

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:** For SPA apps
- **Authorization Code flow:** For mobile and server based web apps
- **Authorization Code with PKCE:** New standard for mobile and web apps
- **Client credentials flow:** For server to server auth

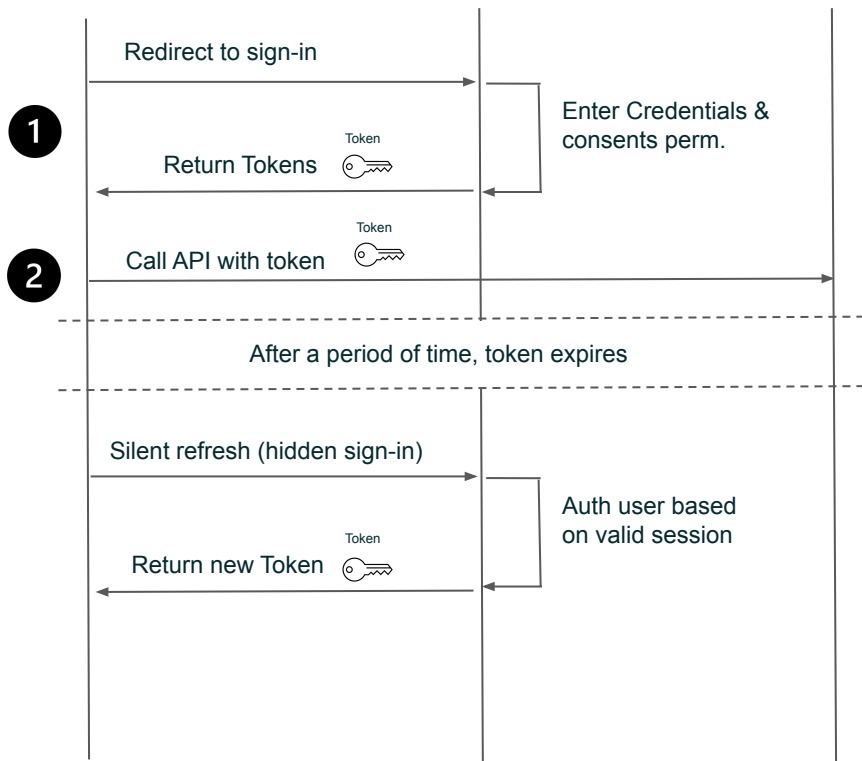
Browser



Auth Server



API Server



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.

(some auth servers prohibit this flow entirely)

Browser



Auth Server

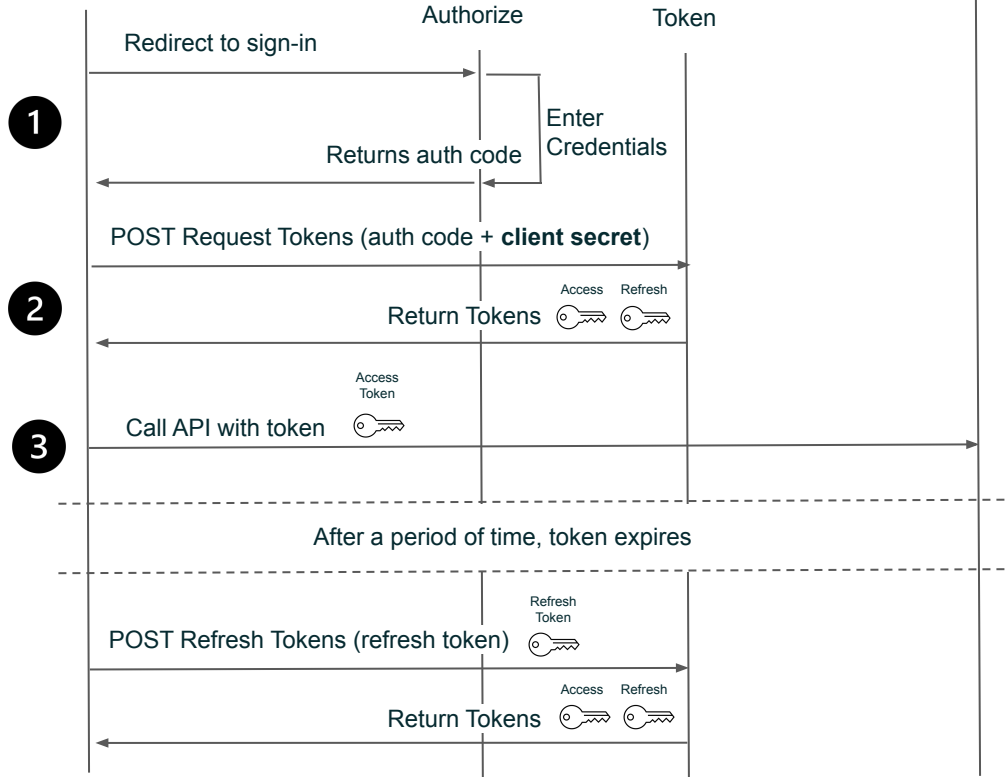


API Server



Authorization Code Flow

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



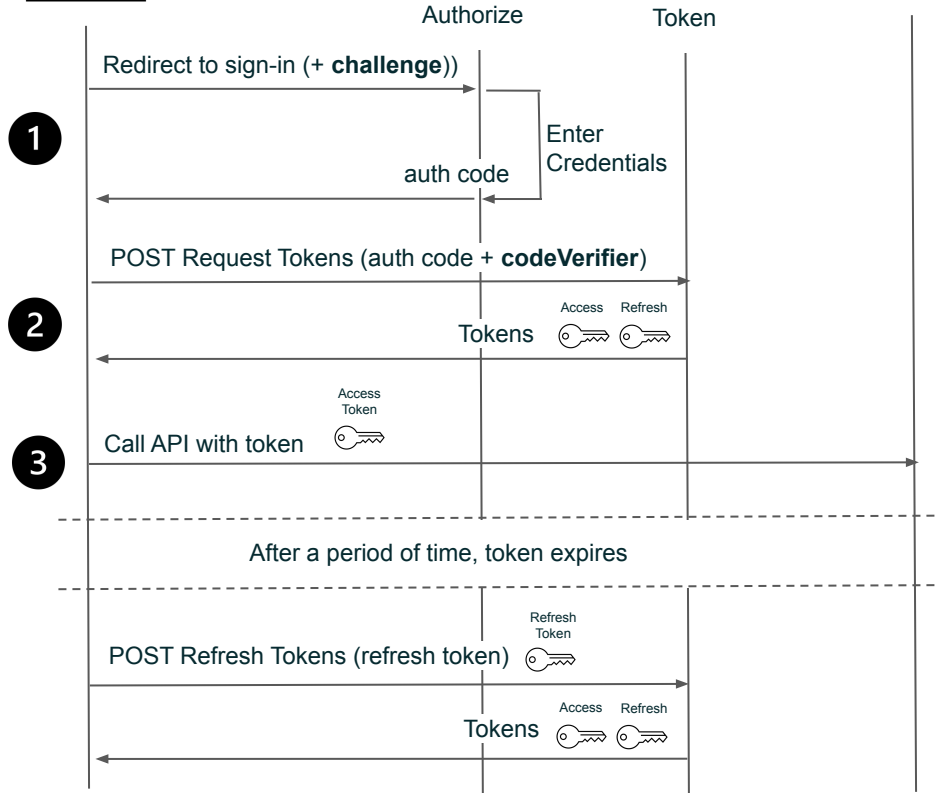
Browser



Auth Server



Your Server



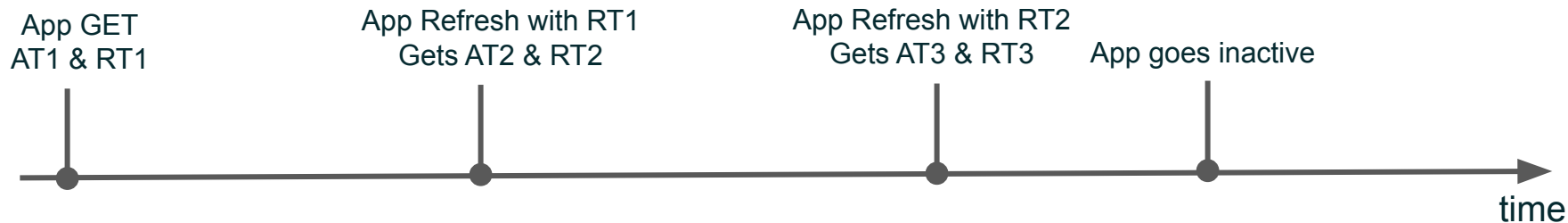
Authorization Code Flow with PKCE

(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))
```

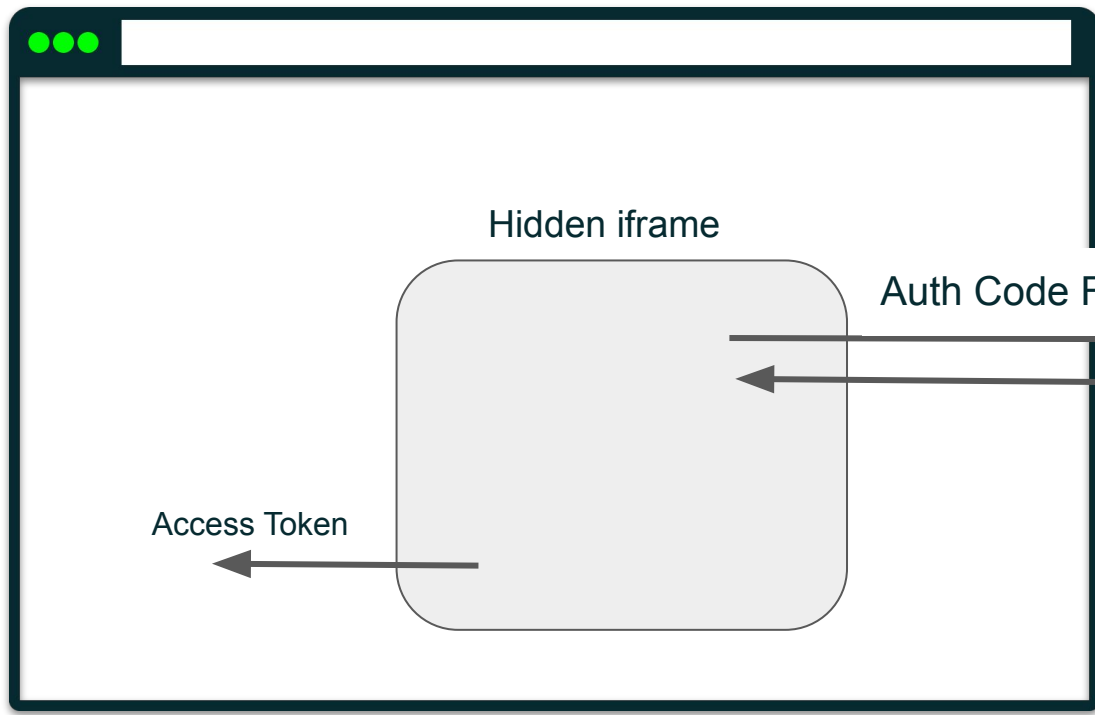
Refresh Token Rotation



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

Silent Refresh

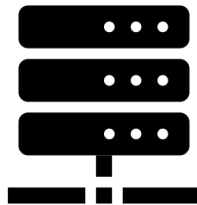
Frontend



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

Auth Code Flow

Auth Server



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

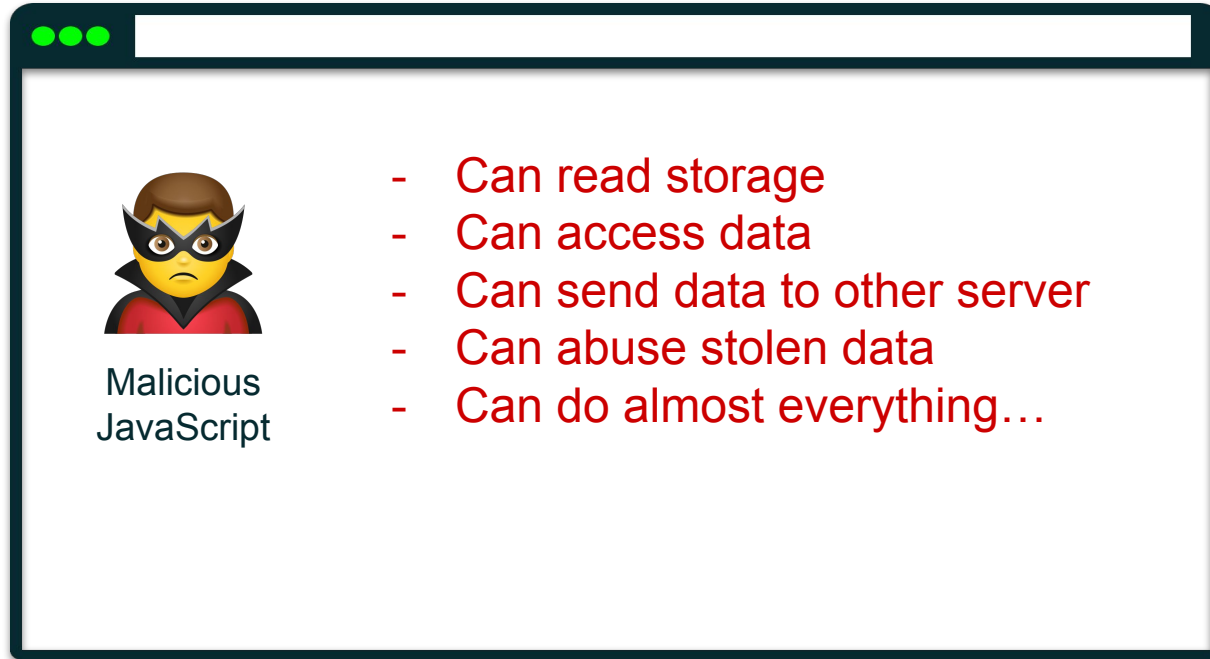
In the **draft of OAuth 2.1** the use of the **PKCE** extension for native apps has been **recommended** to all kinds of OAuth clients, including web applications and other confidential clients in order **to avoid** malicious browser extensions to perform OAuth 2.0 **code injection attack**

Attacking OAuth 2.0 in frontend



Malicious JavaScript (XSS)

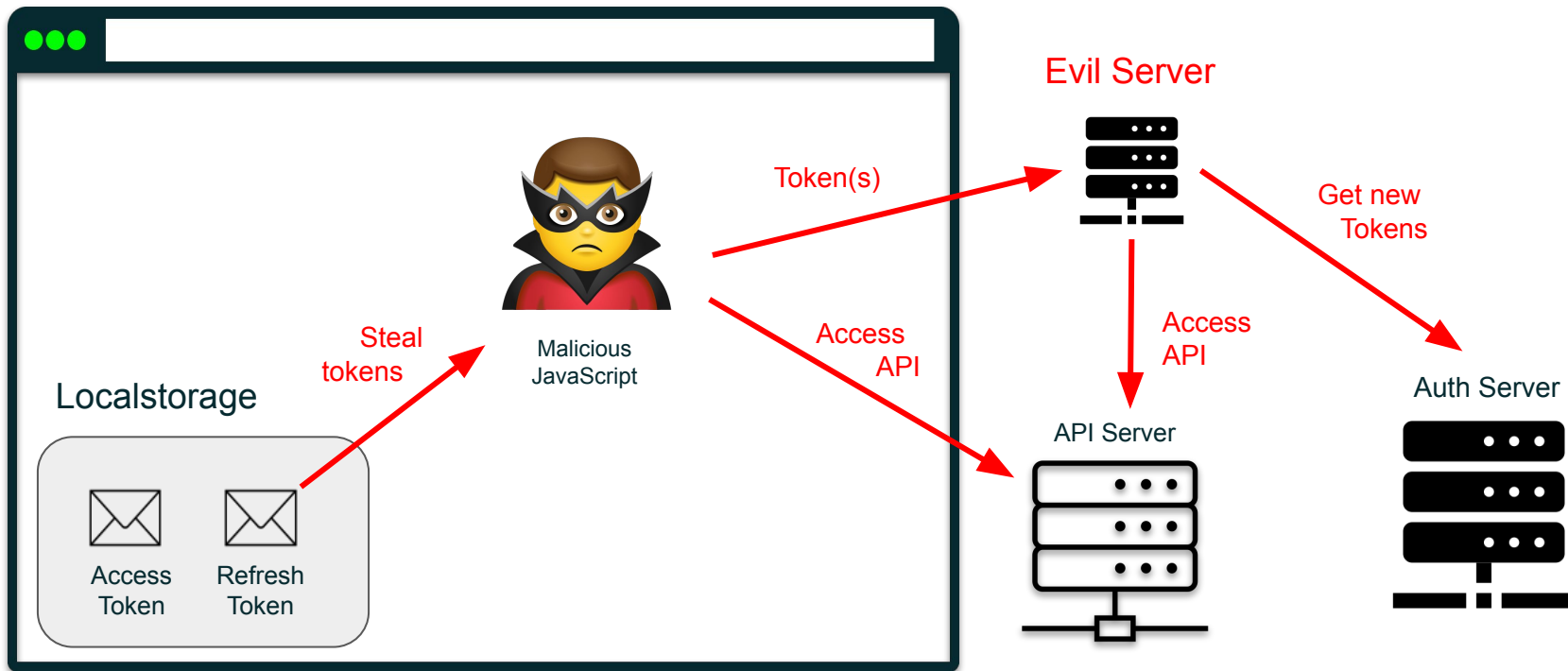
Frontend



The browser cannot distinguish the malicious code from legitimate code.

Stealing tokens

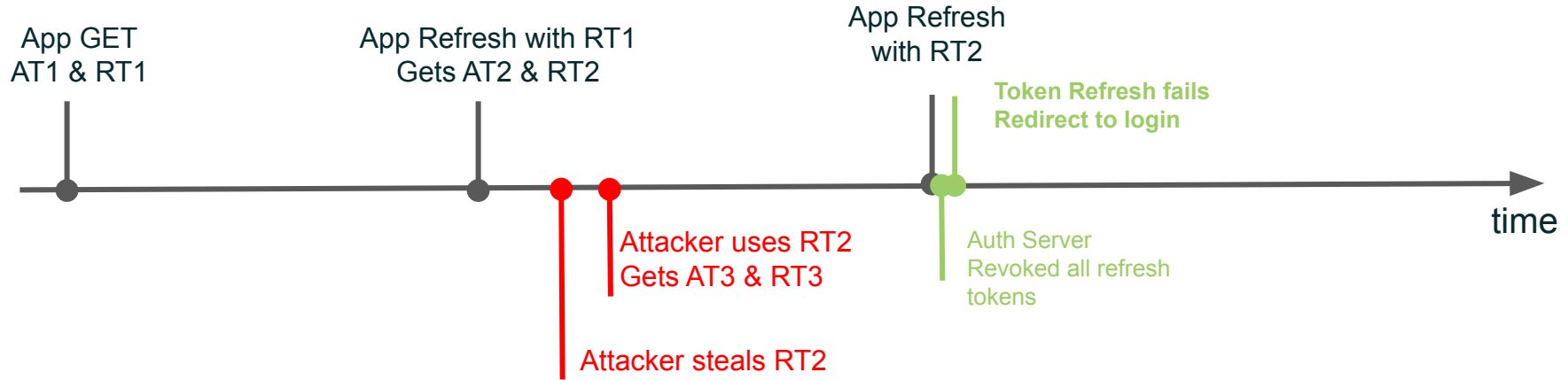
Frontend



Hacker can steal tokens and access api

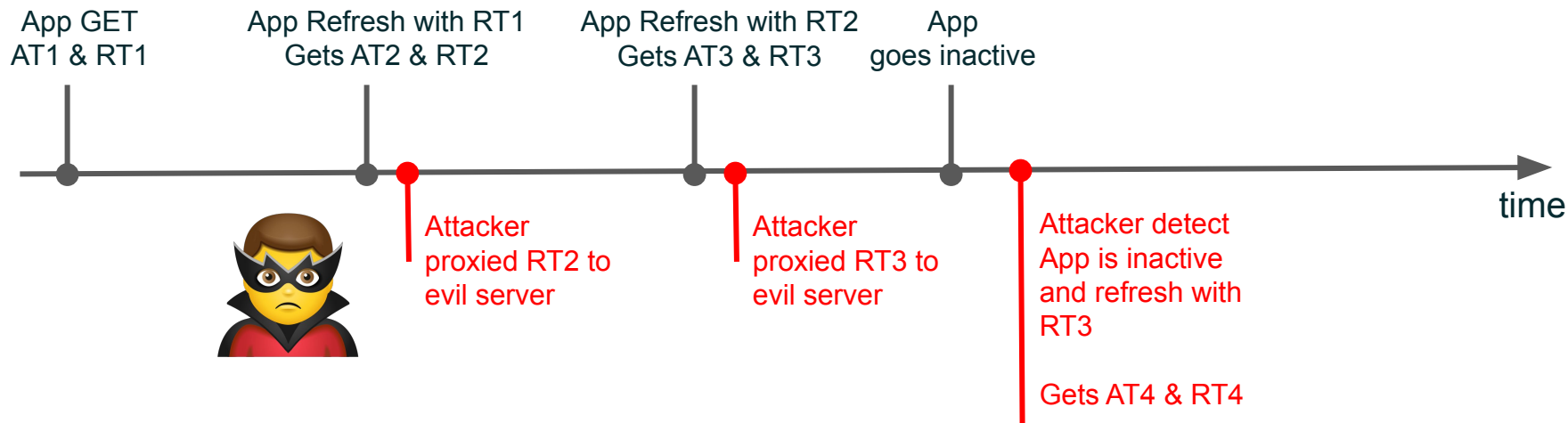


Refresh Token Rotation - Stealing Token



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

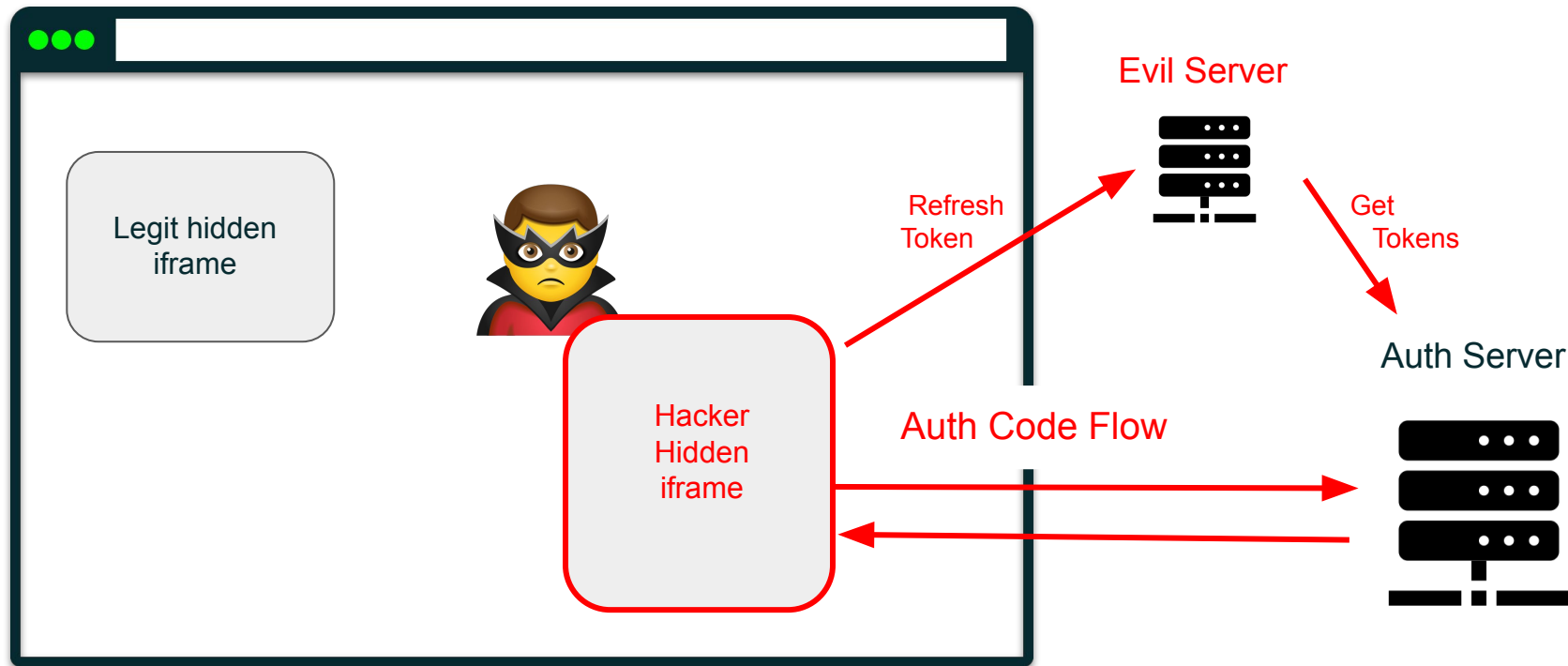
Bypassing Refresh Token Rotation Protection



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

Frontend



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

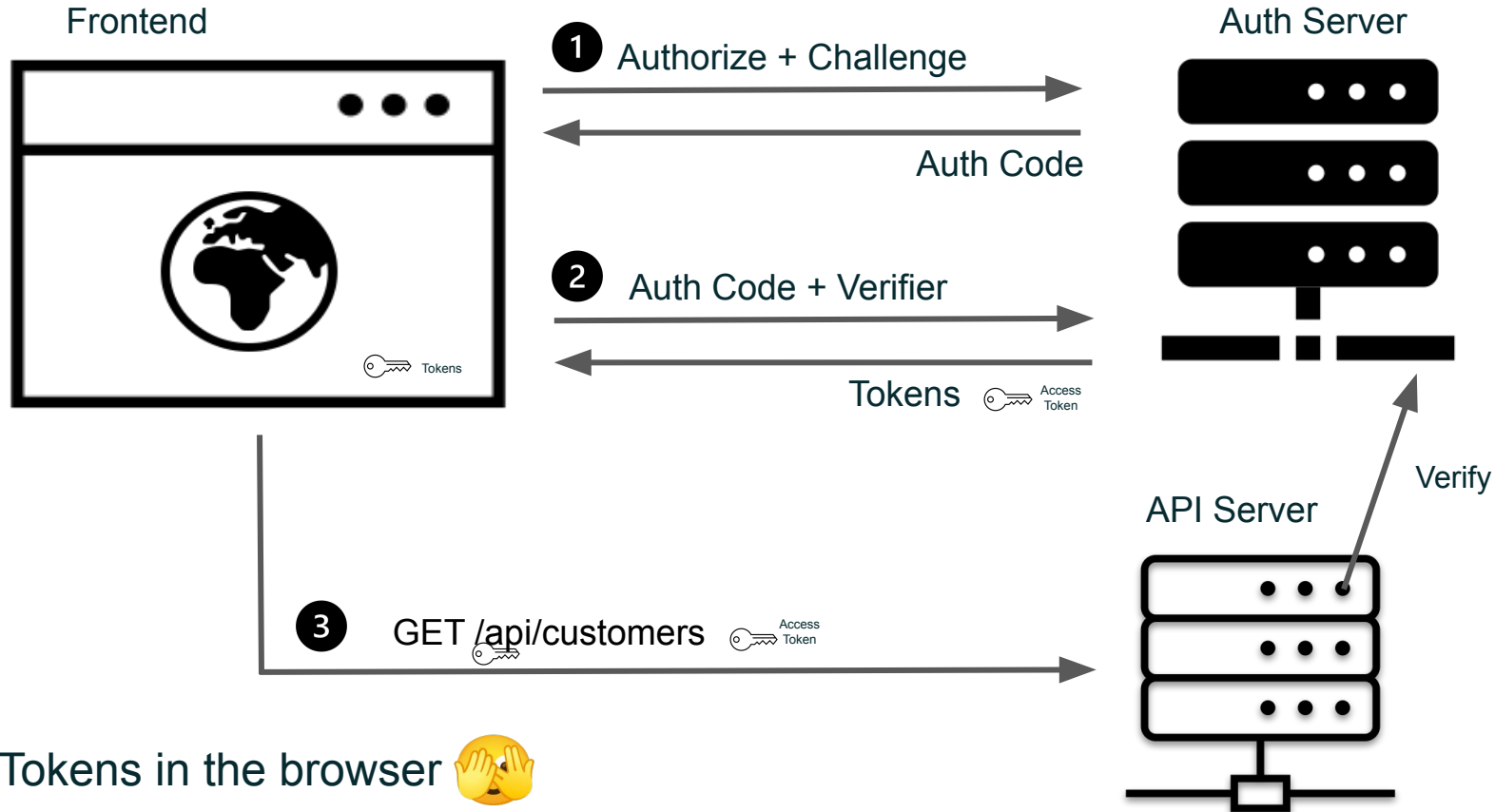
Securing solely frontend applications with OAuth 2.0 is insufficient for comprehensive protection.

XSS and the is game over

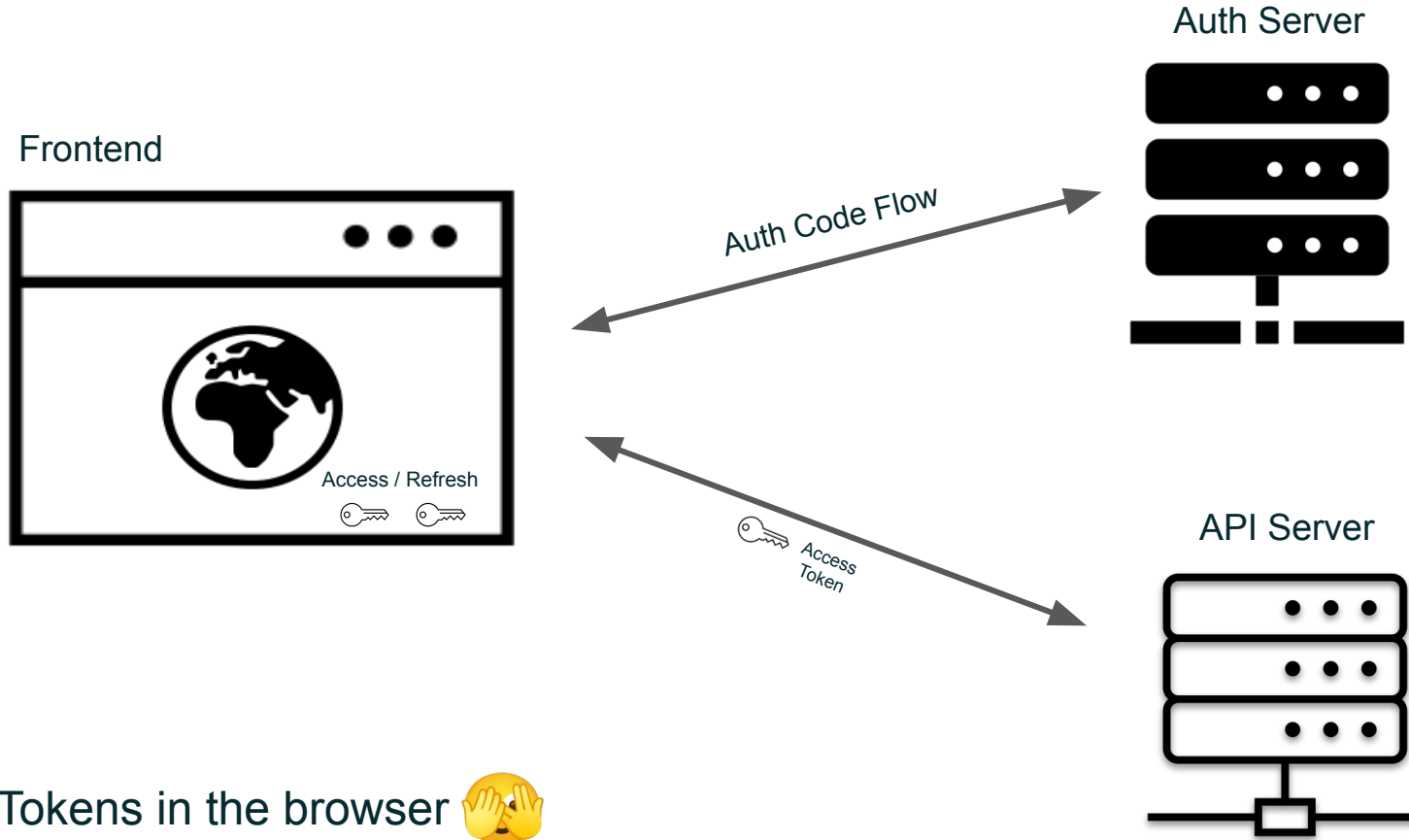
The backend-for-frontend (BFF) pattern



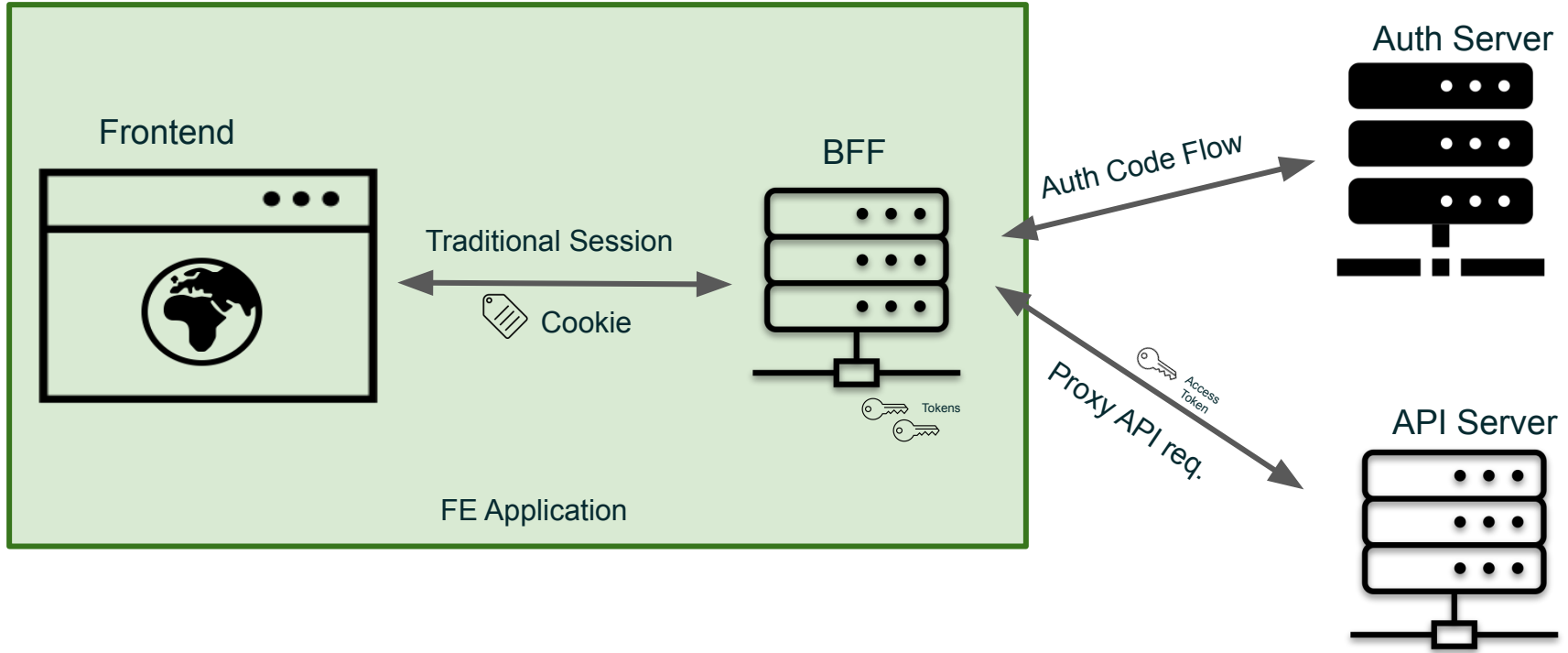
Authorization Code in Front-end (not safe)



Authorization Code in Front-end

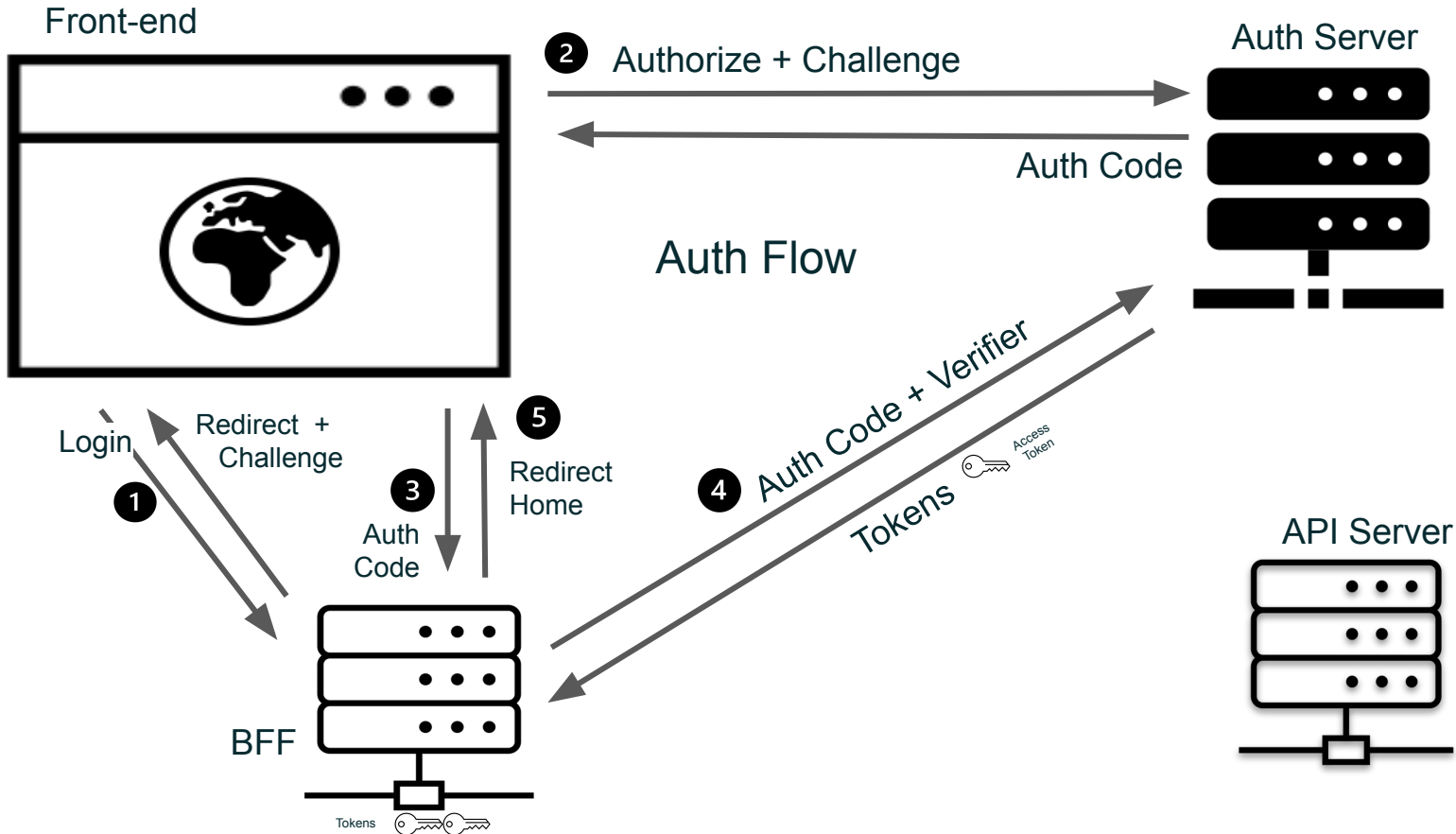


Authorization Code with Backend for Frontend (BFF)

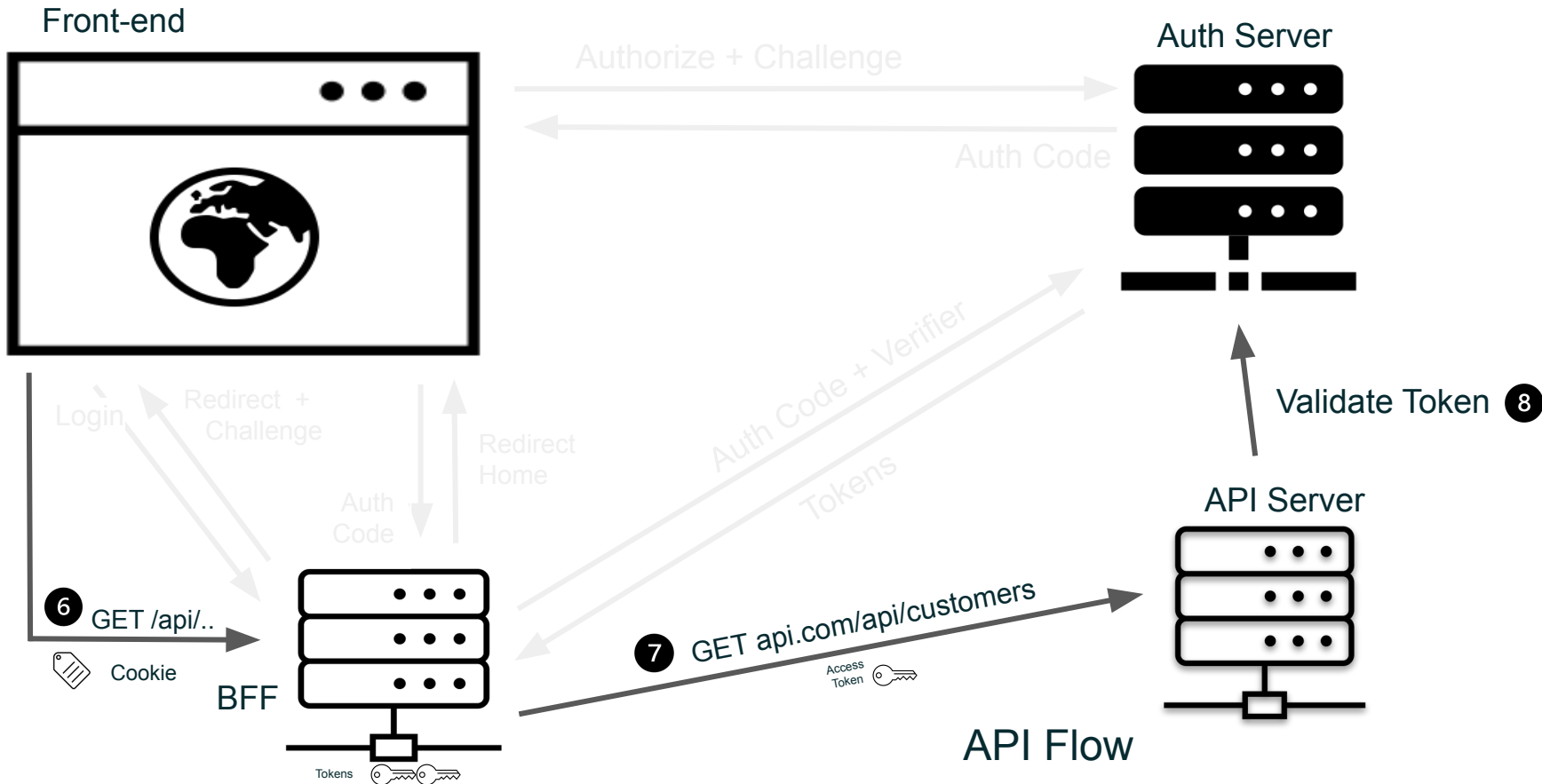


Tokens in the BE

Authorization Code with BFF

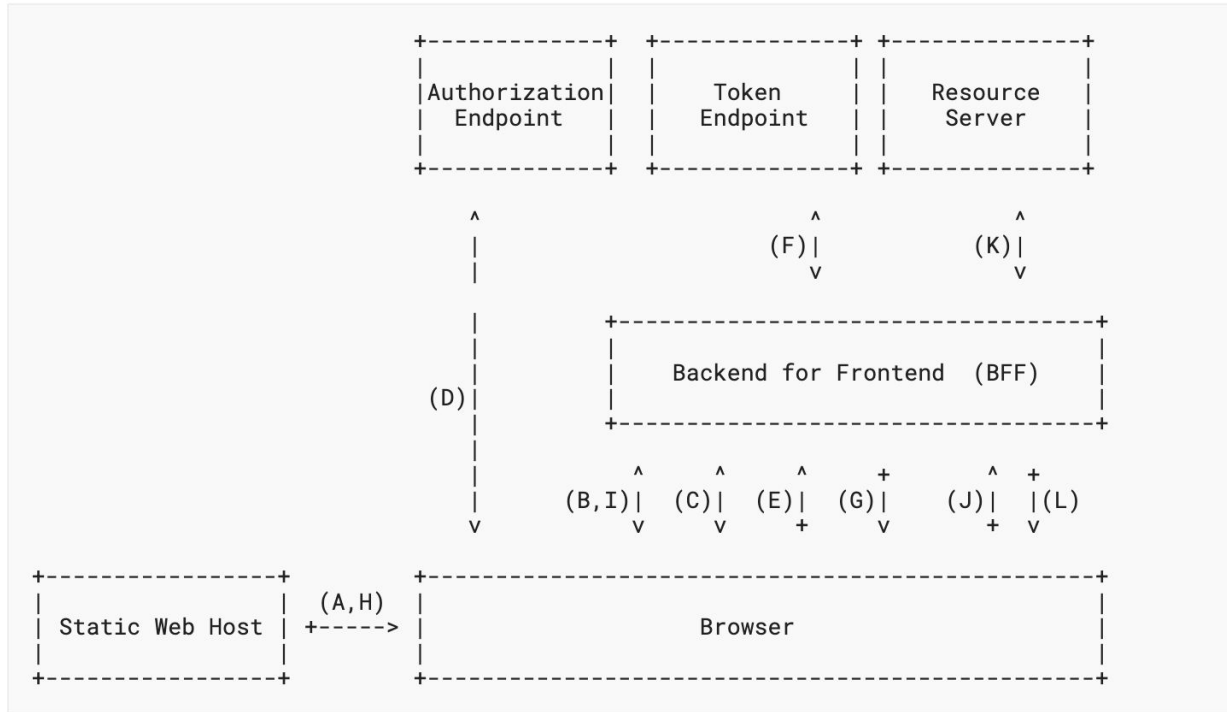


Authorization Code with BFF



Specification: OAuth 2.0 for Browser-Based Apps

6.1.1. Application Architecture



BFF Solutions



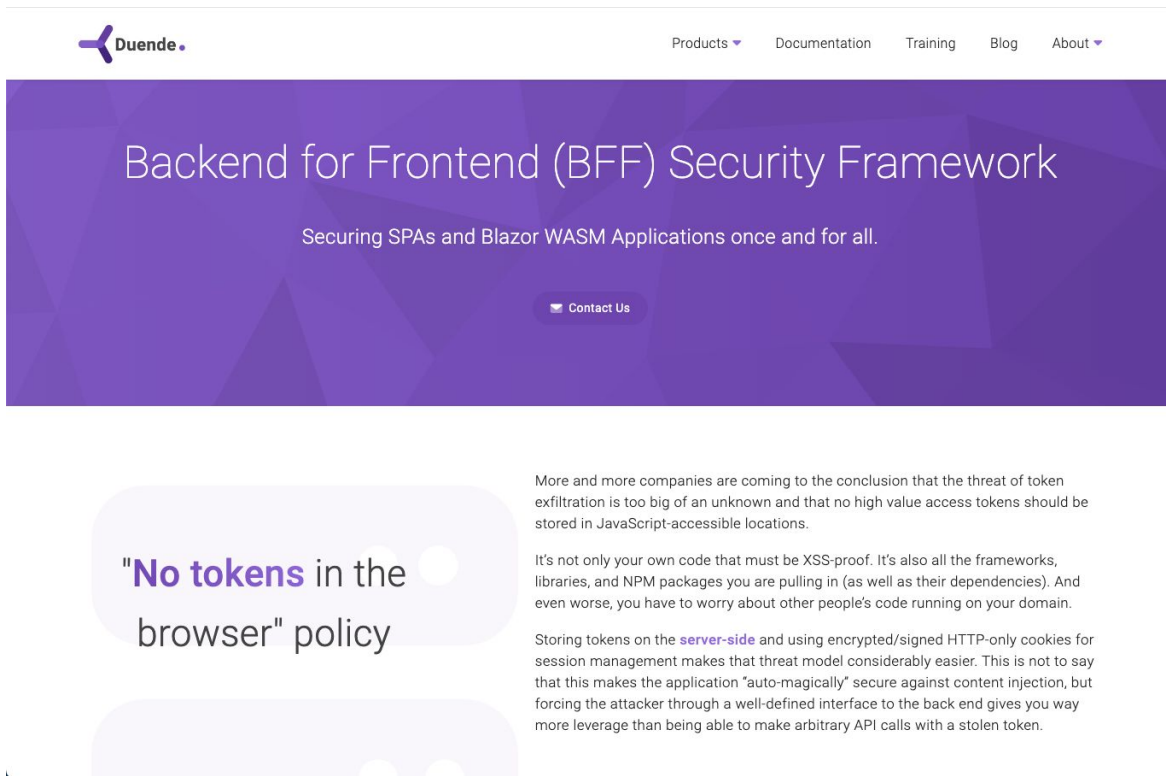
Full stack JavaScript frameworks

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](#), SvelteKit & 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



The image shows the landing page for the Duende Backend for Frontend (BFF) Security Framework. The page has a purple header with the Duende logo on the left and navigation links for Products, Documentation, Training, Blog, and About on the right. The main content area has a purple background with a geometric pattern. It features the title "Backend for Frontend (BFF) Security Framework" and the tagline "Securing SPAs and Blazor WASM Applications once and for all." Below this is a "Contact Us" button. The lower section has a light purple background with a quote on the left and explanatory text on the right.

Duende.

Products ▾ Documentation Training Blog About ▾

Backend for Frontend (BFF) Security Framework

Securing SPAs and Blazor WASM Applications once and for all.

✉ Contact Us

"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.

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 additional 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](#)