OAuth for Security Critical Applications

Dr. Torsten Lodderstedt, yes.com

About me

- Identity Practitioner
  - Consumer Identity Management (Architect, PO, VP)
  - Open Banking Ecosystem (CTO)
  - Freelancer
- Standards Guy
  - OAuth 2.0 Threat Model and Security Considerations (RFC 6819)
  - ...
  - OAuth 2.0 Security Best Current Practice
  - OAuth 2.0 Pushed Authorization Requests
  - OAuth 2.0 Rich Authorization Requests
  - OAuth 2.1
  - OpenID Connect 4 Identity Assurance
  - OpenID 4 Verifiable Credentials
  - Global Assured Identity Network (GAIN)

OAuth 2.0 Universe

RFC 8693
OAuth 2.0 Token Exchange

RFC 8628
OAuth 2.0 Device Authorization Grant

RFC 8252
OAuth 2.0 for Native Apps

RFC 8414
OAuth 2.0 Authorization Server Metadata

RFC 7800
Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

RFC 8176
Authentication Method Reference Values

RFC 7523
JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants

RFC 7662
OAuth 2.0 Token Introspection

RFC 7636
Proof Key for Code Exchange by OAuth Public Clients

RFC 7522
Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants

RFC 7592
OAuth 2.0 Dynamic Client Registration Management Protocol

RFC 7591
OAuth 2.0 Dynamic Client Registration Protocol

RFC 7521
Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants

RFC 7009
OAuth 2.0 Token Revocation

RFC 7519
JSON Web Token (JWT)

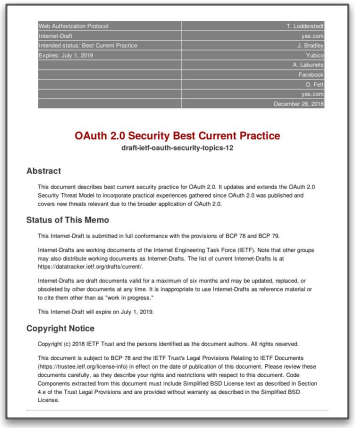
RFC 6750
The OAuth 2.0 Authorization Framework: Bearer Token Usage

RFC 6749
The OAuth 2.0 Authorization Framework

RFC 6819
OAuth 2.0 Threat Model and Security Considerations

OAuth 2.0 Security Best Current Practice

- Refines and enhances security guidance for OAuth 2.0 implementers
- Updates, but does not replace:
  - OAuth 2.0 Threat Model and Security Considerations (RFC 6819)
  - OAuth 2.0 Security Considerations (RFC 6749 & 6750)



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.

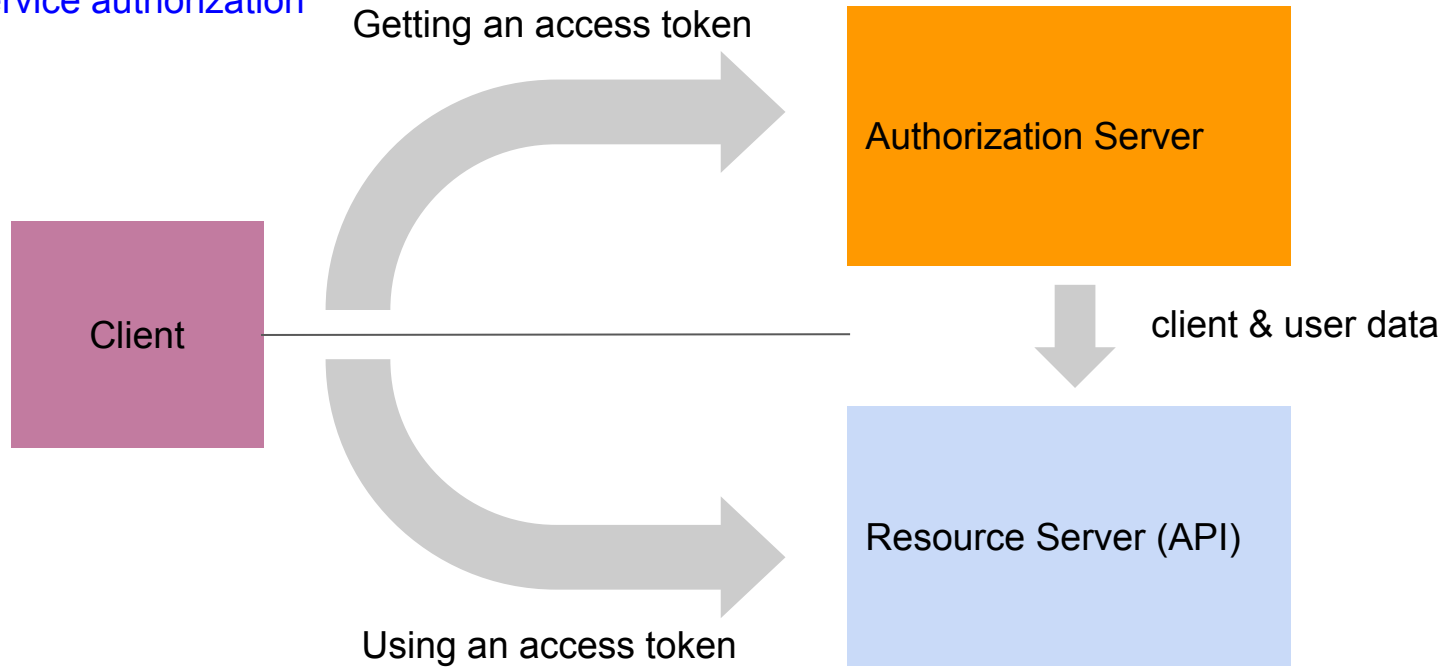
This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this 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

https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics

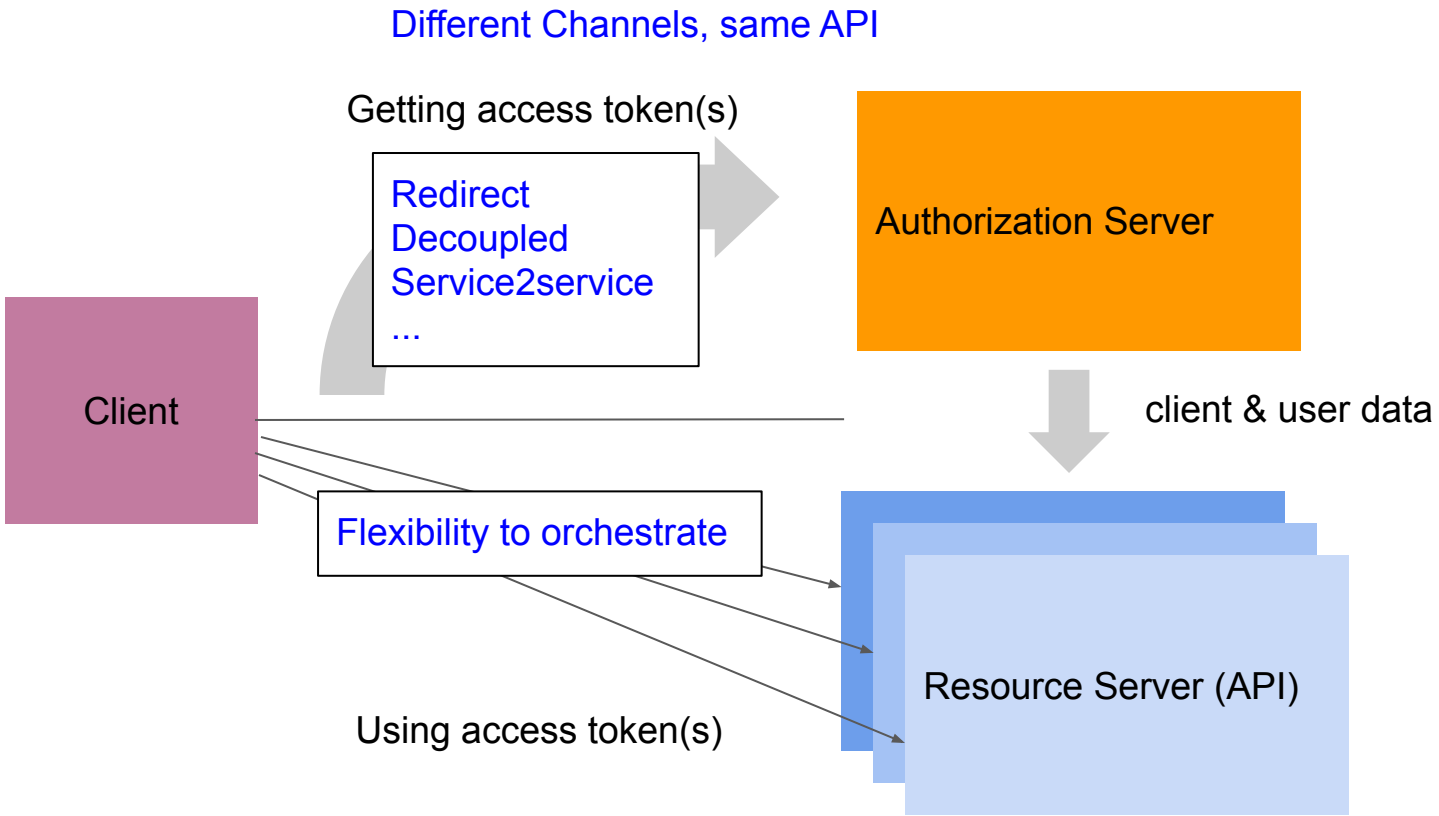
OAuth 2 Design Principles

Authorization Server acts as trusted 3rd party for Resource Server - decoupling user authentication & consent from service authorization



Huge potential regarding software architecture, security, and user experience

Software Architecture

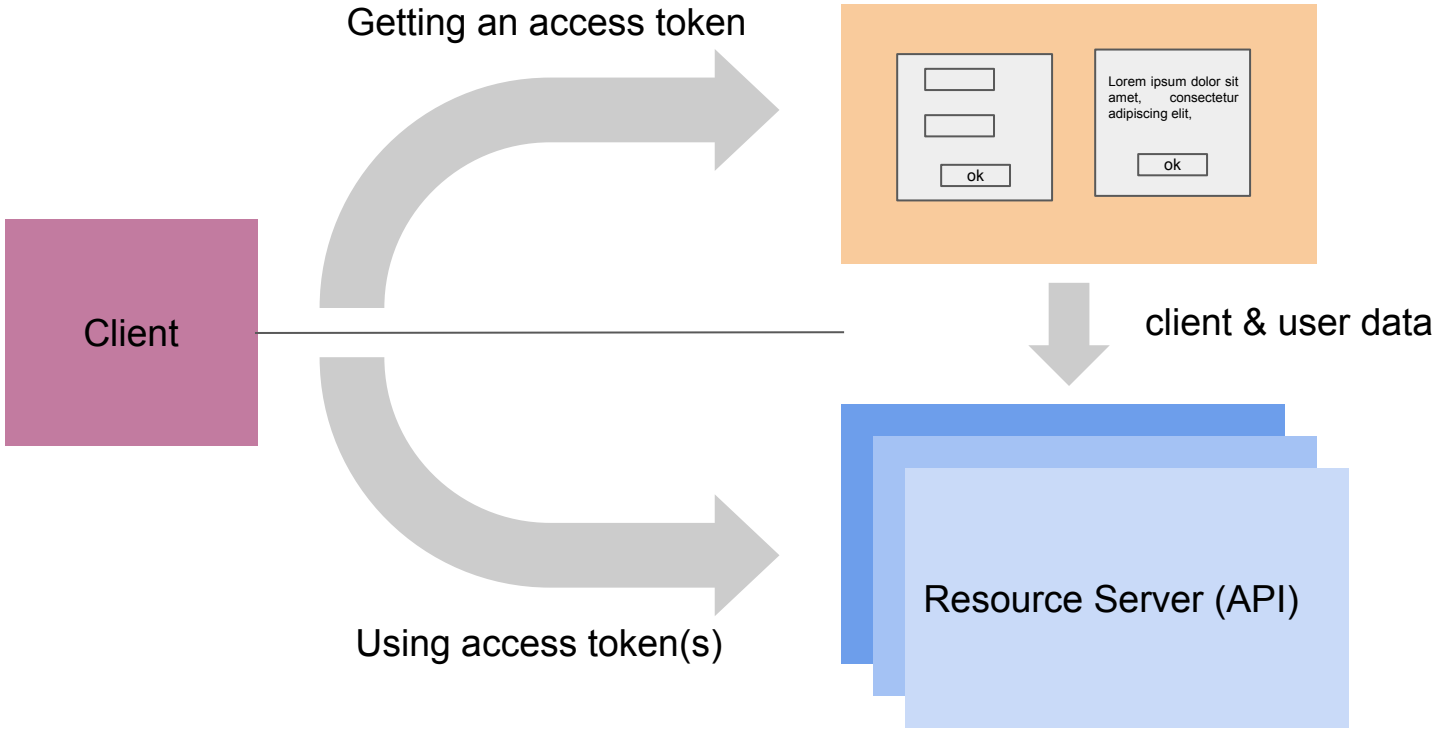


Security

- AS handles user credentials in frontend process
  - APIs do not handle user credentials
  - Clients do not handle user credentials
- AS handles user consent
- AS may handle authorization centrally
- Results in reduced attack surface
- Origin bound credentials (cookies, certs, FIDO) and 3rd party Logins can be used in API scenarios

User Experience

User may authorize access to multiple APIs (including 2FA) once, client uses access token multiple times

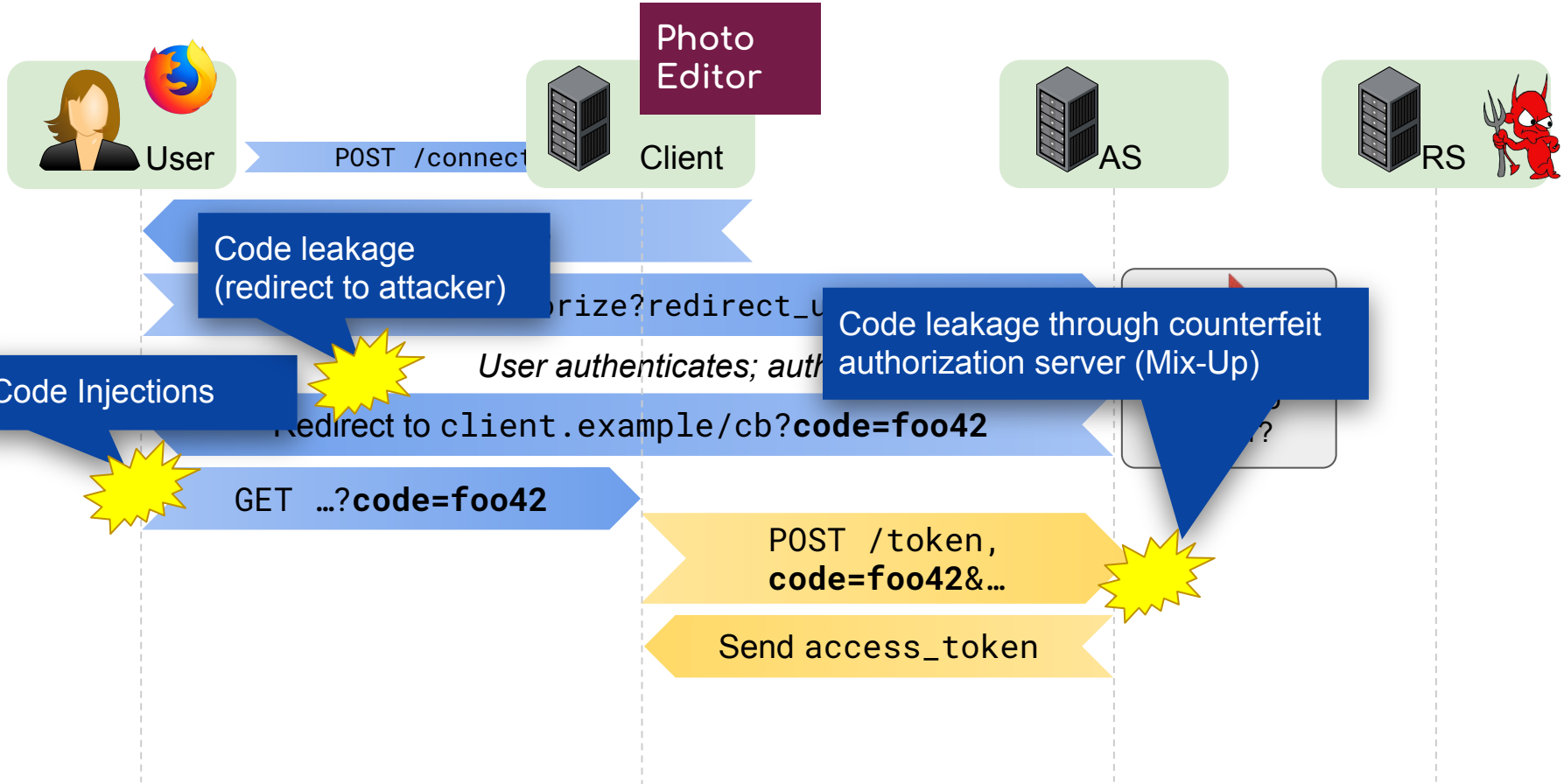


Topics

- OAuth Code Flow (re-enforced)
- Client Authentication
- Token Leakage and Replay Prevention
- Introspection vs Structure Access Tokens
- Privilege Enforcement
- Audience Restriction
- Server Metadata

OAuth Code Flow (re-enforced)

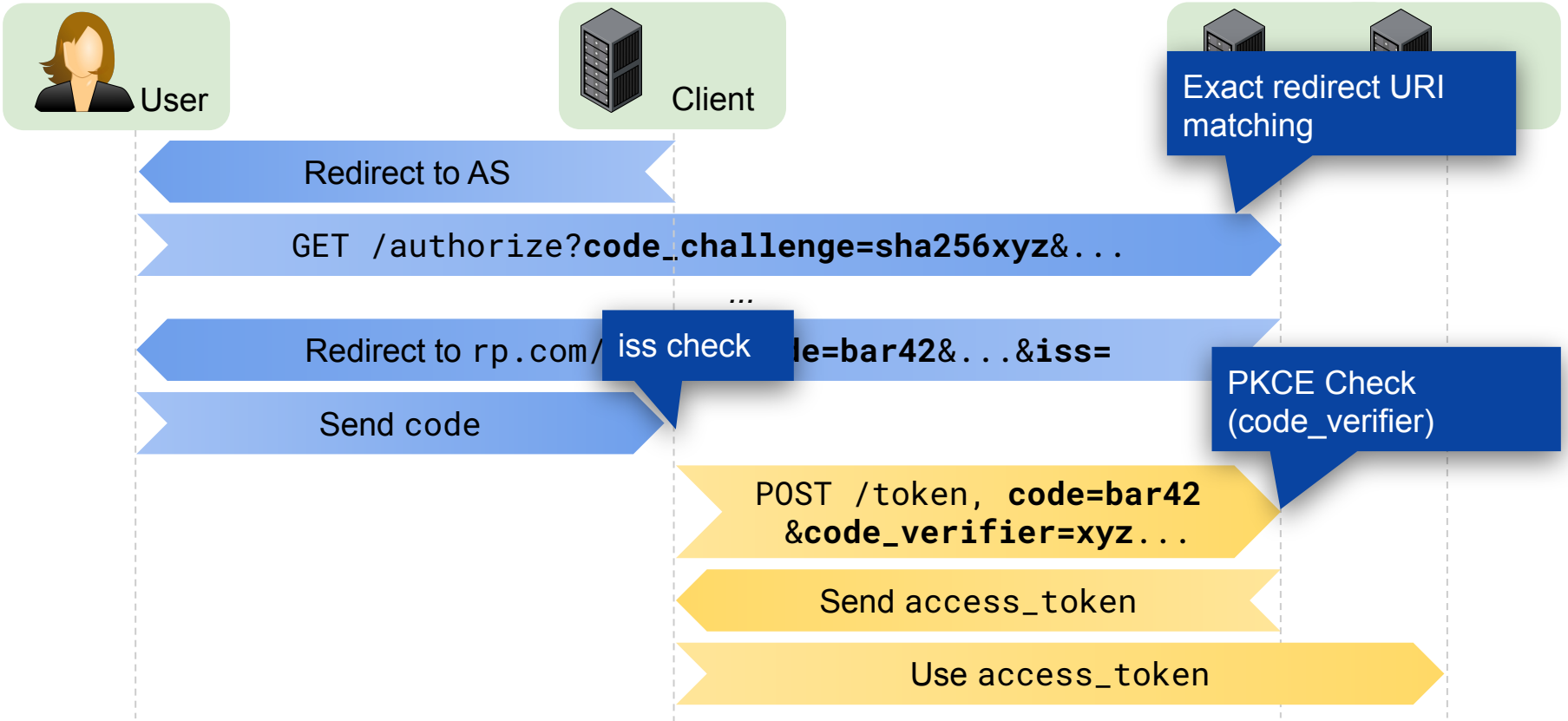
OAuth Code Flow (Threats)



Hardened Authorization Code Grant

- Security improvements with
  - OAuth Security Guidelines (https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics)
  - and OAuth 2.1 (https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1)
- Exact redirect URI matching (Code Leakage)
- Proof Key for Code Exchange - PKCE (Code Replay & CSRF)
- “iss” response parameter (Mix Up - Conditional)

Authorization Code Grant (Security BCP/OAuth 2.1)



Client Authentication

Why Client Authentication?

- Decide on policy
  - Accessible APIs
  - User consent required? 1st vs 3rd party clients
- Show correct party in user consent
- Ensure only vetted parties can access your API
- Ensure the bill is paid

Client Authentication Options

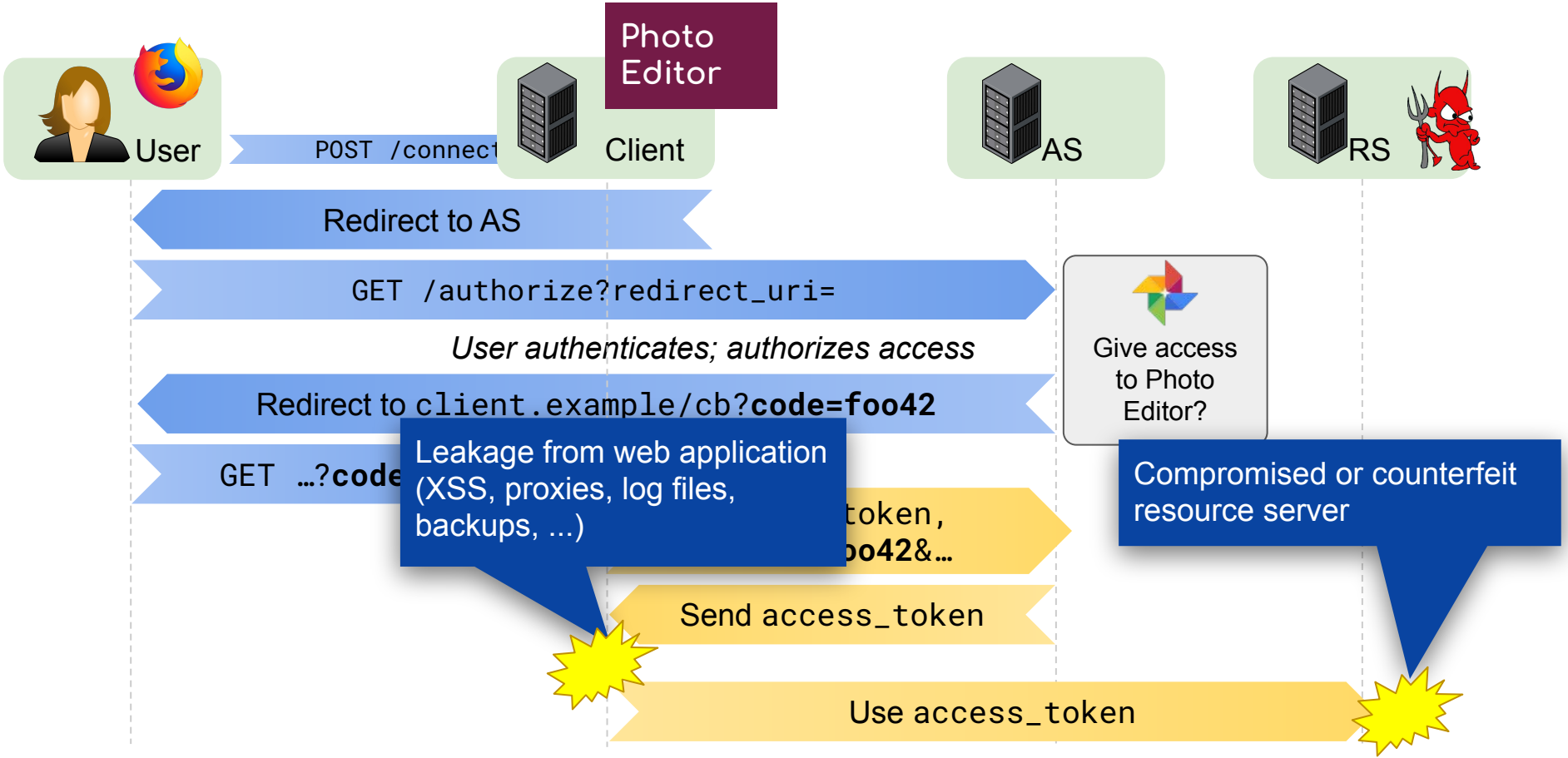
- Client Secret (RFC 6749)
  - Basic Authorization Header or POST message parameter
  - Simple, shared secret, secret sent over the wire
- TLS Client Authentication (RFC 8705)
  - Self-signed certificates or public key infrastructure (PKI)
  - Can be simple, secret known to client only
  - Integrates well with token replay detection (next topic)
- Signed JWTs (RFC 7523)
  - private_key_jwt: Signature using private key corresponding to pre-registered JSON Web Key
    - Secure and simple to use, secret known to client only
  - client_secret_jwt: HMAC of shared secret
    - Secure and simple to use, however shared secret

Comparison

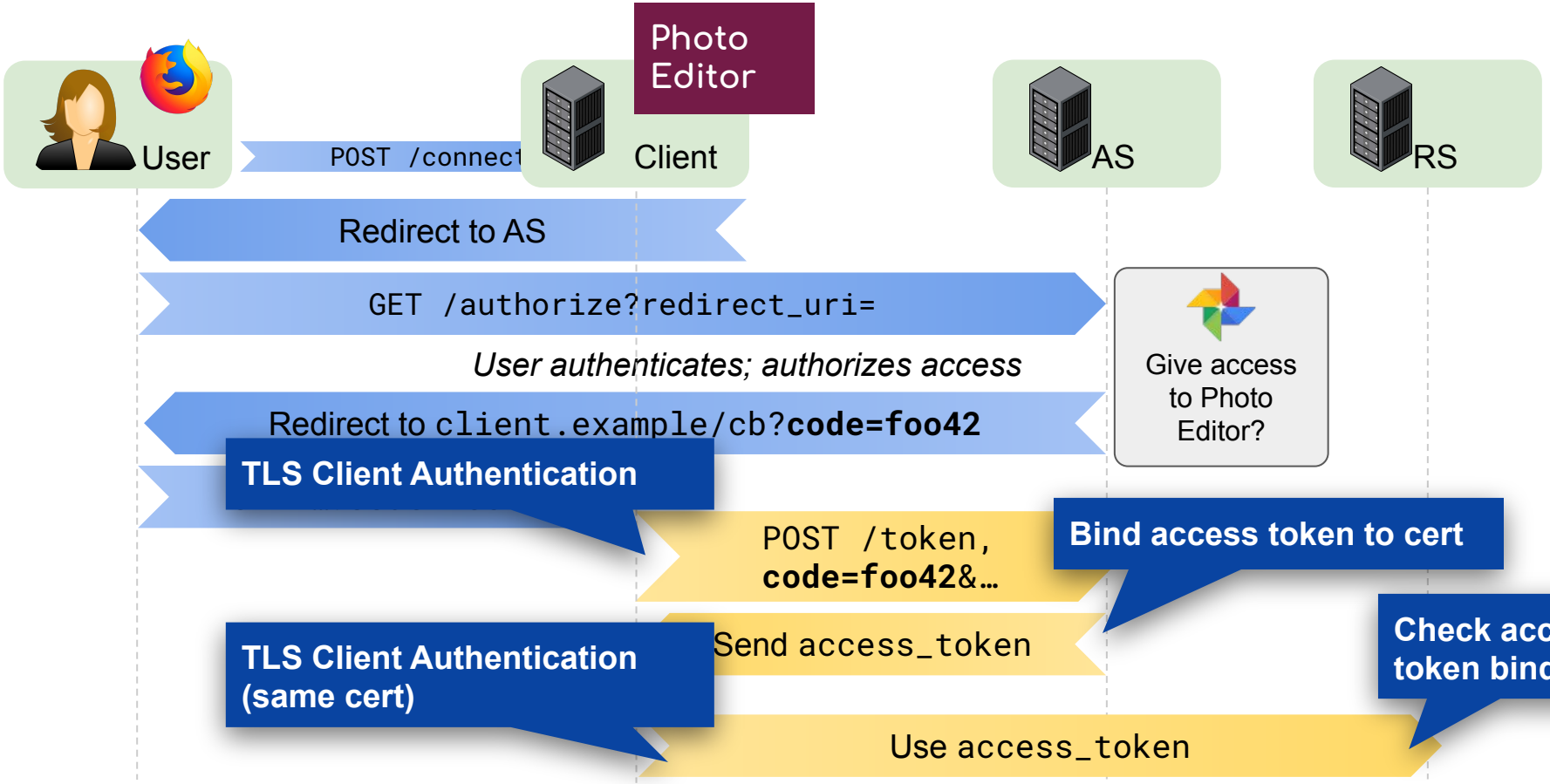
Method	Complexity	Who knows secret?	Secret on the wire
Client Secret	Simple	Shared Secret	yes
mTLS 4 OAuth	Depends	Client only	no
private_key_jwt	Increased, but manageable	Client only	no
client_secret_jwt	Increased, but manageable	Shared Secret	no

Token Leakage & Replay Prevention

Token Leakage



Sender-Constrained Access Tokens (mTLS)



Access Token Binding Check (self-signed certs)

Reverse Proxy Configuration (NGINX)

Turn on optional client TLS w/o trust chain validation

Pass client TLS cert to RS

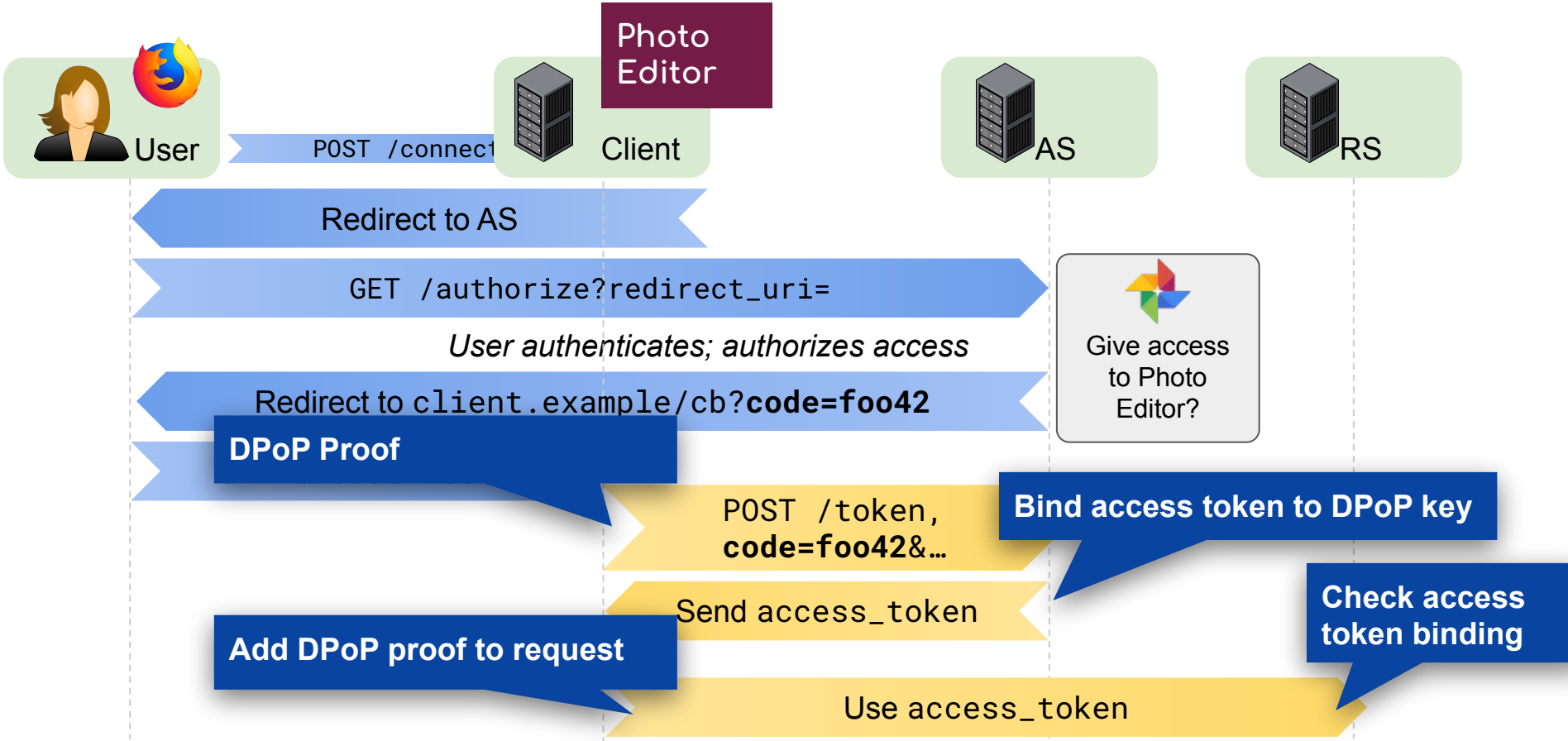
```
server {
  …
  ssl_verify_client optional_no_ca;
  location /example/ {
    proxy_set_header x-client-x509-cert-alaelul8geiqu3ohog1mafa4ecu9ahsh $ssl_client_cert;
    proxy_pass <app server>;
  }
}
```

Access Token Content

Cert fingerprint (SHA 256)

```
{
  "iss":"ht
  "sub":"ty
  "exp":1493726
  "cnf":{
    "x5t#S256":"bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}
```

Sender-Constrained Access Tokens (DPoP)



DPoP-protected Request

```
GET /protectedresource HTTP/1.1
Host: resource.example.org
Authorization: DPoP Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiJlMWozVl9iS2ljOC1MQUVCIiwiaHRtIj
 oiR0VUIiwiaHR1IjoiaHR0cHM6Ly9yZXNvdXJjZS5leGFtcGxlLm9yZy9wcm90ZWN0Z
 WRyZXNvdXJjZSIsImlhdCI6MTU2MjI2MjYxOCwiYXRoIjoiZlVIeU8ycjJaM0RaNTNF
 c05yV0JiMHhXWG9hTnk1OUlpS0NBcWtzbVFFbyJ9.2oW9RP35yRqzhrtNP86L-Ey71E
 OptxRimPPToA1plemAgR6pxHF8y6-yqyVnmcw6Fy1dqd-jfxSYoMxhAJpLjA
```

Access Token Content

```
{
  "sub":"someone@example.com",
  "iss":"https://server.example.com",
  "nbf":1562262611,
  "exp":1562266216,
  "cnf":
  {
    "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"
  }
}
```

Key fingerprint (SHA 256)

DPoP-Proof

```
{
 "typ":"dpop+jwt",
 "alg":"ES256",
 "jwk": {
   "kty":"EC",
   "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
   "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
   "crv":"P-256"
 }
}
.
{
  "jti":"e1j3V_bKic8-LA
  "htm":"GET",
  "htu":"https://resource.example.org/protectedresource",
  "iat":1562262618,
  "ath":"fUHyO2r2Z3DZ53EsNrWBb0xWXoaNy59IiKCAqksmQEo",
  "nonce": "eyJ7S_zG.eyJH0-Z.HX4w-7v"
}
```

identifier

HTTP method

protected resource

Access token hash

RS provided nonce

mTLS vs DPoP

mTLS

- Utilizes TLS/HTTPS stack
- Client setup straightforward with self-signed certs (works with Postman)
- PKI might cause issues
- Server setup might be challenging in managed infrastructures

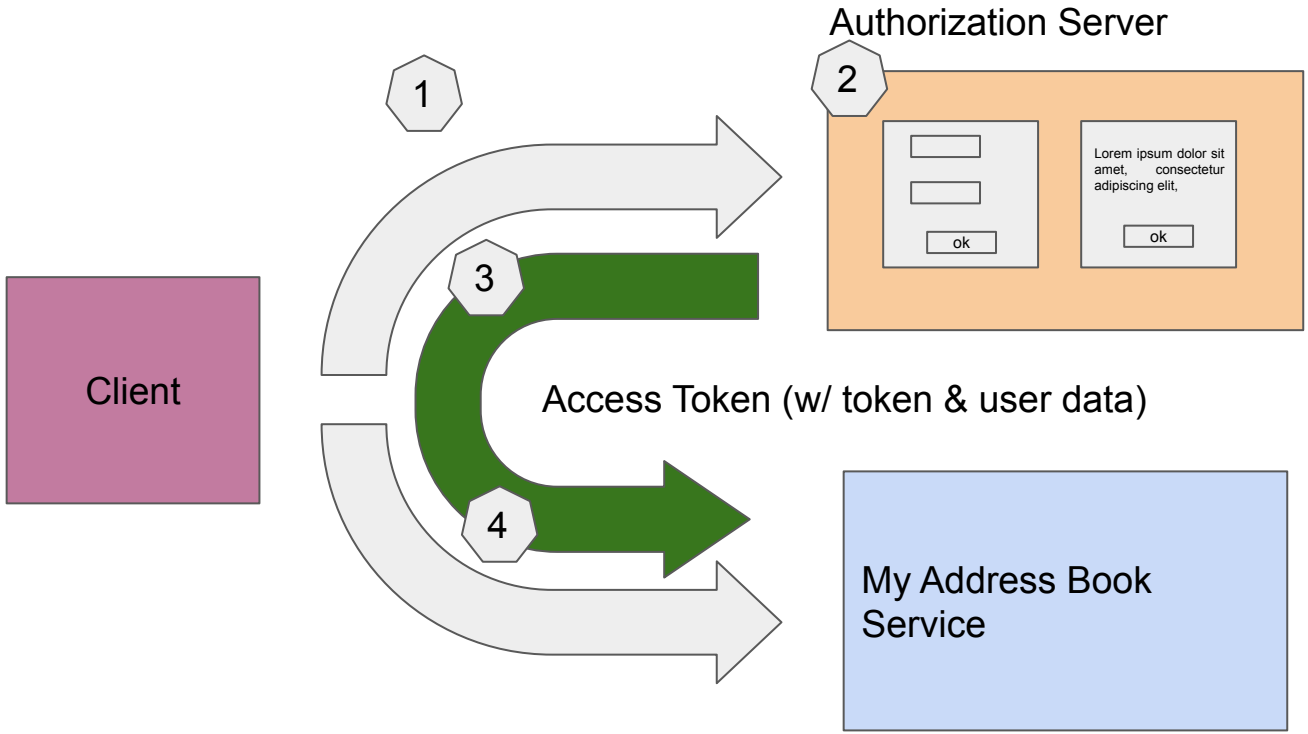
DPoP

- Utilizes application level signatures
- Works on top of any managed infrastructure
- Requires dedicated support in OAuth library
- Replay detection might require server-side nonces

- Both methods are designed to work for public clients and for confidential clients (in conjunction with any client authentication method)

Structured Access Tokens vs Token Introspection

Structured Access Tokens (e.g. JWTs)

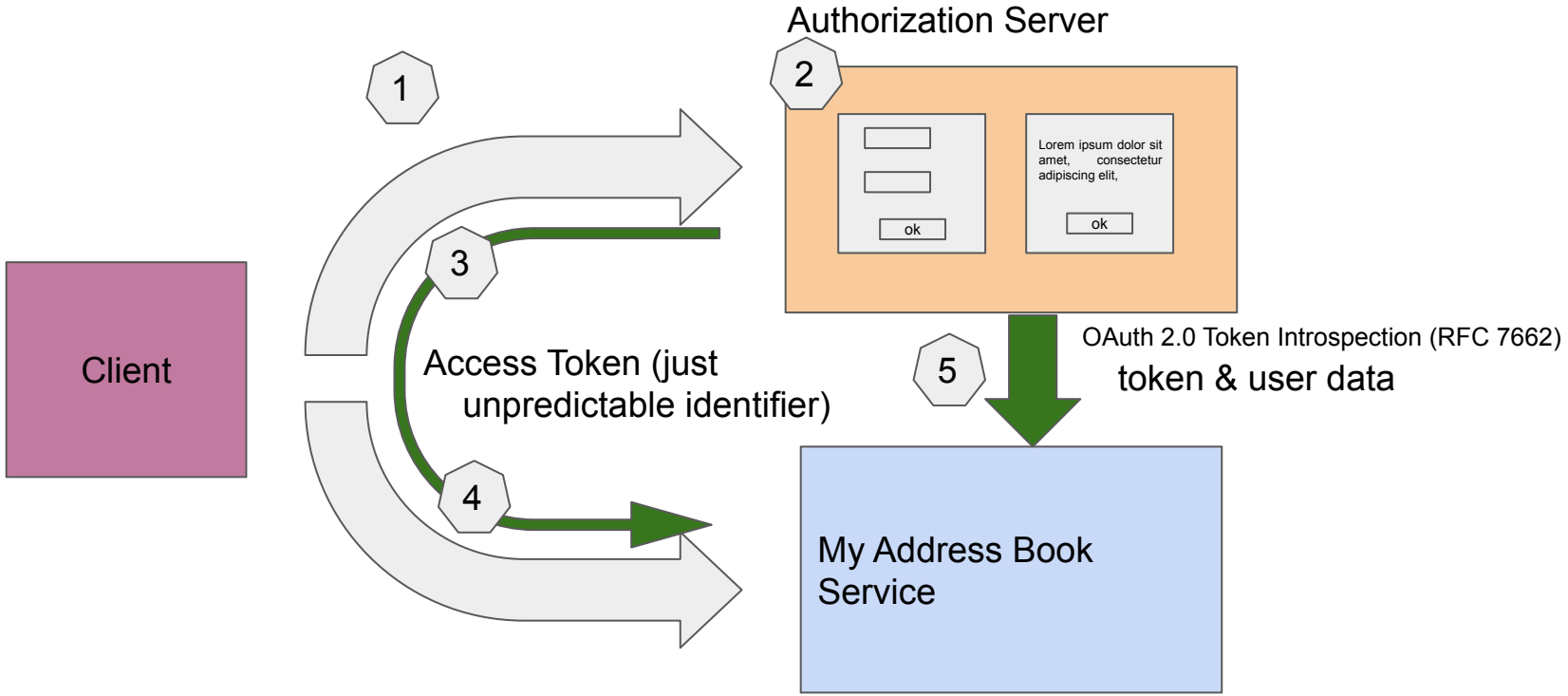


JWT-based Access Tokens

- Signed and (optionally) encrypted tokens
- May contain any user data required to authorize and perform API requests
- Recommended reads:
  - JSON Web Token (JWT/ RFC 7519)
  - Profile for OAuth 2.0 Access Tokens (RFC 906)
- (Some of the) standard claims
  - **iss**: token issuer
  - **sub**: token subject (user id)
  - **aud**: token audience
  - **exp**: token expiration
  - **scope**: delegated scope

```
{
   "iss":"https://as.example.com/",
   "sub":" 5ba552d67",
   "aud":"https://ab.example.com/",
   "exp":1544645174,
   "client_id":"s6BhdRkqt3",
   "scope":"read",
   "role":"helpdesk",
   "email":"max@company.com"
}
```

Token Introspection



Token Introspection Example

```
POST /introspect HTTP/1.1
Host: server.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

token=mF_9.B5f-4.1JqM
```

```
HTTP/1.1 200 OK
Content-Type: applicat

{
  "active": true,
  "iss":"https://as.example.com/",
  "sub":" 5ba552d67",
  "aud":"https://ab.example.com/",
  "exp":1544645174,
  "client_id":"s6BhdRkqt3",
  "scope":"read",
  "role":"helpdesk"
}
```

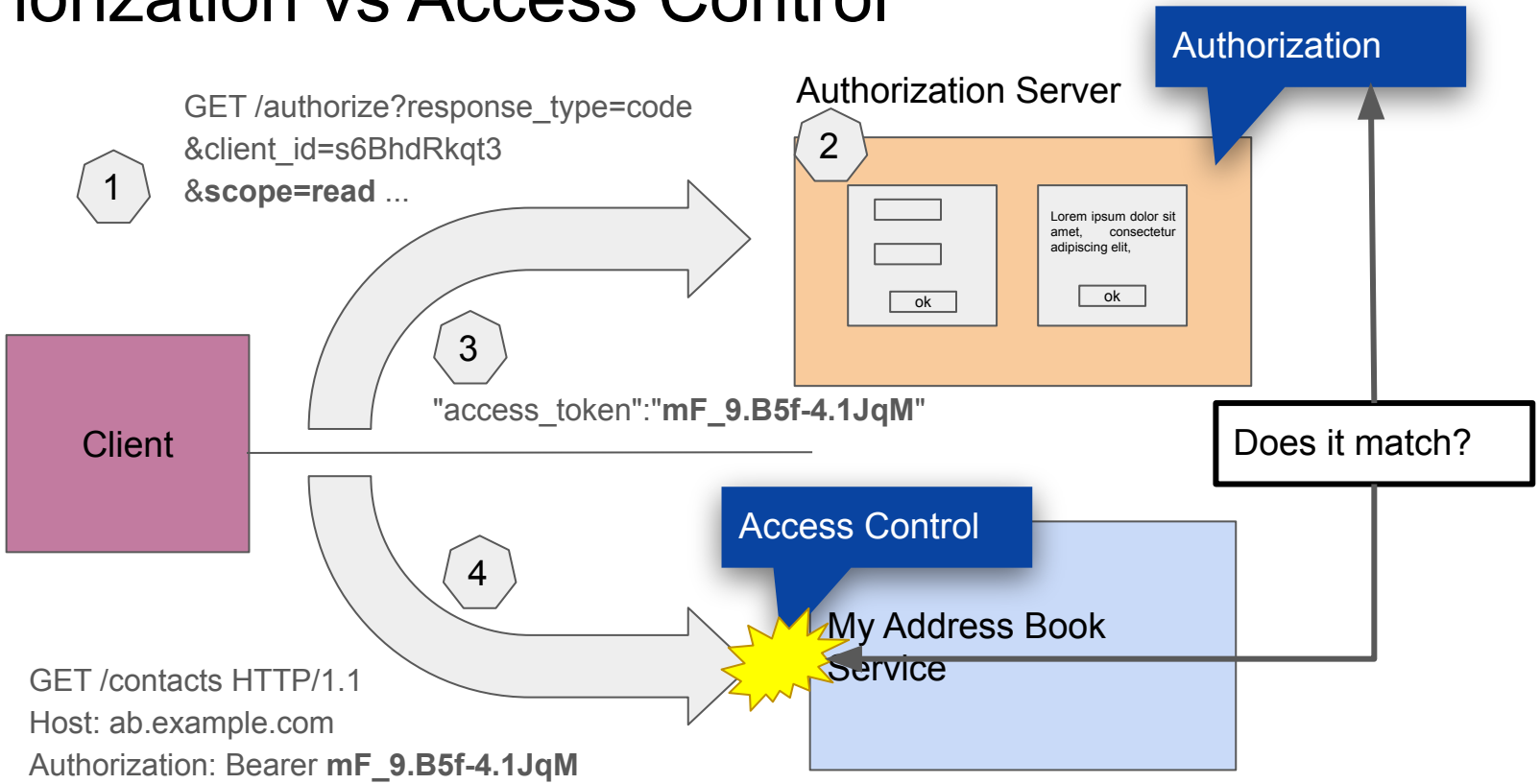
Is this access token still valid?

Comparison

	Structured Tokens	Token Introspection
API Performance	No impact, RS can meet access control decisions locally	RS needs to callback to AS
Scalability	Excellent, since no state required at RS	Depends on AS's scalability
Client 2 API Performance	Access tokens can be huge, potential impact on lower latency networks	No impact, since access tokens are very small
Integrity	Based on digital signatures or HMACs	Based on random numbers and TLS server authentication
Revocation	Difficult to implement (Refresh Tokens as alternative)	Easy to implement
Privacy	Client: token encryption RS: RS-specific access tokens required (RAR)	Client: token does not contain PII RS: RS-specific introspection responses

Privileges Enforcement

Authorization vs Access Control



Convey granted privileges to RS

- Options
  - Put the data into the access token (e.g. JSON Web Tokens)
  - Query data from AS during access control process (e.g. Token Introspection)
- Let’s discuss it with JSON Web Tokens (JWT)

Access control based on JWT

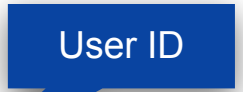
Access Token in JWT format

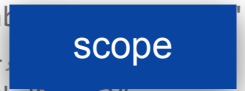
Header

```
{
   "typ":"at+JWT",
   "alg":"RS256",
   "kid":"RjEwOwOA"
}
```

Payload

```
{
   "iss":"https://as.example.com/",
   "sub":" 5ba552d67",
   "aud":"https://ab...",
   "exp":1544645174,
   "client_id":"s6Bhd...qt3",
   "scope":"read"
}
```






```
GET /contacts/@me/@all HTTP/1.1
Host: ab.example.com
Authorization: Bearer mF_9.B5f-4.1JqM

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

[
   {
      "id":"1234",
      "displayName":"Contact ABC"
   },
   ...
]
```

```
POST /contacts/@me/1234 HTTP/1.1
Host: ab.example.com
Content-Type: application/json;charset=UTF-8
Authorization: Bearer mF_9.B5f-4.1JqM

{
      "id":"1234",
      "displayName":"Something else"
}

HTTP/1.1 403 Forbidden
```

User Data in Access Token

```
{
   "iss":"https://as.example
   "sub":" 5ba552d67",
   "aud":"https://ab.example.c
   "exp":1544645174,
   "client_id":"s6BhdRkqt3",
   "scope":"read"
}
```

No user data - Requires user database lookup

```
GET /contacts/@fred.firestone/@all HTTP/1.1
Host: ab.example.com
Authorization: Bearer mF_9.B5f-4.1JqM

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

[
   {
      "id":"3456",
      "displayName":"Contact WWA"
   },
   ...
```

```
{
   "iss":"https://as.example.com/",
   "sub":" 5ba552d67",
   "aud":"https://ab.examp
   "exp":1544645174,
   "client_id":"s6BhdRkqt3",
   "scope":"read",
   "role":"helpdesk"
}
```

User privileges are provided in JWT, no additional lookup needed

Access Token Validity

Base64 Encoded Representation


```
eyJhbGciOiJSUzI1NiIsInR5cCI6ImF0K0pXVCIsImtp
ZCI6IlJqRXdPd09BIn0.eyJpc3MiOiJodHRwczovL2Fz
LmV4YW1wbGUuY29tLyIsInN1YiI6IiA1YmE1NTJkNjci
LCJhdWQiOiJodHRwczovL2FiLmV4YW1wbGUuY29tLyIs
ImV4cCI6MTU0NDY0NTE3NCwiY2xpZW50X2lkIjoiczZC
aGRSa3F0MyIsInNjb3BlIjoicmVhZCJ9.hm3ZKFVu-u2
DxavIy3tcLqIICWlZAP0Ht_GvB7awgBEmagdhLxRTCgF
ZQbPOSSXXa0EzirmtXkWCxo-_raYF5NiKgfWX2Hhj1ux
ukNelJi0L3GXqD6AKzVvSU5Q0pY_kWcCxyyUpU0YbeEx
FS25bl6q9kkON3z7nRiNFWfkQOhSHqip7cD1k7HNZXwz
NNQ-0qy083EP8fxD929zDRR-YUgiUR6bjrNekDcI9TR
MjLzcSHy_REL5PYgjZxwzwM_XsfU
PsIh02ZCrJcYA4ONjD-Mijlkzlx
wX0hMW1Egk5ktcF7bD513zA
```


Signature

Is the signature valid?

Header


```
{
   "typ":"at+JWT",
   "alg":"RS256",
   "kid":"RjEwOwOA"
}
```


Is this an access token?

Payload


```
{
   "iss":"https://as.example.com/",
   "sub":" 5ba552d67",
   "aud":"https://ab.exam
   "exp":1544645174,
   "client_id":"s6BhdRkqt3",
   "scope":"read"
}
```

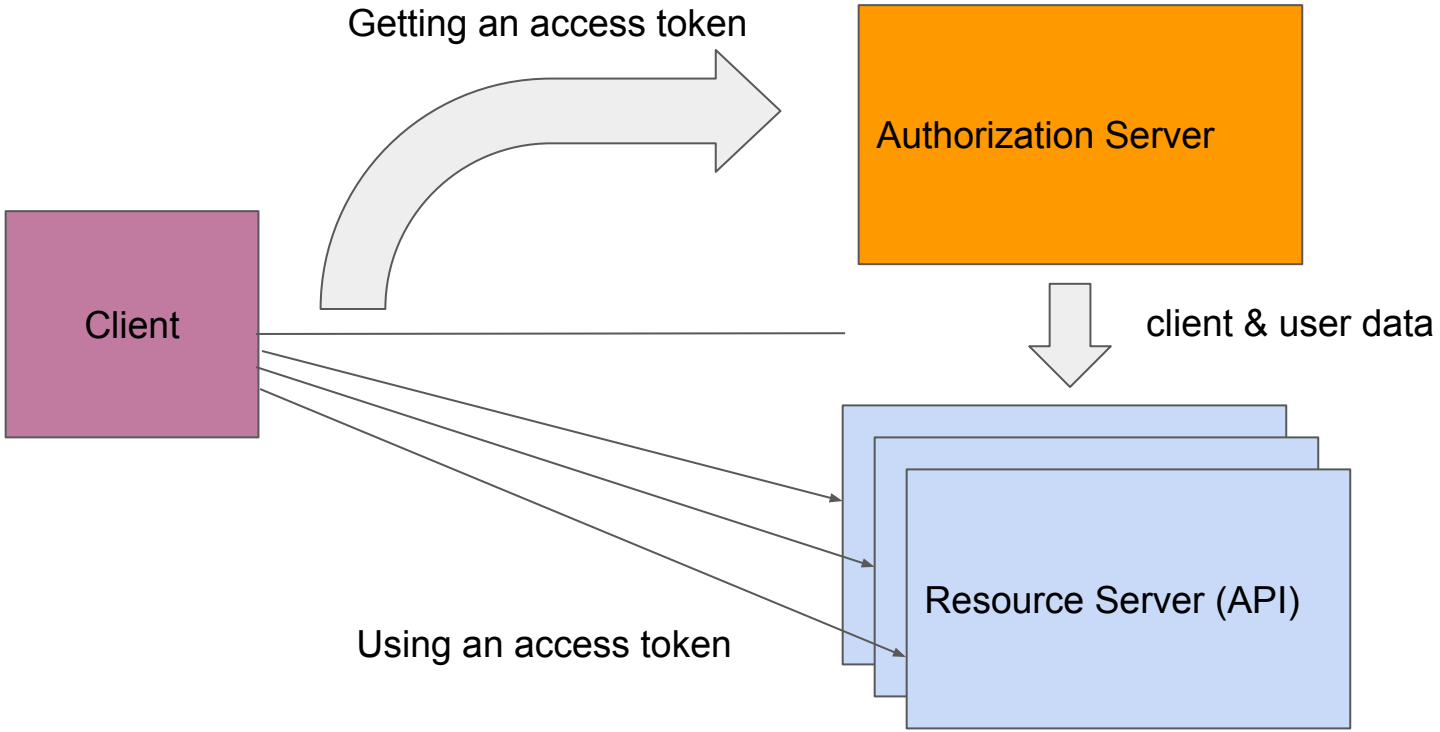

An authorization server I trust?

Is that me?

Is token still valid?

Audience Restriction

One Client, Multiple Resource Servers



What if the AS protects multiple RSs?

You want to make sure:

- That every RS is only provided with the data it needs.
- That a RS cannot replay a token it received from a legitimate client with another RS.
- That the RS unambiguously can determine the privileges a client has wrt to this RS.

Audience Restriction

RS-specific access tokens

- Typically used in conjunction with JWTs (and other structured access tokens)
- Audience set to specific RS (audience restriction), so RS cannot use access token somewhere else
- Privacy ensured since token only contains data relevant for particular RS
- Per-RS encryption keys
- How does client request such a token?
  - resource indicators (RFC 8707)
  - “locations” element in “authorization_details”

```
"authorization_details":[
      {
         "type":"payment_initiation",
         "locations":[
            "https://api.example.com/payments"
         ],
         "instructedAmount":{
            "currency":"GBP",
            "amount":"31.94"
         },
         "creditorName":"Merchant",
         …
]
```

RS-specific token introspection response

- Single token across resource servers
- Every RS authenticates towards AS in introspection request
- AS responds with RS-specific introspection response
- Response only contains data relevant for particular RS and only if the access token is good for that particular RS

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "active":true,
   "iss":"https://as.example.com/",
   "sub":" 5ba552d67",
   "aud":"https://api.example.com/payments",
   "exp":1544645174,
   "client_id":"s6BhdRkqt3",
   "authorization_details":[
      {
        "type":"payment_initiation",
        "locations":[
       "https://api.example.com/payments"],
         "instructedAmount":{
            "currency":"GBP",
            "amount":"31.94"
         },
         "creditorName":"Merchant"
      }
   ]
}
```

Server Metadata make your life easier

- OAuth 2.0 Authorization Server Metadata (RFC 8414)
- Client obtains endpoint URLs and other metadata from well defined URL
- More efficient and secure than manual configuration

```
https://server.example.com/
.well-known/oauth-authorization-server
```



```
{
  "issuer":"https://server.example.com",
  "authorization_endpoint":
    "https://server.example.com/authorize",
  "token_endpoint":
    "https://server.example.com/token",
  "token_endpoint_auth_methods_supported":[
    "client_secret_basic",
    "private_key_jwt"
  ],
  "scopes_supported":[
    "contacts",
    "cloud"
  ],
  "response_types_supported":[
    "code"
  ]
```

Q&A