How (Not) to Use OAuth

Dr. Daniel Fett
@dfett42

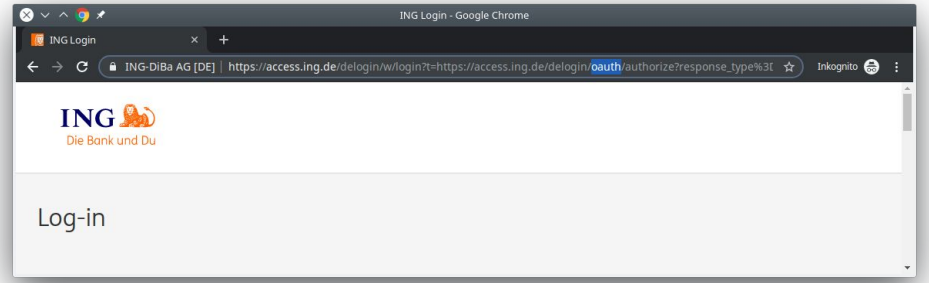
Secure
Cologne
Talks

yes®

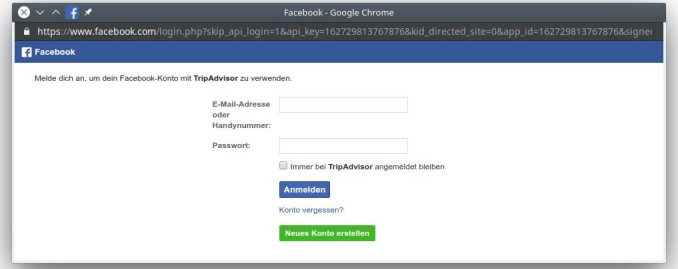
Who is familiar with OAuth?

OAuth 2.0 in the Wild

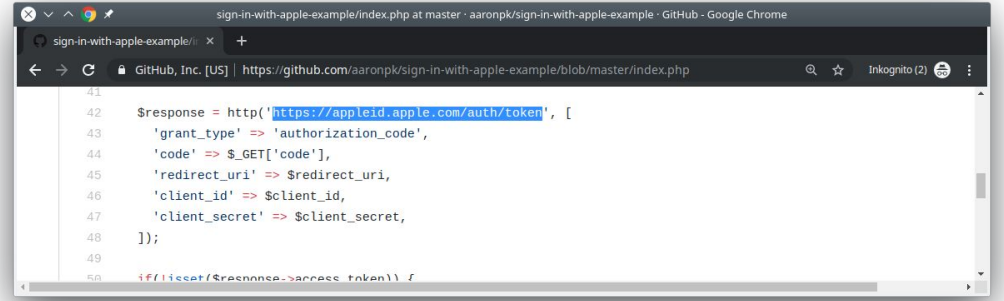
and friends



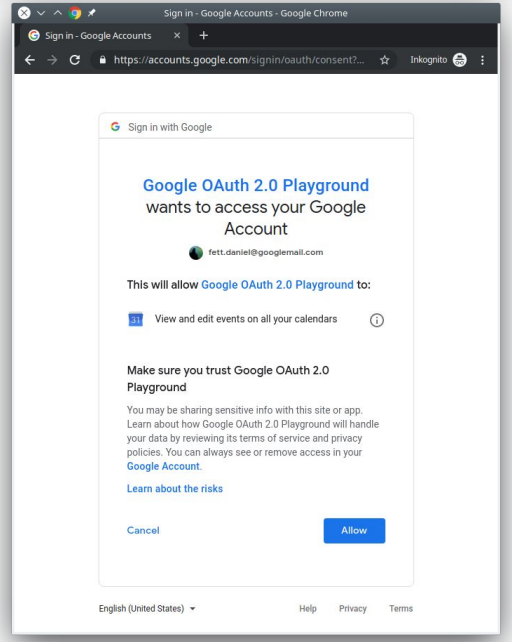
Banking



Facebook



Apple



Google

OAuth is a standard
for federated authorization

Authorization

User

authorizes

Photo Editor

Client

to access

Google Photos account

Authorization Server & Resource Server

Authentication

User

authenticates to

airbnb

Relying Party

using identity from

Identity Provider

Say OAuth is an Authentication standard again.

I dare you. I double dare you.

Authorization

OAuth (RFC6749+RFC6750)

User

authorizes

Photo Editor

Client

to access

Google Photos account

Authorization Server & Resource Server

Authentication

OAuth + OpenID Connect

User

authenticates to

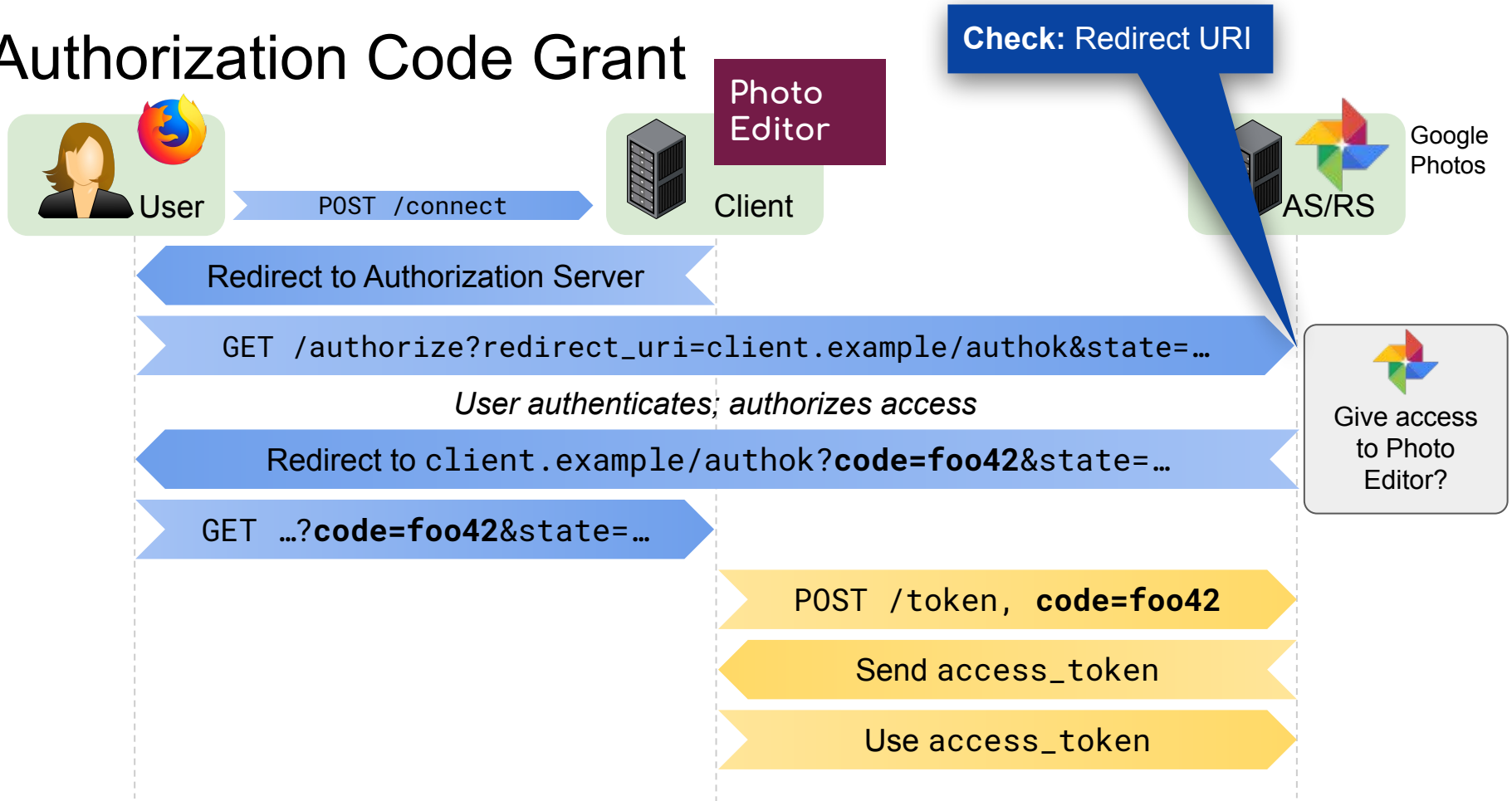
airbnb

Relying Party

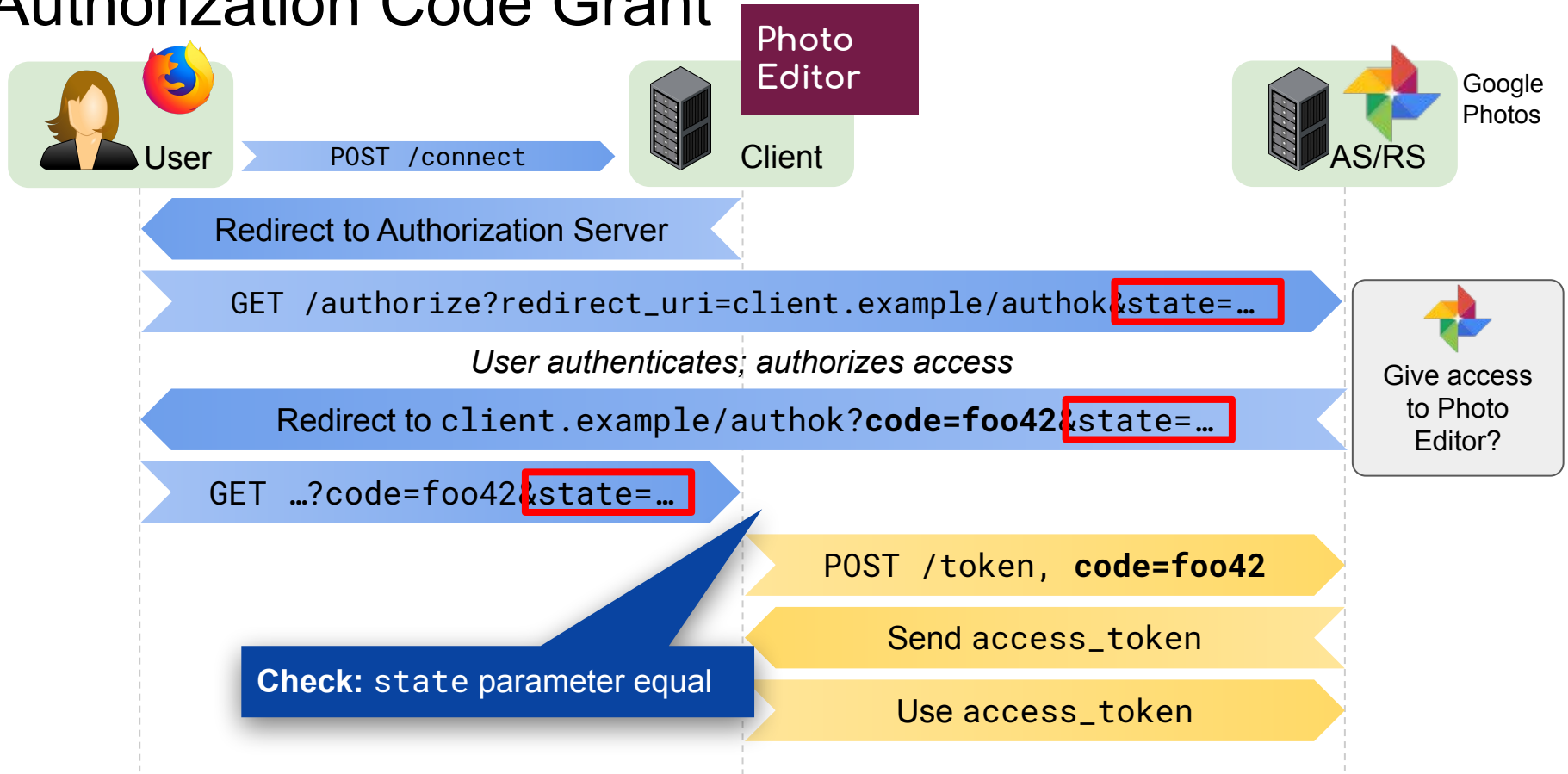
using identity from

Identity Provider

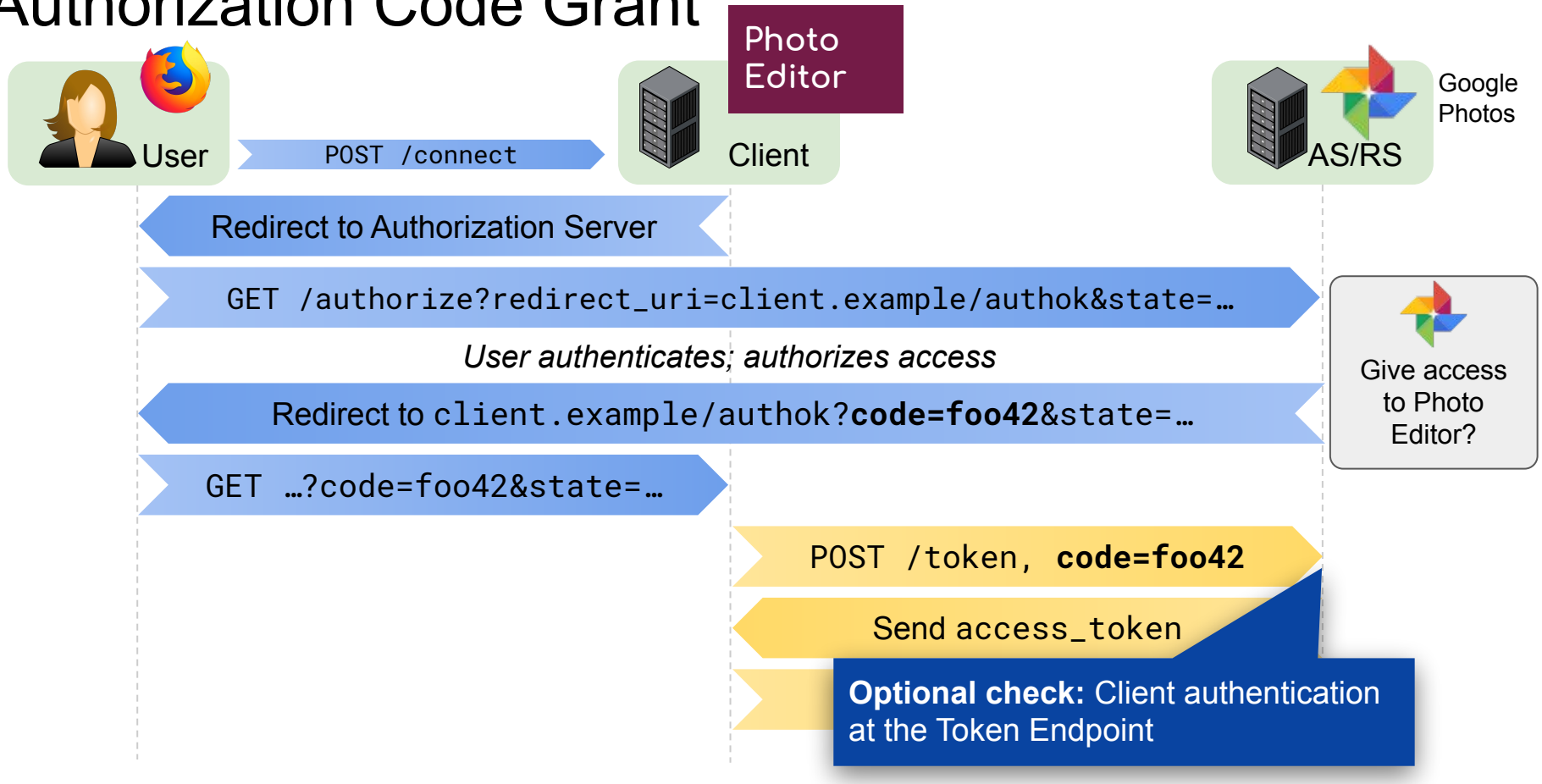
Authorization Code Grant



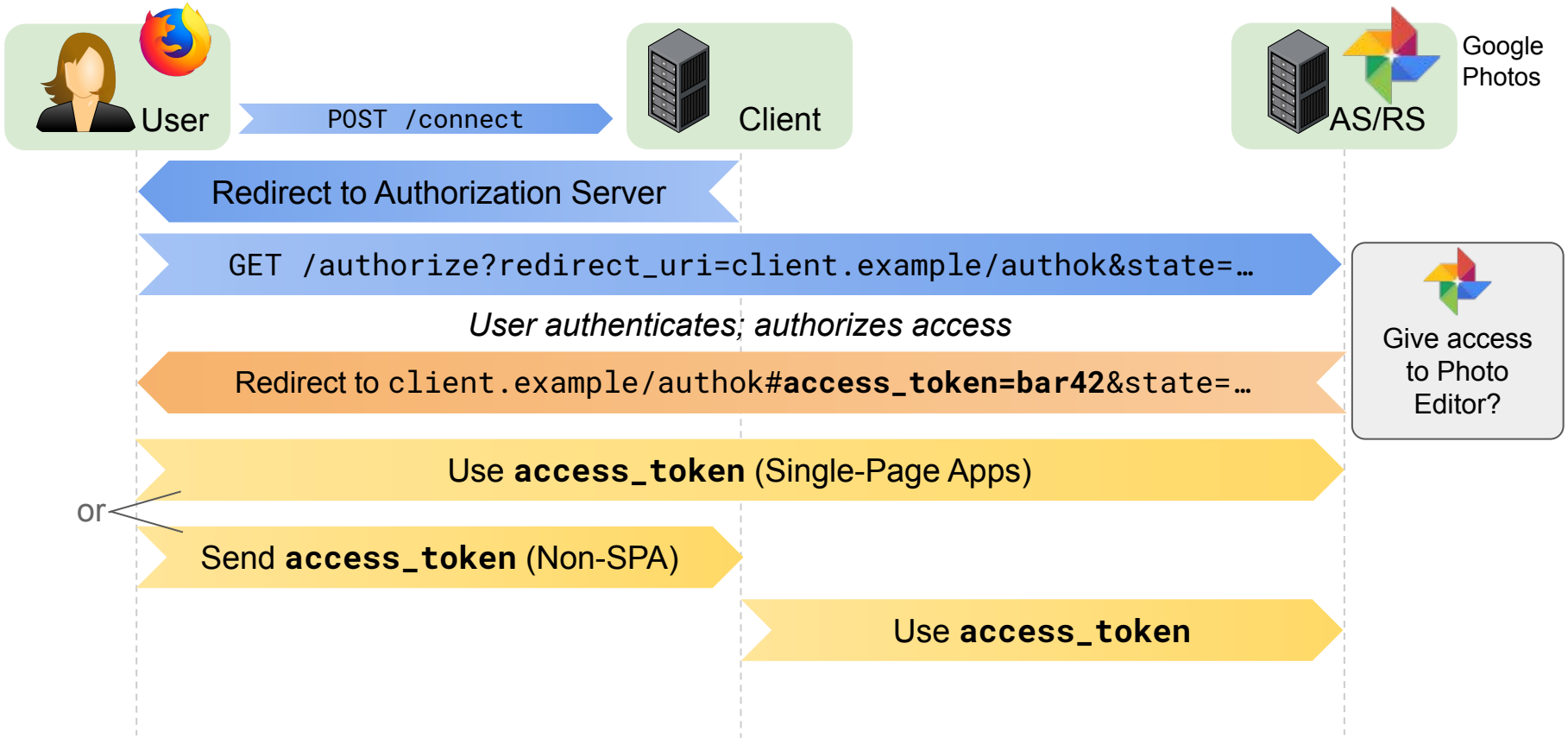
Authorization Code Grant



Authorization Code Grant



Implicit Grant — the “simpler OAuth”?



Seven Years after RFC6749:

Security Challenges for OAuth

Challenge 1: Implementation Flaws

- We still see many implementation flaws

Challenge 1: Implementation Flaws

- We still see many implementation flaws
- Known anti-patterns are still used
  - Insufficient redirect URI checking (code/token is redirected to attacker)
  - `state` parameter is not used properly to defend against CSRF
  - …

- [Li et al., 2014]
  60 chinese clients, **more than half** vulnerable to CSRF
- [Yang et al., 2016]
  Out of 405 clients, **55%** do not handle `state` (CSRF protection) correctly
- [Shebab et al., 2015]
  25% of OAuth clients in Alexa Top 10000 vulnerable to CSRF
- [Chen et al., 2014]
  89 of 149 mobile clients vulnerable to one or more attacks
- [Wang et al., 2013]
  Vulnerabilities in Facebook PHP SDK and other OAuth SDKs
- [Sun et al., 2012]
  96 Clients, **almost all** vulnerable to one or more attacks

Challenge 1: Implementation Flaws

- We still see many implementation flaws
- Known anti-patterns are still used
  - Insufficient redirect URI checking (code/token is redirected to attacker)
  - `state` parameter is not used properly to defend against CSRF
  - …
- Technological changes bring new problems
  - E.g., URI fragment handling in browsers changed
    → Vulnerability when used with open redirectors

Open Redirector: Parameterized, unchecked redirection. E.g.:

`https://client.example/anything?resume_at=https://evil.example`

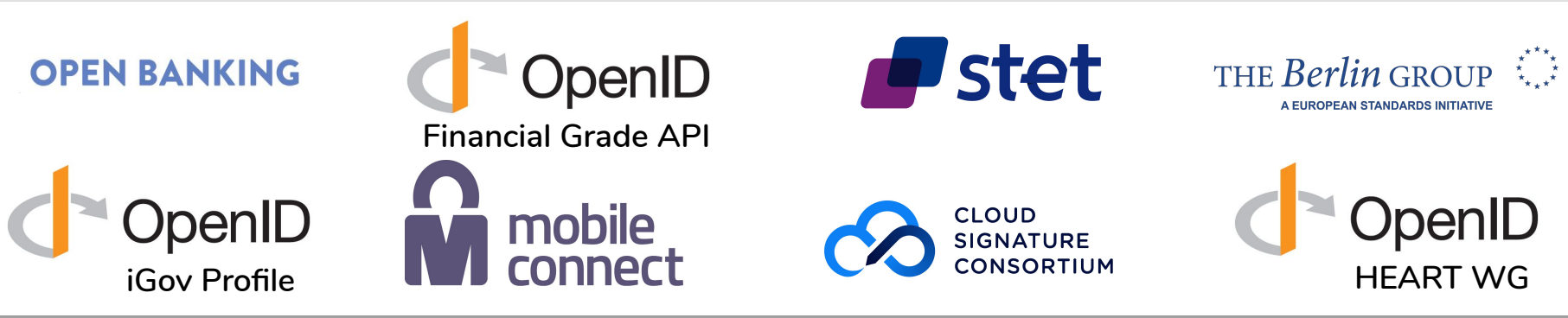
Redirects to `https://evil.example`

Challenge 1: Implementation Flaws

- We still see many implementation flaws
- Known anti-patterns are still used
  - Insufficient redirect URI checking (code/token is redirected to attacker)
  - `state` parameter is not used properly to defend against CSRF
  - …
- Technological changes bring new problems
  - E.g., URI fragment handling in browsers changed
    → Vulnerability when used with open redirectors

Challenge 2: High-Stakes Environments

New Use Cases, e.g., Open Banking, require a very high level of security

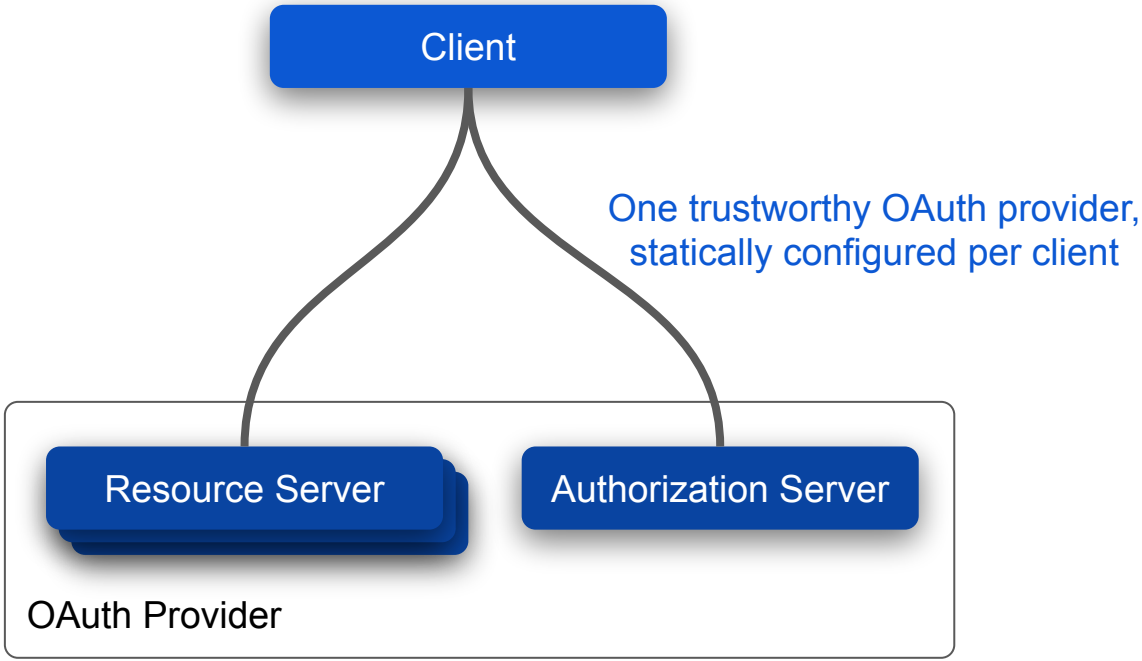


Also: eIDAS/QES (legally binding electronic signatures)

Far beyond the scope of the original security threat model!

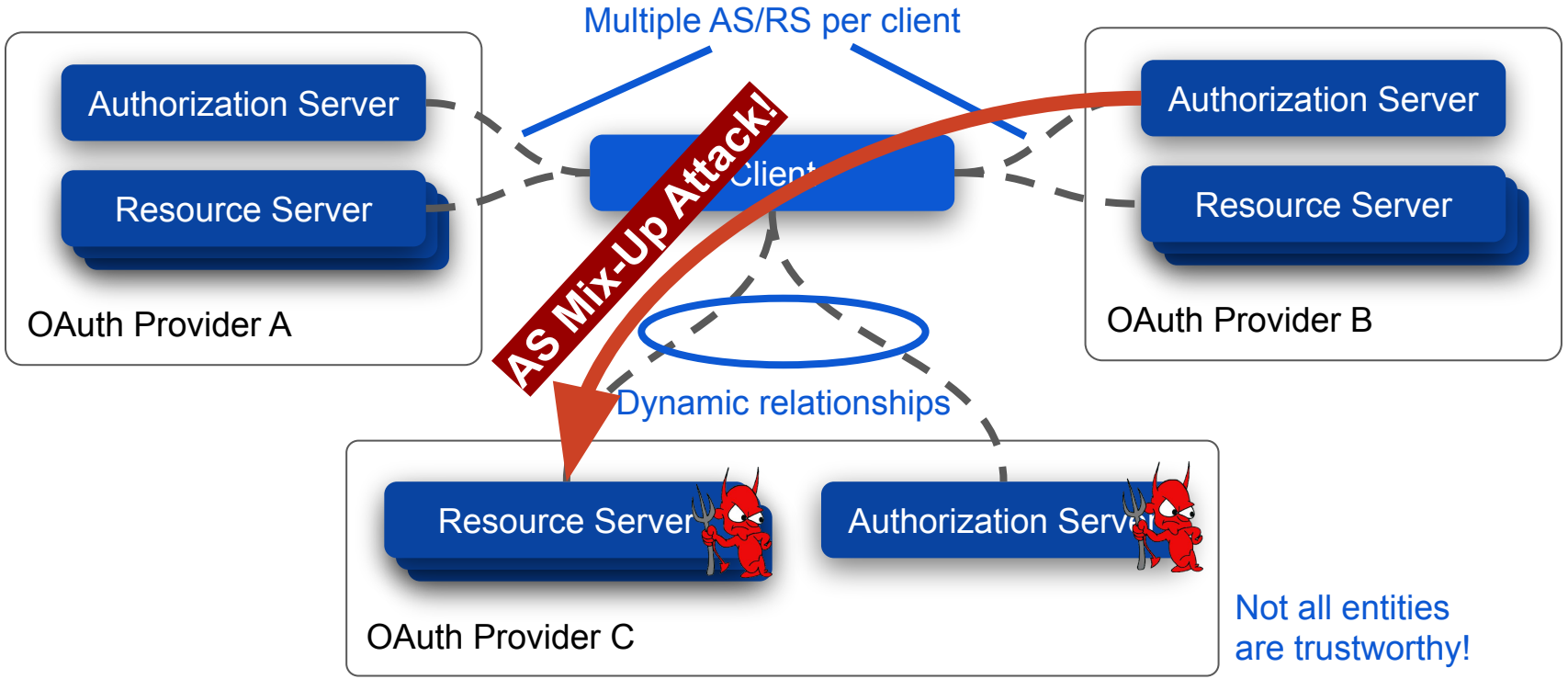
Challenge 3: Dynamic and Complex Setups

Originally anticipated:



Challenge 3: Dynamic and Complex Setups

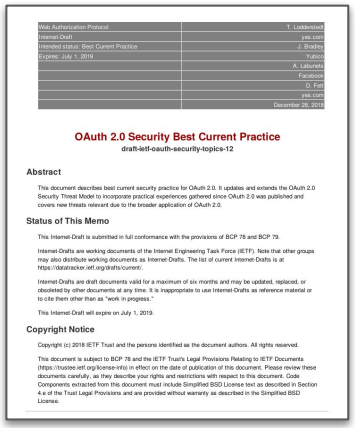
Today:



How to address these challenges?

OAuth 2.0 Security Best Current Practice RFC

- Under development at the IETF
- Refined and enhanced security guidance for OAuth 2.0 implementers
- Complements existing security guidance in RFCs 6749, 6750, and 6819



Web Authorization Protocol	T. Lodderstedt
Internet-Draft	yes.com
Intended status: Best Current Practice	J. Bradley
Expires: July 1, 2019	Yubico
	A. Labunets
	Facebook
	D. Fett
	yes.com
	December 28, 2018

OAuth 2.0 Security Best Current Practice

draft-ietf-oauth-security-topics-12

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

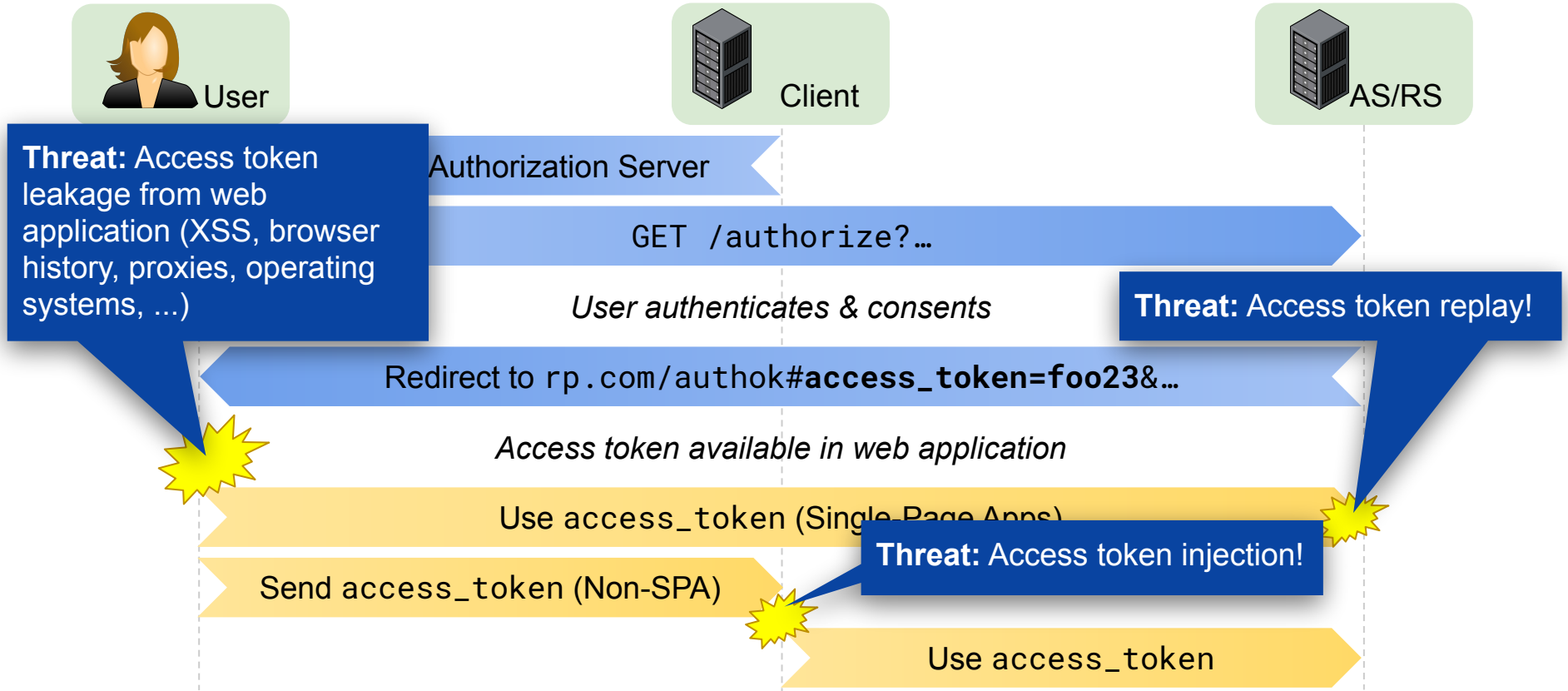
- Updated, more comprehensive Threat Model
- Description of Attacks and Mitigations
- Simple and actionable recommendations

Input from **practice** and **formal analysis**

IETF®

The Seven Most Important Recommendations in the OAuth Security BCP

① Do not use the OAuth Implicit Grant any longer!



The Implicit Grant ...

- sends **powerful** and **potentially long-lived** tokens through the browser,
- lacks features for **sender-constraining** access tokens,
- provides no protection against access token **replay and injection**, and
- provides no **defense in depth** against XSS, URL leaks, etc.!

Why is Implicit even in RFC6749?

No Cross-Origin Resource Sharing in 2012!
⇒ No way of (easily) using OAuth in SPAs.

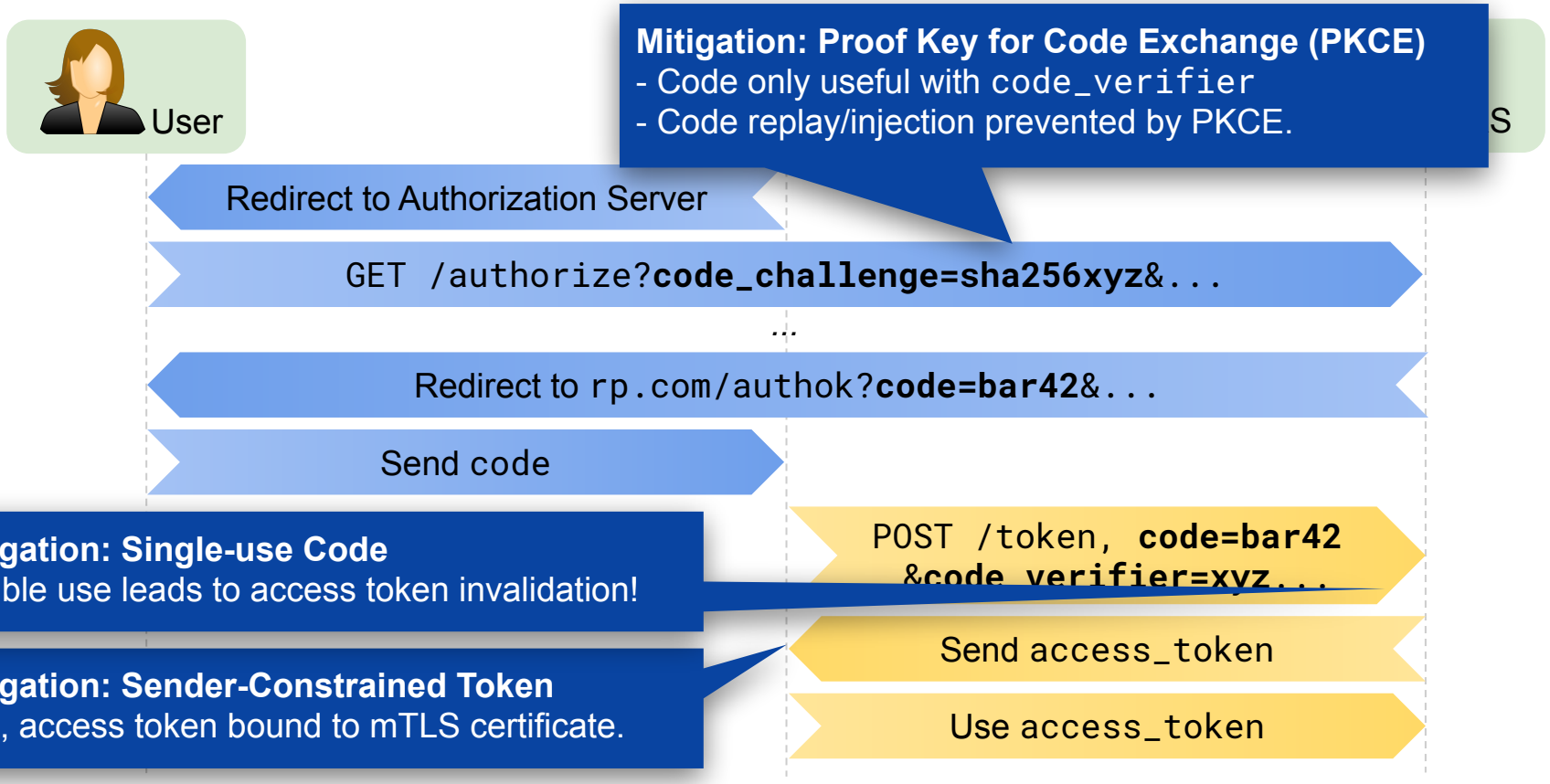
⇒ Not needed in 2019!

Recommendation

“Clients SHOULD NOT use the implicit grant [...]”

“Clients SHOULD instead use the response type code (aka authorization code grant type) [...]”

Use the Authorization Code Grant!



Authorization Code Grant with PKCE & mTLS …

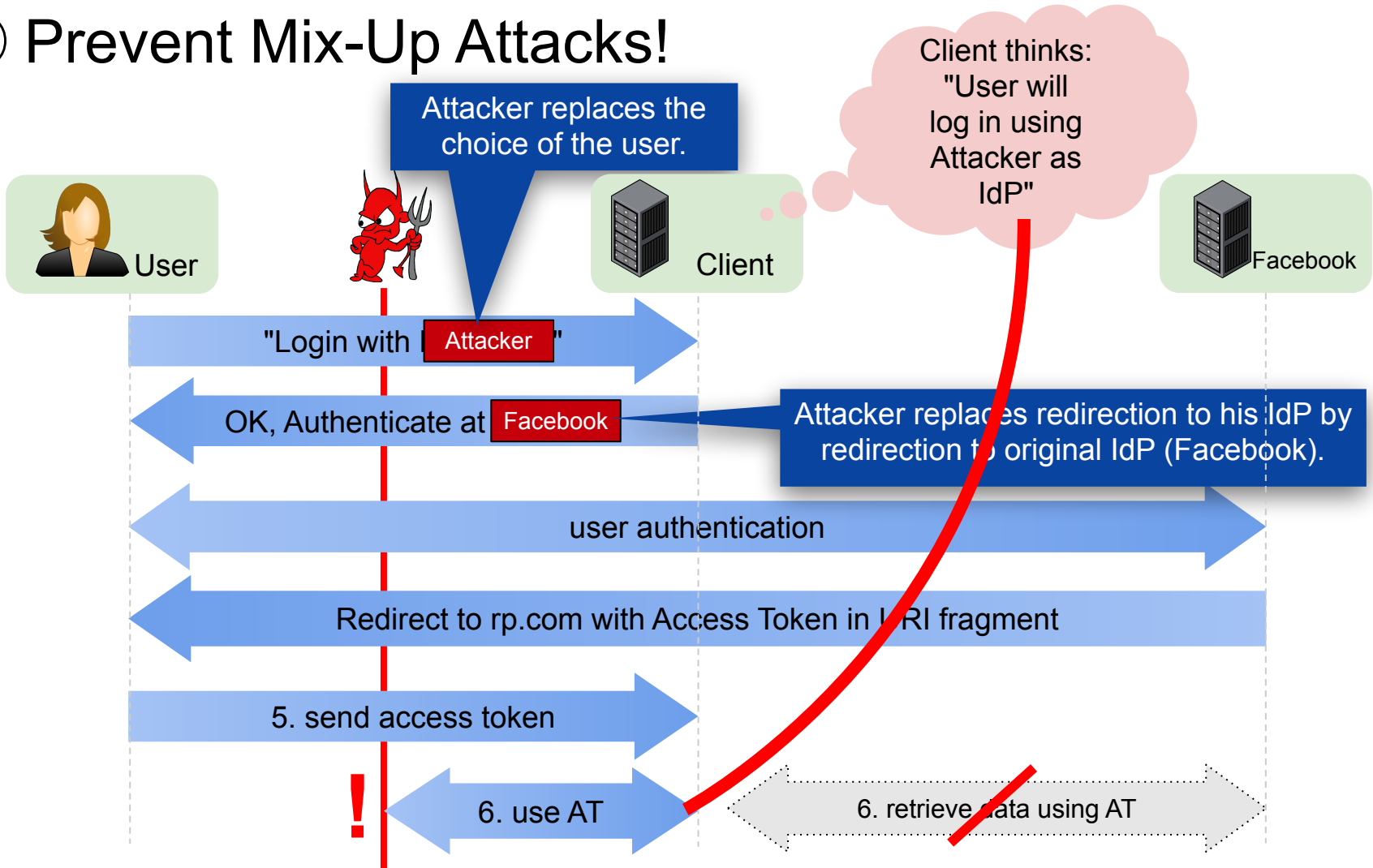
- protects against **code and token replay and injection,**
- supports **sender-constraining** of access tokens,
- provides **defense in depth!**

Recommendation

“Clients utilizing the authorization grant type MUST use PKCE [...]”

“Authorization servers SHOULD use TLS-based methods for sender-constrained access tokens [...]”

② Prevent Mix-Up Attacks!



② Prevent Mix-Up Attacks!

- Clients MUST be able to see originator of authorization response
  - Clients SHOULD use a **separate redirect URI** for each AS
  - Alternative: issuer in authorization response for OpenID Connect
- Clients MUST keep track of desired AS (“explicit tracking”)

③ Stop Redirects Gone Wild!

- Enforce exact redirect URI matching
  - Simpler to implement on AS side
  - Adds protection layer against open redirection
- Clients MUST avoid open redirectors!
  - Use whitelisting of target URLs
  - or authenticate redirection request

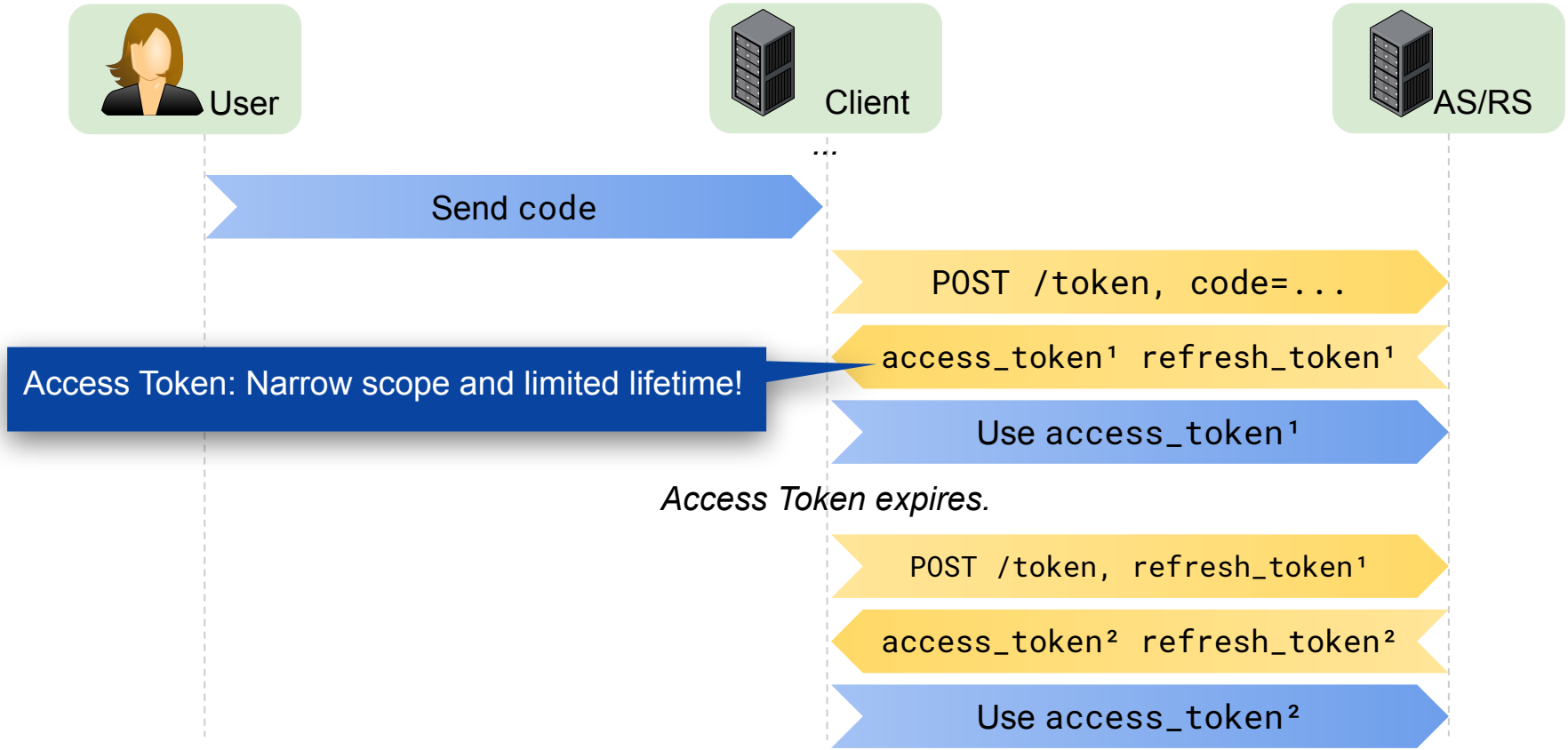
④ Prevent CSRF Attacks!

- CSRF attacks MUST be prevented
- RFC 6749 and RFC 6819 recommend use of `state` parameter
- Updated advice:
  - If PKCE is used, `state` is not needed for CSRF protection
  - `state` can again be used for application state

⑤ Limit Privileges of Access Tokens!

- Sender-constraining (mTLS, HTTP Token Binding, or DPoP)
- Receiver-constraining (only valid for certain RS)
- Reduce scope and lifetime and use refresh tokens - defense in depth!

Refresh Tokens

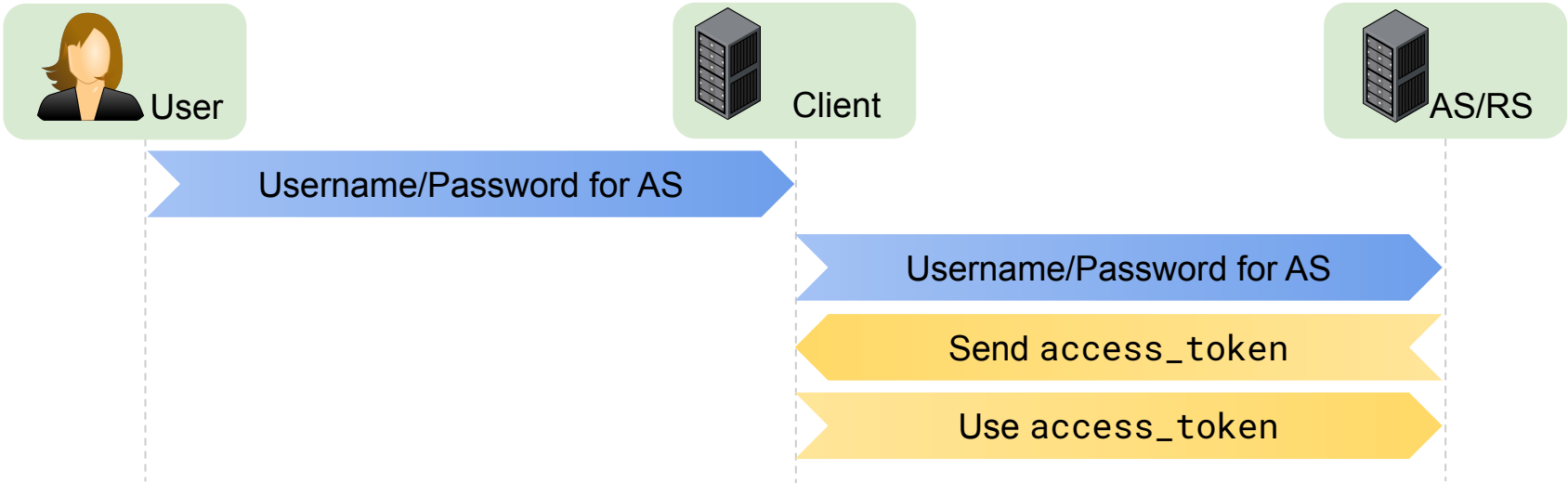


⑥ Protect Refresh Tokens!

- Attractive target since refresh tokens represent overall grant
- Requirement: Protection from theft and replay
  - Client Binding and Authentication
    - Confidential clients only
  - Sender-Constrained Refresh Tokens
    - mTLS and DPoP now support this even for public clients

⑦ Do not use the R.O.P.C.G.* any longer!

*Resource Owner Password Credentials Grant



- Client sees username/password of user
- Complicated or impossible to integrate 2-factor-authentication
- Stopgap solution for migrating to OAuth flows
- Grant name too long, even for Germans ;-)

What else?

- Do not use HTTP status code 307 for redirections
  - User credentials may be leaked to an attacker
- Aim to prevent code leakage from referrer headers and browser history
  - E.g., referrer policies, browser history manipulations, etc.
  - Already common practice among implementers
  - Only one of many lines of defense now
- Use client authentication if possible
  - Client authenticates at the token endpoint
  - More protection for authorization code

Should I even use OAuth?

Absolutely!

- Standards are good
  - Battle-proven libraries
  - Interoperability
- Years of experience, dozens of security analyses
- Custom-built solutions prone to repeat even the most simple vulnerabilities
- Protection against strong attackers
- Formal proof of security
- But:
  - Read the security advice, including the BCP draft
  - Implement the latest security features
  - Know your threat model

Dr. Daniel Fett
yes.com
danielf@yes.com
@dfett42

Q&A!

Latest Draft, papers, etc.: `https://danielfett.de` → Publications

yes®