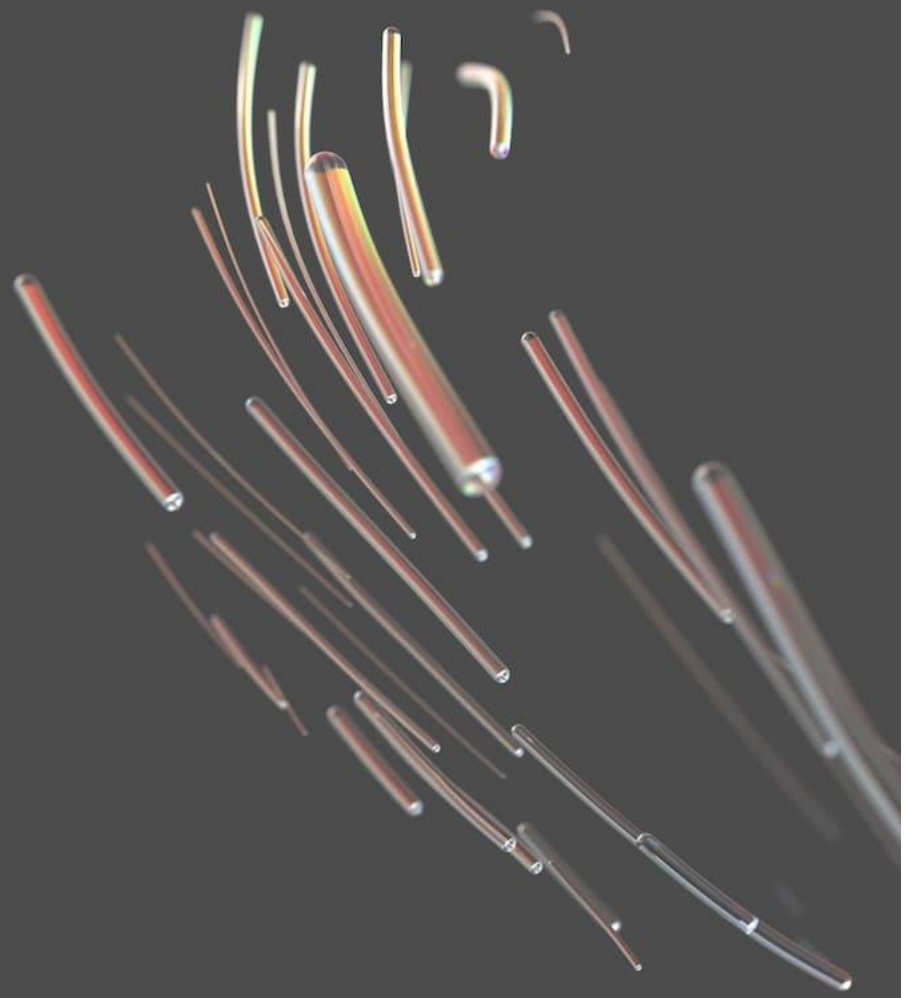


innoQ Technology Day / 20.11.2025

OAuth 2.1

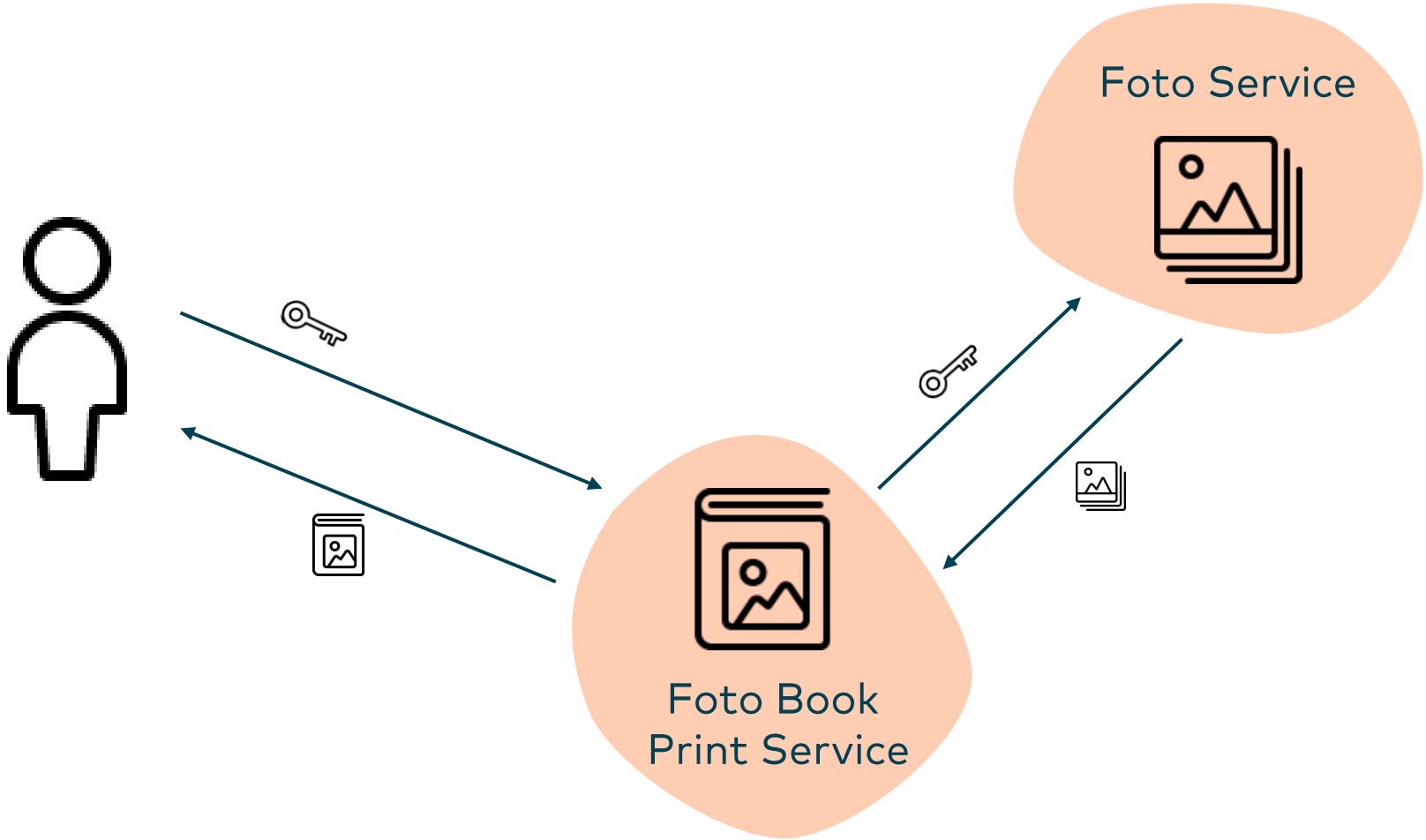
Evolution, Revolution, does it matter at all?

INNOQ

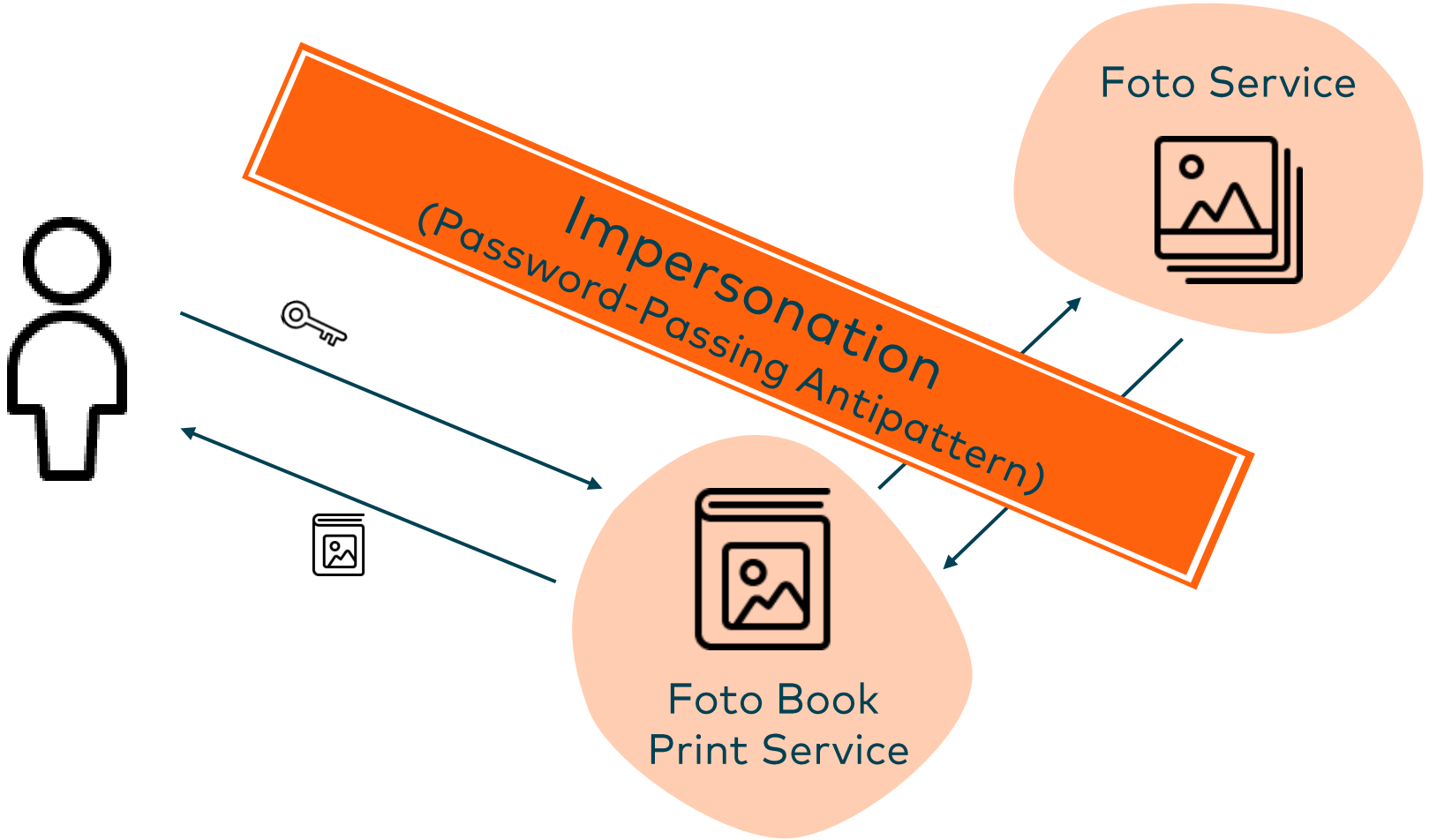
Dimitrij Drus
dadrus

Once upon a time ...

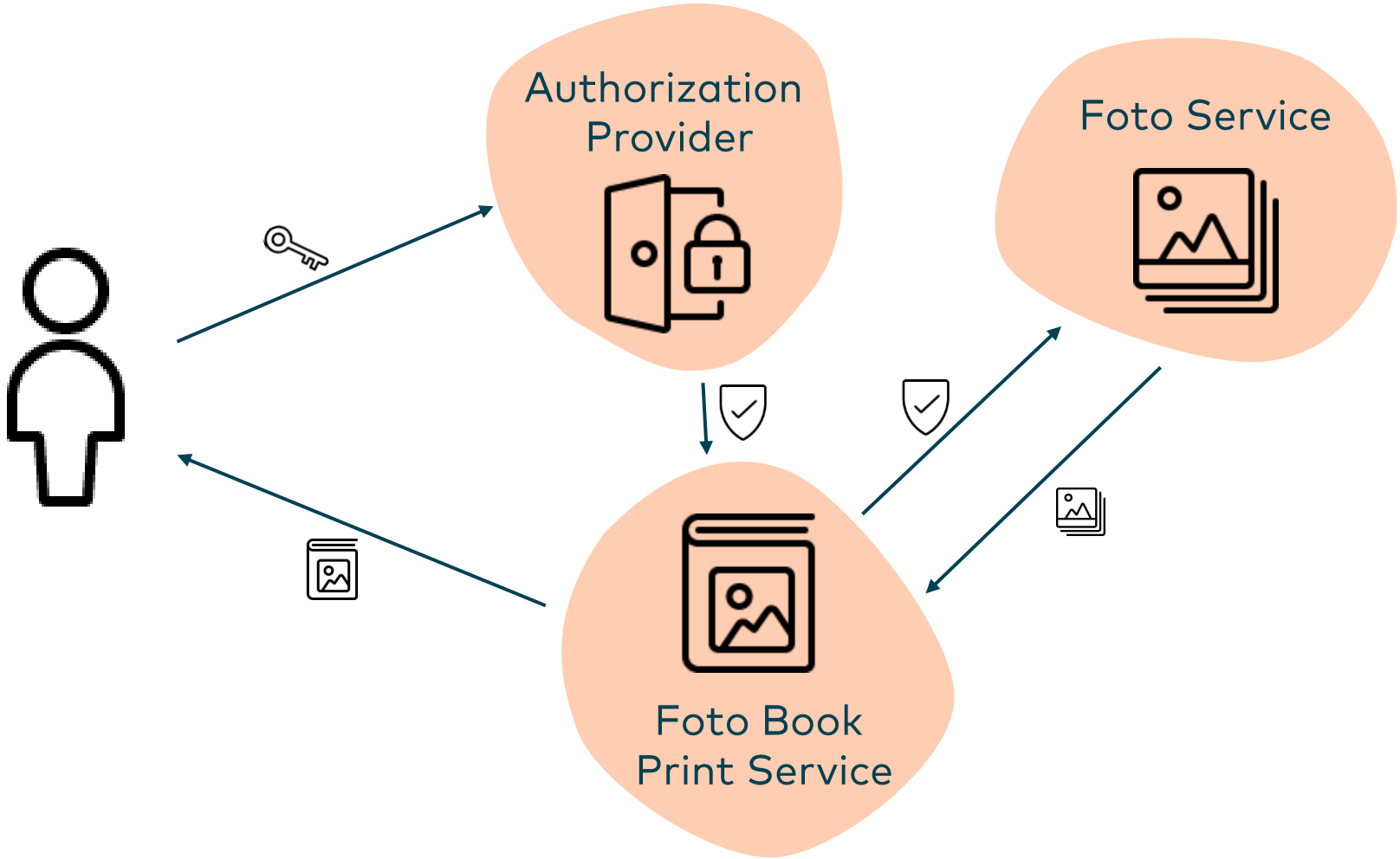
The pre-OAuth World



The pre-OAuth World



The OAuth 1.0 World



Internet Engineering Task Force (IETF)
Request for Comments: 5849
Category: Informational
ISSN: 2070-1721

E. Hammer-Lahav, Ed.
April 2010

The OAuth 1.0 Protocol

Abstract

OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

OAuth 1.0 Resource Request

```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Host: photos.example.net
Authorization: OAuth realm="Photos",
   oauth_consumer_key="dpf43f3p2l4k3l03",    # reference to the long-term shared secret of the client
   oauth_token="nnch734d00sl2jdk",           # reference to the consent related key
   oauth_signature_method="HMAC-SHA1",
   oauth_timestamp="137131202",              # allows the server to decide how long to keep nonces
   oauth_nonce="chapoH",                     # identification of the requests
   oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

- Signature key is a combination of the long-term shared secret between the client and the authorization server, and the key created by the authorization server for the user concent.
- „Token“ is actually a key reference
- Requests can be tracked back to a specifc client
- Replay protection

The OAuth 1.0 Challenges

- **Secret coordination**
  No defined mechanism for sharing or validating token secrets, leading to tightly coupled deployments
- **Transport-level ambiguity**
  Spec didn't mandate TLS; some believed signatures made it optional, which led to insecure implementations
- **Weak extensibility**
  Hard to adopt to mobile, native, browser based or IoT clients, or introduce new cryptographic methods
- **Interoperability**
  Inconsistent signature base string handling, parameter encoding, and cryptography use

Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: 5849
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft

October 2012

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

OAuth 1.0:

Protocol
Not Extensible - Single Flow
Client Bound Tokens
No Token Expiary

OAuth 2.0:

Framework
Extensible - Multiple Flows (4+)
Bearer Tokens
Token Expiary & Mgmt

OAuth 2.0 Resource Request

```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Host: photos.example.net
Authorization: Bearer mF_9.B5f-4.1JqM
```

- A string representing an access authorization issued to the client
- No protection at all
- Like cash, whoever holds the token is its rightful owner

The Road to OAuth 2.0

OAuth 2.0 (without Signatures) is Bad for the Web

By **Eran Hammer**
Wednesday September 15, 2010

OAuth 2.0 drops signatures and cryptography in favor of bearer tokens, similar to how cookies work. As the OAuth 2.0 editor, I'm associated with the OAuth 2.0 protocol more than most, and the assumption is that I agree with the decisions and directions the protocol is taking. While being an editor gives you a certain degree of temporary control over the specification, at the end decisions are made by the group as a whole (as they should).

And as a whole, the OAuth community has made a big mistake about the future direction of the protocol. A mistake that is going to make OAuth 2.0 a much less significant agent of change on the web.

OAuth Bearer Tokens are a Terrible Idea

By **Eran Hammer**
Wednesday September 29, 2010

My last post about the **lack of signature support in OAuth 2.0** stirred some very good discussions and showed wide support for including a signature mechanism in OAuth 2.0. The discussions on the working group focused on the best way to include signature support, and a bit on the different approached to signatures (more on that in another post).

However, the discussion failed to highlight the fundamental problem with supporting bearer tokens at all. In my **previous post** I suggested that bearer tokens over HTTPS are *fine for now*. I'd like to take that back and explain why OAuth bearer tokens are a really bad idea.

OAuth 2.0 and the Road to Hell

By **Eran Hammer**
Thursday July 26, 2012

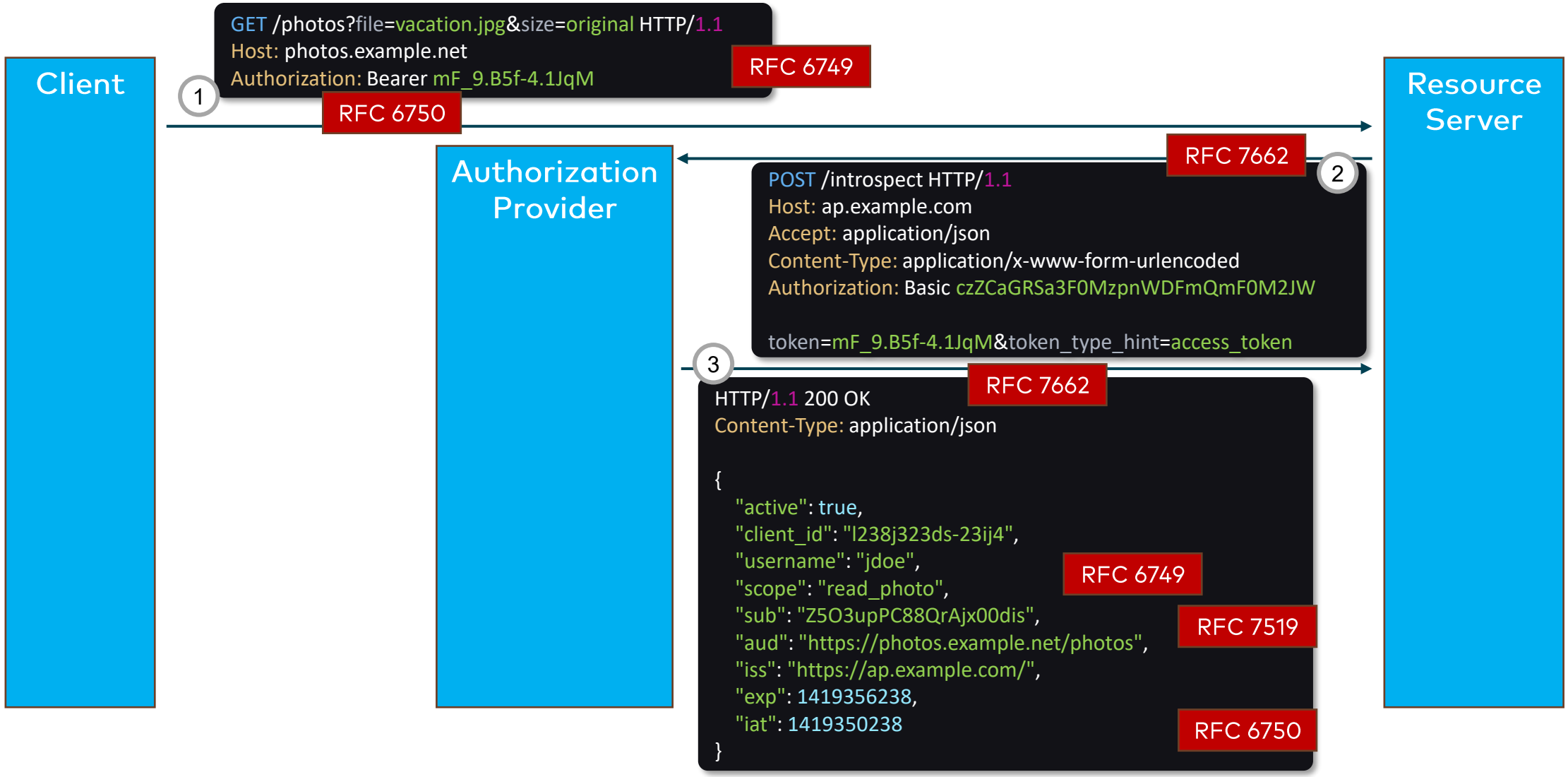
They say the road to hell is paved with good intentions. Well, that's **OAuth 2.0**.

Last month I reached the painful conclusion that I can no longer be associated with the OAuth 2.0 standard. I resigned my role as lead author and editor, **withdraw my name from the specification**, and left the working group. Removing my name from a document I have painstakingly labored over for three years and over two dozen drafts was not easy. Deciding to move on from an effort I have led for over five years was agonizing.

But OAuth2.0 is so simple... right?

PROPOSED STANDARD
Errata Exist
Stream: Internet Engineering Task Force (IETF)
RFC: 9728
Category: Standards Track
Published: April 2025
ISSN: 2070-1721
Authors: M.B. Jones, Self-Issued Consulting; P. Hunt, Independent Identity, Inc.; A. Parecki, Okta
RFC 9728
OAuth 2.0 Protected Resource Metadata
Abstract
This specification defines a metadata format that an OAuth 2.0 client or authorization server can use to obtain the information needed to interact with an OAuth 2.0 protected resource.
BEST CURRENT PRACTICE
Errata Exist
Stream: Internet Engineering Task Force (IETF)
RFC: 8725
BCP: 225
Updates: 7519
Category: Best Current Practice
Published: February 2020
ISSN: 2070-1721
Authors: Y. Sheffer, Intuit; D. Hardt; M. Jones, Microsoft
RFC 8725
JSON Web Token Best Current Practices
Abstract
Stream: Internet Engineering Task Force (IETF)
RFC: 9101
Category: Standards Track
Published: August 2021
ISSN: 2070-1721
Authors: N. Sakimura, NAT.Consulting; J. Bradley, Yubico; M. Jones, Microsoft
RFC 9101
The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)
Abstract
Stream: Internet Engineering Task Force (IETF)
RFC: 9701
Category: Standards Track
Published: January 2025
ISSN: 2070-1721
Authors: T. Lodderstedt, Ed., yes.com AG; V. Dzhuvinov, Connect2id Ltd.
RFC 9701
JSON Web Token (JWT) Response for OAuth Token Introspection
Abstract
This specification proposes an additional response secured by JSON Web Token (JWT) for OAuth 2.0 Token Introspection.
Stream: Internet Engineering Task Force (IETF)
RFC: 9700
BCP: 240
Updates: 6749, 6750, 6819
Category: Best Current Practice
Published: January 2025
ISSN: 2070-1721
Authors: T. Lodderstedt, SPRIND; J. Bradley, Yubico; A. Labunets, Independent Researcher; D. Fett, Authlete
RFC 9700
Best Current Practice for OAuth 2.0 Security
Abstract
This document describes best current security practice for OAuth 2.0. It updates and extends the threat model and security advice given in RFCs 6749, 6750, and 6819 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. Further, it deprecates some modes of operation that are deemed less secure or even insecure.
Stream: Internet Engineering Task Force (IETF)
RFC: 9470
Category: Standards Track
Published: September 2023
ISSN: 2070-1721
Authors: V. Bertocci, Auth0/Okta; B. Campbell, Ping Identity
RFC 9470
OAuth 2.0 Step Up Authentication Challenge Protocol
Abstract
It is not uncommon for resource servers to require different authentication strengths or recentness according to the characteristics of a request. This document introduces a mechanism that resource servers can use to signal to a client that the authentication event associated with the access token of the current request does not meet its authentication requirements and, further, how to meet them. This document also codifies a mechanism for a client to request that an authorization server achieve a specific authentication strength or recentness when processing an authorization request.
Category: Standards Track
Published: September 2021
ISSN: 2070-1721
Authors: T. Lodderstedt, yes.com; B. Campbell, Ping Identity; N. Sakimura, NAT.Consulting; D. Tonge, Moneyhub Financial Technology; F. Skokan, Auth0
RFC 9126
OAuth 2.0 Pushed Authorization Requests
Abstract
This document defines the pushed authorization request (PAR) endpoint, which allows clients to push the payload of an OAuth 2.0 authorization request to the authorization server via a direct request and provides them with a request URI that is used as reference to the data in a subsequent call to the authorization endpoint.
Stream: Internet Engineering Task Force (IETF)
RFC: 8707
Category: Standards Track
Published: February 2020
ISSN: 2070-1721
Authors: B. Campbell, Ping Identity; J. Bradley, Yubico; H. Tschofenig, Arm Limited
RFC 8707
Resource Indicators for OAuth 2.0
Abstract
This document specifies an extension to the OAuth 2.0 Authorization Framework defining request parameters that enable a client to explicitly signal to an authorization server about the identity of the protected resource(s) to which it is requesting access.
Stream: Internet Engineering Task Force (IETF)
RFC: 9396
Category: Standards Track
Published: May 2023
ISSN: 2070-1721
Authors: T. Lodderstedt, yes.com; J. Richer, Bespoke Engineering; B. Campbell, Ping Identity
RFC 9396
OAuth 2.0 Rich Authorization Requests
Abstract
This document specifies a new parameter authorization_details that is used to carry fine-grained authorization data in OAuth messages.

The Access Token Usage Case



But what's new in OAuth 2.1?

Updated by: 9700
Internet Engineering Task Force (IETF)
Request for Comments: 6819
Category: Informational
ISSN: 2070-1721

INFORMATIONAL
Errata Exist

T. Lodderstedt, Ed.
Deutsche Telekom AG
M. McGloin
IBM
P. Hunt
Oracle Corporation
January 2013

OAuth 2.0 Threat Model and Security Considerations

Abstract

This document gives additional security considerations for (
beyond those in the OAuth 2.0 specification, based on a com
threat model for the OAuth 2.0 protocol.

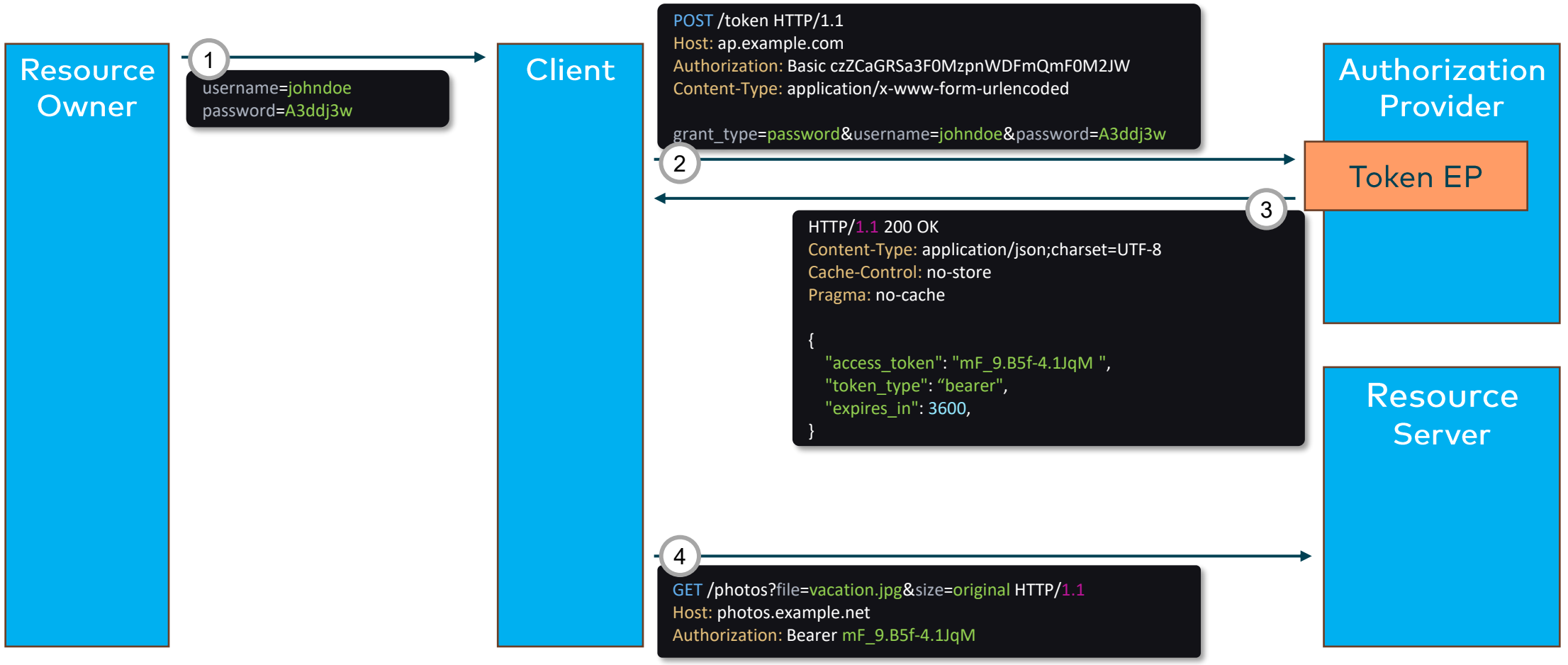
Stream: Internet Engineering Task Force (IETF)
RFC: 9700
BCP: 240
Updates: 6749, 6750, 6819
Category: Best Current Practice
Published: January 2025
ISSN: 2070-1721
Authors: T. Lodderstedt, *SPRIND*; J. Bradley, *Yubico*; A. Labunets, *Independent Researcher*; D. Fett, *Authlete*

RFC 9700
Best Current Practice for OAuth 2.0 Security

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the threat model and security advice given in RFCs 6749, 6750, and 6819 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. Further, it deprecates some modes of operation that are deemed less secure or even insecure.

Resource Owner Password Credentials Grant



Resource Owner Password Credentials Grant

RFC 6819

- Legacy/migration reasons
- Maintains the password anti-pattern
- User does not have control over the authorization process
- Can affect resources beyond the scope of the client

RFC 9700

- MUST NOT be used
- not designed to work with multi-factor authentication protocols

Resource Owner Password Credentials Grant

RFC 6819

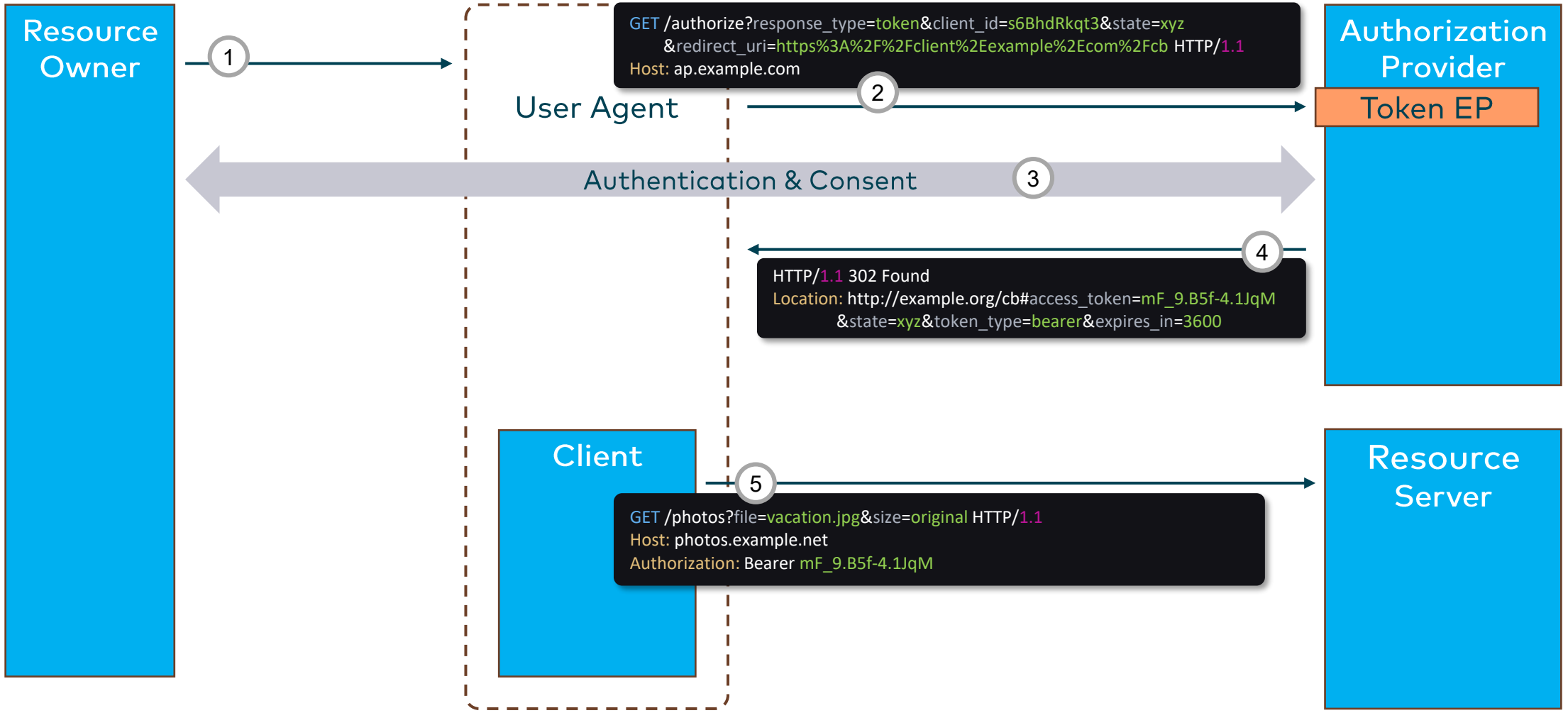
- Legacy/migration req
- Maintains the passwor
- User does not have control o
  authorization process
- Can affect resources beyond the
  scope of the client

RFC 9700

- MUST NOT be used
- not designed to work with multi-factor authentication protocols

Banned from OAuth 2.1

Implicit Grant



Implicit Grant

RFC 6819

- access token is directly returned to the client (No CORS in those days)
- Token can leak from browser history
- Malicious client fraud
- Manipulation of client
- CSRF attack against the redirect-uri
- User identity abuse

RFC 9700

- No sender-constraints
  - vulnerable to access token leakage
  - vulnerable to access token replay
- Clients SHOULD NOT use the implicit grant
- Clients MUST prevent CSRF attacks

Implicit Grant

RFC 6819

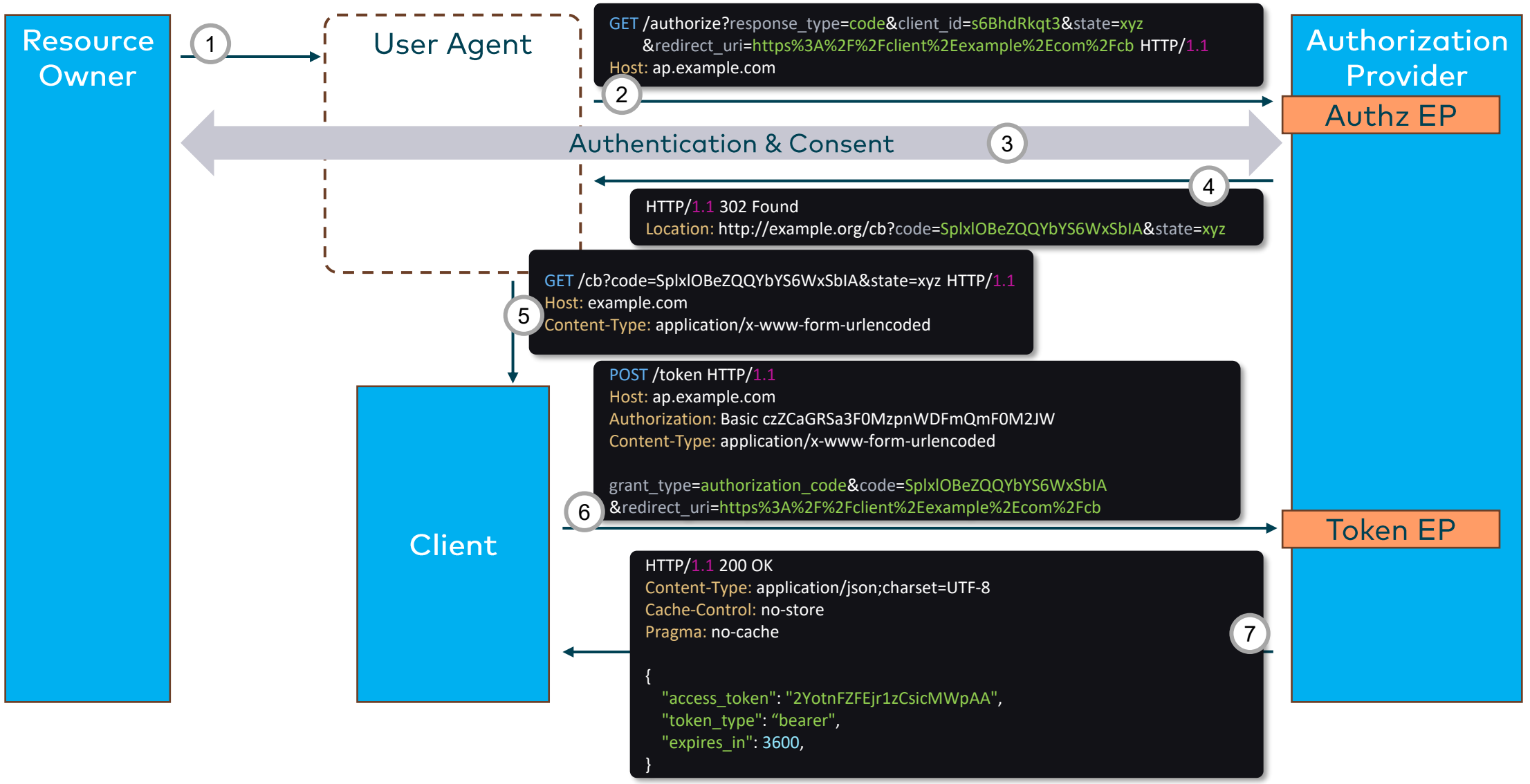
- access token is directly returned to the client (No CORS in these days)
- Token can leak from b istory
- Malicious client fraud
- Manipulation of client
- CSRF attack against the redirect
- User identity abuse

RFC 9700

- No sender-constraints
  - vulnerable to access token leakage
  - vulnerable to access token replay
- Clients SHOULD NOT use the implicit grant
- lients MUST prevent CSRF attacks

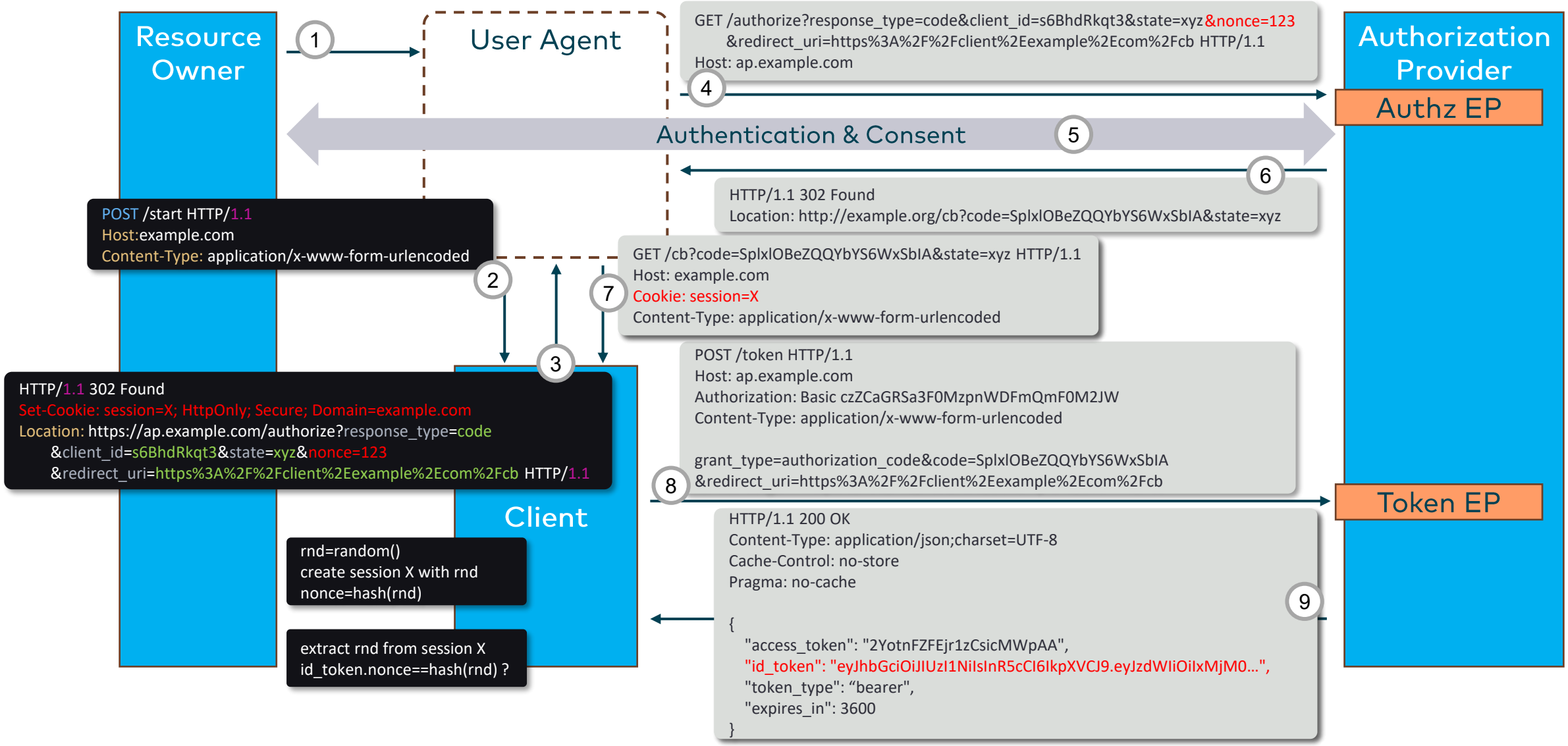
Banned from OAuth 2.1

Authorization Code Grant



No worries, Authorization Code Grant remains

OIDC to the Rescue



Internet Engineering Task Force (IETF)
Request for Comments: 8252
BCP: 212
Updates: 6749
Category: Best Current Practice
ISSN: 2070-1721

BEST CURRENT PRACTICE
Errata Exist

W. Denniss
Google
J. Bradley
Ping Identity

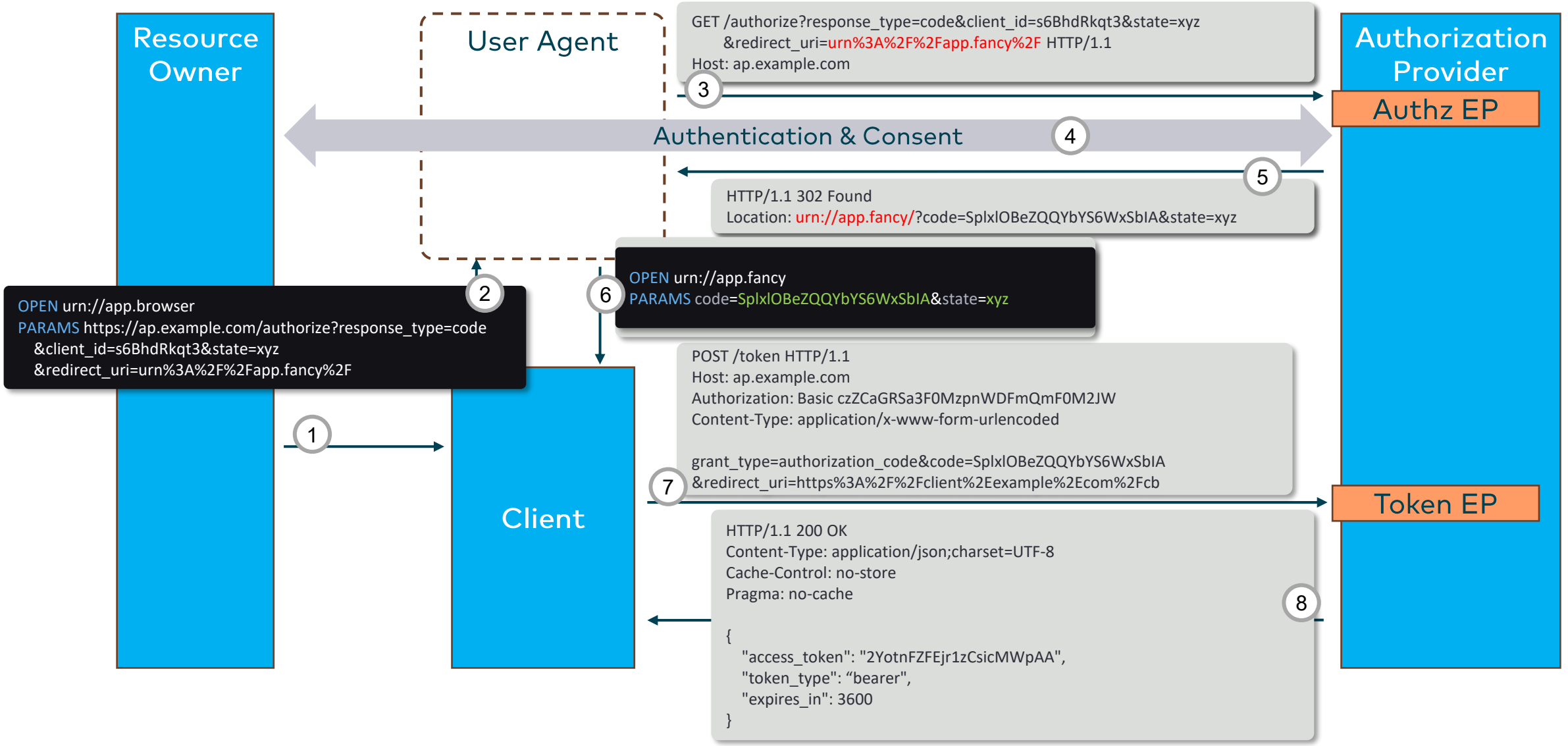
October 2017

OAuth 2.0 for Native Apps

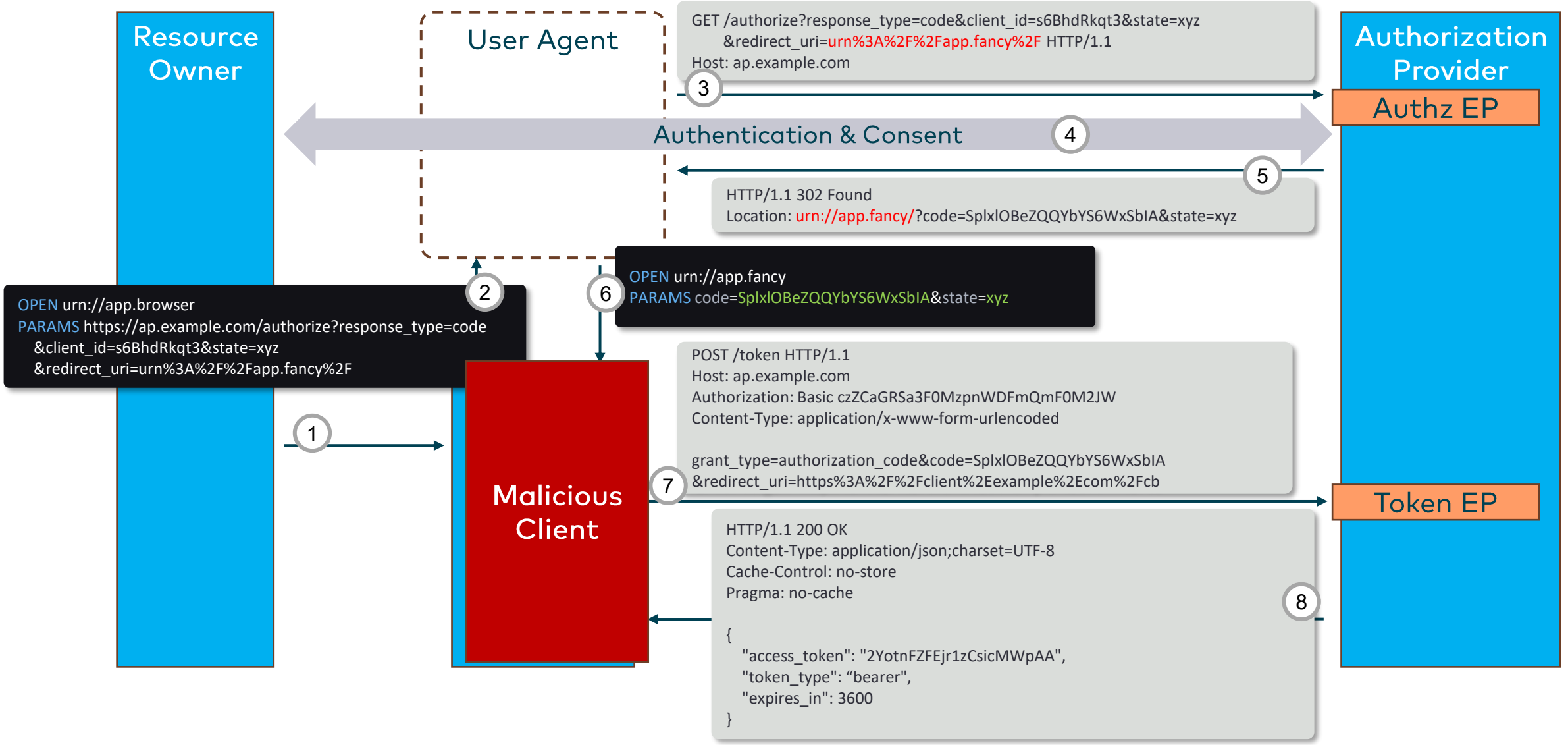
Abstract

OAuth 2.0 authorization requests from native apps should only be made through external user-agents, primarily the user's browser. This specification details the security and usability reasons why this is the case and how native apps and authorization servers can implement this best practice.

Mobile and Native Apps



Mobile and Native Apps



Internet Engineering Task Force (IETF)
Request for Comments: 7636
Category: Standards Track
ISSN: 2070-1721

PROPOSED STANDARD
Errata Exist

N. Sakimura, Ed.
Nomura Research Institute
J. Bradley
Ping Identity
N. Agarwal
Google

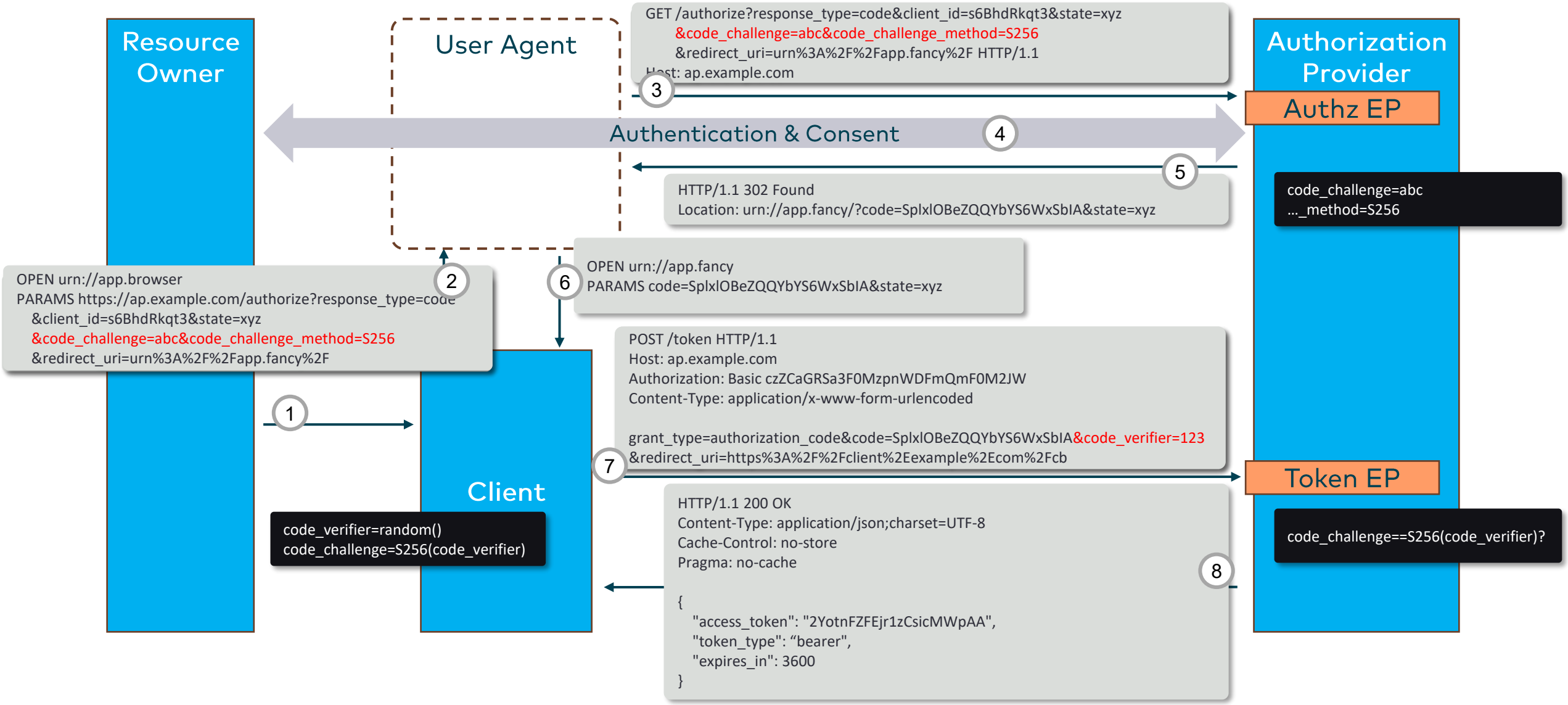
September 2015

Proof Key for Code Exchange by OAuth Public Clients

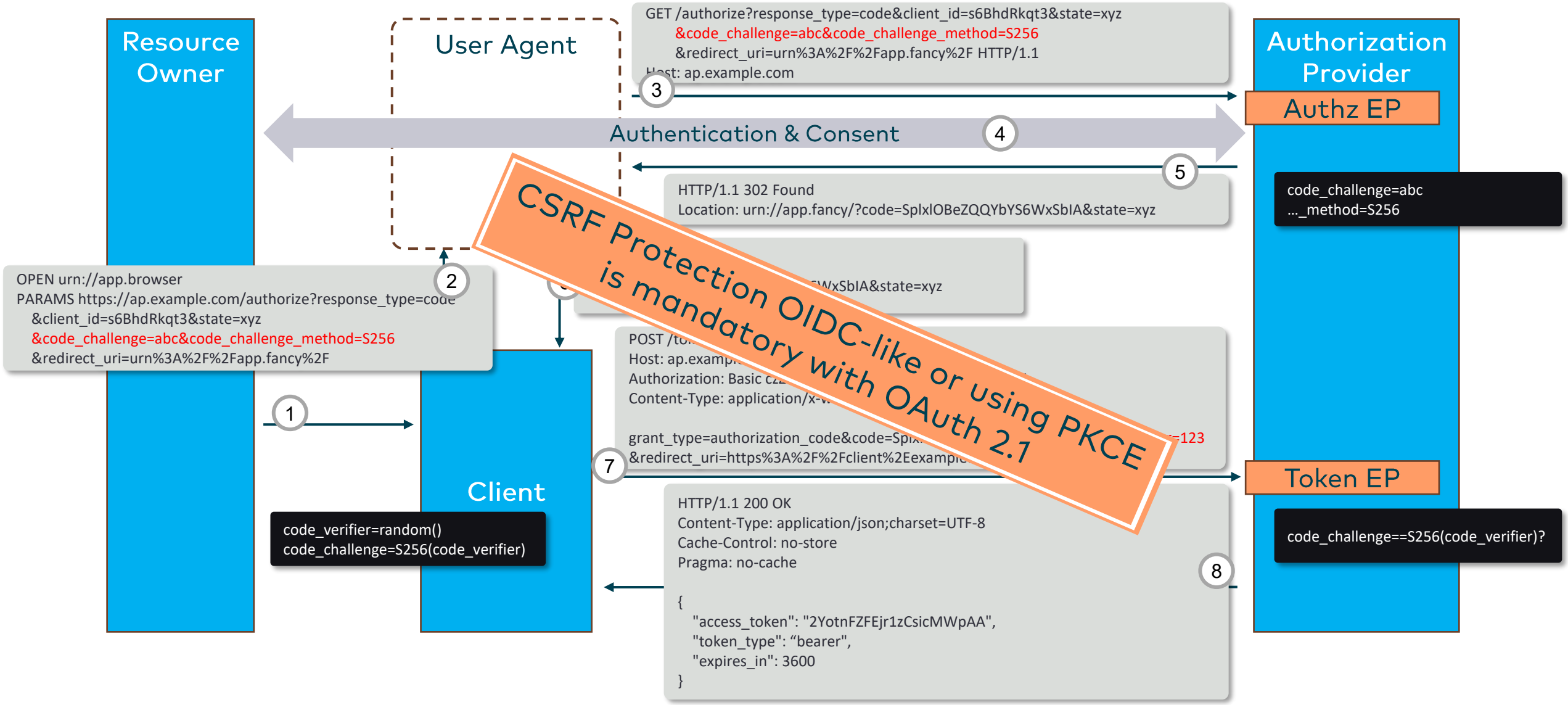
Abstract

OAuth 2.0 public clients utilizing the Authorization Code Grant are susceptible to the authorization code interception attack. This specification describes the attack as well as a technique to mitigate against the threat through the use of Proof Key for Code Exchange (PKCE, pronounced "pixy").

PKCE to the Rescue

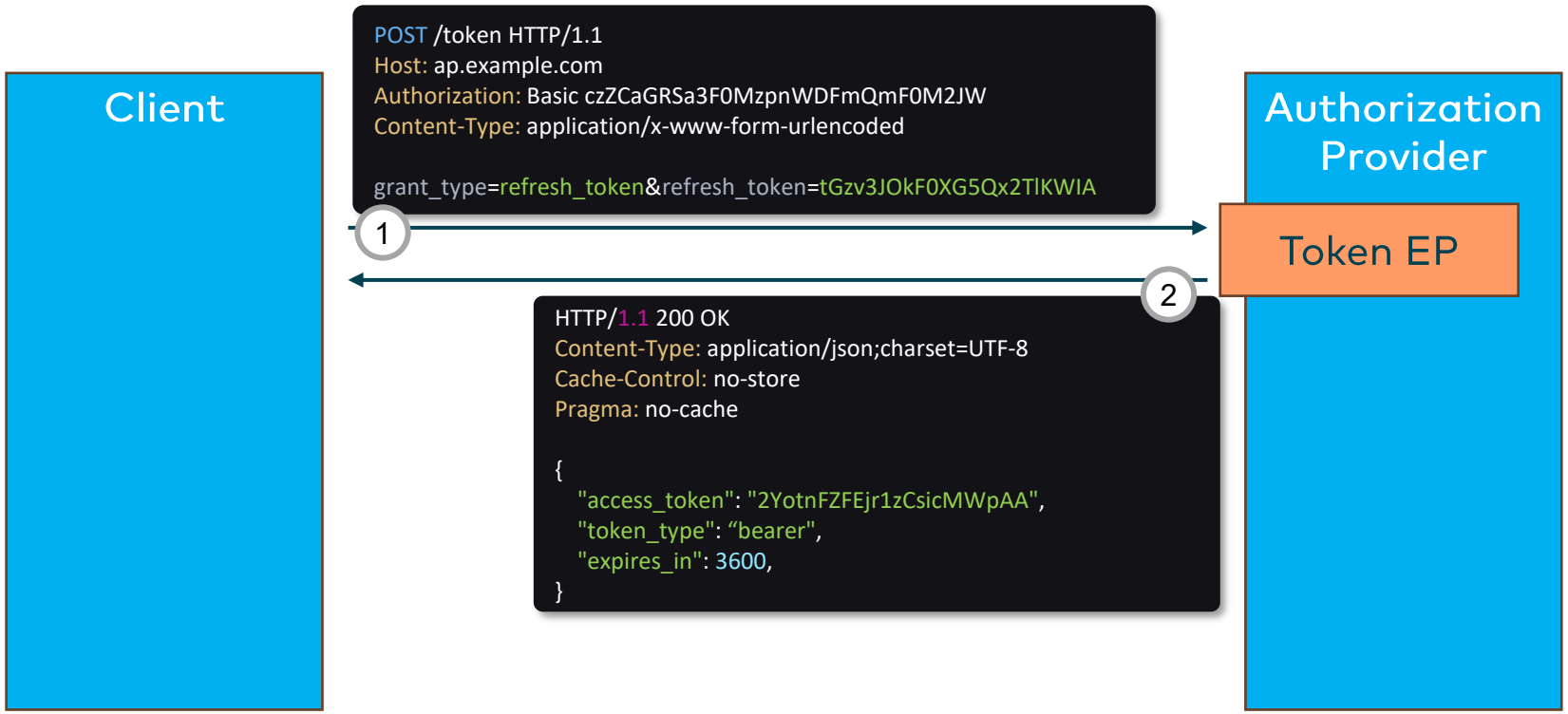


PKCE to the Rescue



Refresh-Tokens Recap

Refresh-Token Grant



Refresh-Token Grant

Token handling
in browser based apps

Malicious JavaScript Code

Has the same privileges as the legitimate application code. Thus, it

- can steal data from the current page,
- interact with other same-origin browsing contexts,
- send requests to a backend from within the application's origin,
- steal data from origin-based storage mechanisms (e.g., LocalStorage, IndexedDB),
- tamper with the regular execution flow of the application, like removing or overriding event listeners, modifying the behavior of built-in functions (prototype pollution), etc.

Malicious JavaScript Consequences

- **Client Hijacking**
  This effectively allows the attacker to perform any operations that the legitimate client application can perform.
- **Exploiting Stolen Refresh Tokens**
  Refresh tokens can be abused by running a Refresh Token flow with the authorization server. The response contains an access token giving the attacker the ability to access protected resources, effectively enabling long-term impersonation of the user to resource servers.
- **Exploiting Stolen Access Tokens**
  Attacker can send arbitrary requests to any resource server that considers the access token to be valid. In essence, abusing a stolen access token enables short-term impersonation of the user to resource servers until the token expires or is revoked.

Architecture Patterns

- **Token Mediating Backend**
  A JavaScript application relying on a backend component for handling OAuth responsibilities, but calling resource servers directly using the access token

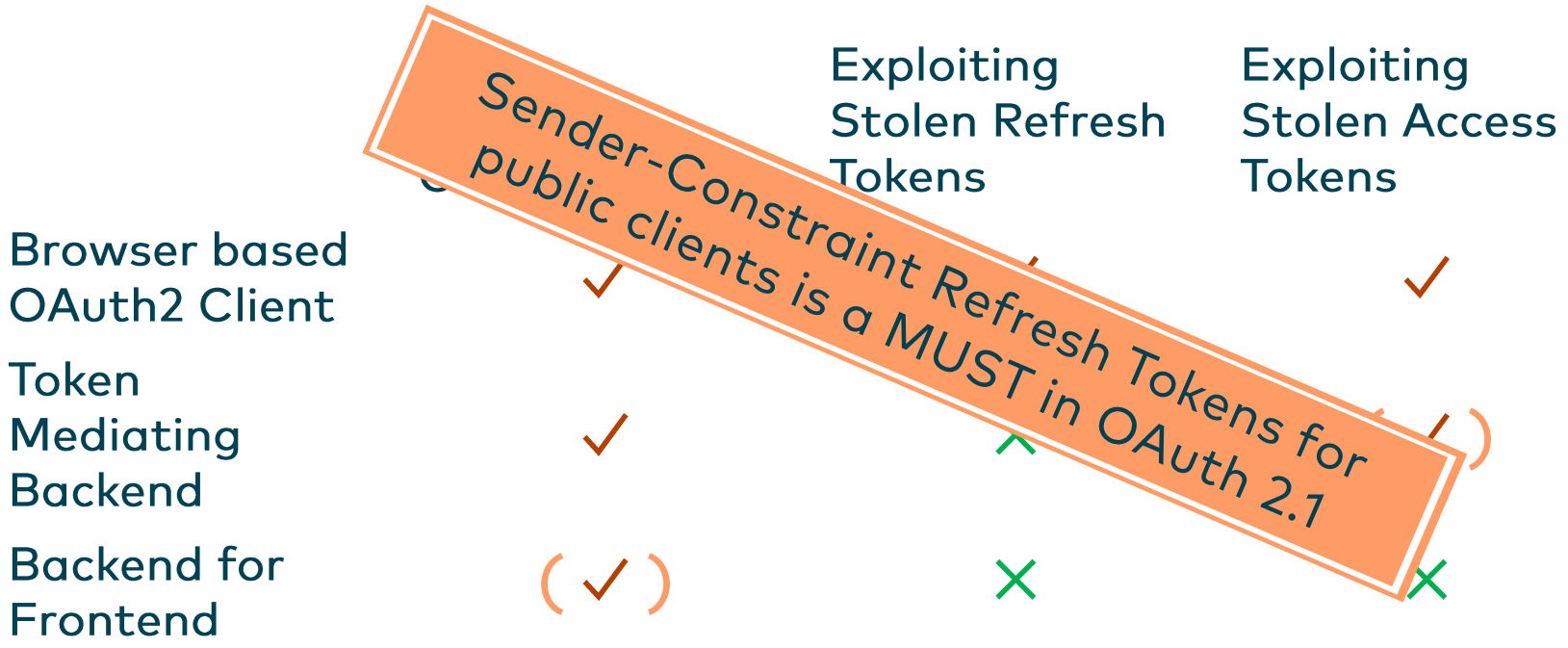
- **Backend for Frontend**
  A JavaScript application that relies on a backend component for handling OAuth responsibilities and proxies all requests through that backend component

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

Architecture Patterns Assessment

	Client Hijacking	Exploiting Stolen Refresh Tokens	Exploiting Stolen Access Tokens
Browser based OAuth2 Client	✓	✓	✓
Token Mediating Backend	✓	✗	(✓)
Backend for Frontend	(✓)	✗	✗

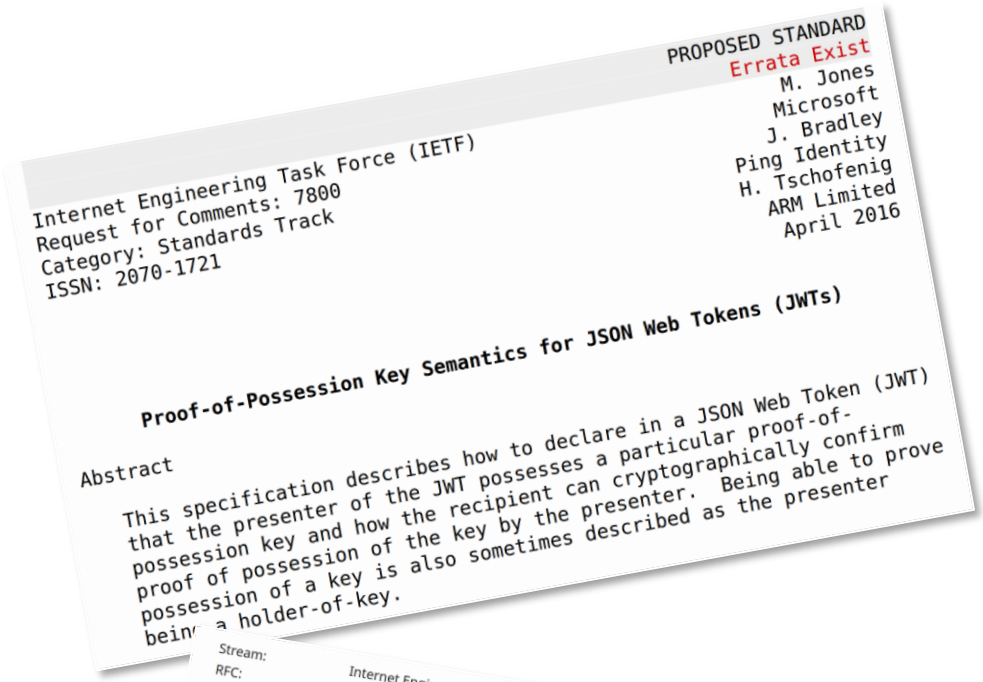
Architecture Patterns Assessment



„Bearer Tokens are a Terrible Idea“

Indeed, they are...

- Circle CI: Stolen Application Keys (2022)
- JetBrains: OAuth Token Vulnerability (2024)
- Hugging Face: Exposed Access Tokens via Unauthorized Access (2024)
- Microsoft Midnight Blizzard: OAuth App Abuse (2024)
- Internet Archive: Token Exploit (2024)
- Cloudflare: Authentication Token Theft (2024)
- New York Times: Over-Privileged GitHub Token (2024)
- Salesforce & Drift: OAuth Token Abuse (2025)
- Microsoft Entra ID: nOAuth Vulnerability (2025)
- Red Hat: Exposed GitLab Tokens (2025)
- ...



Internet Engineering Task Force (IETF)
Request for Comments: 7800
Category: Standards Track
ISSN: 2070-1721

PROPOSED STANDARD
Errata Exist

M. Jones
Microsoft
J. Bradley
Ping Identity
H. Tschofenig
ARM Limited
April 2016

Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

Abstract

This specification describes how to declare in a JSON Web Token (JWT) that the presenter of the JWT possesses a particular proof-of-possession key and how the recipient can cryptographically confirm proof of possession of the key by the presenter. Being able to prove possession of a key is also sometimes described as the presenter being a holder-of-key.

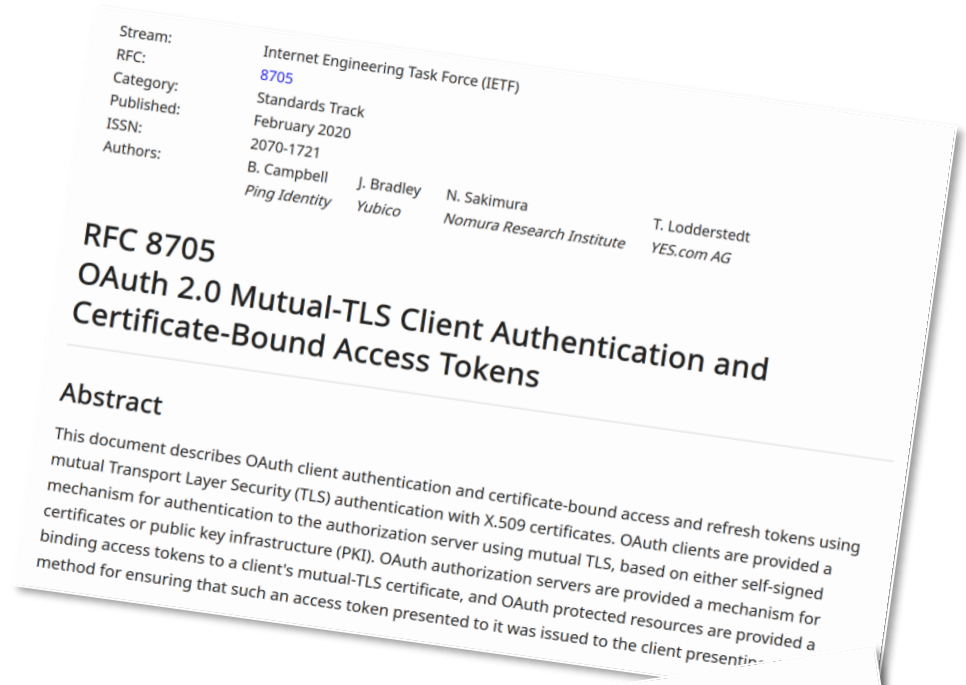
Stream: Internet Engineering Task Force (IETF)
RFC: 9101
Category: Standards Track
Published: August 2021
ISSN: 2070-1721
Authors: N. Sakimura (NAT.Consulting), J. Bradley (Yubico), M. Jones (Microsoft)

RFC 9101
The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that authorization request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that a) the communication through the user agents is not integrity protected and thus, the parameters can be tainted, b) the source of the communication is not authenticated, and c) the communication through the user agents can be monitored. Because of these weaknesses, several attacks to the protocol have now been put forward.

This document introduces the ability to send request parameters in a JSON Web Token (JWT) instead, which allows the request to be signed with JSON Web Signature (JWS) and encrypted with JSON Web Encryption (JWE) so that the integrity, source authentication, and confidentiality properties of the authorization request are attained. The request can be sent by value or by reference.

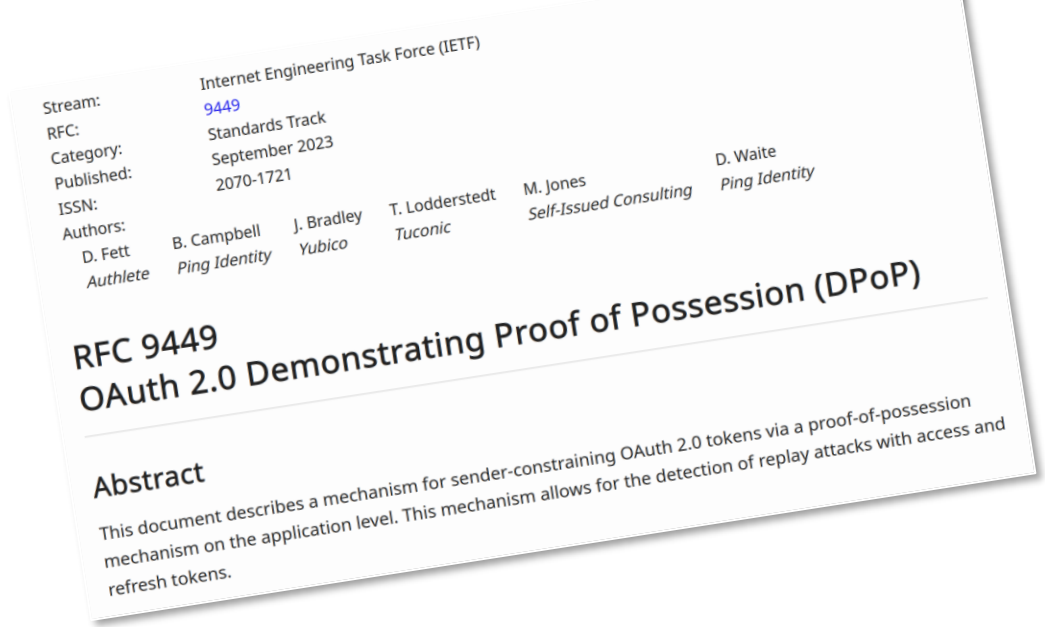


Stream: Internet Engineering Task Force (IETF)
RFC: 8705
Category: Standards Track
Published: February 2020
ISSN: 2070-1721
Authors: B. Campbell (Ping Identity), J. Bradley (Yubico), N. Sakimura (Nomura Research Institute), T. Lodderstedt (YES.com AG)

RFC 8705
OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

Abstract

This document describes OAuth client authentication and certificate-bound access and refresh tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates. OAuth clients are provided a mechanism for authentication to the authorization server using mutual TLS, based on either self-signed certificates or public key infrastructure (PKI). OAuth authorization servers are provided a mechanism for binding access tokens to a client's mutual-TLS certificate, and OAuth protected resources are provided a method for ensuring that such an access token presented to it was issued to the client presentin

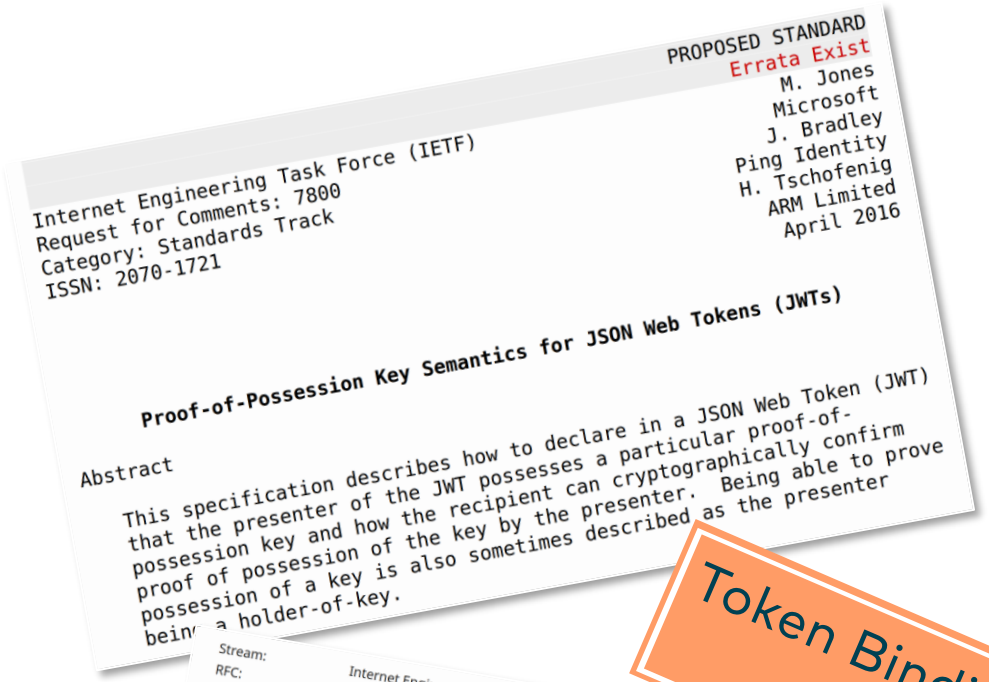


Stream: Internet Engineering Task Force (IETF)
RFC: 9449
Category: Standards Track
Published: September 2023
ISSN: 2070-1721
Authors: D. Fett (Authlete), B. Campbell (Ping Identity), J. Bradley (Yubico), T. Lodderstedt (Tuconic), M. Jones (Self-Issued Consulting), D. Waite (Ping Identity)

RFC 9449
OAuth 2.0 Demonstrating Proof of Possession (DPoP)

Abstract

This document describes a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. This mechanism allows for the detection of replay attacks with access and refresh tokens.



PROPOSED STANDARD
Errata Exist

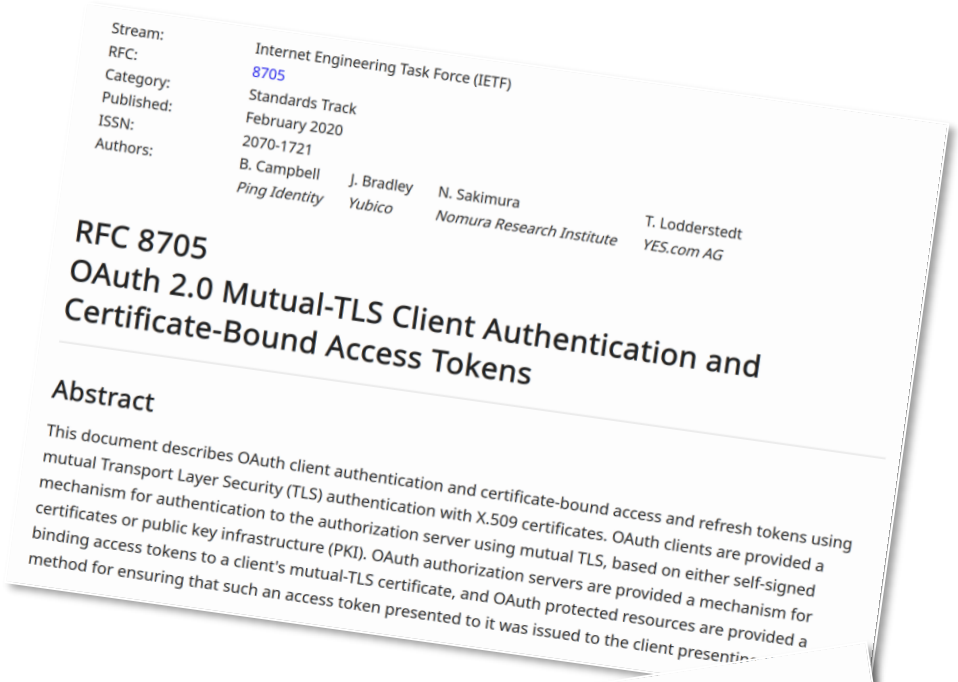
Internet Engineering Task Force (IETF)
Request for Comments: 7800
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft
J. Bradley
Ping Identity
H. Tschofenig
ARM Limited
April 2016

Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

Abstract

This specification describes how to declare in a JSON Web Token (JWT) that the presenter of the JWT possesses a particular proof-of-possession key and how the recipient can cryptographically confirm proof of possession of the key by the presenter. Being able to prove possession of a key is also sometimes described as the presenter being a holder-of-key.

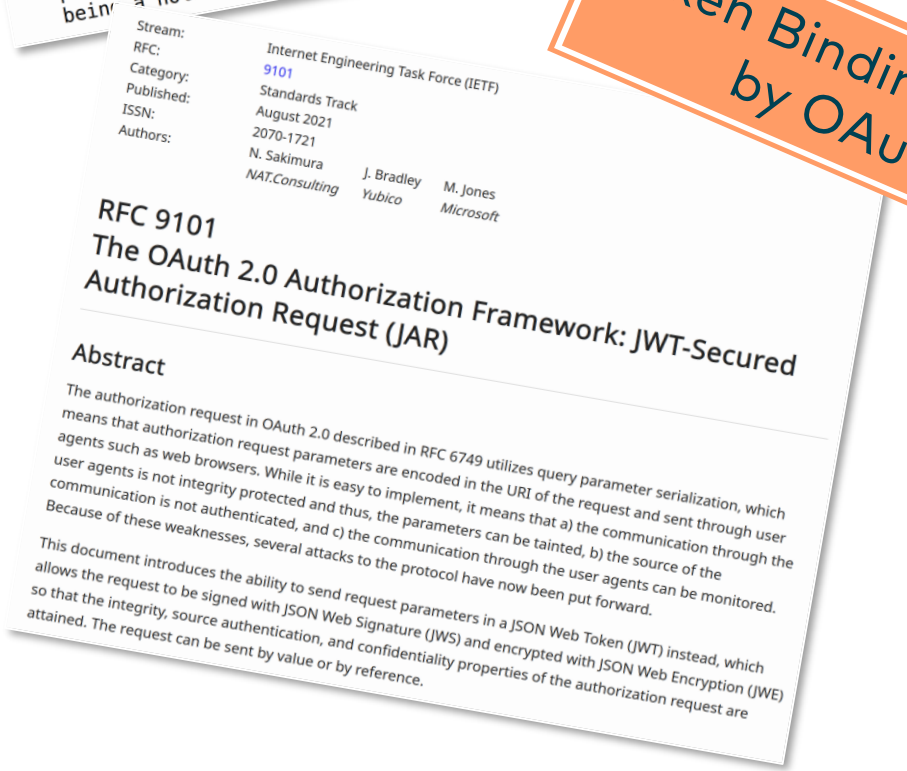


Stream: Internet Engineering Task Force (IETF)
RFC: 8705
Category: Standards Track
Published: February 2020
ISSN: 2070-1721
Authors: B. Campbell (Ping Identity), J. Bradley (Yubico), N. Sakimura (Nomura Research Institute), T. Lodderstedt (YES.com AG)

RFC 8705
OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

Abstract

This document describes OAuth client authentication and certificate-bound access and refresh tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates. OAuth clients are provided a mechanism for authentication to the authorization server using mutual TLS, based on either self-signed certificates or public key infrastructure (PKI). OAuth authorization servers are provided a mechanism for binding access tokens to a client's mutual-TLS certificate, and OAuth protected resources are provided a method for ensuring that such an access token presented to it was issued to the client presentin



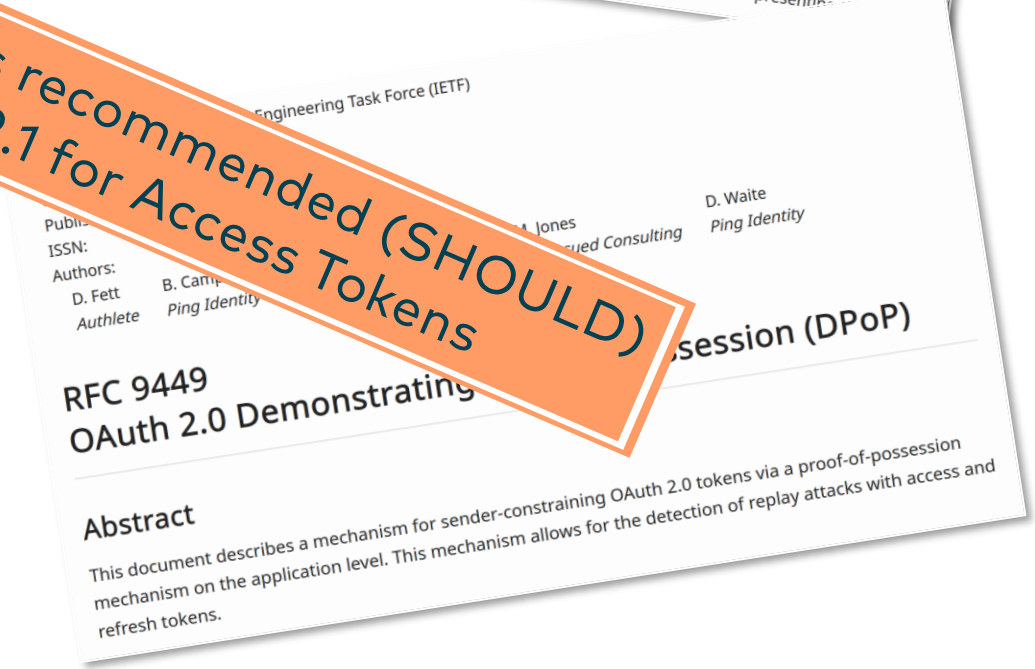
Stream: Internet Engineering Task Force (IETF)
RFC: 9101
Category: Standards Track
Published: August 2021
ISSN: 2070-1721
Authors: N. Sakimura (NAT.Consulting), J. Bradley (Yubico), M. Jones (Microsoft)

RFC 9101
The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that authorization request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that a) the communication through the user agents is not integrity protected and thus, the parameters can be tainted, b) the source of the communication is not authenticated, and c) the communication through the user agents can be monitored. Because of these weaknesses, several attacks to the protocol have now been put forward.

This document introduces the ability to send request parameters in a JSON Web Token (JWT) instead, which allows the request to be signed with JSON Web Signature (JWS) and encrypted with JSON Web Encryption (JWE) so that the integrity, source authentication, and confidentiality properties of the authorization request are attained. The request can be sent by value or by reference.



Engineering Task Force (IETF)

Publi
ISSN:
Authors: D. Fett (Authlete), B. Cam (Ping Identity), A. Jones (ued Consulting), D. Waite (Ping Identity)

RFC 9449
OAuth 2.0 Demonstrating session (DPoP)

Abstract

This document describes a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. This mechanism allows for the detection of replay attacks with access and refresh tokens.

Token Binding is recommended (SHOULD) by OAuth 2.1 for Access Tokens

DR;TL

OAuth 2.1 Updates

Addressed

- No Resource Owner Password Credentials Grant
- No Implicit Grant
- Mandatory CSRF protection for the redirect-uri (with PKCE or other means)
- Access Token binding (DPoP, or MPoP) is highly recommended
- Exact Redirect URI matching
- No Bearer Tokens in query strings
- Sender-constraint Refresh Tokens
- No Refresh-Token replays

Not (yet) addressed

- Weak CSRF protection – e.g. plain method with PKCE
- BFF is only mentioned
- No clarification for 3rd and 1st party use cases
- Token Lifecycle Management with PoP not addressed

Thank you! Questions?

Dimitrij Drus
dimitrij.drus@innoq.com
+49 175 2227084
dadrus

INNOQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Wendenstraße 130
20573 Hamburg

Spichernstrasse 44
50672 Köln