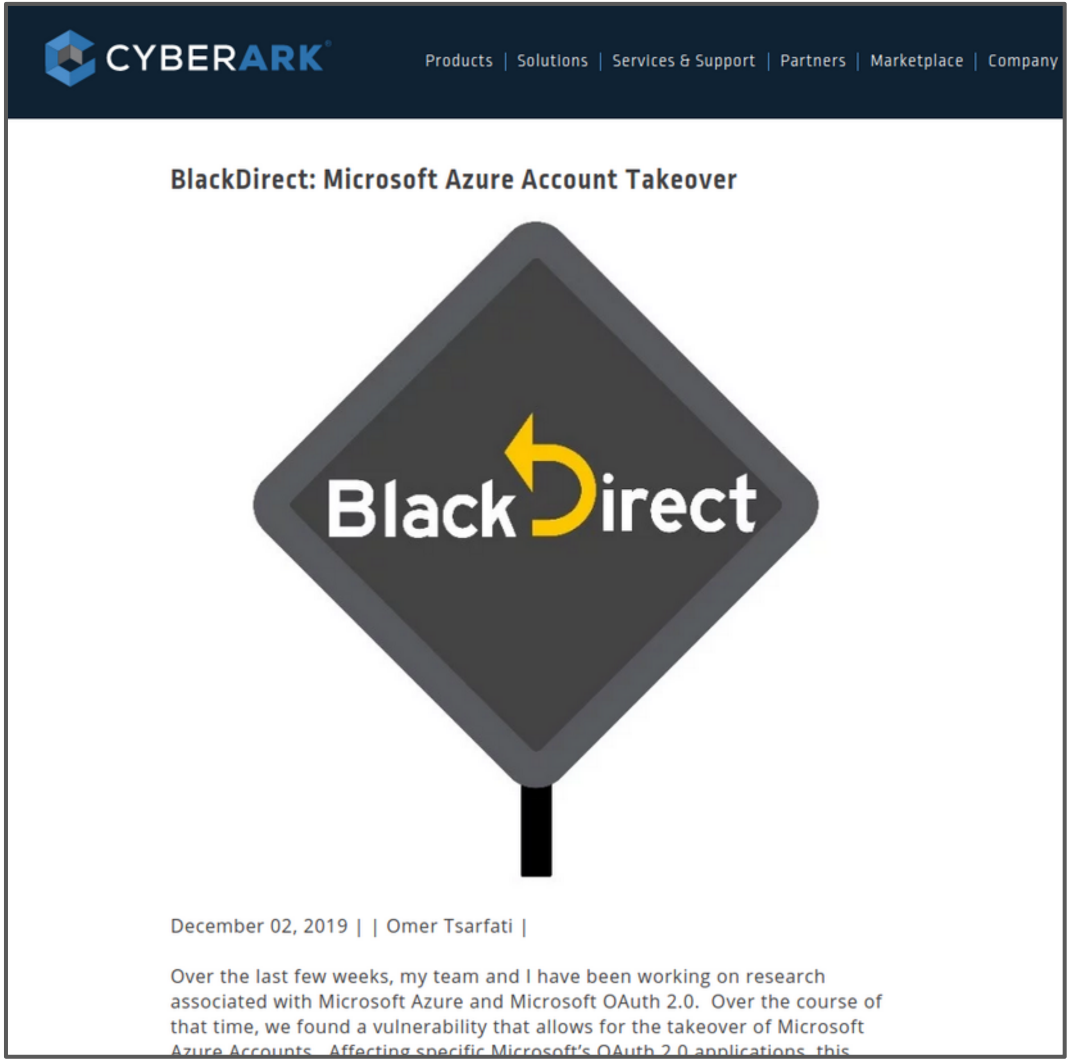
How (Not) to Use OAuth

Dr. Daniel Fett

@dfett42

REPLY
SPIKE


CYBERARK
Products | Solutions | Services & Support | Partners | Marketplace | Company
BlackDirect

BlackDirect: Microsoft Azure Account Takeover

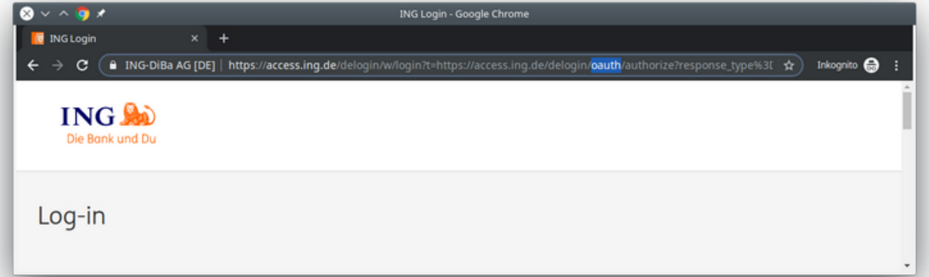
December 02, 2019 | | Omer Tsarfati |

Over the last few weeks, my team and I have been working on research associated with Microsoft Azure and Microsoft OAuth 2.0. Over the course of that time, we found a vulnerability that allows for the takeover of Microsoft Azure Accounts. Affecting specific Microsoft's OAuth 2.0 applications, this

Who is familiar with OAuth?

OAuth 2.0

OAuth 2.0 in the Wild

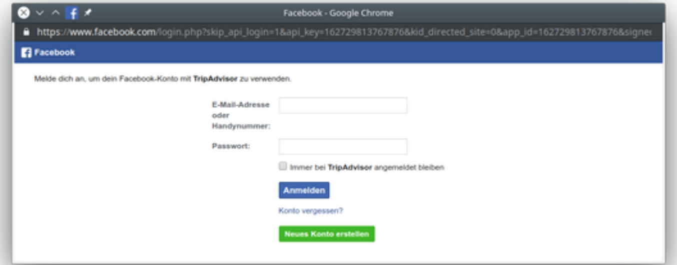


ING Login - Google Chrome

ING-DiBa AG [DE] | https://access.ing.de/delogin/w/login?t=https://access.ing.de/delogin/oauth/authorize?response_type%3[

Log-in

Banking



Facebook - Google Chrome

https://www.facebook.com/login.php?skip_api_login=1&api_key=162729813767876&kid_directed_site=0&app_id=162729813767876&signe

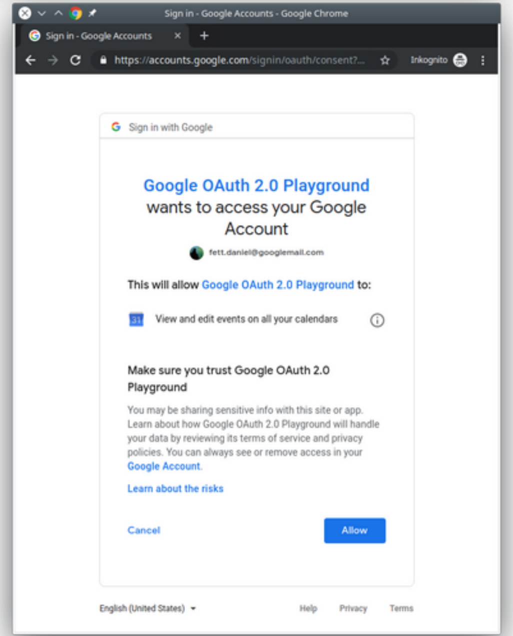
Melde dich an, um dein Facebook-Konto mit **TripAdvisor** zu verwenden.

E-Mail-Adresse oder Handynummer:

Passwort:

□ Immer bei **TripAdvisor** angemeldet bleiben

Facebook



Sign in - Google Accounts - Google Chrome

https://accounts.google.com/signin/oauth/consent?...

Google OAuth 2.0 Playground wants to access your Google Account

fett.daniel@googlemail.com

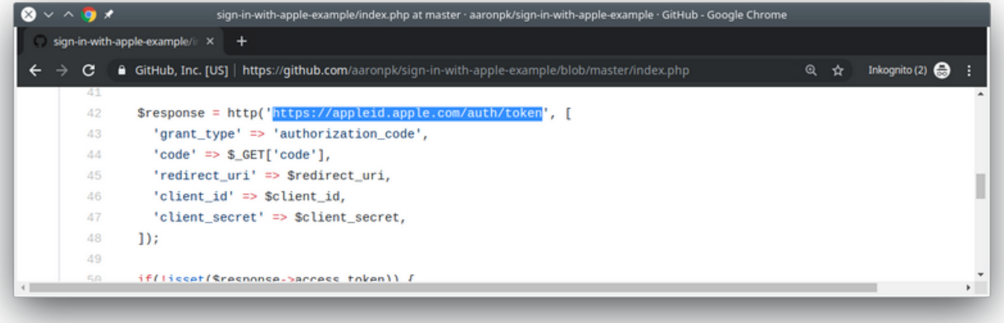
This will allow Google OAuth 2.0 Playground to:

View and edit events on all your calendars

Make sure you trust Google OAuth 2.0 Playground

You may be sharing sensitive info with this site or app. Learn about how Google OAuth 2.0 Playground will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Google



sign-in-with-apple-example/index.php at master · aaronpk/sign-in-with-apple-example · GitHub - Google Chrome

GitHub, Inc. [US] | https://github.com/aaronpk/sign-in-with-apple-example/blob/master/index.php

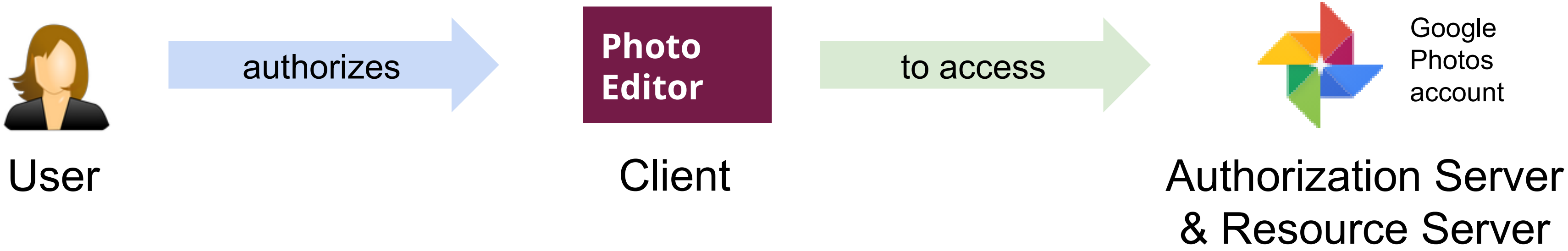
```
$response = http('https://appleid.apple.com/auth/token', [
  'grant_type' => 'authorization_code',
  'code' => $_GET['code'],
  'redirect_uri' => $redirect_uri,
  'client_id' => $client_id,
  'client_secret' => $client_secret,
]);

if(!isset($response->access_token)) {
```

Apple

OAuth is a standard
for federated authorization

Authorization



User

authorizes

Photo Editor

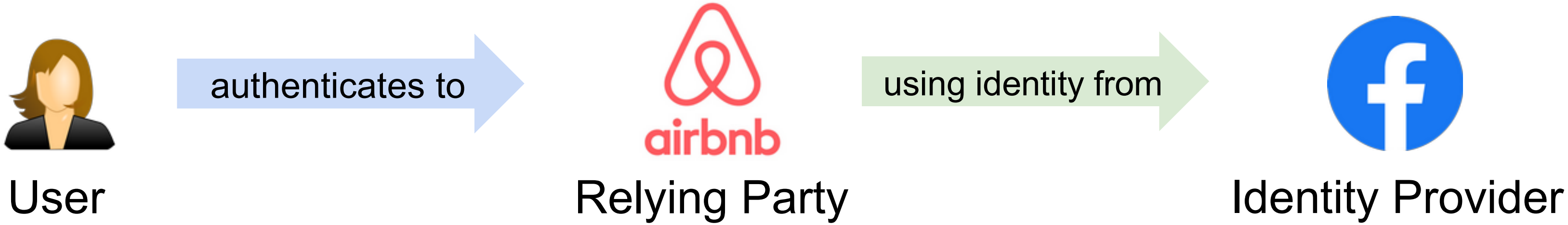
Client

to access

Google Photos account

Authorization Server & Resource Server

Authentication



User

authenticates to

airbnb

Relying Party

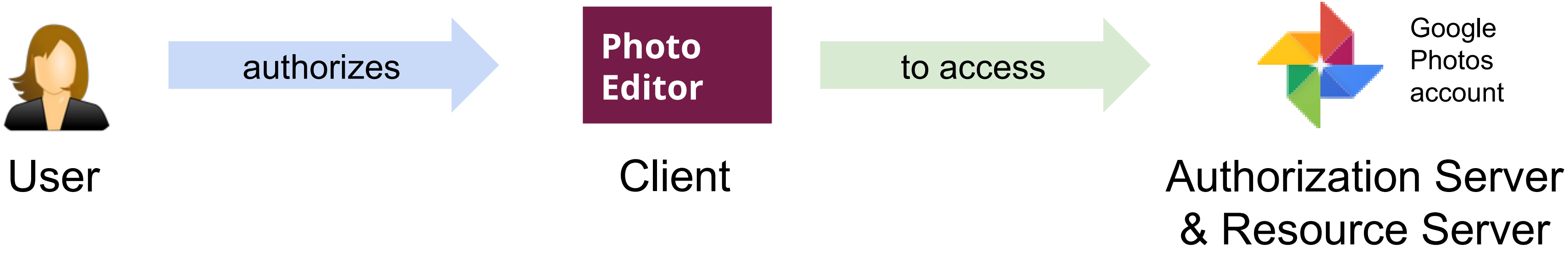
using identity from

Identity Provider

Say OAuth is an Authentication standard again.

I dare you. I double dare you.

Authorization OAuth



User

authorizes

Photo Editor

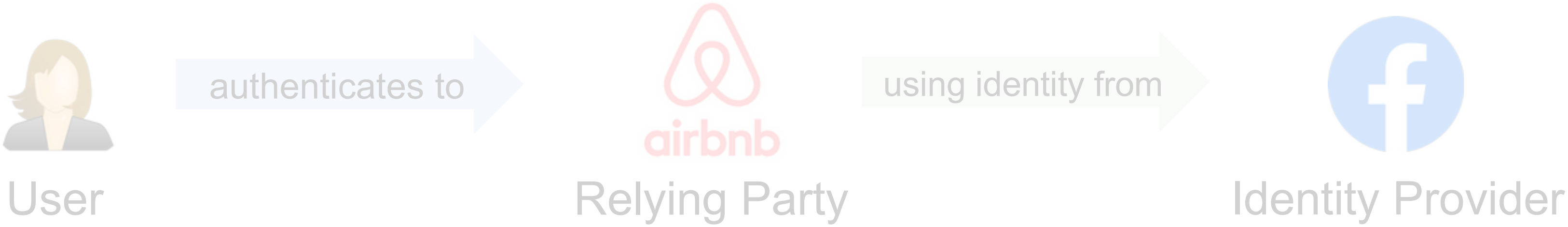
Client

to access

Google Photos account

Authorization Server & Resource Server

Authentication OAuth OpenID



User

authenticates to

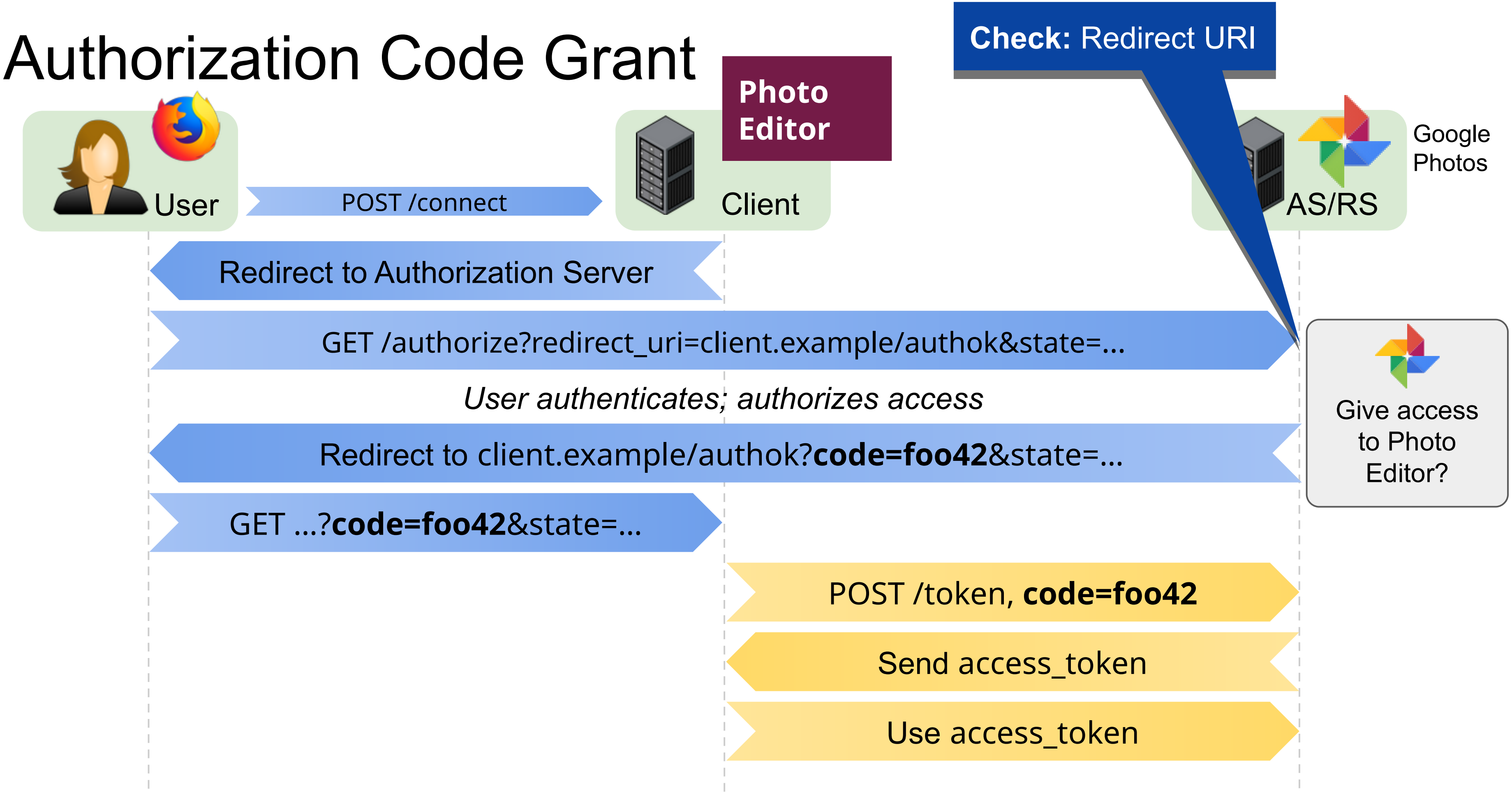
airbnb

Relying Party

using identity from

Identity Provider

Authorization Code Grant



Check: Redirect URI

Photo Editor

User

POST /connect

Client

Google Photos

AS/RS

Redirect to Authorization Server

GET /authorize?redirect_uri=client.example/authok&state=...

User authenticates; authorizes access

Give access to Photo Editor?

Redirect to client.example/authok?**code=foo42**&state=...

GET ...?**code=foo42**&state=...

POST /token, **code=foo42**

Send access_token

Use access_token

Authorization Code Grant

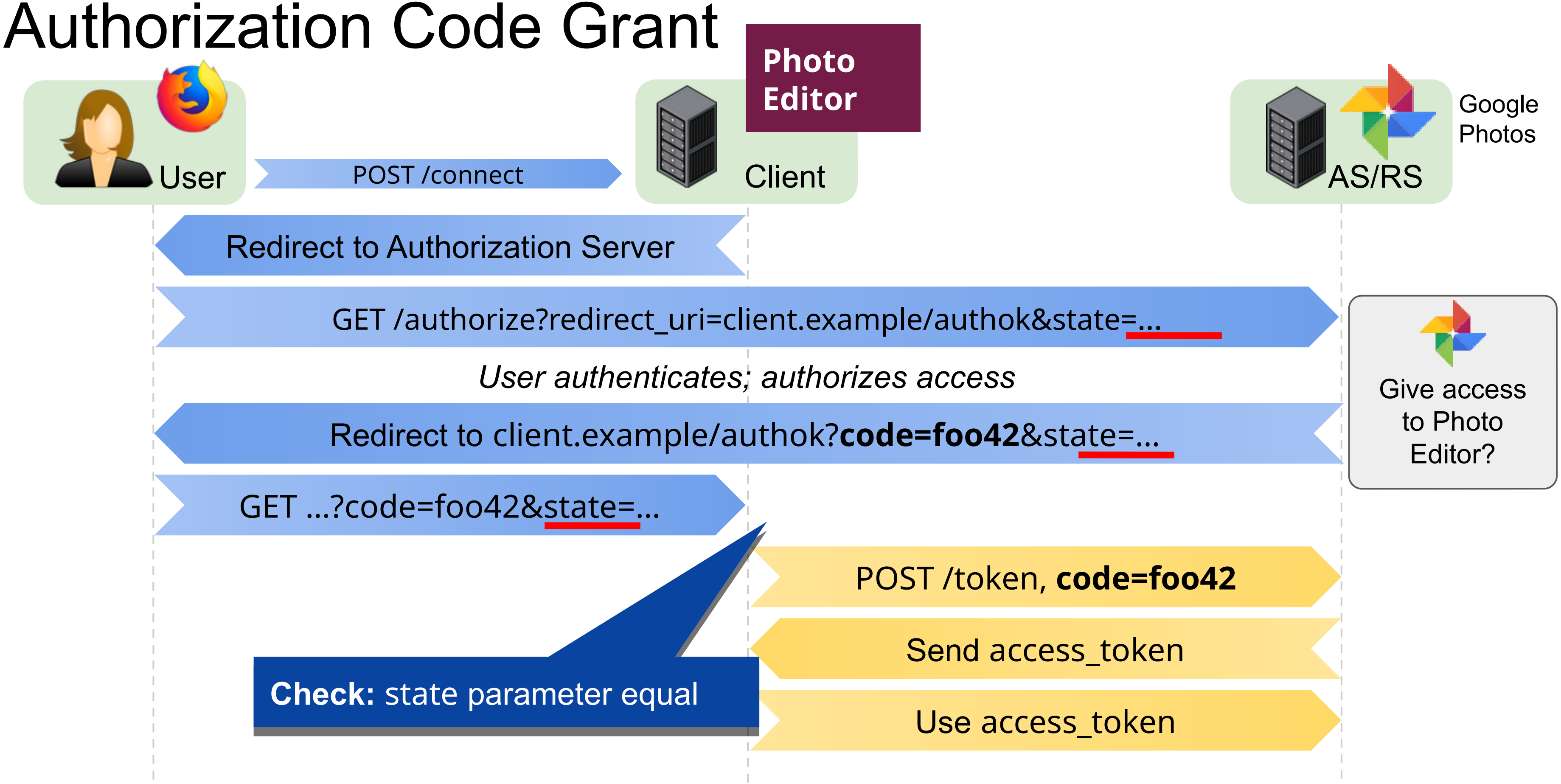


Photo Editor

User

POST /connect

Client

AS/RS

Google Photos

Redirect to Authorization Server

GET /authorize?redirect_uri=client.example/authok&state=...

User authenticates; authorizes access

Give access to Photo Editor?

Redirect to client.example/authok?**code=foo42**&state=...

GET ...?code=foo42&state=...

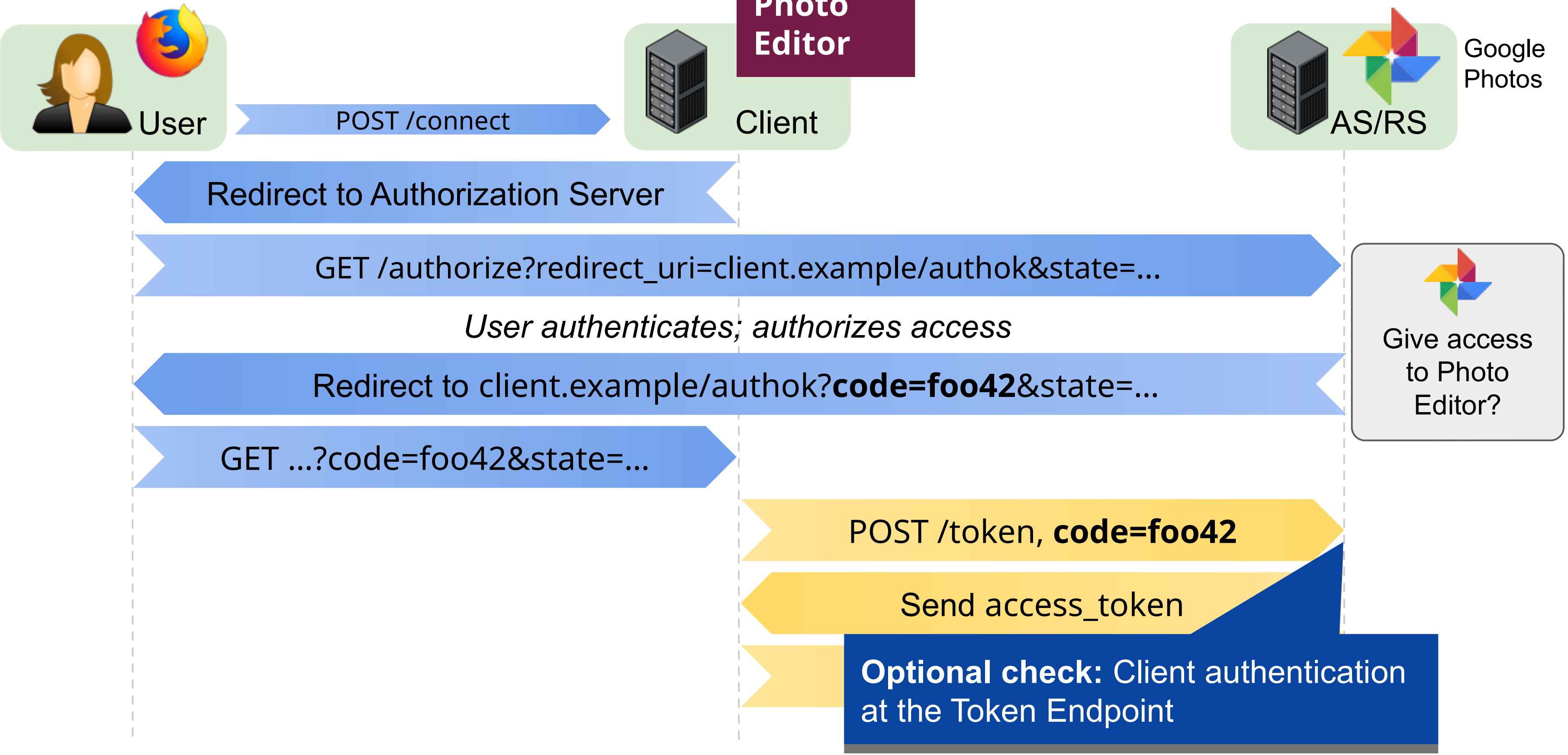
POST /token, **code=foo42**

Send access_token

Use access_token

Check: state parameter equal

Authorization Code Grant



User

Photo Editor

Client

AS/RS

Google Photos

POST /connect

Redirect to Authorization Server

GET /authorize?redirect_uri=client.example/authok&state=...

User authenticates; authorizes access

Give access to Photo Editor?

Redirect to client.example/authok?**code=foo42**&state=...

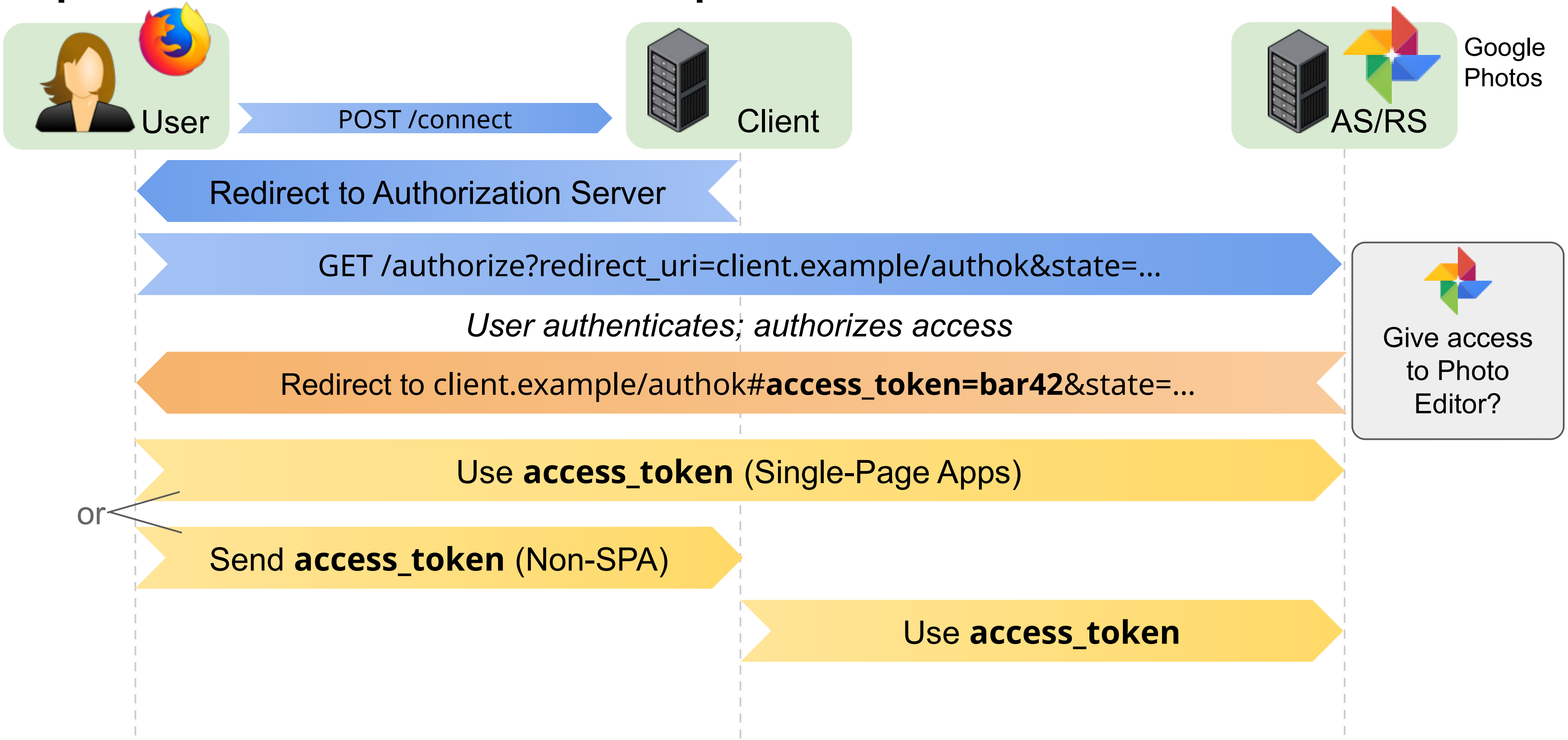
GET ...?code=foo42&state=...

POST /token, **code=foo42**

Send access_token

Optional check: Client authentication at the Token Endpoint

Implicit Grant — the “simpler OAuth”?



User

Client

AS/RS

Google Photos

POST /connect

Redirect to Authorization Server

GET /authorize?redirect_uri=client.example/authok&state=...

User authenticates; authorizes access

Give access to Photo Editor?

Redirect to client.example/authok#**access_token=bar42**&state=...

Use **access_token** (Single-Page Apps)

or

Send **access_token** (Non-SPA)

Use **access_token**

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

Challenge 2: High-Stakes Environments

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

OPEN BANKING

OpenID
Financial Grade API

stet

THE Berlin GROUP
A EUROPEAN STANDARDS INITIATIVE

OpenID
iGov Profile

mobile connect

CLOUD SIGNATURE CONSORTIUM

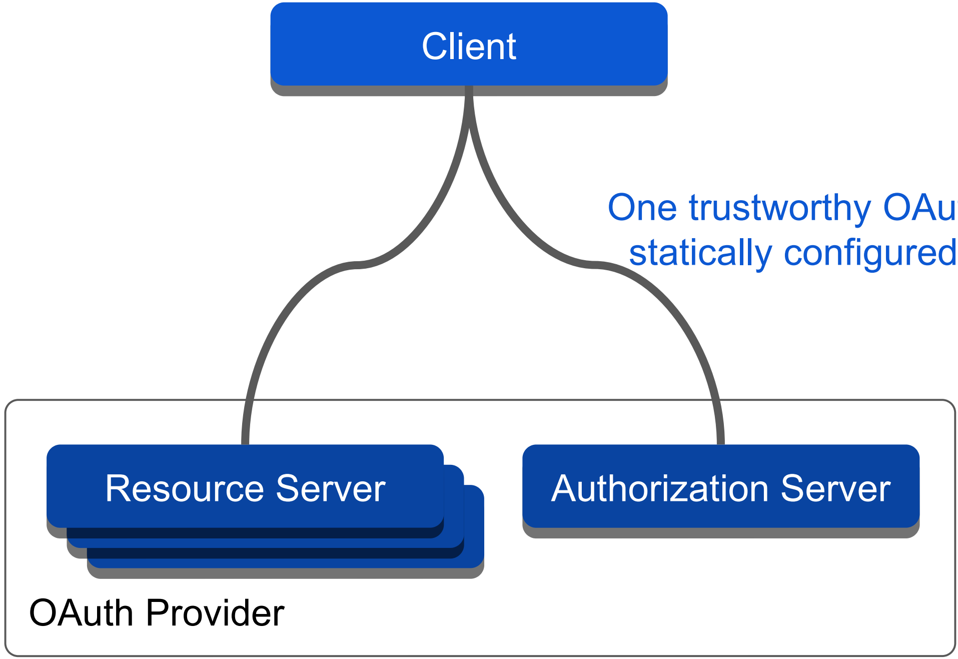
yes®

Also: eIDAS/QES (legally binding electronic signatures)

Far beyond the scope of the original security threat model!

Challenge 3: Dynamic and Complex Setups

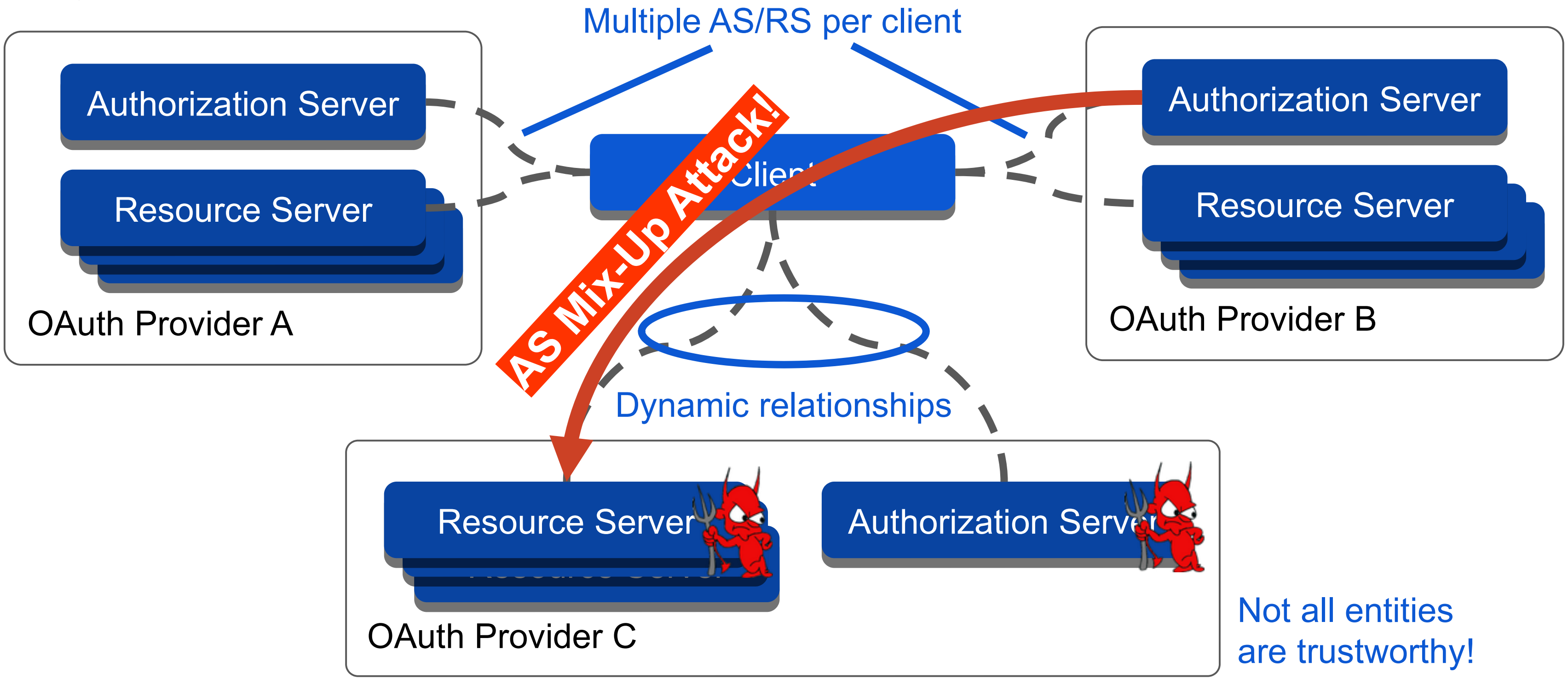
Originally anticipated:



One trustworthy OAuth provider, statically configured per client

Challenge 3: Dynamic and Complex Setups

Today:

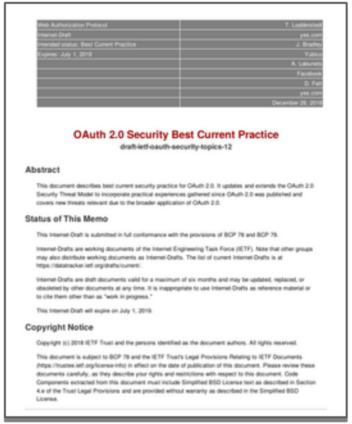


Not all entities are trustworthy!

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

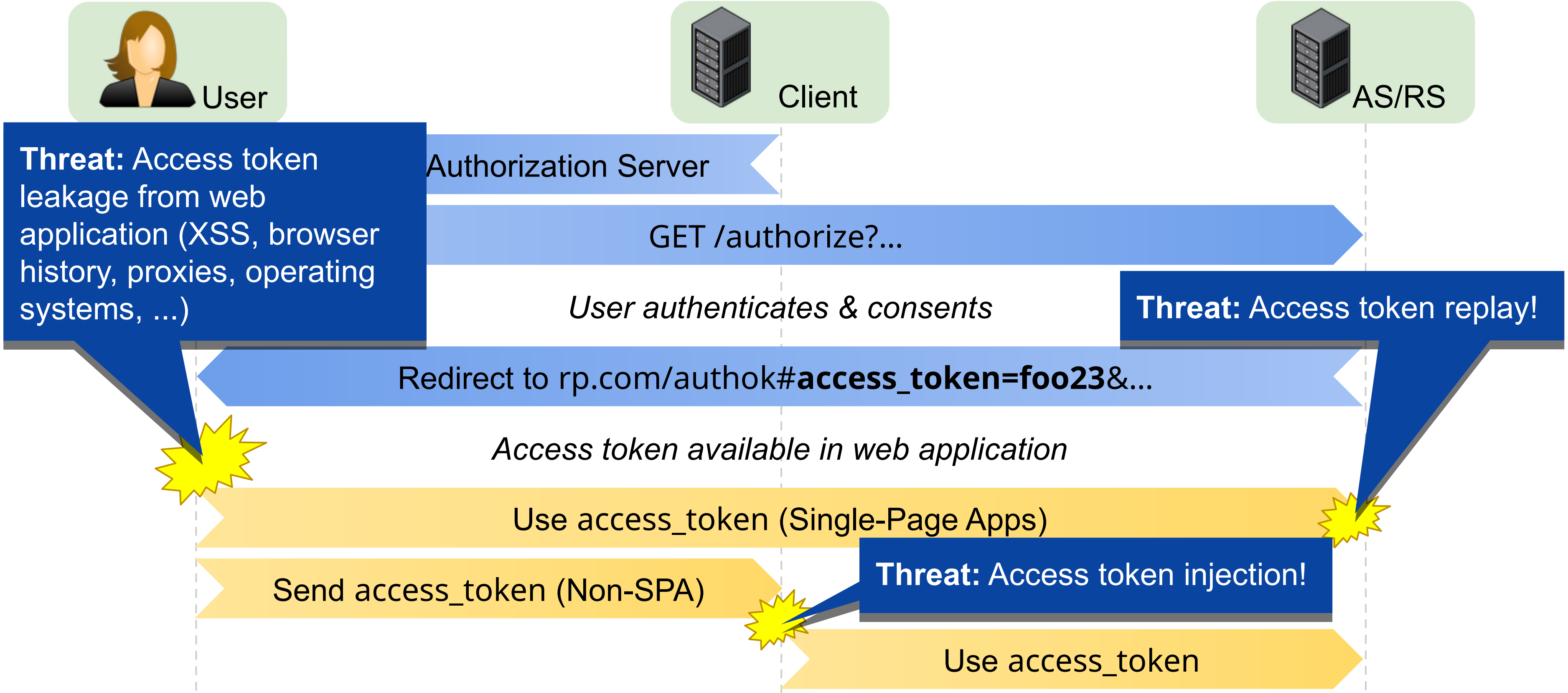
Input from **practice** and **formal analysis**

The Five Most Important

Recommendations

in the OAuth Security BCP

① Do not use the OAuth Implicit Grant any longer!



User

Client

AS/RS

Threat: Access token leakage from web application (XSS, browser history, proxies, operating systems, ...)

Authorization Server

GET /authorize?...

User authenticates & consents

Threat: Access token replay!

Redirect to rp.com/authok#**access_token=foo23**&...

Access token available in web application

Use access_token (Single-Page Apps)

Send access_token (Non-SPA)

Threat: Access token injection!

Use access_token

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.

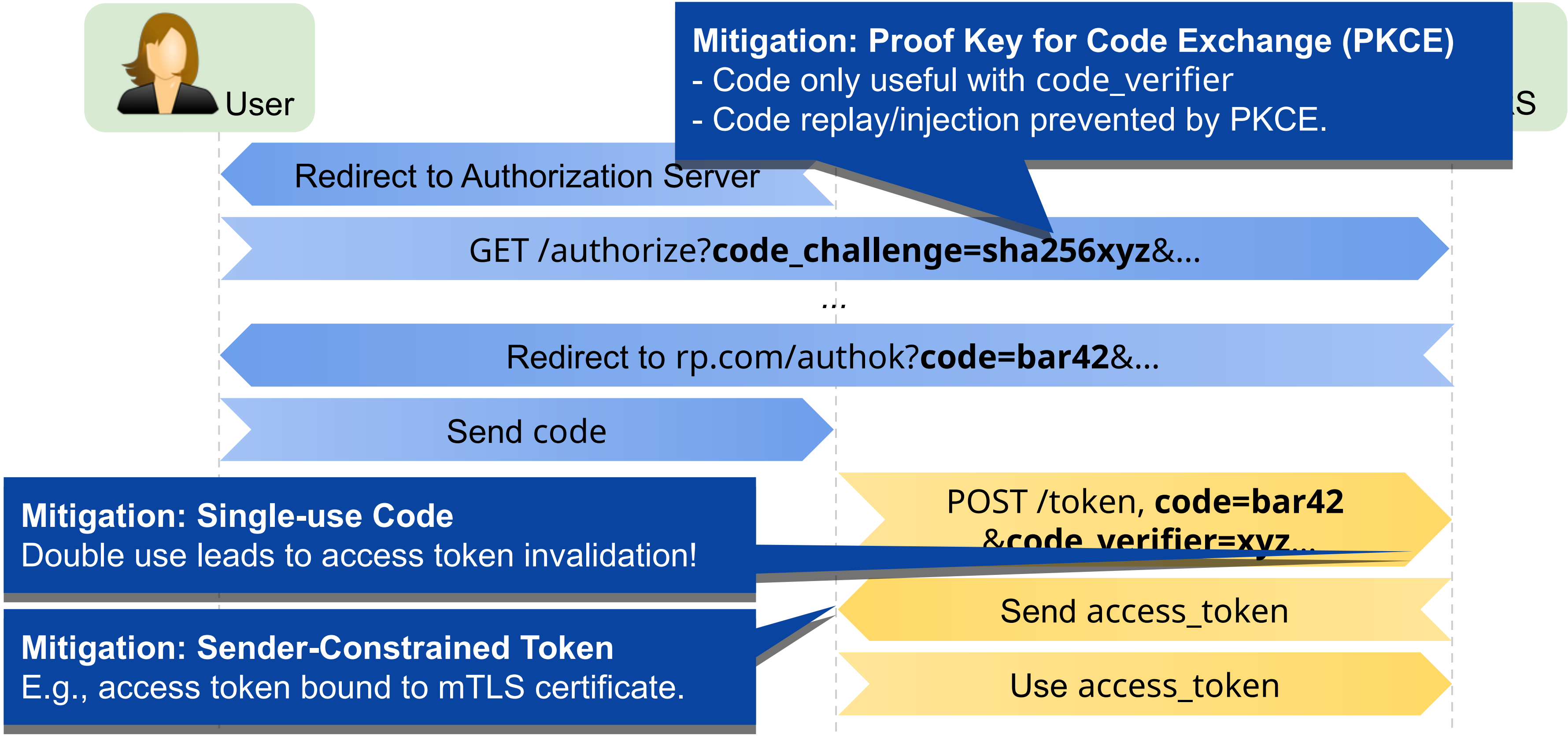
⇒ Not needed in 2019!

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!



User

S

Redirect to Authorization Server

GET /authorize?**code_challenge=sha256xyz**&...

...

Redirect to rp.com/authok?**code=bar42**&...

Send code

POST /token, **code=bar42**
&code_verifier=xyz...

Send access_token

Use access_token

Mitigation: Proof Key for Code Exchange (PKCE)
- Code only useful with code_verifier
- Code replay/injection prevented by PKCE.

Mitigation: Single-use Code
Double use leads to access token invalidation!

Mitigation: Sender-Constrained Token
E.g., access token bound to mTLS certificate.

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 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

③ Prevent CSRF Attacks!

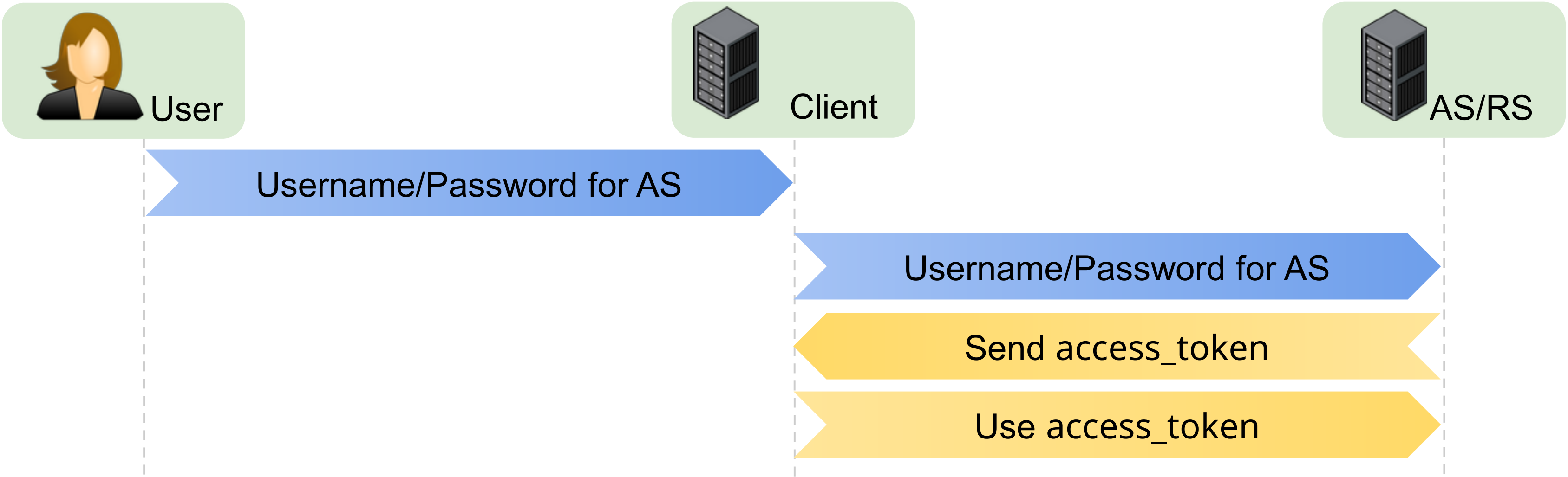
- CSRF attacks MUST be prevented
- RFC 6749 and RFC 6819 recommend use of `state` parameter
- Updated advice:
  - If PKCE is used, `state` is not needed for CSRF protection
  - `state` can again be used for application state

④ 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!

⑤ 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

What else?

- Prevent Mix-Up attacks!
- Protect Refresh Tokens!
- 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
  - Libraries (save time & money; battle-proven code)
  - 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
  - Don’t roll your own ~~crypto~~ OAuth!
  - Know your threat model

Formal Analysis

- Analysis based on formal models of systems
- “Offline testing of application logic”
  - Before writing a single line of code
  - Finds regressions caused by technological changes
- Successfully used for cryptographic protocols
  - Recently used for TLS 1.3
  - Helps to write precise specifications
  - Provides security guarantees - within limits
- Not common for web applications/standards yet
- → Proof for OAuth 2.0

Dr. Daniel Fett
Security Specialist at yes.com
mail@danielfett.de
@dfett42

REPLY
SPIKE

Links, latest BCP draft, papers, etc.:

`https://danielfett.de`

Now:
Q&A

OAuth Security Workshop
Trondheim, July 22-24
`https://osw2020.com`

seccamp
cologne

Security Barcamp
Cologne, Apr. 4-5
`https://sec.camp`

Research Papers

[Fett et al., 2014] Daniel Fett, Ralf Küsters, and Guido Schmitz. “An Expressive Model for the Web Infrastructure: Definition and Application to the BrowserID SSO System”.

[Fett et al., 2016] Daniel Fett, Ralf Küsters, and Guido Schmitz. “A Comprehensive Formal Security Analysis of OAuth 2.0”.

[Li et al., 2014] Wanpeng Li and Chris J. Mitchell. “Security issues in OAuth 2.0 SSO implementations”.

[Yang et al., 2016] Ronghai Yang et al. “Model-based Security Testing: An Empirical Study on OAuth 2.0 Implementations”.

[Shebab et al., 2015] Mohamed Shehab and Fadi Mohsen. “Towards Enhancing the Security of OAuth Implementations in Smart Phones”.

[Chen et al., 2014] Eric Y. Chen et al. “OAuth Demystified for Mobile Application Developers”.

[Wang et al., 2013] Rui Wang et al. “Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization”.

[Sun et al., 2012] San-Tsai Sun and Konstantin Beznosov. “The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems”.