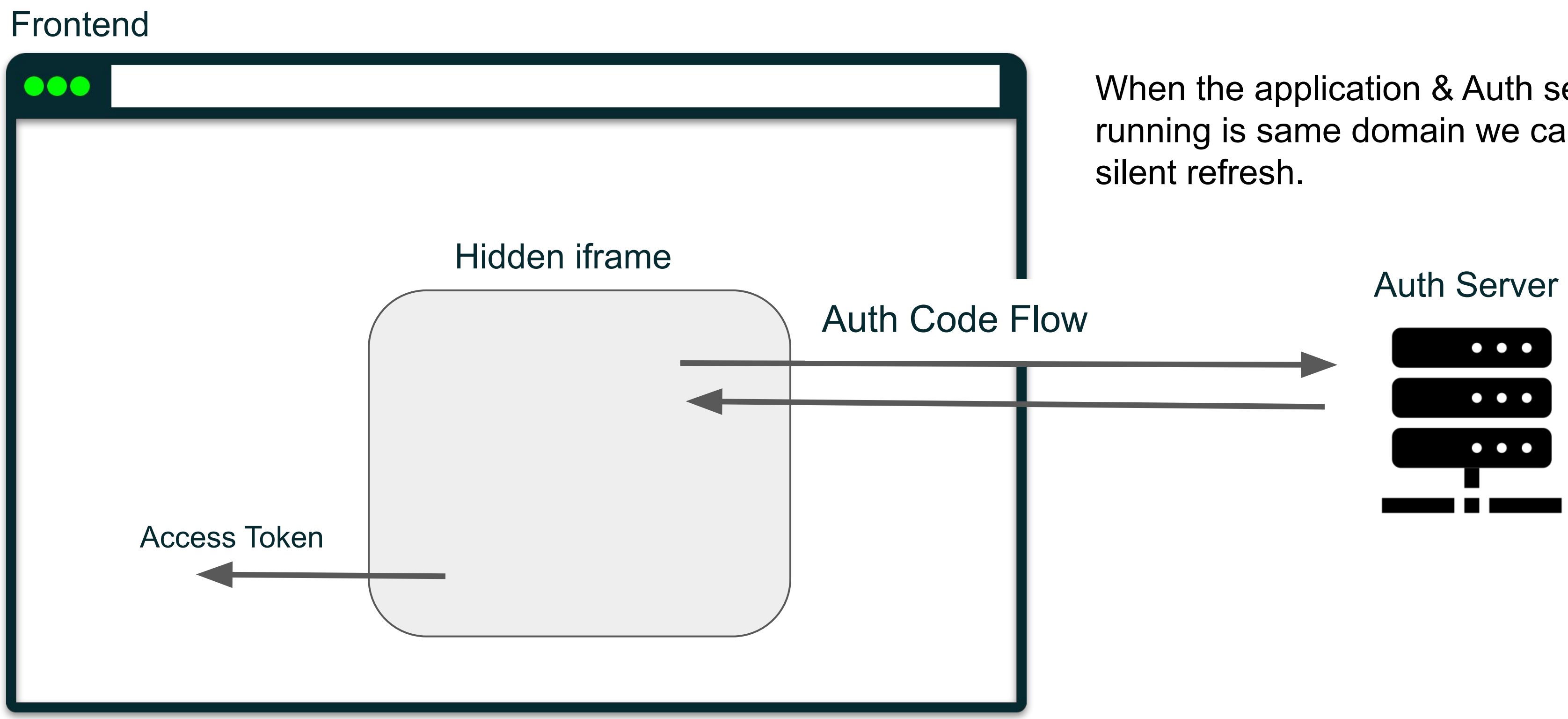
codeVerifier = random({length: 128})
challenge = base64UrlEncode(sha256Hash(codeVerifier))

Refresh Token Rotation



Good practise to keep **access token short-lived** and use **refresh token rotation.**

Silent Refresh



When the application & Auth server running is same domain we can use silent refresh.

With Silent Refresh we can avoid using refresh tokens

OAuth 2.1 Flows

OAuth 2.1 removes insecure flows, so only 2 remains

- Implicit flow
- **Client credentials flow**
- Authorization Code
- **Authorization Code with PKCE**

In the **draft of OAuth 2.1** the use of the **PKCE** extension for native apps has been **recommended** to all kinds of OAuth clients, including web applications and other confidential clients in order **to avoid** malicious browser extensions to perform OAuth 2.0 **code injection attack**

Attacking OAuth 2.0 in frontend

Malicious JavaScript (XSS)

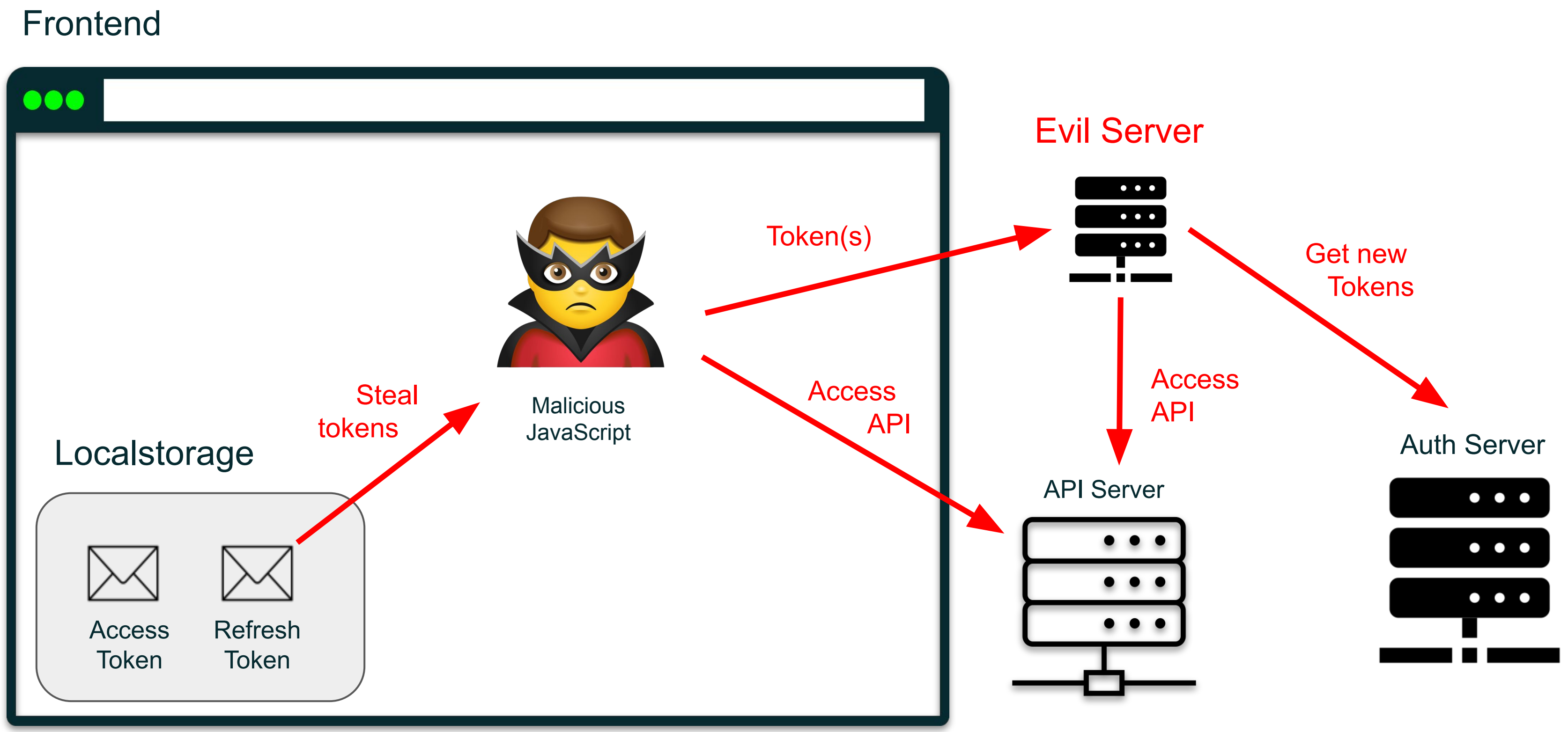
Frontend

Malicious JavaScript

- Can read storage
- Can access data
- Can send data to other server
- Can abuse stolen data
- Can do almost everything…

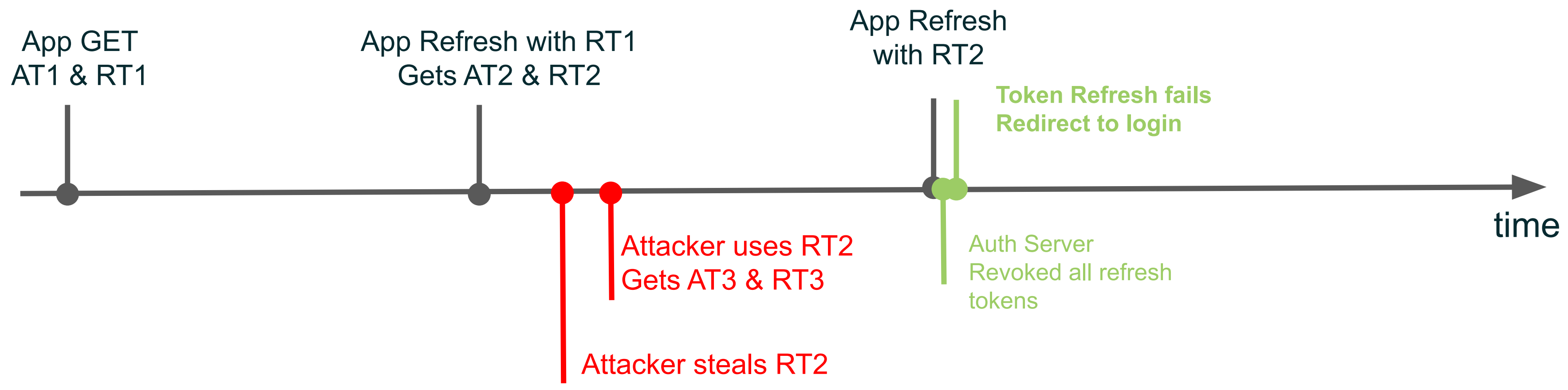
The browser cannot distinguish it malicious code from legitimate code.

Stealing tokens



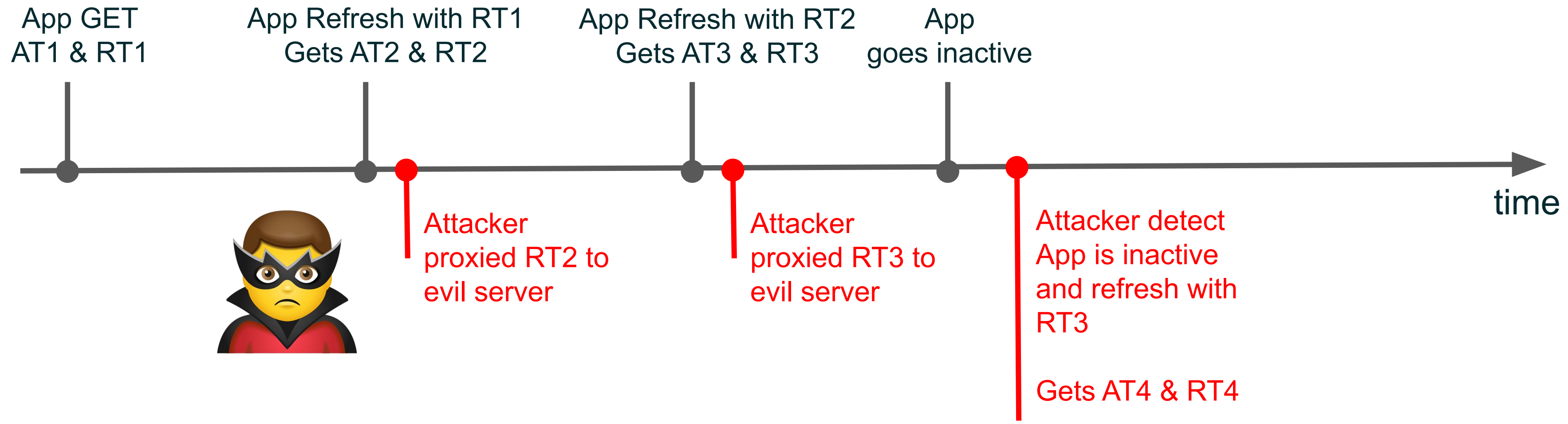
Hacker can steal tokens and access api 🫣

Refresh Token Rotation - Stealing Token



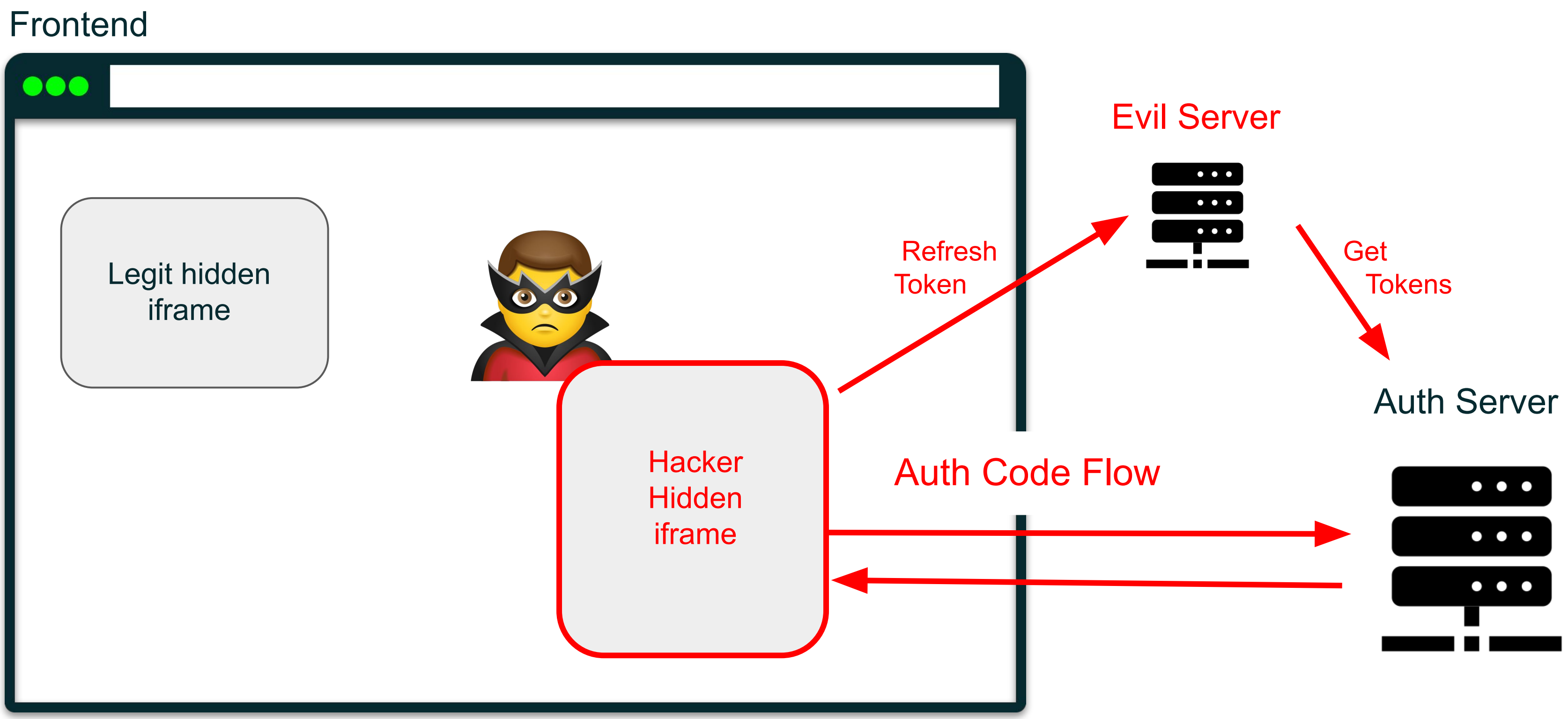
When the auth server detect a re-use of the refresh token, something is wrong and revokes all refresh tokens to prevent abuse.

Bypassing Refresh Token Rotation Protection



Attacker steals Refresh Token and send it to evil server (under his control) and waits until application becomes inactive. Then the attacker start using the access- and refresh tokens.

Stealing tokens with Silent Refresh



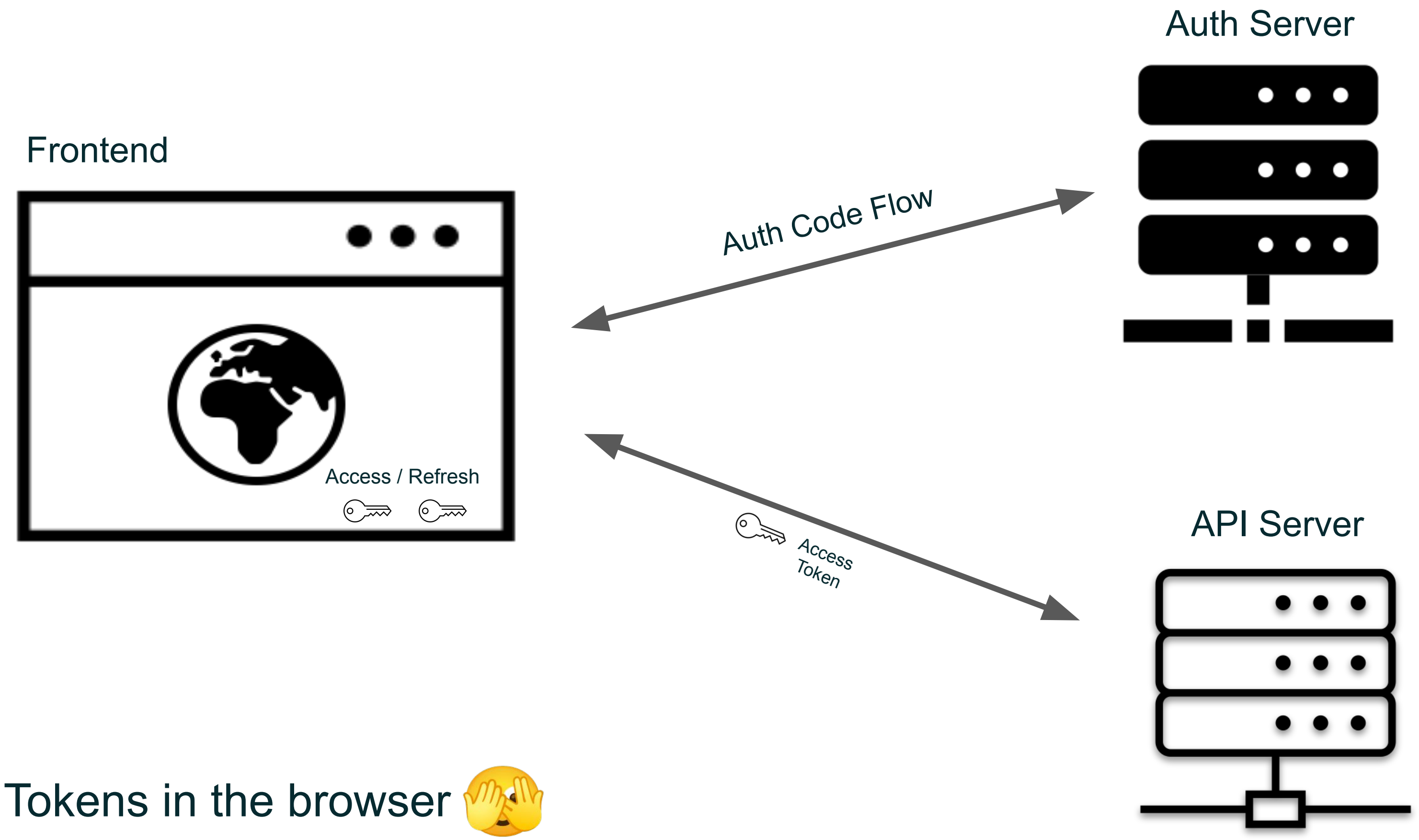
Hacker can issues a separate iframe and request his own access/refresh token.

Securing solely frontend applications with OAuth 2.0 is insufficient for comprehensive protection.

XSS is game over

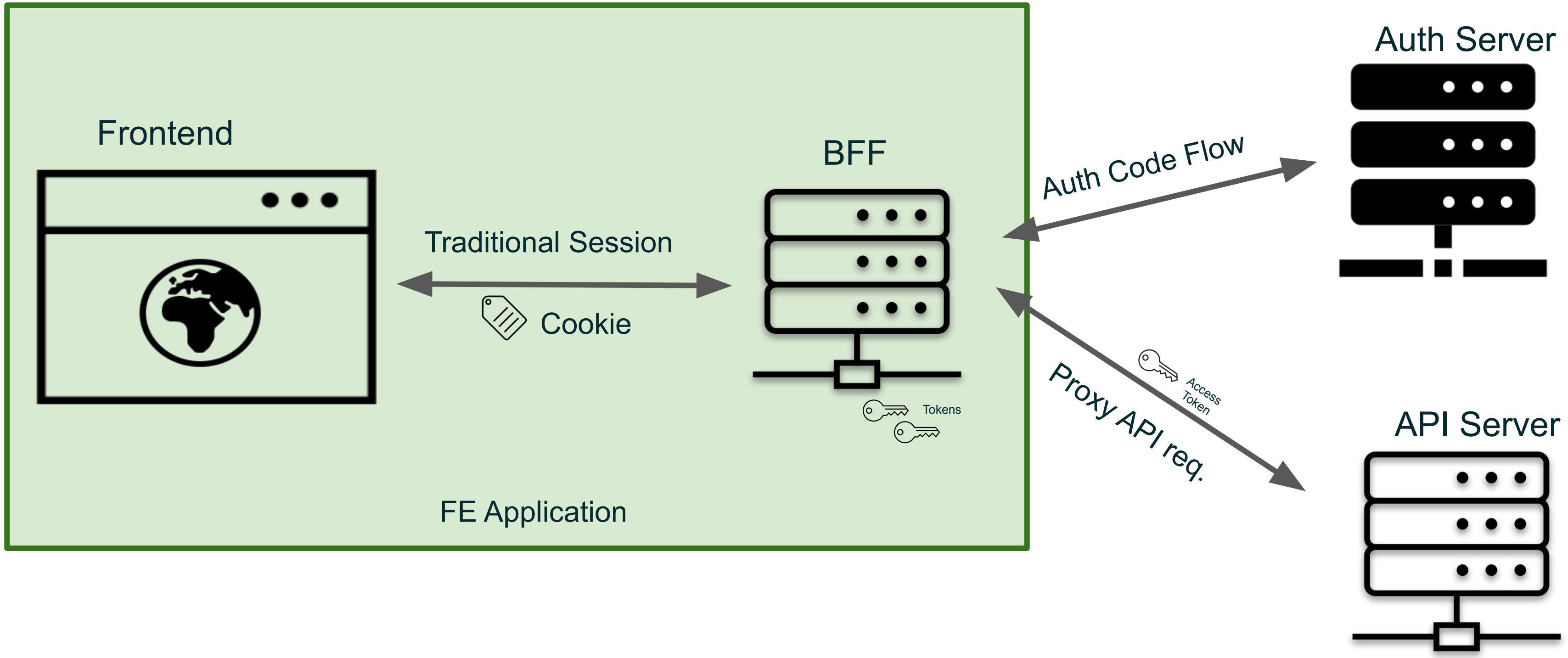
The backend-for-frontend (BFF) pattern

Authorization Code in Front-end



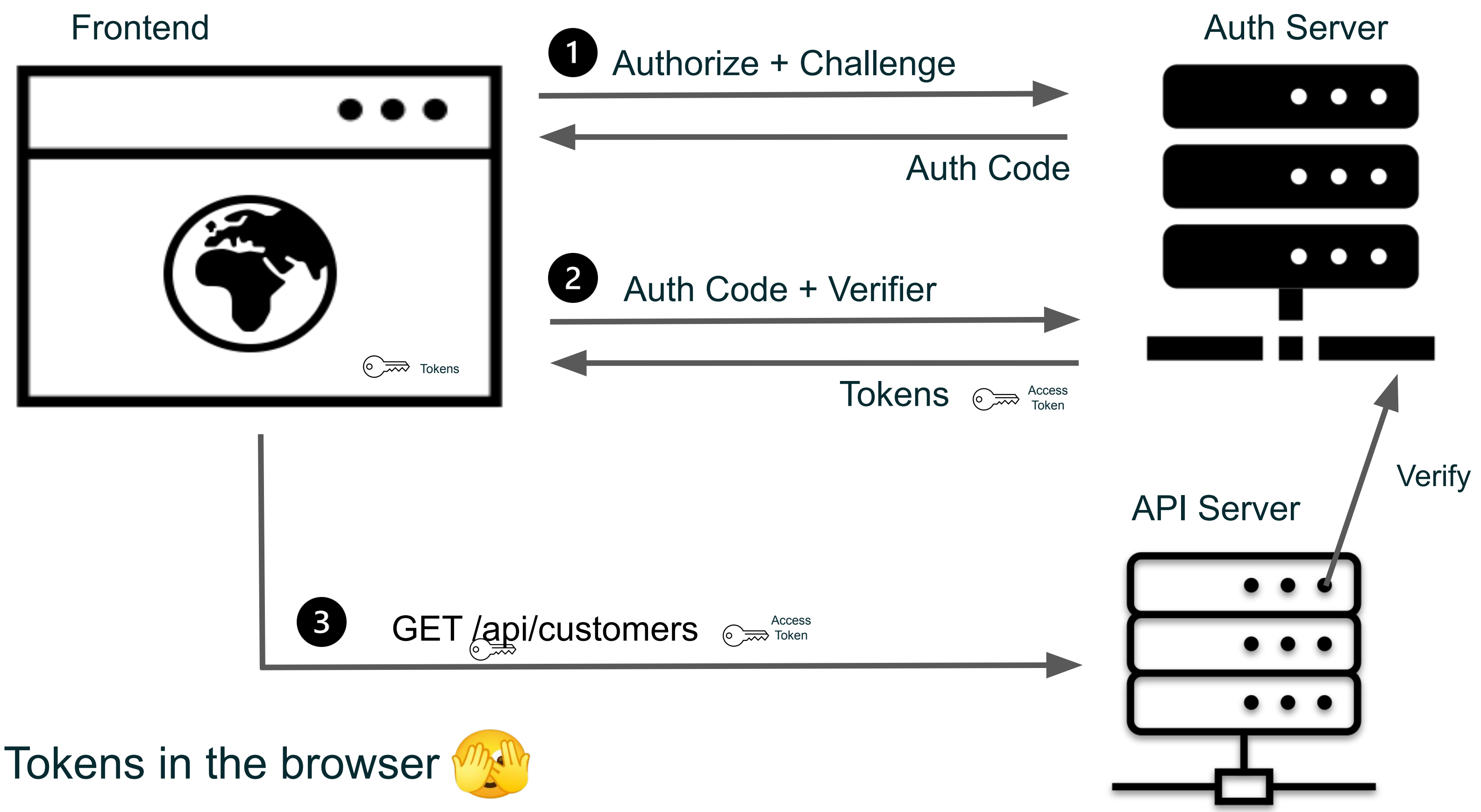
Tokens in the browser 🫣

Authorization Code with Backend for Frontend (BFF)



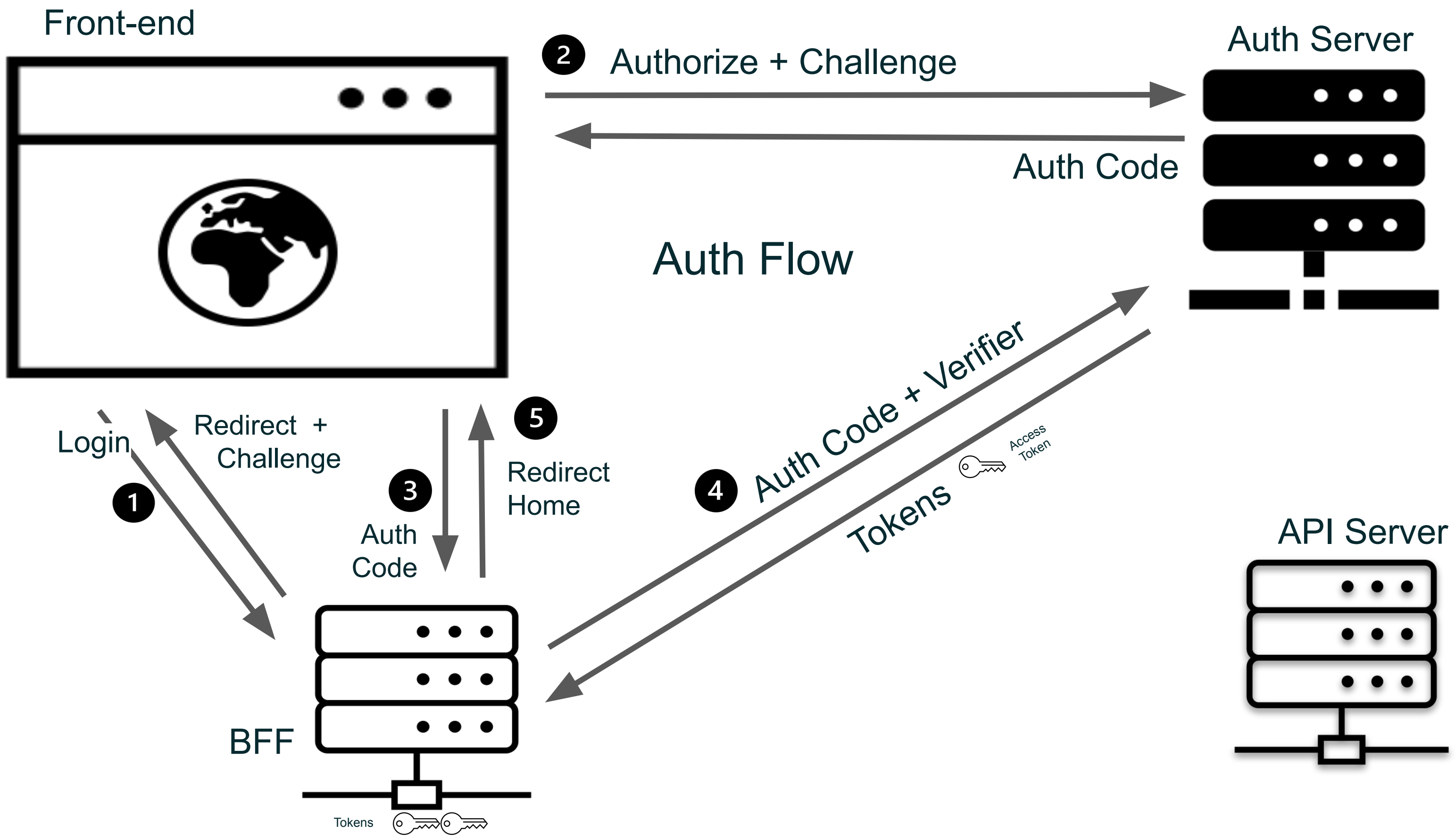
Tokens in the BE

Authorization Code in Front-end (not safe)

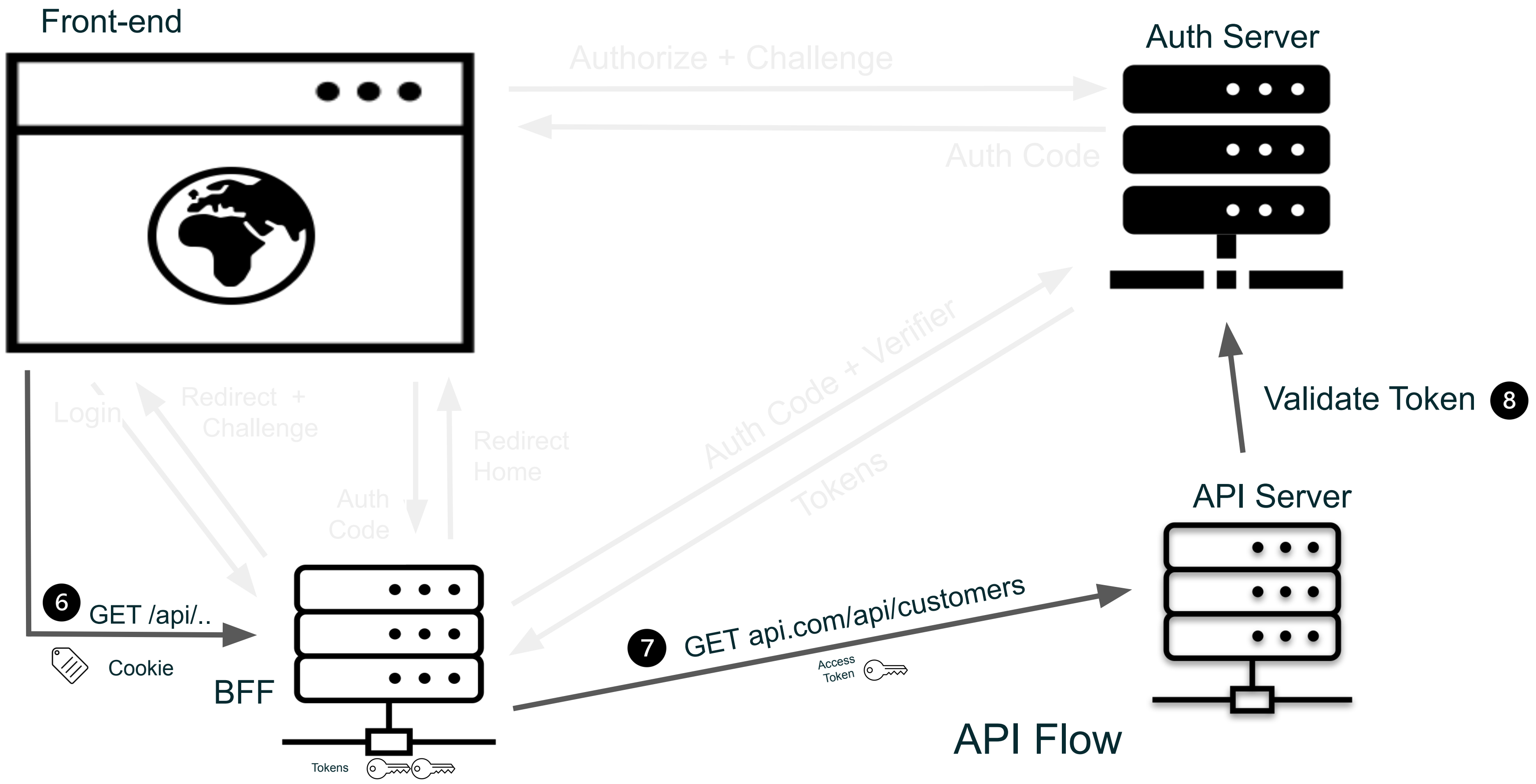


Tokens in the browser 🫣

Authorization Code with BFF

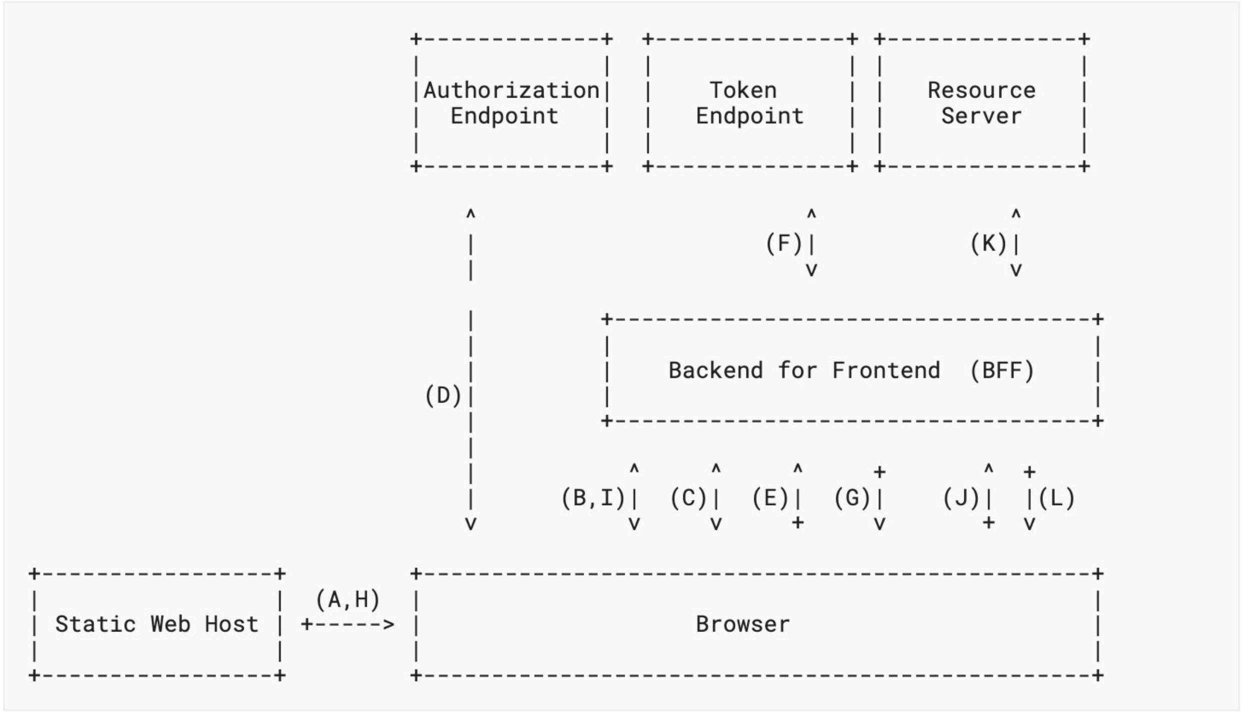


Authorization Code with BFF



API Flow

Specification: OAuth 2.0 for Browser-Based Apps



6.1.1. Application Architecture

```
                            +-------------+  +--------------+ +--------------+
                            |             |  |              | |              |
                            |Authorization|  |    Token     | |   Resource   |
                            |  Endpoint   |  |   Endpoint   | |    Server    |
                            |             |  |              | |              |
                            +-------------+  +--------------+ +--------------+

                                ^                        ^              ^
                                |                     (F)|           (K)|
                                |                        v              v

                                |         +-----------------------------------+
                                |         |                                   |
                                |         |    Backend for Frontend  (BFF)    |
                             (D)|         |                                   |
                                |         +-----------------------------------+
                                |
                                |           ^     ^     ^     +       ^  +
                                |      (B,I)|  (C)|  (E)|  (G)|    (J)|  |(L)
                                v           v     v     +     v       +  v

+-----------------+         +-------------------------------------------------+
|                 |  (A,H)  |                                                 |
| Static Web Host | +-----> |                    Browser                      |
|                 |         |                                                 |
+-----------------+         +-------------------------------------------------+
```

https://www.ietf.org/archive/id/draft-ietf-oauth-browser-based-apps-17.html

BFF Solutions

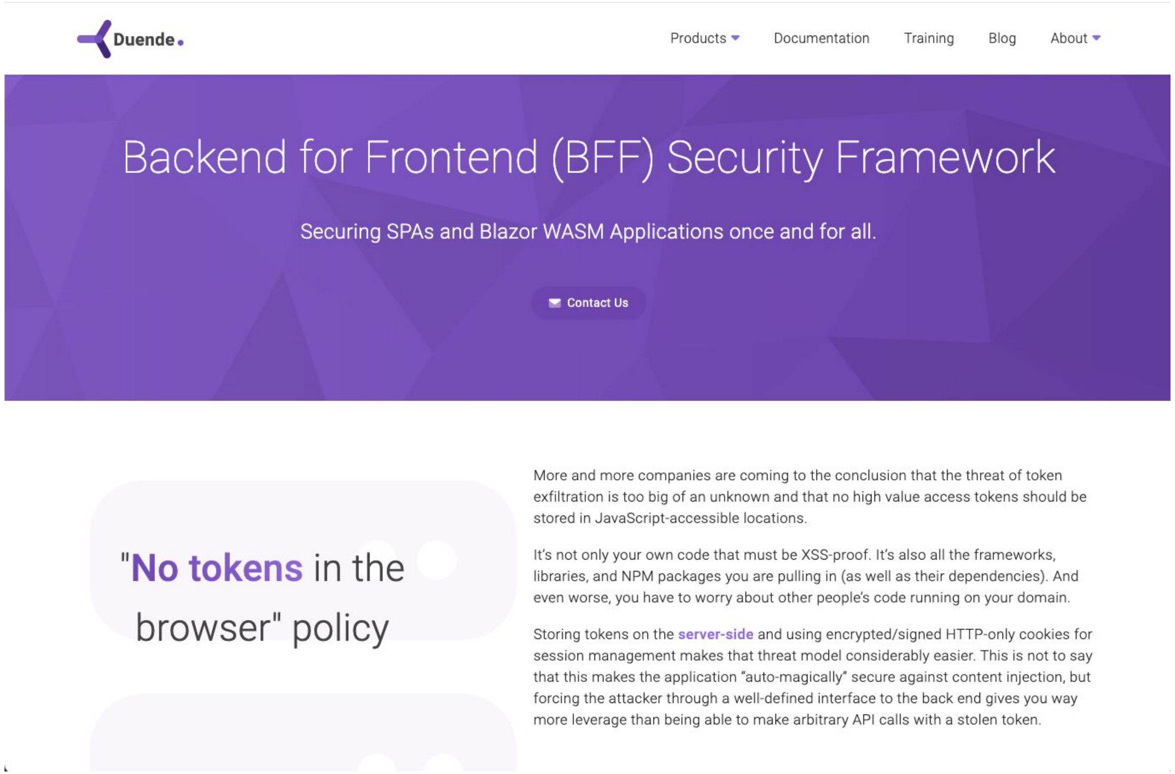
Full stack JavaScript frameworks

Full-stack JavaScript frameworks are optimally suited for a secure BFF architecture.

- Auth.js delivers server side oAuth 2.0 authentication for Nuxt, Next.js, SveltKit & Solid Start.
- Remix Auth provide a simple & secure authentication for Remix

Full-stack JavaScript frameworks are also capable of effectively managing stringent **Content Security Policies**, with the use of nonces for enhanced security.

Duende - BFF Security Framework for .NET



Backend for Frontend (BFF) Security Framework

Securing SPAs and Blazor WASM Applications once and for all.

"**No tokens** in the browser" policy

More and more companies are coming to the conclusion that the threat of token exfiltration is too big of an unknown and that no high value access tokens should be stored in JavaScript-accessible locations.

It's not only your own code that must be XSS-proof. It's also all the frameworks, libraries, and NPM packages you are pulling in (as well as their dependencies). And even worse, you have to worry about other people's code running on your domain.

Storing tokens on the **server-side** and using encrypted/signed HTTP-only cookies for session management makes that threat model considerably easier. This is not to say that this makes the application "auto-magically" secure against content injection, but forcing the attacker through a well-defined interface to the back end gives you way more leverage than being able to make arbitrary API calls with a stolen token.

https://duendesoftware.com/products/bff

Key Takeaways

Key takeaways

Securing OAuth 2.0 in the browser alone is NOT possible

A secure BFF keeps tokens out of the browser, which significantly increases security

A Secure BFF reduces the consequences of an attach to session riding but don’t blocks it.

Follow secure coding guidelines to fix XSS in your applications

Further reading

Articles

- OAuth 2.0 for browser-based Apps
- Why avoiding LocalStorage for tokens is the wrong solution
- Securing SPAs using the BFF Pattern (.NET)
- An in-depth look at refresh tokens in the browser
- Comparing the BFF Security architecture with an SPA using a public API.

Presentations

- Additional talks on SPA and API security
- Introduction to OAuth 2.0 and OpenID Connect By Philippe De Ryck (3h)

Further reading

Videos

- The insecurity of OAuth 2.0 in frontends - Philippe de Ryck
- Securing SPAs and Blazor Applications using the BFF (Backend for Frontend) Pattern - Dominick Baier

Tools

- OAuth 2.0 Playground

Specifications

- Map of OAuth 2.0 Specs
- OAuth 2.0 for Browser-Based Apps