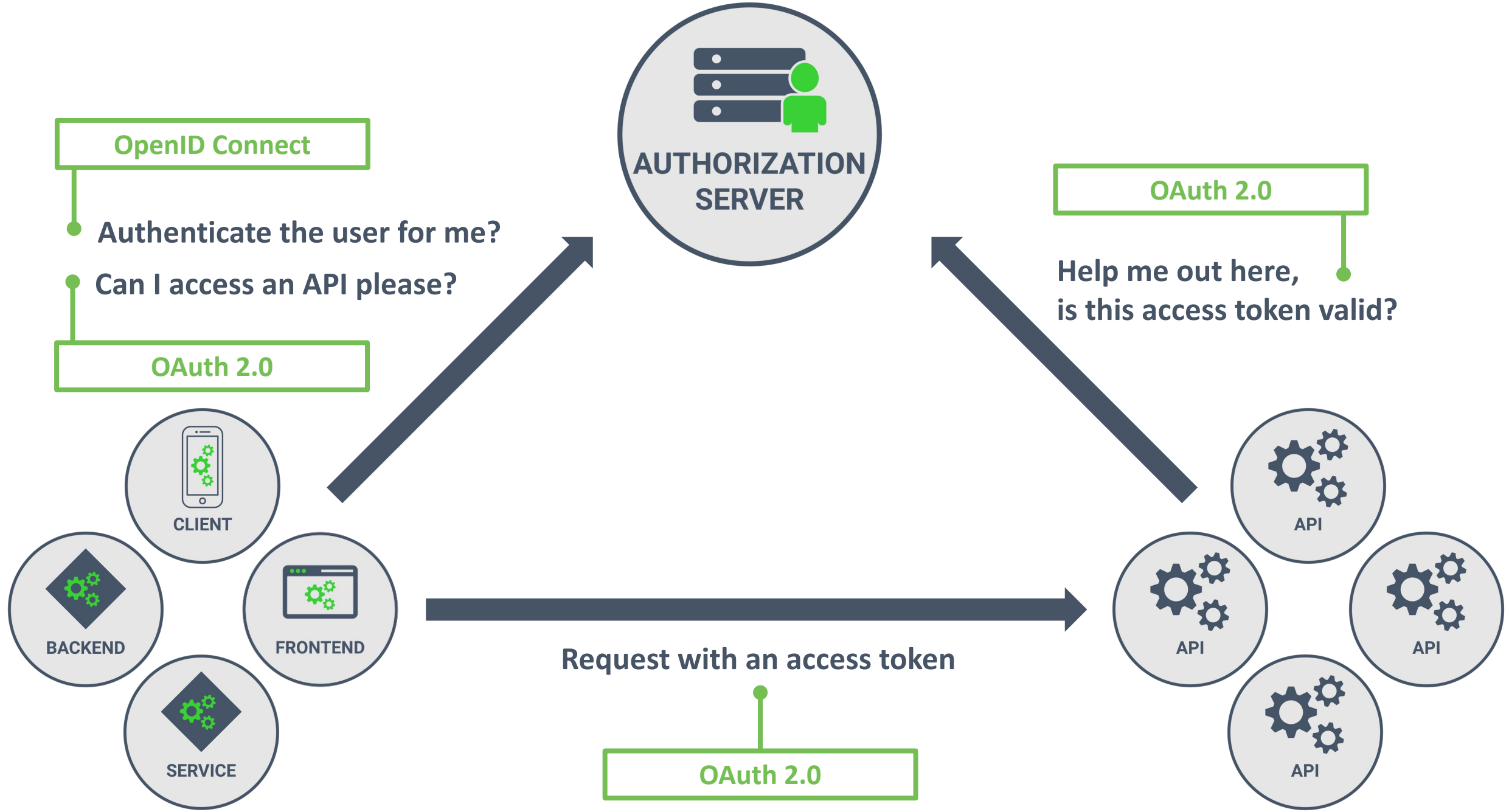
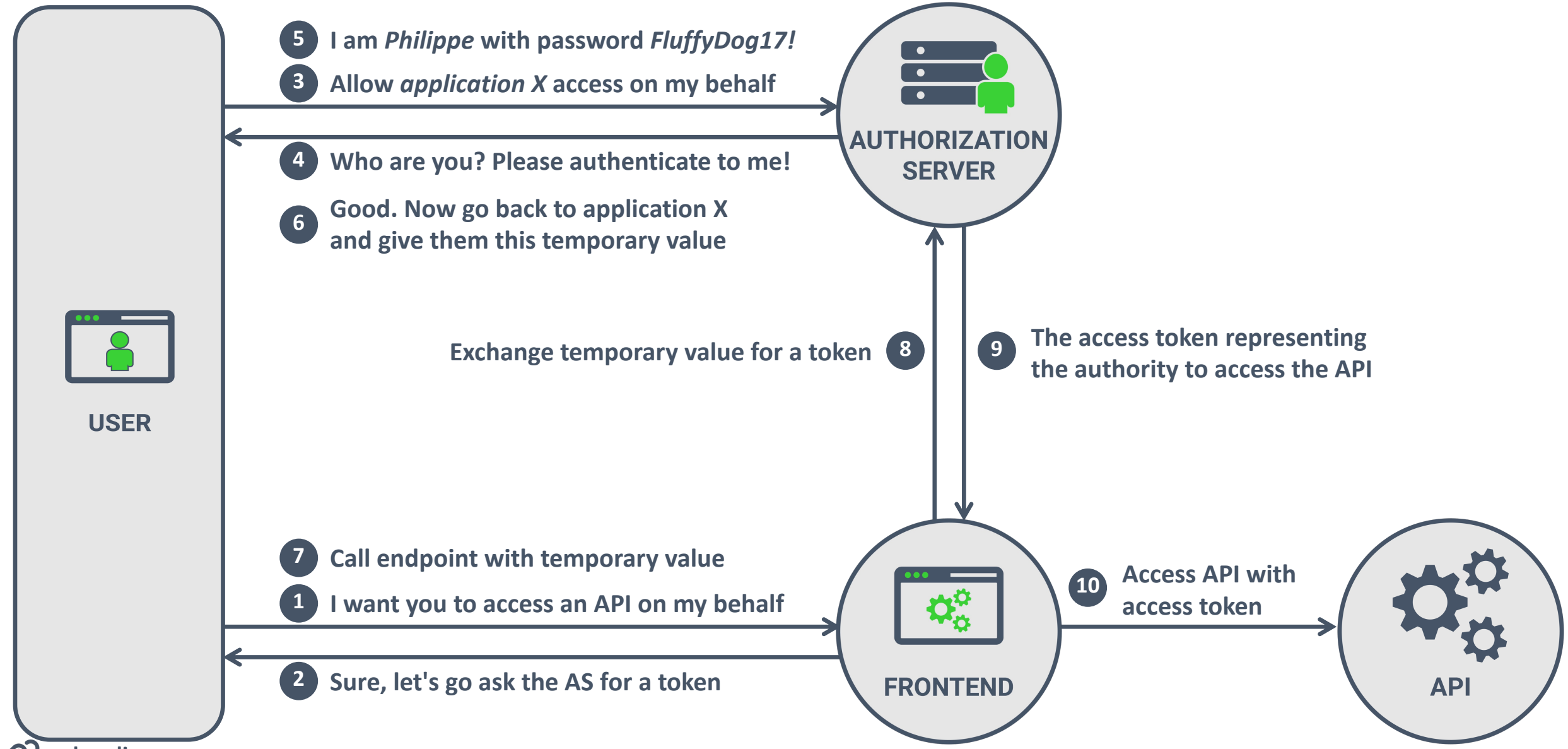
Breaking and securing OAuth 2.0 in frontends

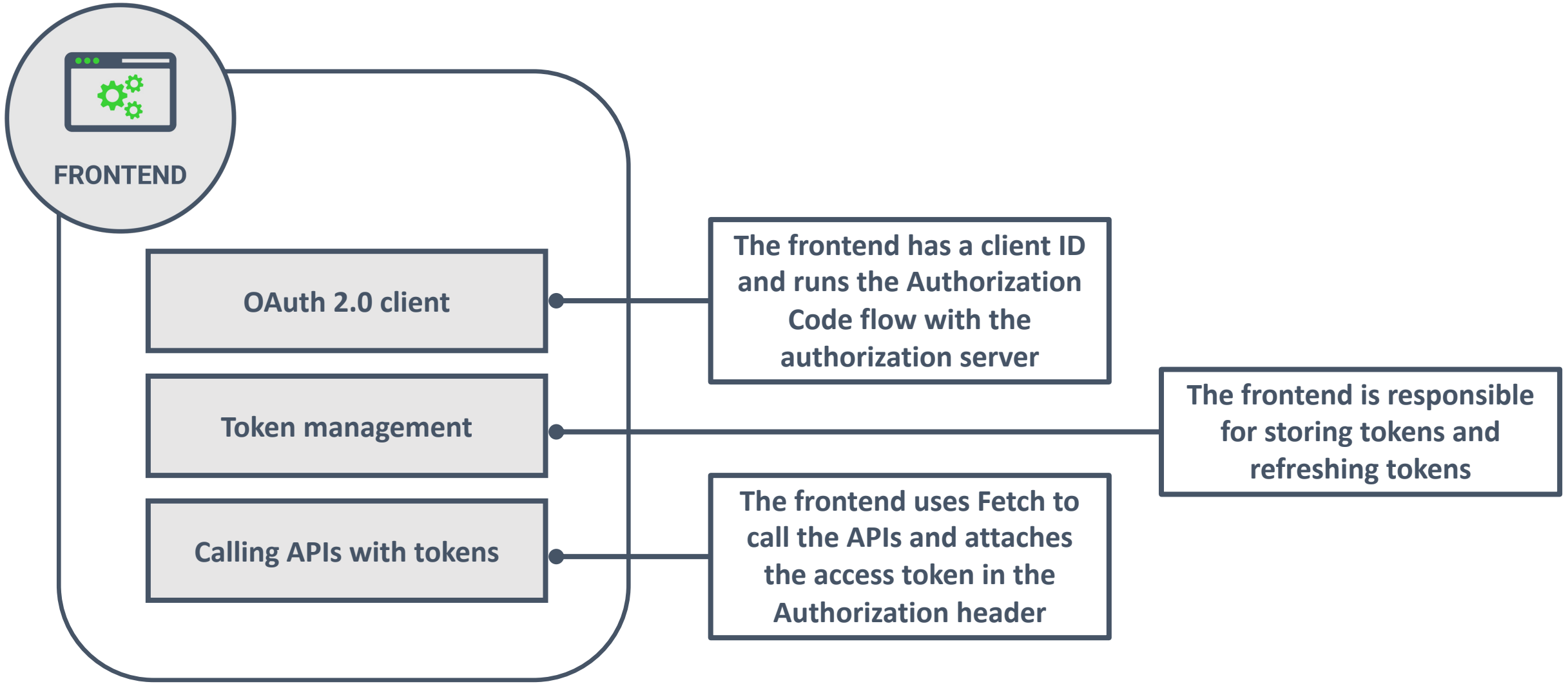
Dr. Philippe De Ryck

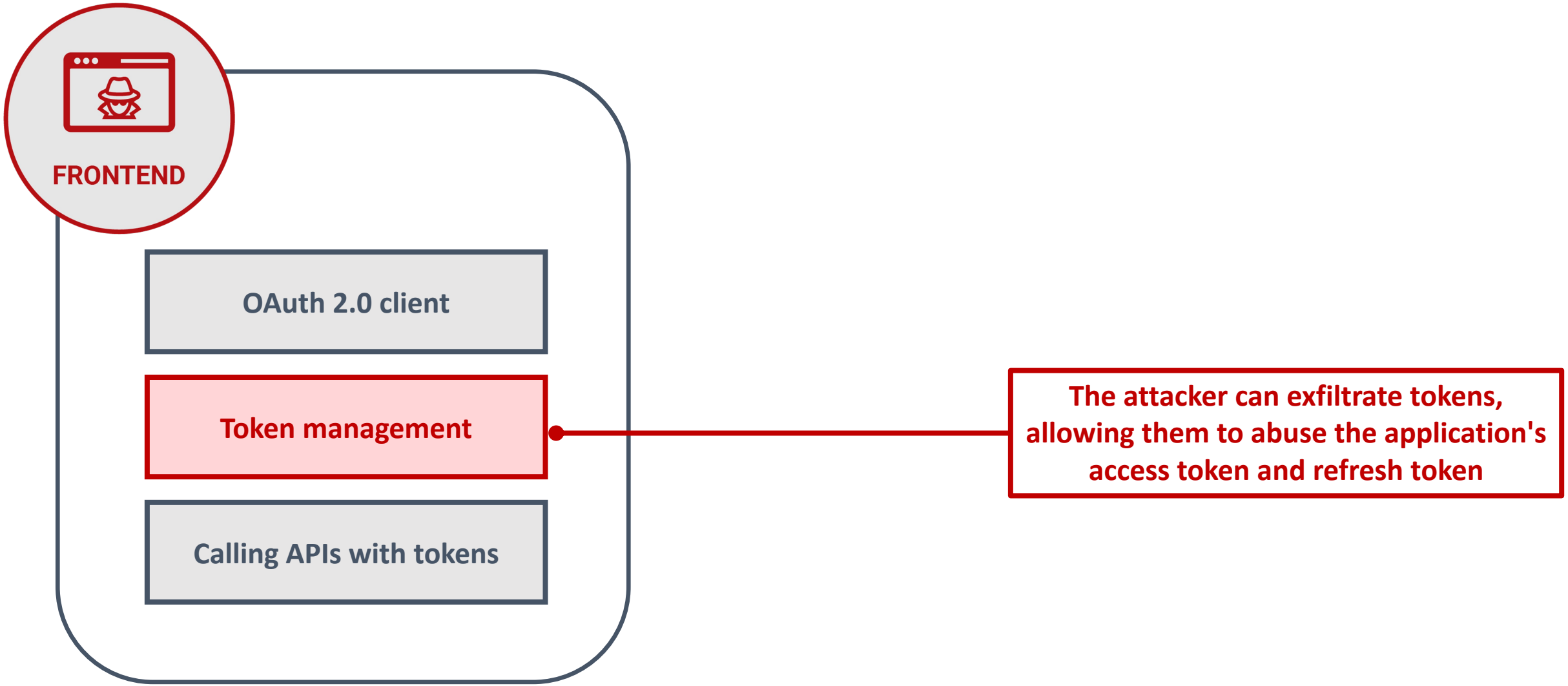
https://Pragmatic Web Security.com

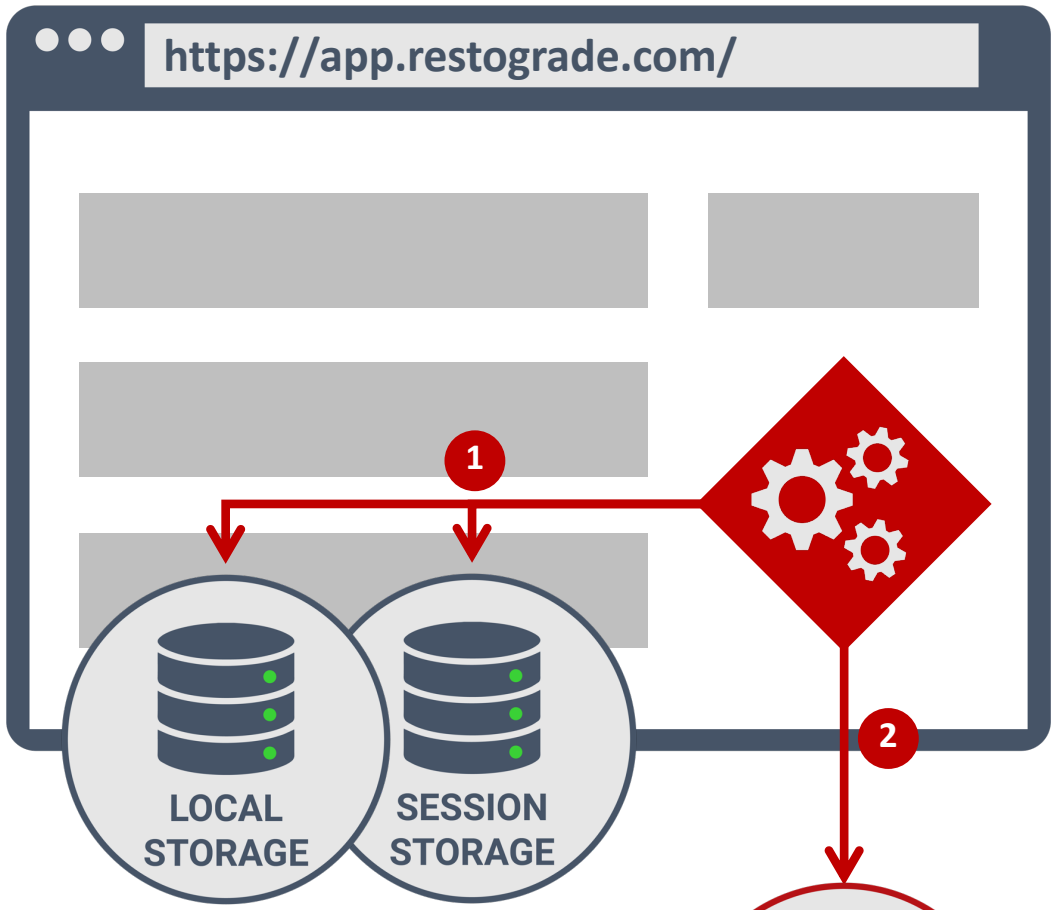

OpenID Connect
Authenticate the user for me?
Can I access an API please?
OAuth 2.0
AUTHORIZATION SERVER
OAuth 2.0
Help me out here,
is this access token valid?
CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Request with an access token
OAuth 2.0

THE CONCEPT OF THE OAUTH 2.0 AUTHORIZATION CODE FLOW




FRONTEND
OAuth 2.0 client
Token management
Calling APIs with tokens
The frontend has a client ID and runs the Authorization Code flow with the authorization server
The frontend is responsible for storing tokens and refreshing tokens
The frontend uses Fetch to call the APIs and attaches the access token in the Authorization header


FRONTEND
OAuth 2.0 client
Token management
Calling APIs with tokens
The attacker can exfiltrate tokens, allowing them to abuse the application's access token and refresh token



1. Request all data from storage or memory
2. Send data to a server controlled by the attacker
3. Abuse the stolen data (access token, refresh token)

- Short-lived access tokens reduce the impact of stolen access tokens
- Refresh token rotation prevents re-use of stolen refresh tokens

This pattern is a highly common practice for implementing OAuth 2.0 in frontends

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-browser-based-apps-22
Published: 17 January 2025
Intended Status: Best Current Practice
Expires: 21 July 2025

A. Parecki
Okta
D. Waite
Ping Identity
P. De Ryck
Pragmatic Web Security

OAuth 2.0 for Browser-Based Applications

Abstract

This specification details the threats, attack consequences, security considerations and best practices that must be taken into account when developing browser-based applications that use OAuth 2.0.

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

I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

SecAppDev

SecAppDev organizer

I help developers with security

- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Expert security advisory services**

https://pdr.online

GRAB A COPY OF THE SLIDES ...

https://pragmaticwebsecurity.com/talks

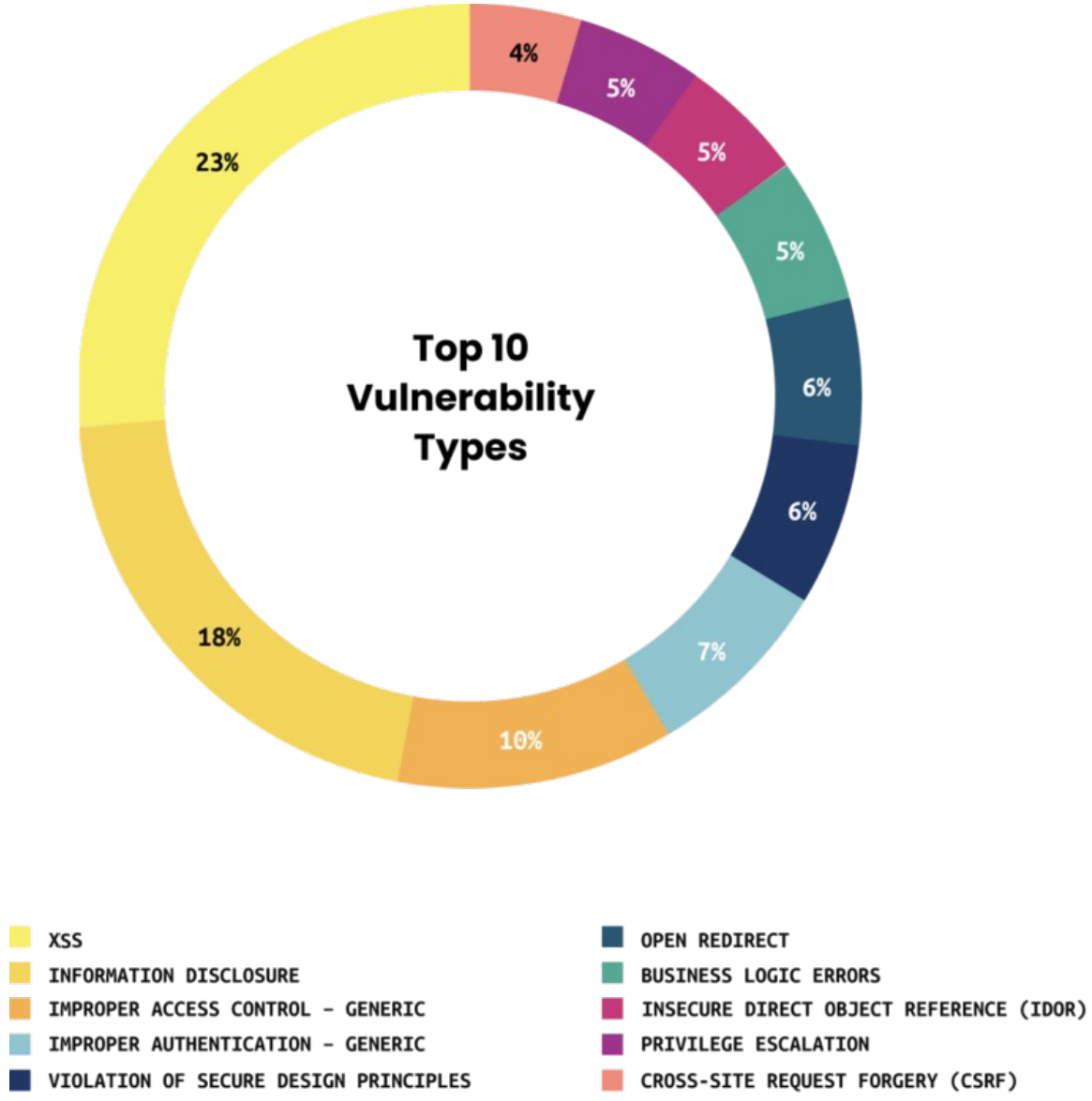
/in/PhilippeDeRyck

@philippederyck.bsky.social

Website icons created by Uniconlabs - Flaticon

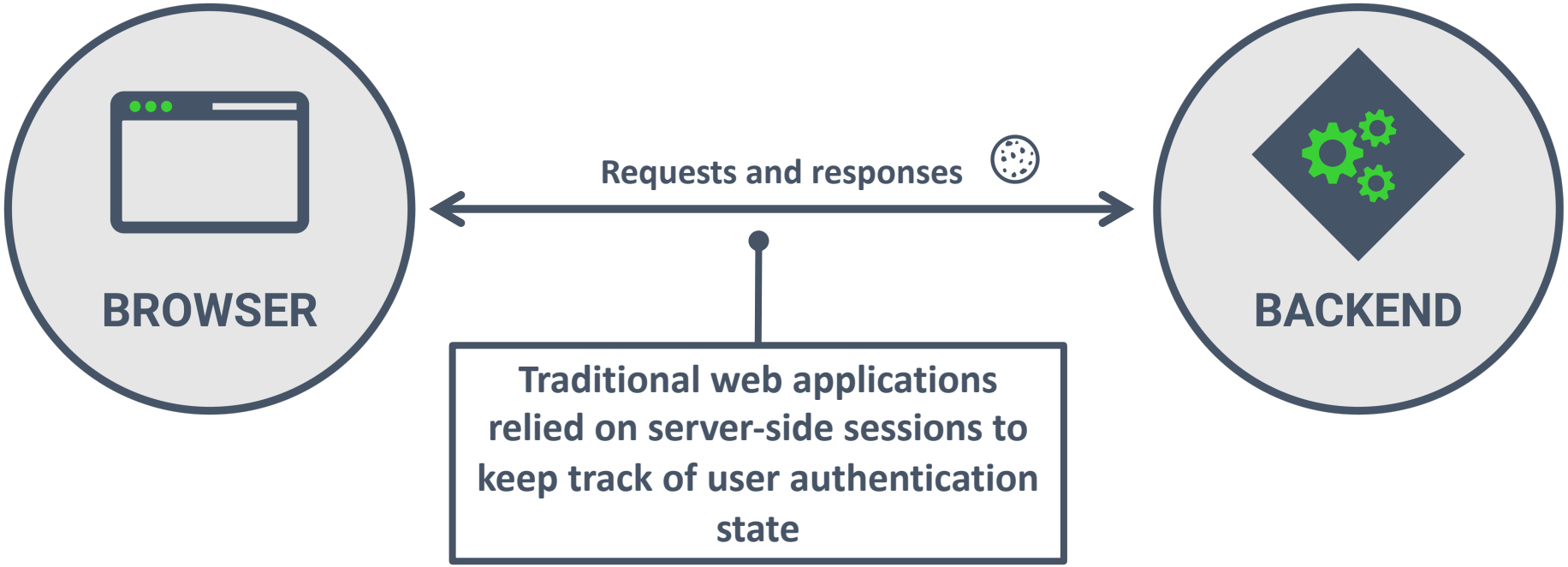
JavaScript

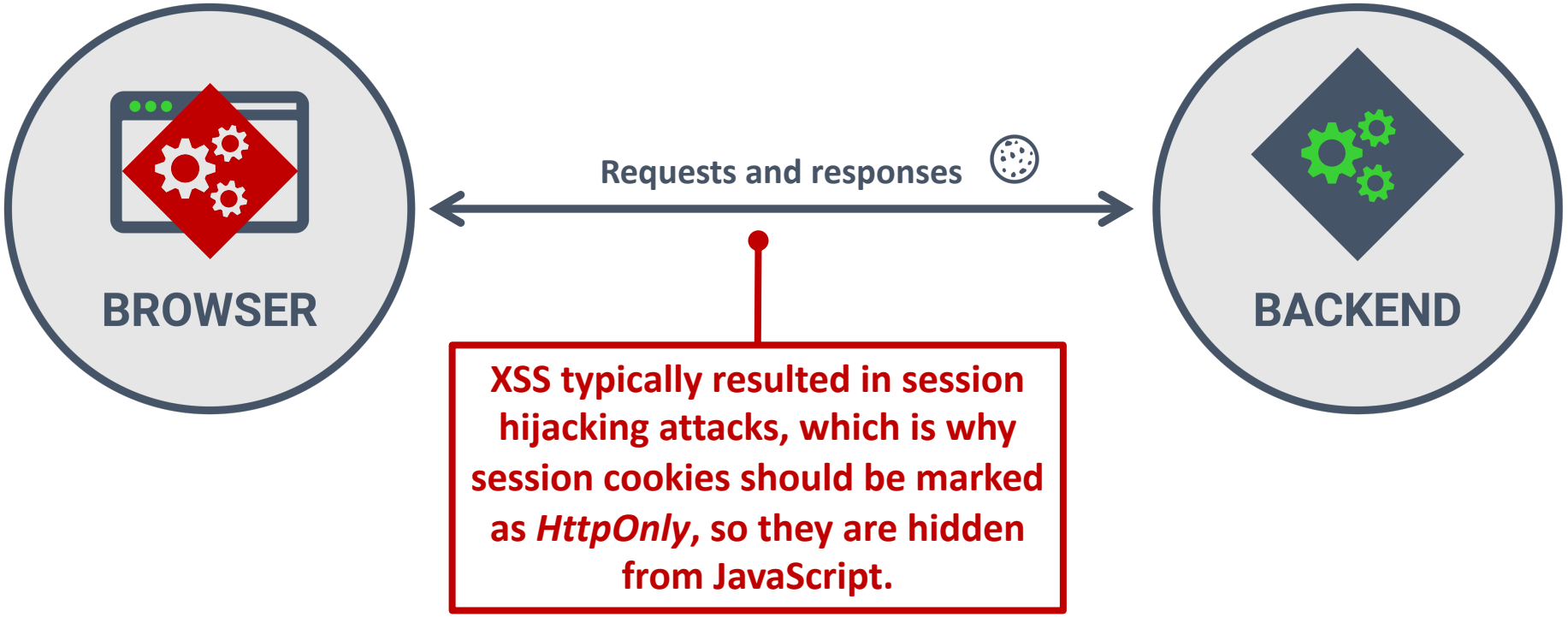
Malicious JavaScript

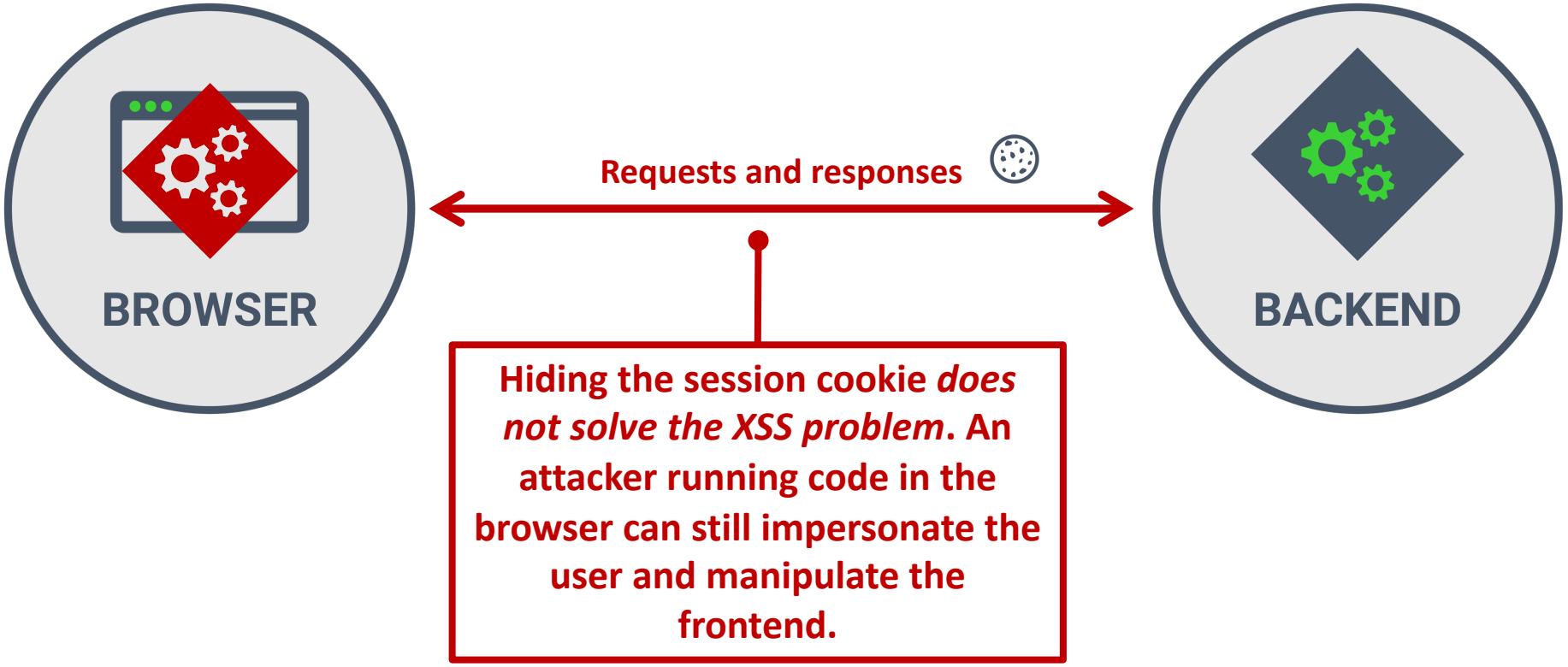


https://www.hackerone.com/top-ten-vulnerabilities

XSS has been a problem for a long time


BROWSER
Requests and responses
BACKEND
Traditional web applications relied on server-side sessions to keep track of user authentication state


BROWSER
Requests and responses
BACKEND
XSS typically resulted in session hijacking attacks, which is why session cookies should be marked as *HttpOnly*, so they are hidden from JavaScript.

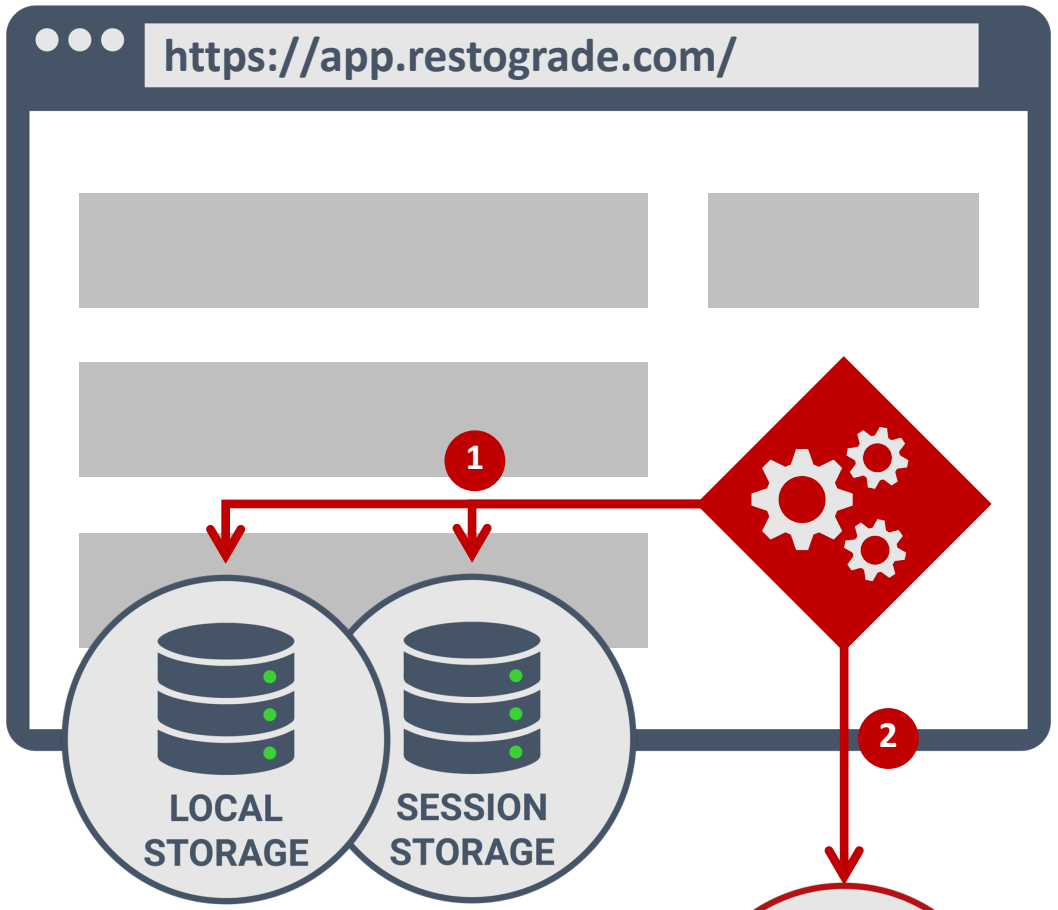

BROWSER
Requests and responses
BACKEND
Hiding the session cookie *does not solve the XSS problem*. An attacker running code in the browser can still impersonate the user and manipulate the frontend.

XSS HAS ALWAYS BEEN A PROBLEM

Traditional web applications already suffered from XSS, with session hijacking as a common consequence.

Even then there was a misbelief that HttpOnly cookies *addressed the problem. However, once the malicious code runs, the attacker controls the client and can deceive or impersonate the user ...*

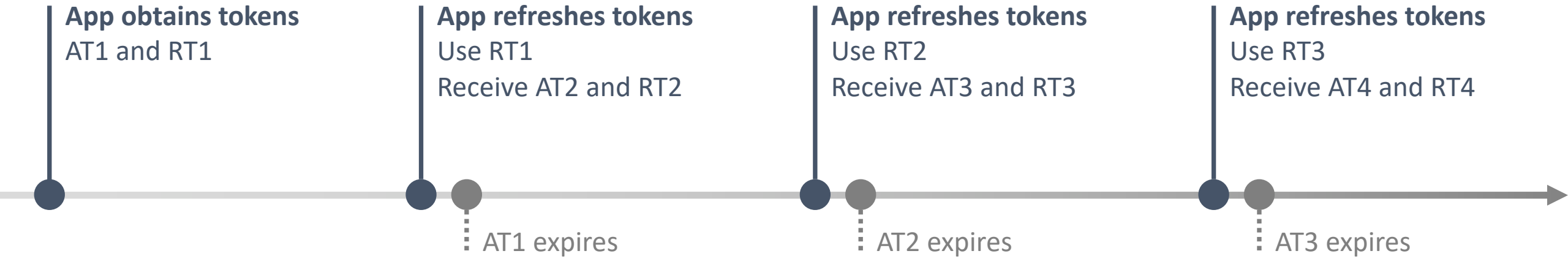

FRONTEND



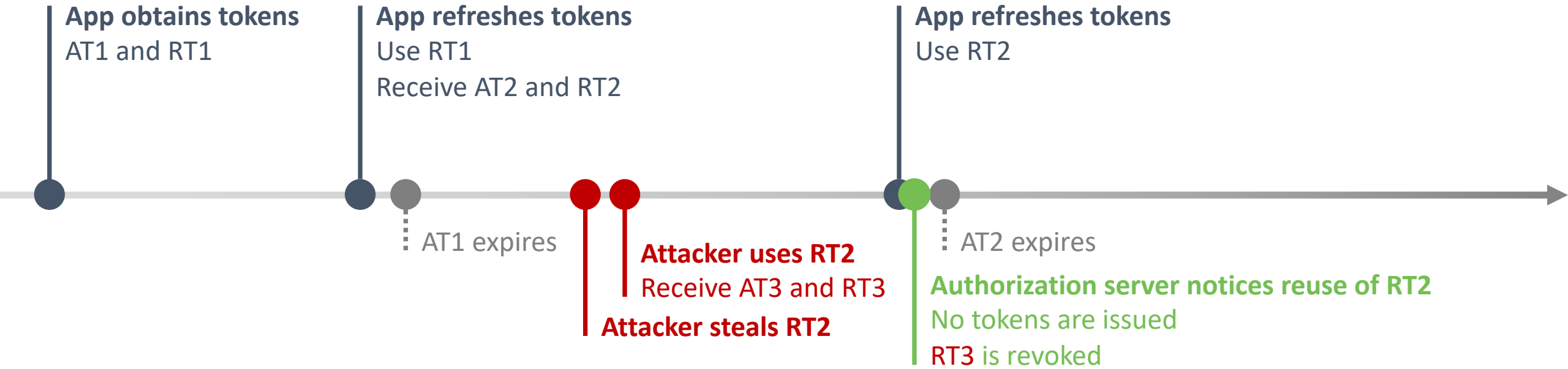
1. Request all data from storage or memory
2. Send data to a server controlled by the attacker
3. Abuse the stolen data (access token, refresh token)

- Short-lived access tokens reduce the impact of stolen access tokens
- Refresh token rotation prevents re-use of stolen refresh tokens

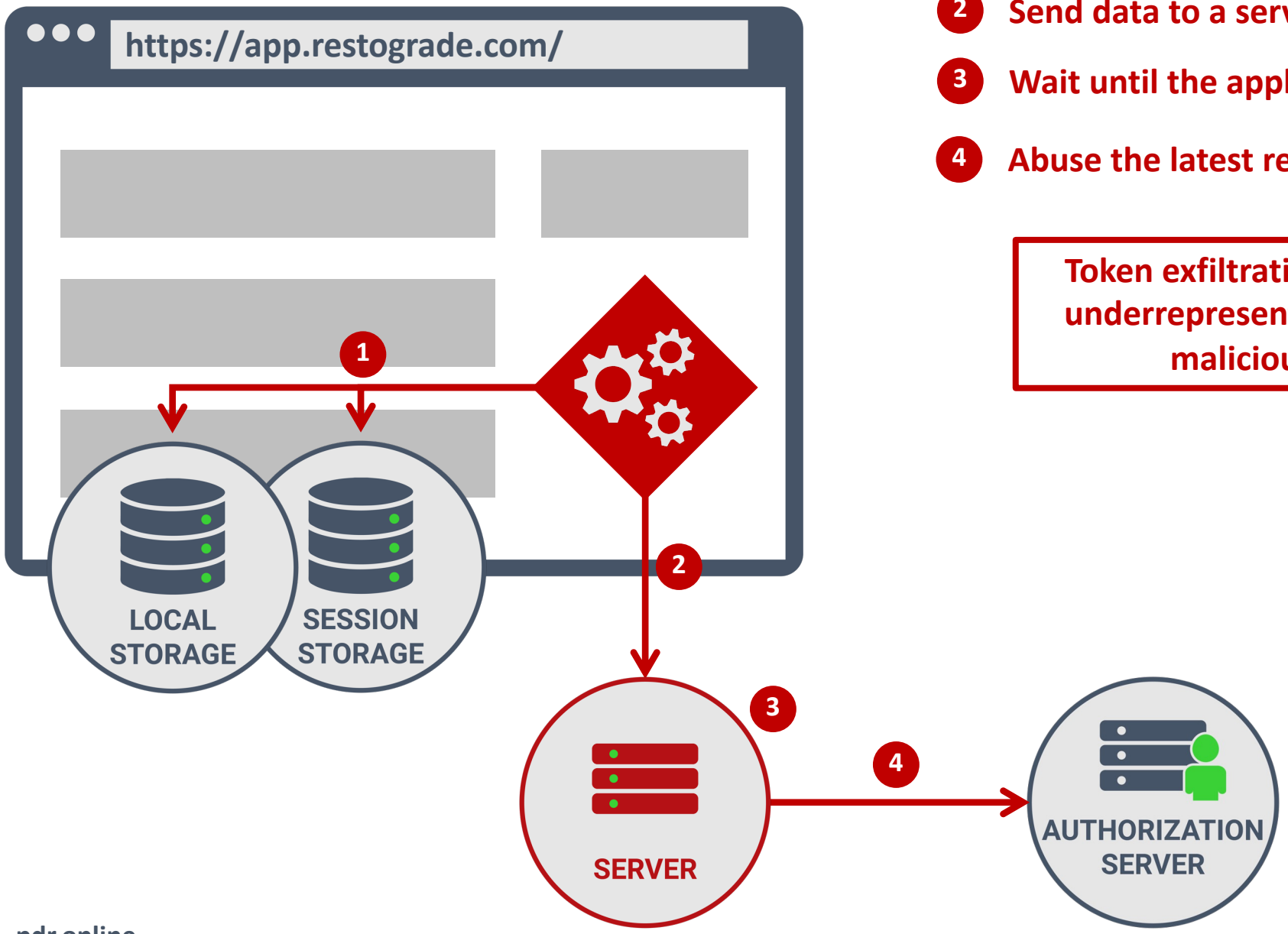
Refresh token rotation



Detecting refresh token abuse



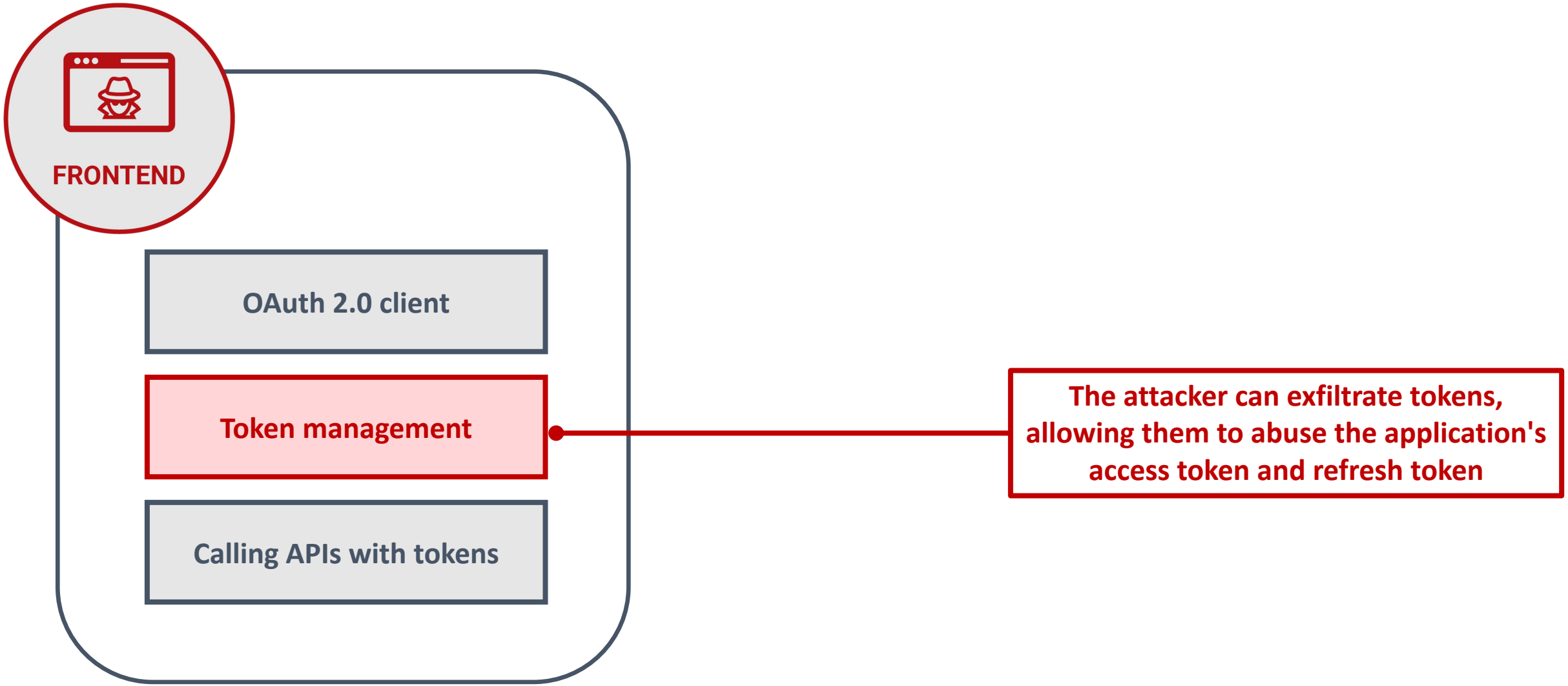
What happens with *Refresh Token Rotation* if a stolen refresh token is never used twice?

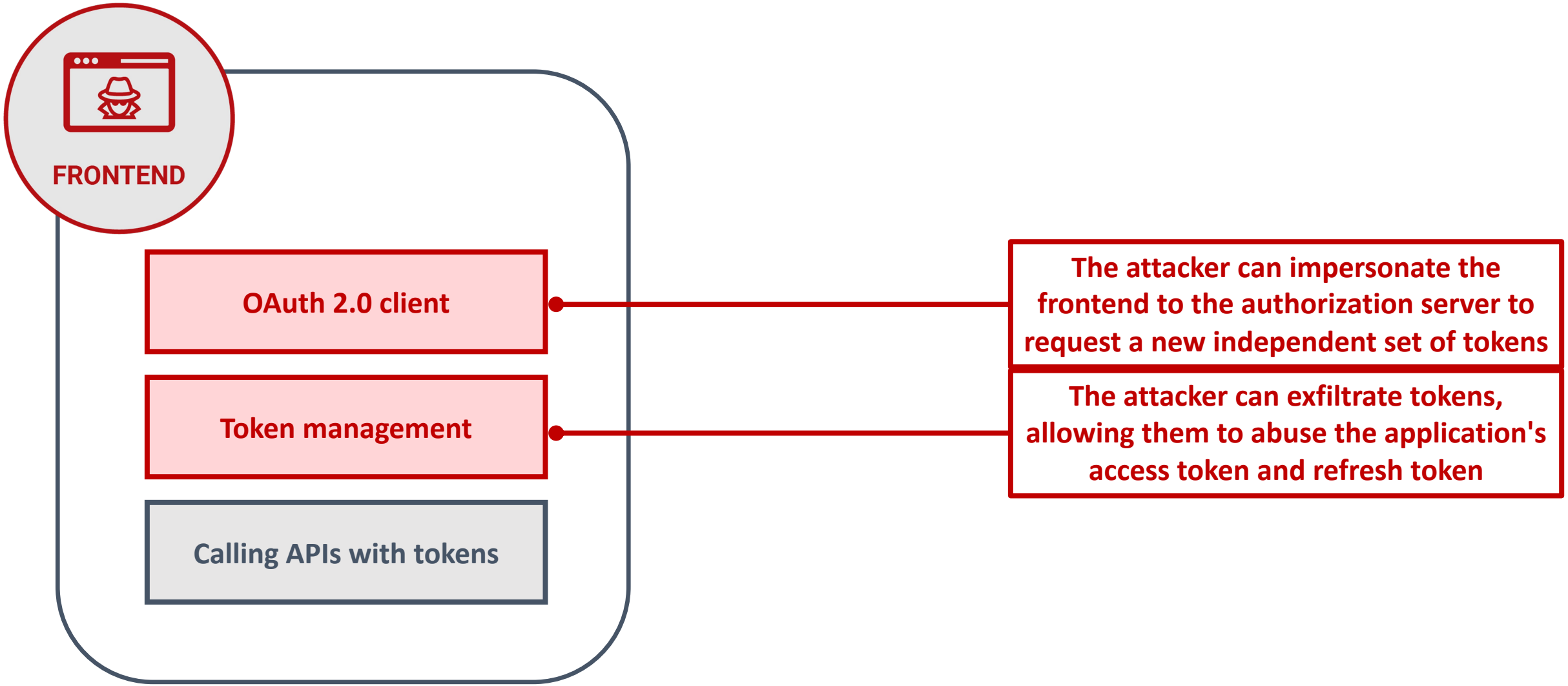


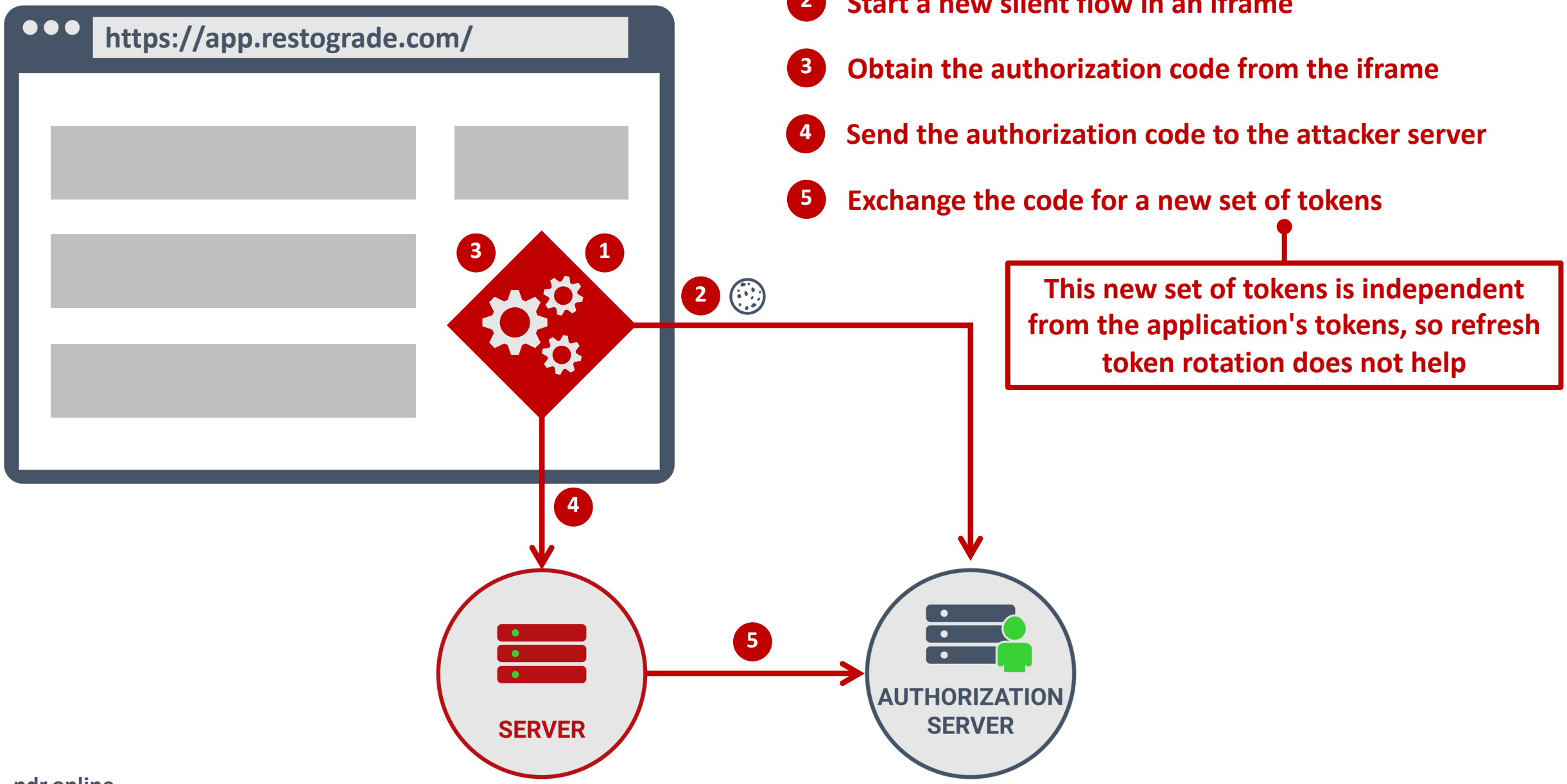
1. Request all data from storage or memory
2. Send data to a server controlled by the attacker
3. Wait until the application goes offline
4. Abuse the latest refresh token

Token exfiltration attacks severely underrepresent the capabilities of malicious JavaScript

Sidestepping refresh token rotation


FRONTEND
OAuth 2.0 client
Token management
Calling APIs with tokens
The attacker can exfiltrate tokens, allowing them to abuse the application's access token and refresh token


FRONTEND
OAuth 2.0 client
Token management
Calling APIs with tokens
The attacker can impersonate the frontend to the authorization server to request a new independent set of tokens
The attacker can exfiltrate tokens, allowing them to abuse the application's access token and refresh token


1 Setup a handler to receive a code from an iframe
2 Start a new silent flow in an iframe
3 Obtain the authorization code from the iframe
4 Send the authorization code to the attacker server
5 Exchange the code for a new set of tokens
This new set of tokens is independent from the application's tokens, so refresh token rotation does not help
https://app.restograde.com/
3
1
2
4
5
SERVER
AUTHORIZATION SERVER

Requesting a fresh set of tokens

Additional security measures, such as DPoP do not work either, since the attacker can provide their own DPoP proofs

You cannot secure browser-only flows

The security of OAuth 2.0 flows in the browser relies on the integrity of the frontend application and its origin (redirect URI).

When the attacker controls that origin, it's game over. Even proof-of-possession mechanisms cannot save you.

Threats to frontend OAuth 2.0 clients

Attack scenario	Example	Duration of attack
Single-execution token theft	One-time payload stealing an access token or refresh token from the running application	**Access tokens:** limited to token lifetime **Refresh tokens:** limited to detection with rotation
Persistent token theft	Continuously stealing access tokens or refresh tokens from the running application	**Access tokens:** as long as the user is online or the application is open **Refresh tokens:** limited to token lifetime after the user goes offline
Acquisition and extraction of new tokens	Running a silent Authorization Code flow to obtain a fresh access token and refresh token	The lifetime of the new refresh token (typically multiple hours or longer)
Proxying requests via the user's browser	Triggering API calls from within the frontend, authenticated by the application's access token	As long as the user is online or application is open

THREATS TO SERVER-SIDE OAUTH 2.0 CLIENTS

Attack scenario	Example	Duration of attack
~~**Single-execution token theft**~~	~~One-time payload stealing an access token or refresh token from the ru~~	~~**Access tokens:** limited to token lifetime~~ ~~**Refresh tokens:** limited to detection~~
~~**Persistent token theft**~~	~~Contin~~ ~~tokens or refresh tokens from the running application~~	~~is online~~ ~~**Refresh tokens:** limited to token lifetime after user goes offline~~
~~**Acquisition and extraction of new tokens**~~	~~Runni~~ ~~Code~~ ~~token~~	~~sh~~ ~~rs or~~
Proxying requests via the user's browser	Trigge fronte applic	

Server-side applications keep access tokens and refresh tokens in server-side storage (e.g., a database). An attacker executing malicious JS in the browser cannot access server-side token storage.

Server-side OAuth 2.0 clients need to authenticate their interactions with the authorization server, making it impossible for the attacker to exchange a stolen authorization code.

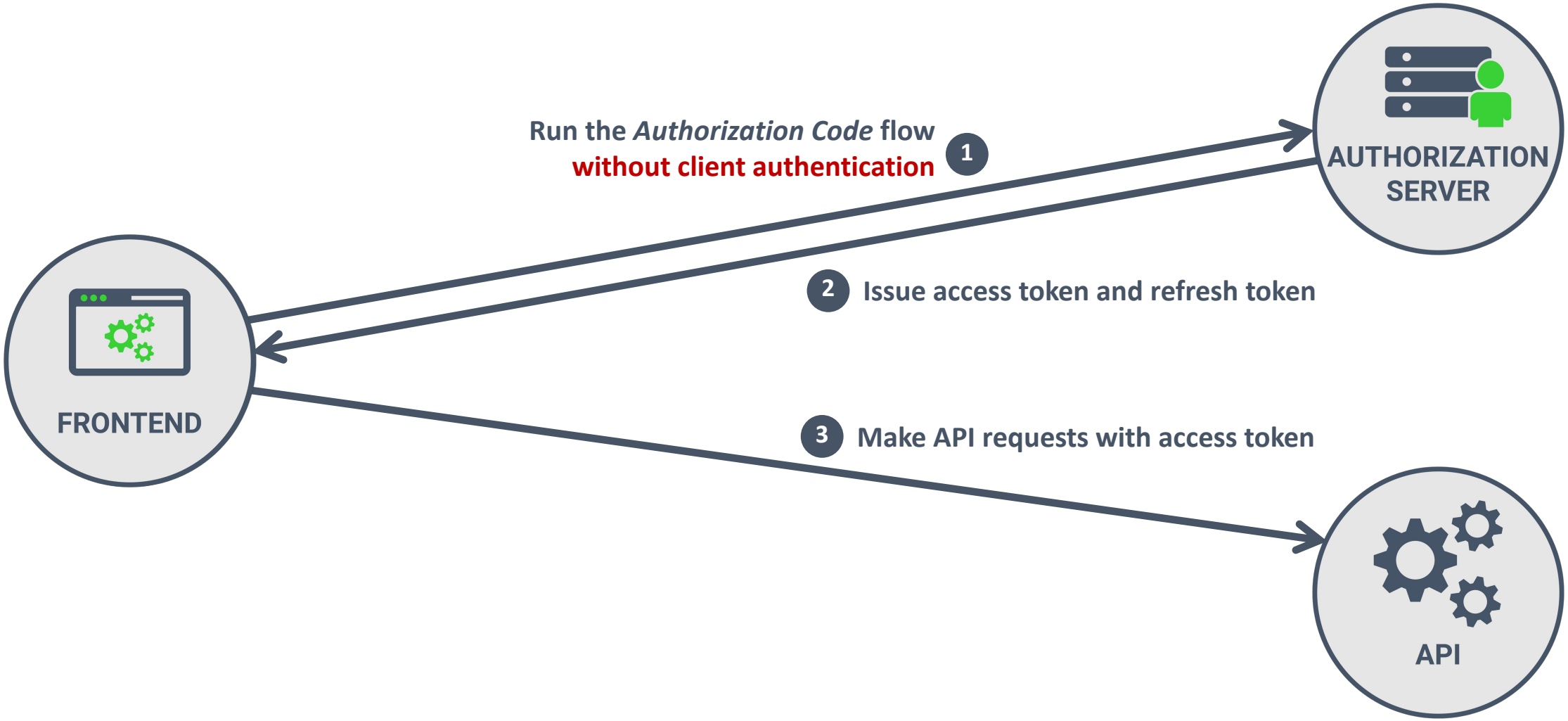
The attacker controlling pages in the browser can still send requests to the backend, which may result in data exfiltration or the execution of operations.

OAuth in frontends increases the attack surface

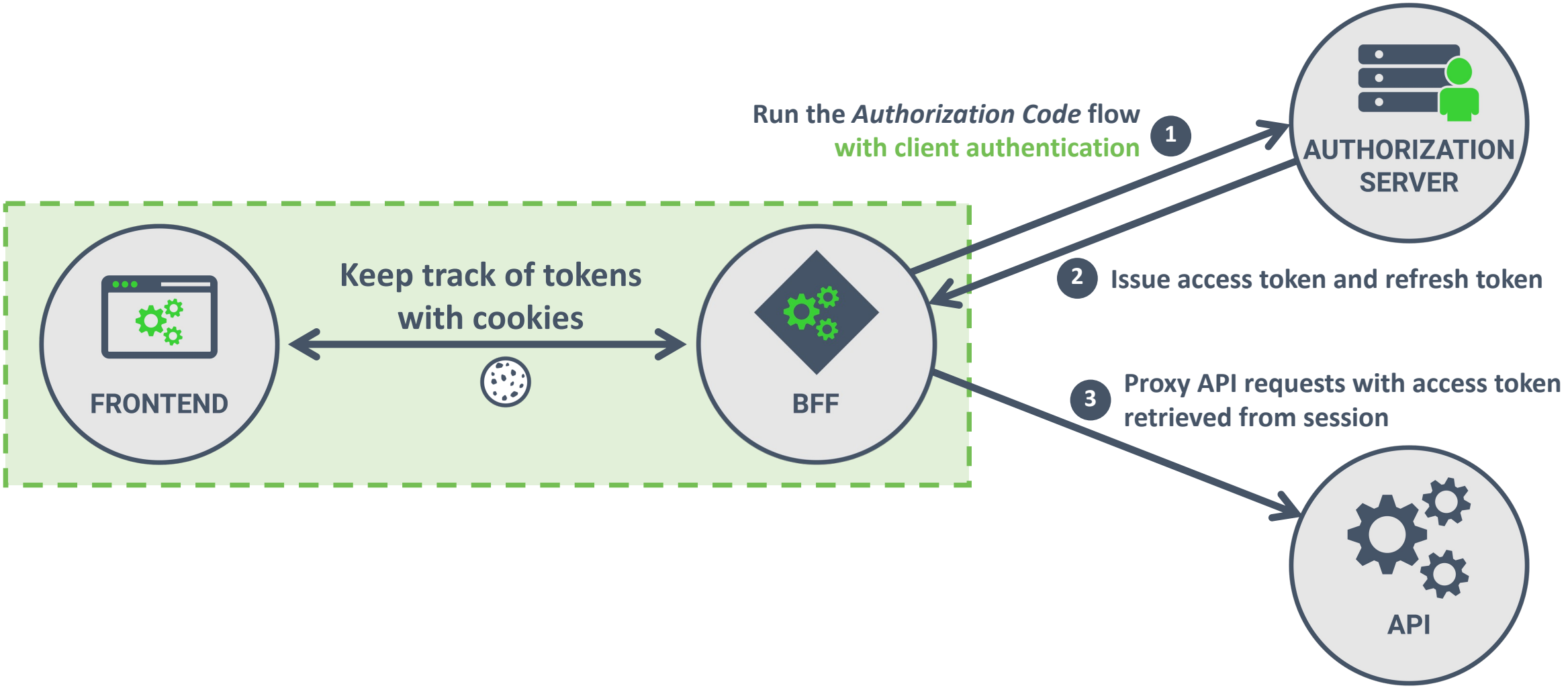
By using OAuth 2.0 in frontend applications, the attack surface of the application increases.

Attackers can impersonate the frontend application, allowing them to independently act in the name of the user for the lifetime of the refresh token.

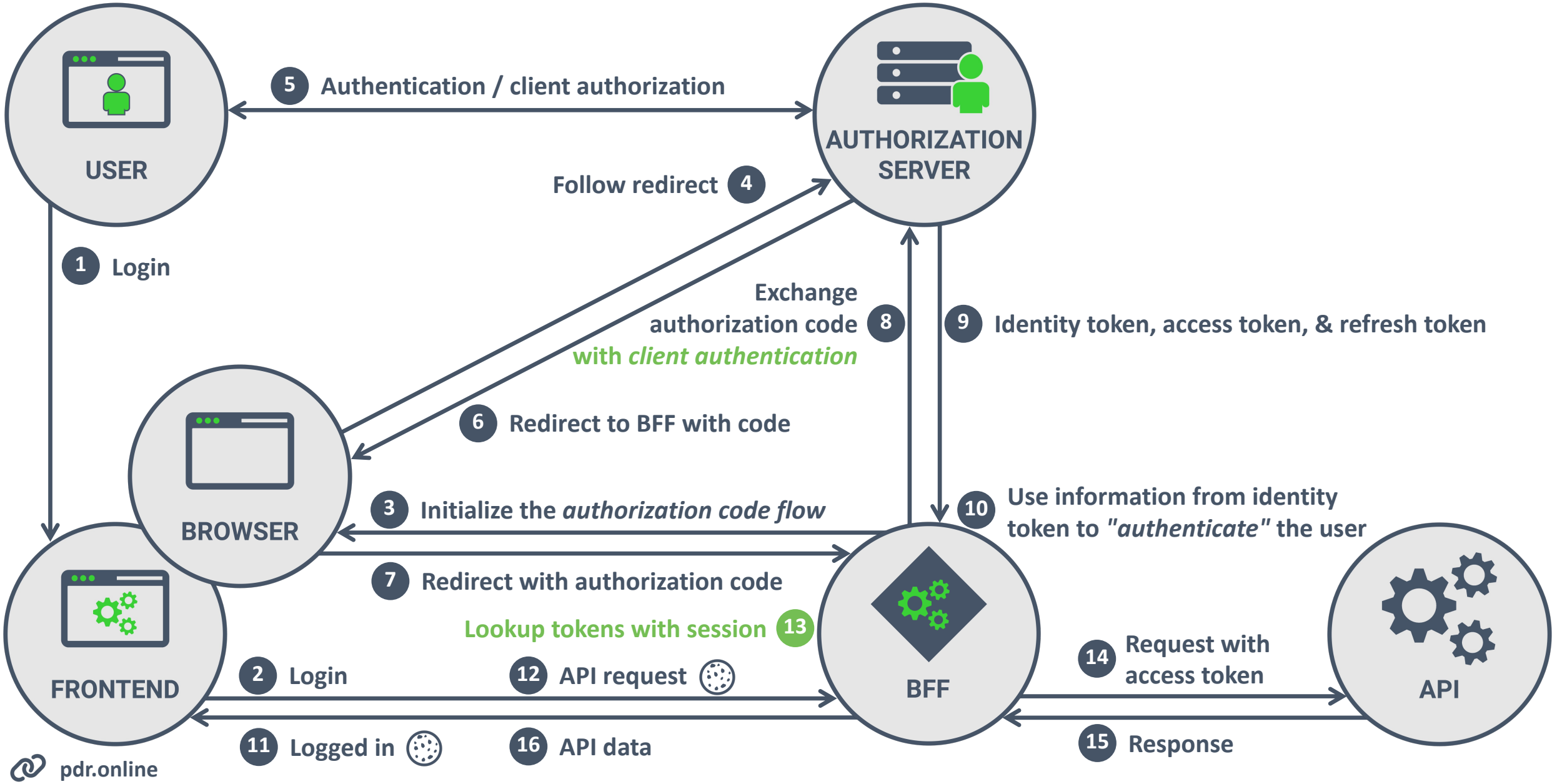
Can we have the security of backend OAuth clients in our frontend applications?


Run the *Authorization Code* flow
without client authentication
1
AUTHORIZATION SERVER
2
Issue access token and refresh token
FRONTEND
3
Make API requests with access token
API

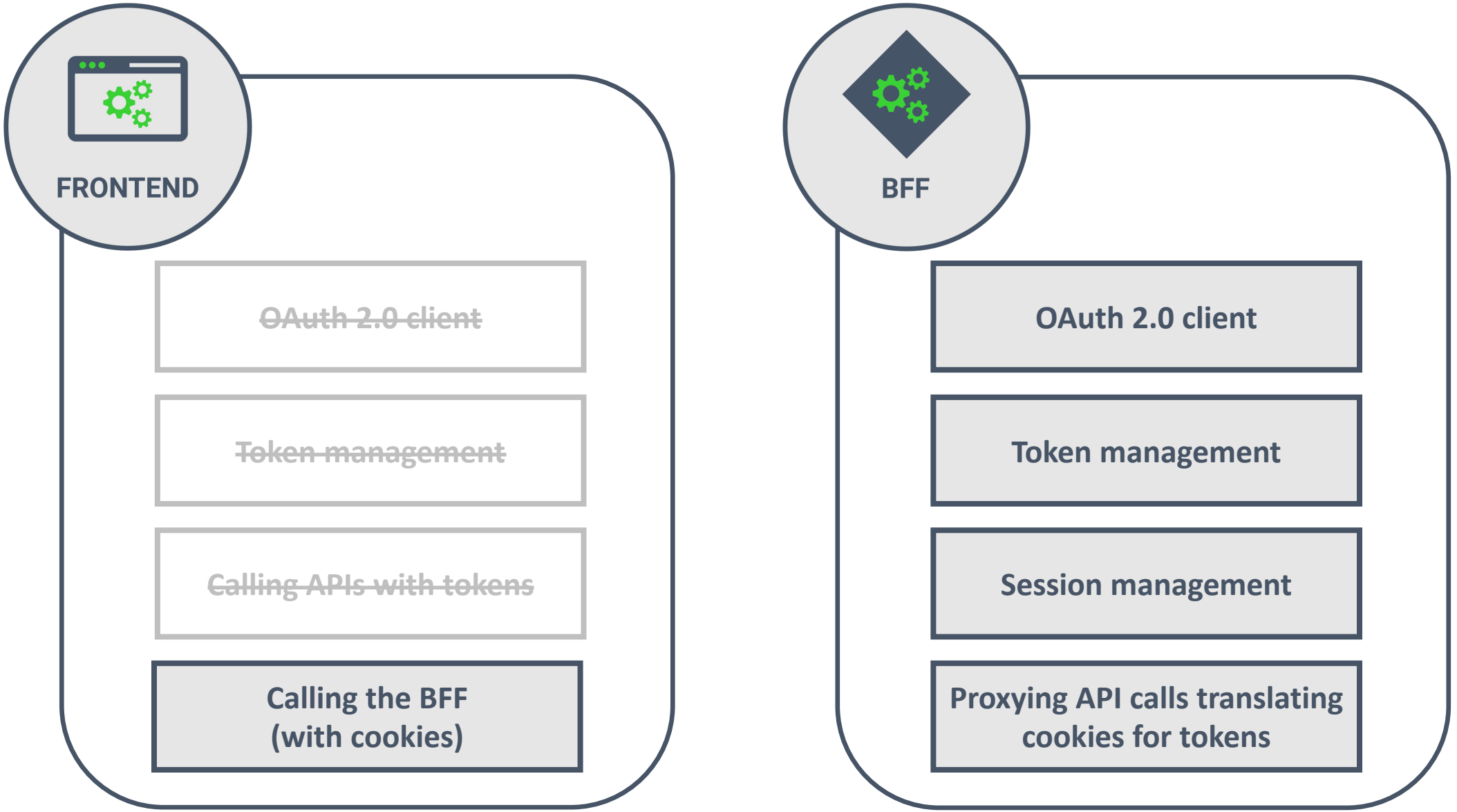
THE CONCEPT OF A BACKEND-FOR-FRONTEND (BFF)



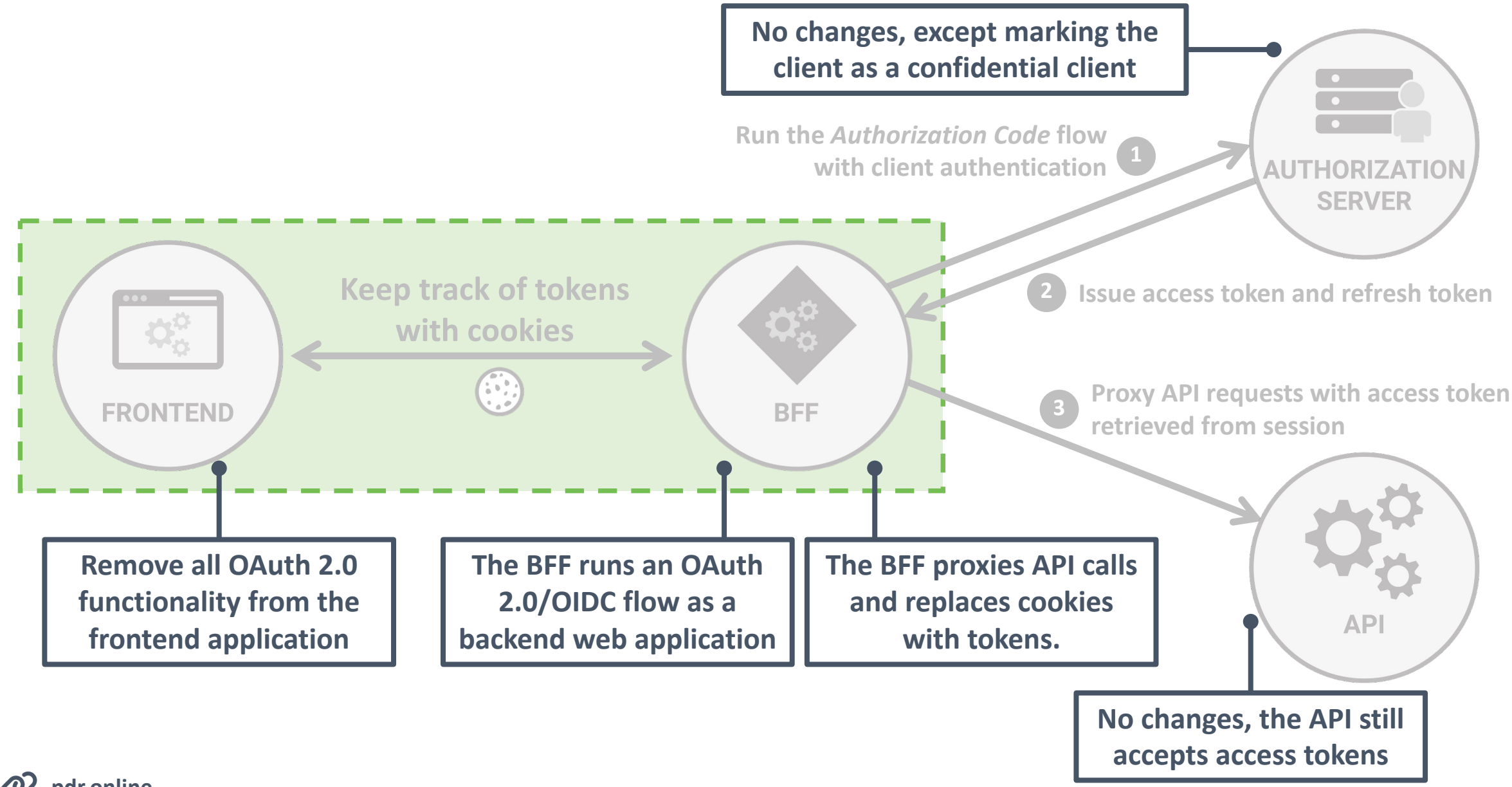
THE DETAILS OF A BACKEND-FOR-FRONTEND

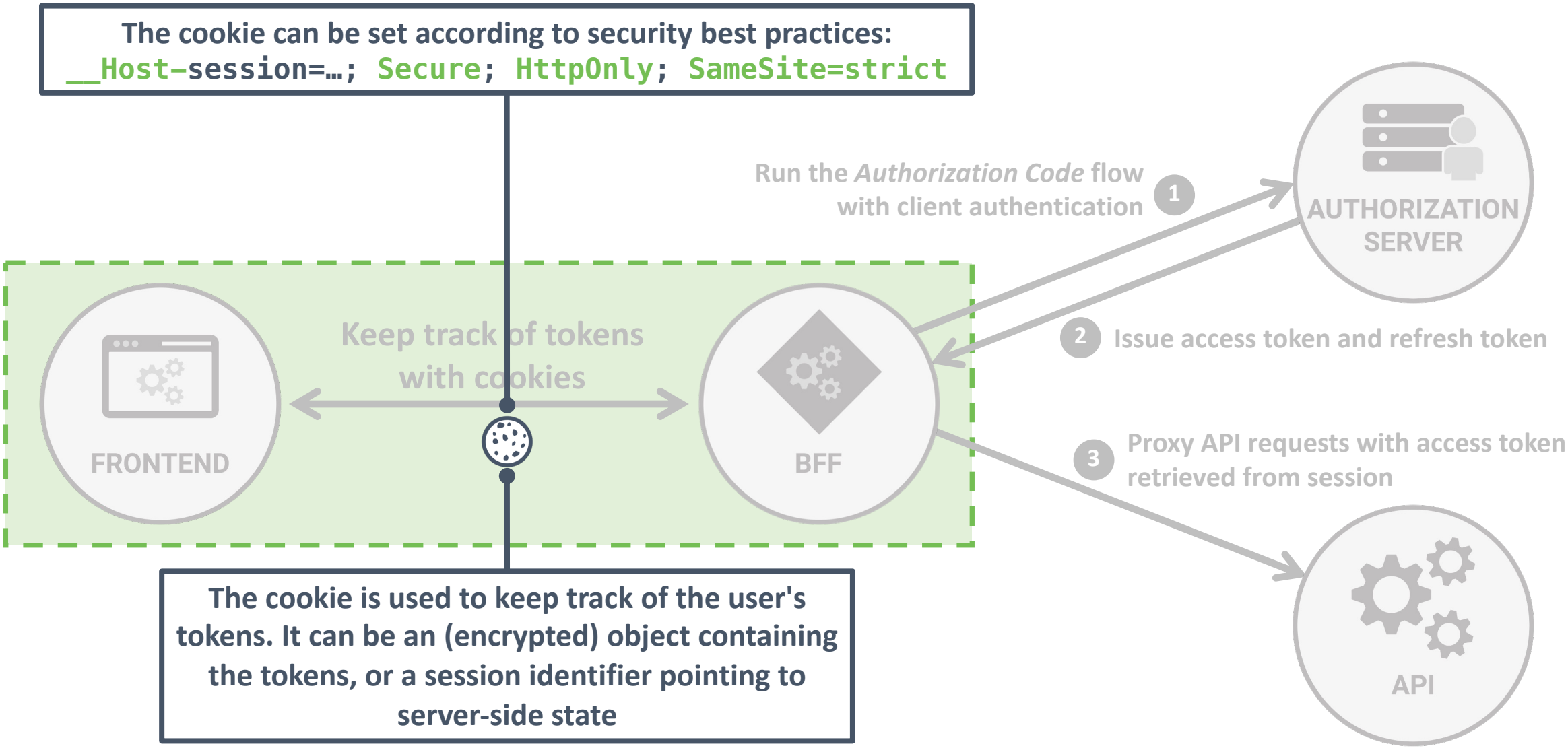


pdr.online


FRONTEND
~~OAuth 2.0 client~~
~~Token management~~
~~Calling APIs with tokens~~
Calling the BFF
(with cookies)
BFF
OAuth 2.0 client
Token management
Session management
Proxying API calls translating
cookies for tokens

That sounds great, but doesn't this require major changes to my existing application?


No changes, except marking the client as a confidential client
Run the *Authorization Code* flow with client authentication
1
AUTHORIZATION SERVER
2
Issue access token and refresh token
Keep track of tokens with cookies
FRONTEND
BFF
3
Proxy API requests with access token retrieved from session
API
Remove all OAuth 2.0 functionality from the frontend application
The BFF runs an OAuth 2.0/OIDC flow as a backend web application
The BFF proxies API calls and replaces cookies with tokens.
No changes, the API still accepts access tokens


The cookie can be set according to security best practices:
__Host-session=…; Secure; HttpOnly; SameSite=strict
Run the *Authorization Code* flow with client authentication
1
AUTHORIZATION SERVER
2
Issue access token and refresh token
Keep track of tokens with cookies
FRONTEND
BFF
3
Proxy API requests with access token retrieved from session
API
The cookie is used to keep track of the user's tokens. It can be an (encrypted) object containing the tokens, or a session identifier pointing to server-side state

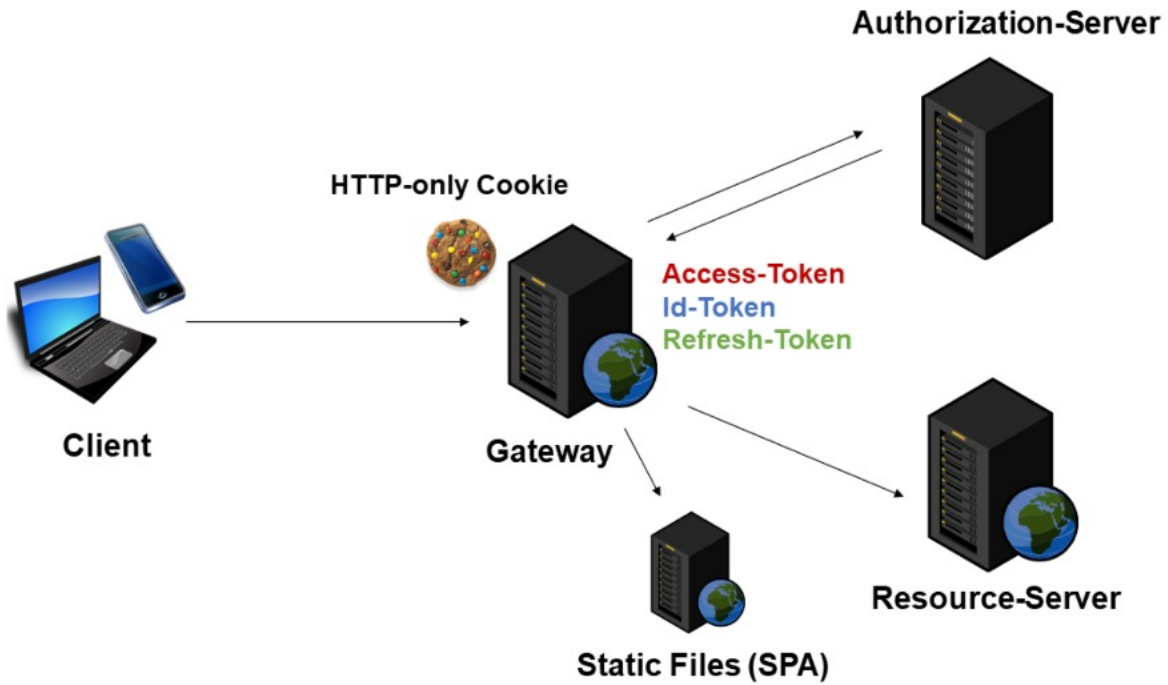
GitHub - manfredsteyer/yarp-a

github.com/manfredsteyer/yarp-auth-proxy

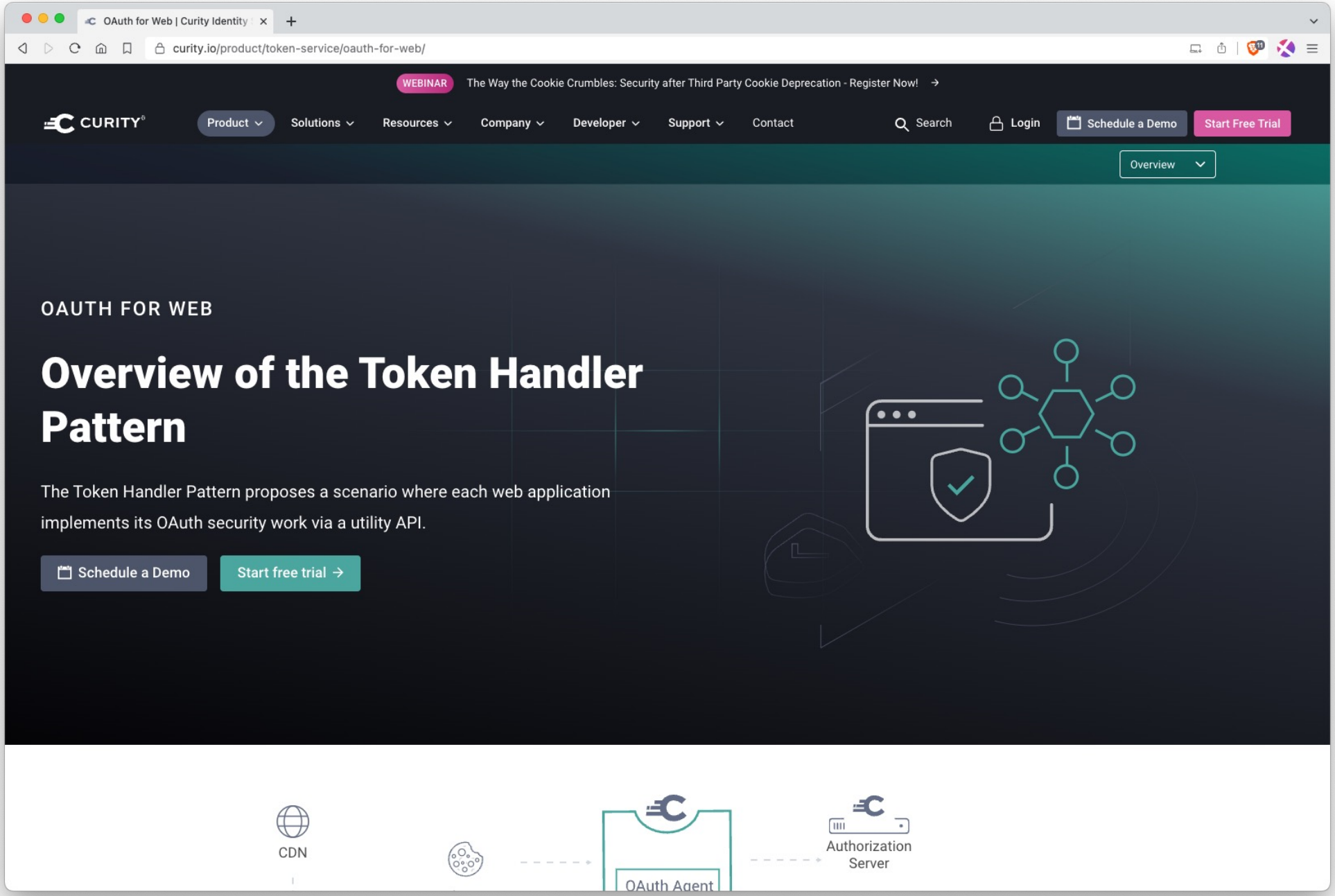
readme.md

Proof of Concept for an Auth Gateway for SPA

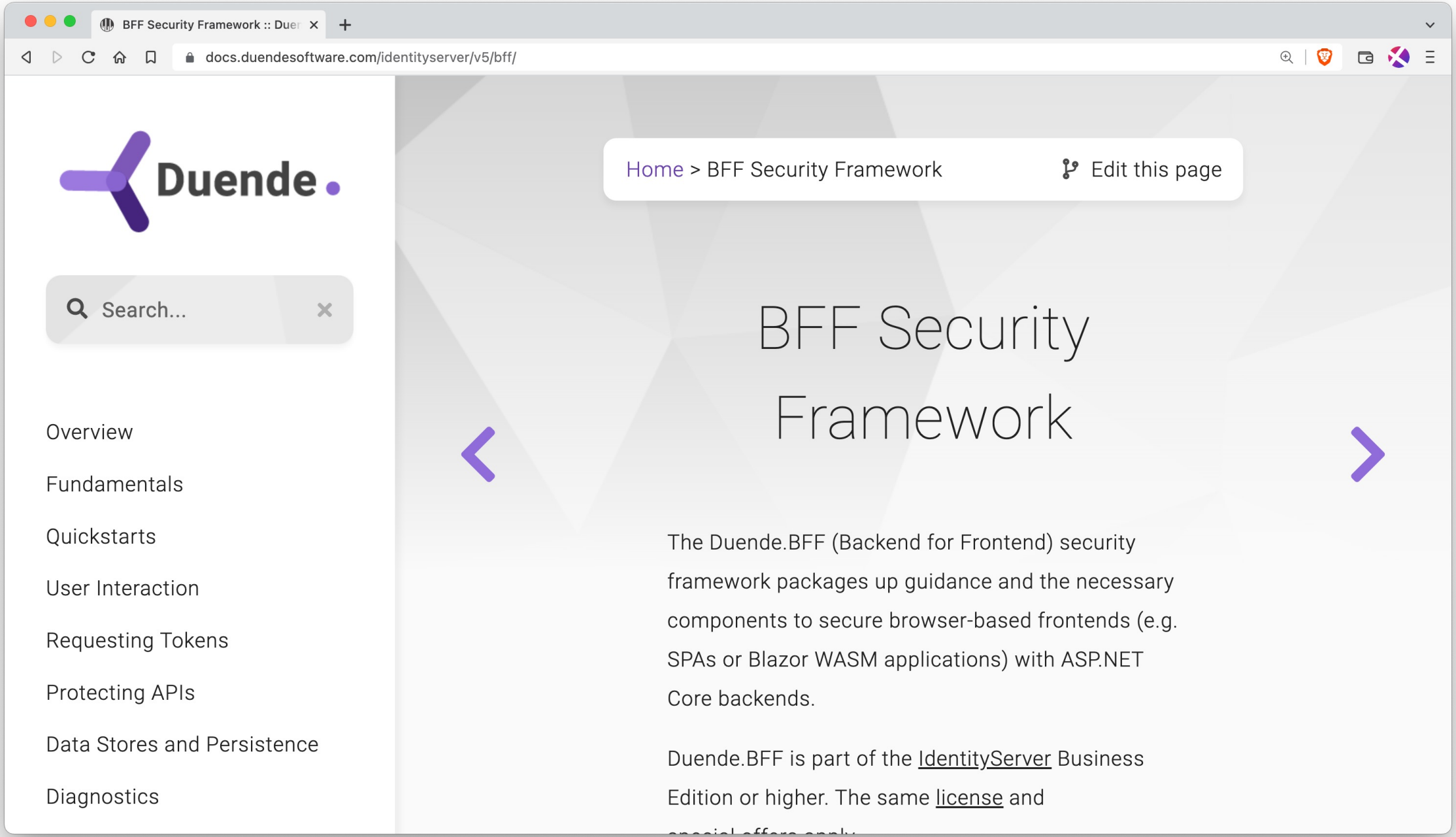
... aka Auth Reverse Proxy ... aka Backend for Frontend (BFF) ... aka Forward Authentication Service ...



> This gateway shifts the use of security standards such as OAuth2 and OpenId Connect to the server side.
> This drastically simplifies the implementation of the SPA and makes your solution more secure.



https://curity.io/resources/learn/token-handler-overview/


BFF Security Framework :: Duer
docs.duendesoftware.com/identityserver/v5/bff/
Duende.
Search...
Overview
Fundamentals
Quickstarts
User Interaction
Requesting Tokens
Protecting APIs
Data Stores and Persistence
Diagnostics
Home > BFF Security Framework
Edit this page
BFF Security Framework
The Duende.BFF (Backend for Frontend) security framework packages up guidance and the necessary components to secure browser-based frontends (e.g. SPAs or Blazor WASM applications) with ASP.NET Core backends.
Duende.BFF is part of the IdentityServer Business Edition or higher. The same license and

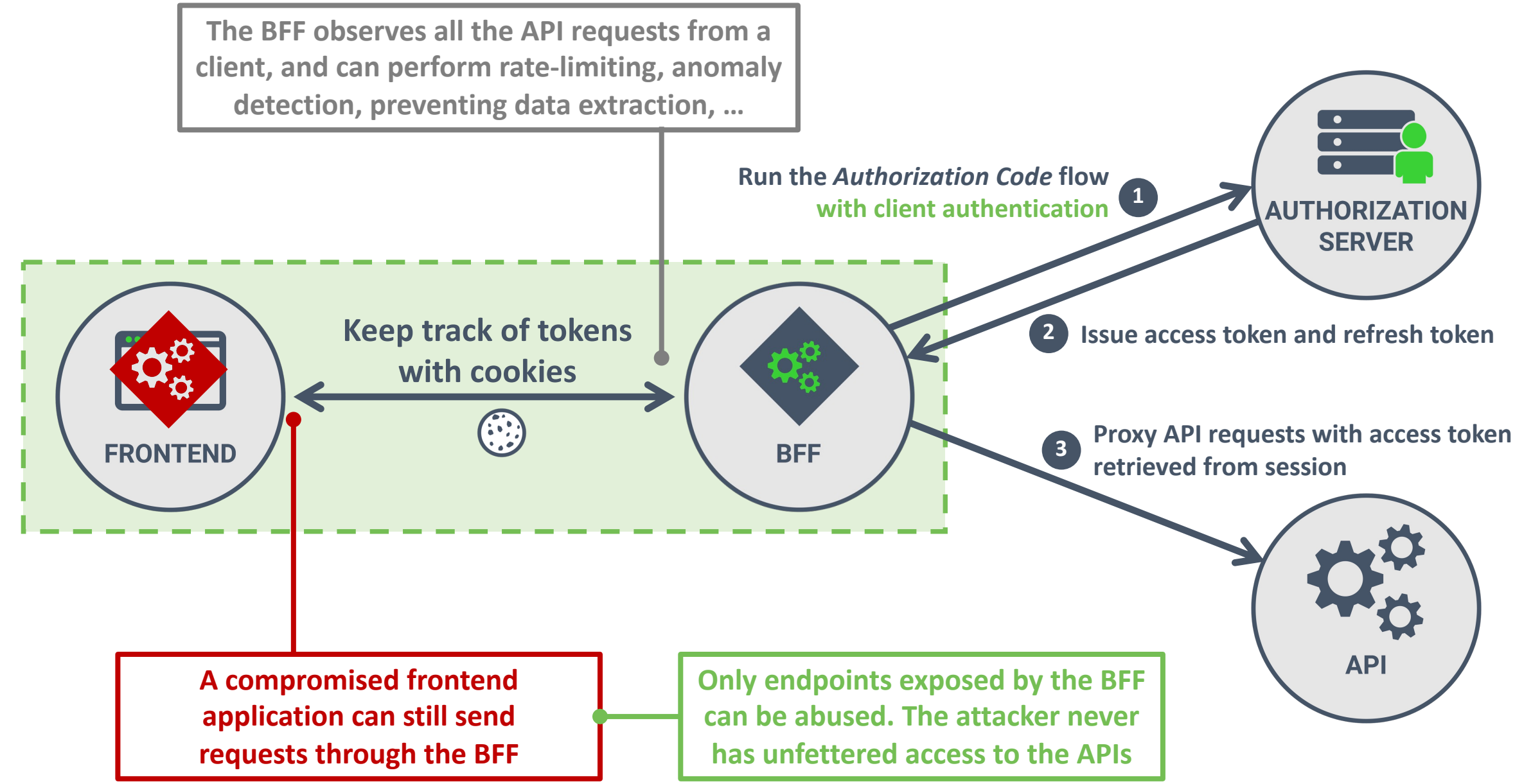
THREATS TO FRONTENDS WITH A BFF

Attack scenario	Example	Duration of attack
~~Single-execution token theft~~	~~One-time payload stealing an access token or refresh token from the ru~~	~~**Access tokens:** limited to token lifetime~~ ~~**Refresh tokens:** limited to detection~~
~~Persistent token theft~~	~~Contin~~ ~~tokens or refresh tokens from the running application~~	~~is online~~ ~~or application is open~~ ~~**Refresh tokens:** limited to token lifetime after user goes offline~~
~~Acquisition and extraction of new tokens~~	~~Runni~~ ~~Code~~ ~~token~~	~~sh~~ ~~rs or~~
Proxying requests via the user's browser	Trigge fronte applic	

Only the BFF has access to the application's tokens, preventing an attacker executing malicious JS in the browser cannot access server-side token storage.

The BFF is a server-side OAuth 2.0 client using authentication on its interactions with the authorization server, making it impossible for the attacker to impersonate the BFF.

The attacker controlling the frontend can still impersonate the legitimate frontend and send requests to the BFF, which will be forward these requests to the APIs.


The BFF observes all the API requests from a client, and can perform rate-limiting, anomaly detection, preventing data extraction, …
Run the *Authorization Code* flow with client authentication
1
AUTHORIZATION SERVER
2
Issue access token and refresh token
Keep track of tokens with cookies
FRONTEND
BFF
3
Proxy API requests with access token retrieved from session
API
A compromised frontend application can still send requests through the BFF
Only endpoints exposed by the BFF can be abused. The attacker never has unfettered access to the APIs

A BFF SIGNIFICANTLY INCREASES SECURITY

For sensitive applications, the security benefits of a BFF far outweigh the costs.

BFFs are often used in banking and healthcare scenarios, and you should do the same.

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-browser-based-apps-22
Published: 17 January 2025
Intended Status: Best Current Practice
Expires: 21 July 2025

A. Parecki
Okta
D. Waite
Ping Identity
P. De Ryck
Pragmatic Web Security

OAuth 2.0 for Browser-Based Applications

Abstract

This specification details the threats, attack consequences, security considerations and best practices that must be taken into account when developing browser-based applications that use OAuth 2.0.

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

Key takeaways

1. Using OAuth 2.0 directly in the browser increases the attack surface
2. Use a BFF to simplify and optimize the security of your frontends
3. Follow secure coding guidelines to fix XSS in your applications

Mastering OAuth 2.0 and OIDC Security (May 2025, EUR)

Live online workshop via Zoom

View schedule

OAuth 2.0 and OpenID Connect have become cornerstone technologies for most modern applications. Unfortunately, these technologies are insanely complex to grasp, making it hard to use them securely.

This workshop takes you on a **step-by-step journey into the world of OAuth 2.0 and OpenID Connect**. We start with understanding best practices for building secure applications with OAuth 2.0 and OIDC. Next, we will level up your OAuth 2.0 security using **the latest state-of-the-art security mechanisms**.

During this two-day hands-on training, spread out over four half days, we'll explore a broad range of OAuth 2.0 and OIDC topics. The outline below illustrates what the workshop will look like.

Day 1

- Introduction to OAuth 2.0 and OpenID Connect
- Architecture patterns using OAuth 2.0 and OpenID Connect
- Best practices for securing OAuth 2.0 and OIDC flows
- Understanding OAuth 2.0 security in frontends
- Breaking OAuth 2.0 security in frontends
- Securing OAuth 2.0 with the Backend-For-Frontend pattern
- Using scopes and permissions in OAuth 2.0
- Securing APIs with OAuth 2.0
- Demos and practical examples throughout the day

Day 2

- Advanced use cases for OAuth 2.0 and OpenID Connect
- Handling delegation scenarios in modern architectures
- Security best practices for confidential OAuth 2.0 clients
- Reducing access token authority with Resource Indicators
- Using sender-constrained tokens with mTLS and DPoP
- Securing OAuth 2.0 flows with JAR and PAR
- Advanced attacks and defenses against OAuth 2.0 flows
- Demos and practical examples throughout the day

This workshop is here to give you the skills you need to design architectures using OAuth 2.0 and OpenID Connect, to assess the security of your applications, and to enhance them using the latest best practices. **In-depth lectures, real-world demos, fun quizzes, and practical examples will guide you through the complex landscape of OAuth 2.0 and OpenID Connect**.

Thank you!

Need training or security guidance?
Reach out to discuss how I can help

https://pragmaticwebsecurity.com