**Engineer**

# OAuth

en måte å gi en app begrenset tilgang uten passordet

## Hva det er

OAuth er en protokoll som lar én applikasjon bruke data en person har hos en annen tjeneste, med personens samtykke og innenfor grenser personen selv kan se, uten at applikasjonen noen gang får vite passordet. Versjonen som brukes i dag, er OAuth 2.0, som IETF publiserte i oktober 2012 som RFC 6749, og den skrives like ofte OAuth 2.0 eller OAuth2 som bare OAuth. Den erstattet OAuth 1.0, og de to versjonene er ikke kompatible.

Spesifikasjonen opererer med fire roller. Ressurseieren er den som kan gi tilgang, som regel en person. Ressursserveren har de beskyttede dataene og svarer på forespørsler som har med et token. Klienten er applikasjonen som vil ha tilgang: en webapp, en mobilapp, et skript, en AI-agent. Autorisasjonsserveren logger eieren inn, ber om samtykke og utsteder tokener. Et lite produkt kan kjøre ressursserveren og autorisasjonsserveren som ett system; i en virksomhet står det oftere én identitetsleverandør foran mange API-er.

Det klienten får, er et tilgangstoken (access token): en tekststreng som står for et bestemt omfang av tilgang i en bestemt periode. Klienten sender tokenet med hver forespørsel til ressursserveren. RFC-en kaller strengen vanligvis ugjennomsiktig for klienten, som bærer den uten å skulle lese den. Autorisasjonsserveren kan også utstede et refresh-token, som klienten bytter inn i et nytt tilgangstoken når det gamle utløper, uten å spørre personen på nytt.

Sitt første token får klienten gjennom en grant, en tillatelsestype. RFC 6749 definerer fire: authorization code, implicit, resource owner password credentials og client credentials. Client credentials er for et program som handler på egne vegne, uten noen person involvert. Authorization code er den som gjelder for alt annet, fordi dagens sikkerhetsanbefalinger har pensjonert implicit og password. Hos IETF samles anbefalingene, sammen med Proof Key for Code Exchange

(PKCE) og reglene for bearer-tokener, i ett OAuth 2.1-dokument. I det sekstende utkastet, datert 3. september 2026, er det fortsatt et Internet-Draft, altså et arbeid underveis og ikke en standard.

OAuth handler om autorisasjon, ikke om autentisering. Et token forteller API-et hva klienten har lov til; det sier ikke klienten noe pålitelig om hvem personen er. Å logge folk inn oppå OAuth er jobben til OpenID Connect, som legger til et token som nettopp er ment for at klienten skal lese det.

Bruk begrepet i en beslutning

Bruk «Hva er OAuth?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

**Se tjenestene våre** Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

**Skrev 1 000 RBAC-regler for å sikre Linux-servere** Ta kontakt →

## Hvorfor det er viktig

RFC 6749 åpner med problemet den ble skrevet for å få slutt på. Skulle en tredjepartsapplikasjon få tilgang til dataene dine, ga du den passordet ditt. Applikasjonen måtte da lagre passordet, gjerne i klartekst. Den nådde alt passordet nådde, så lenge den ville. Ville du stenge ute én applikasjon, måtte du bytte passord, og da stengte du samtidig ute alle de andre. Og ble én av dem kompromittert, fulgte passordet og alt bak det med. OAuth fjerner hvert av disse problemene: applikasjonen har aldri passordet, tokenet har et omfang og en utløpstid, og tilgangen kan trekkes tilbake fra én klient uten å røre resten.

Et team møter OAuth fra begge sider. Eier dere et API, lar OAuth partnere, integrasjoner og kundenes egne verktøy kalle det på vegne av en bruker, uten at noen lagrer andres passord. Bruker dere et, er OAuth det kalendere, e-postkasser, regnskapssystemer og fillagre tilbyr, så klienthalvdelen er kode teamet deres skriver og vedlikeholder. AI-agenter som handler for en person, sitter på samme plass som enhver annen klient: de skal ha et token begrenset til det personen har sagt ja til, aldri personens påloggingsinformasjon.

Scopes gjør prinsippet om minste privilegium til noe en server kan håndheve. Et token som lar en utskriftstjeneste lese bilder, bør ikke la den slette dem, og et token utstedt for ett API bør avvises av et annet. IETFs sikkerhetsveiledning fra

2025, RFC 9700, ber om akkurat det: rettighetene til et tilgangstoken holdt på det minimum bruksområdet krever, og tokener begrenset til ressursserveren de ble utstedt for.

## Slik ser det ut i praksis

RFC-ens eget eksempel er en utskriftstjeneste som vil hente bilder fra en bildedelingstjeneste. Her er den utvekslingen som authorization code-flyten med PKCE, formen dagens anbefalinger ber alle klienter bruke. Aller først lager klienten en tilfeldig code verifier på 43 til 128 tegn og holder den for seg selv. Den hasher verifieren med SHA-256, koder resultatet med base64url og sender det som code challenge når den videresender personens nettleser til autorisasjonsserveren:

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
    &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
    &code_challenge_method=S256 HTTP/1.1
Host: server.example.com
```

Autorisasjonsserveren logger personen inn, viser hva utskriftstjenesten ber om, og sender nettleseren tilbake med en kortlevd autorisasjonskode når personen har godkjent. Klienten sjekker at `state` er verdien den selv sendte:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA
          &state=xyz
```

Koden er verdiløs alene. Klienten løser den inn med en HTTPS-forespørsel til autorisasjonsserverens tokenendepunkt og sender denne gangen selve verifieren:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
```

```
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&client_id=s6BhdRkqt3
&code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Serveren hasher verifieren, sammenligner med challengen den lagret sammen med koden, og først når de to stemmer, returnerer den tokenene, merket slik at mellomlagre lar dem ligge:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Herfra sender klienten `Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA` med hvert kall til bilde-API-et. En time senere har tokenet utløpt, API-et svarer `401 Unauthorized` med `error="invalid_token"` i `WWW-Authenticate`-hodet, og klienten bruker refresh-tokenet til å hente et nytt. Ber klienten om noe utenfor det tildelte scopet, blir svaret i stedet `403 Forbidden` med `insufficient_scope`. Verifieren beskytter mot misbruk av en oppsnappet kode: en kode som kopieres fra enheten under overføringen — angrepet PKCE opprinnelig ble skrevet mot — kan ikke løses inn av noen som mangler verifieren. Selve verifieren ble ikke sendt i autorisasjonsforespørselen eller i omdirigeringen gjennom nettleseren.

Den samme delegeringen er det som lar programvare være med i arbeidet til agenter. På nettstedets side om å gjøre eksisterende programvare om til verktøy AI-agenter kan kalle, gir OAuth den delegerte tilgangen, så en agent handler på vegne av en bruker uten at noe passord skifter hender, og bare innenfor brukerens rettigheter.

## Hvor det går galt

**En slapp sjekk av redirect-URI-en sender koden til noen andre.** En autorisasjonsserver som godtar enhver redirect-URI som begynner med et registrert prefiks eller treffer et jokertegn, kan lures til å levere en kode til en

side angriperen styrer. RFC 9700 krever at autorisasjonsservere sammenligner redirect-URI-er med de registrerte ved eksakt strengsammenligning, med ett unntak for portnumre på localhost for native apper, og forbyr både klienter og servere å eksponere åpne videresendinger som sender nettleseren videre til en vilkårlig URL hentet fra en spørreparameter.

**Klienten tar imot et svar den aldri ba om.** Godtar klienten hvilken som helst kode som havner på redirect-URI-en, kan en angriper dytte sin egen kode gjennom nettleseren til et offer, og offeret ender med å jobbe i angriperens konto. Det er cross-site request forgery rettet mot omdirigeringen. En klient kan la PKCE beskytte seg når den har forsikret seg om at serveren støtter PKCE; ellers sender den en engangsverdi i `state`, bundet til personens nettleserøkt, og avviser et svar som kommer tilbake uten den.

**Tokener reiser i URL-er.** Implicit-flyten legger tilgangstokenet rett i omdirigeringen, der nettleserhistorikken og `Referer`-hoder kan bære det steder det aldri skulle, og ingenting binder det til klienten som ba om det. RFC 9700 sier at klienter ikke bør bruke den, og OAuth 2.1-utkastet har den ikke med i det hele tatt. Det samme gjelder et API som tar imot et token som spørreparameter: ifølge RFC 6750 bør bearer-tokener ikke sendes i sidens URL-er. Bruk authorization code-flyten, og send tokener i `Authorization`-hodet.

**Applikasjonen ber om passordet likevel.** Password-flyten, resource owner password credentials, lar en klient samle inn en persons brukernavn og passord og bytte dem mot et token, noe som undergraver selve grunnen til at OAuth finnes. RFC 9700 slår fast at den ikke skal brukes: den eksponerer påloggingsinformasjonen for klienten, venner folk til å taste passordet sitt andre steder enn hos autorisasjonsserveren, og den ble aldri laget for tofaktorautentisering eller pålogging i flere steg. Send folk til autorisasjonsserveren med authorization code-flyten, og bruk client credentials når det ikke er noen person involvert i det hele tatt.

**Et stjålet token virker for den som har det.** RFC 6750 definerer et bearer-token som et token enhver som har det i sin besittelse, kan bruke, uten ytterligere bevis. Hold tilgangstokener kortlevde, begrens hvert av dem til API-et det ble utstedt for, og bind dem til klienten med gjensidig TLS (RFC 8705) eller DPoP (RFC 9449) der det står mye på spill. Refresh-tokener for offentlige klienter, som mobil- og nettleserapper, må enten være bundet til klienten eller roteres: hver fornyelse gir et nytt token og ugyldiggjør det gamle. Blir en stjålet kopi også

brukt, ender den ene parten opp med å vise fram et ugyldiggjort token, serveren tilbakekaller det aktive, og angrepet stopper, mot at den ekte klienten må be personen om en ny tillatelse.

## Les mer

- RFC 6749: The OAuth 2.0 Authorization Framework — de fire rollene, de fire grant-typene og endepunktene slik de ble publisert i oktober 2012.
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients — code verifier og code challenge, med S256-eksempelet som er brukt ovenfor.
- RFC 9700: Best Current Practice for OAuth 2.0 Security — veiledningen fra januar 2025: eksakt sammenligning av redirect-URI-er, PKCE for alle klienter og farvel til implicit- og password-flyten.
- RFC 6750: Bearer Token Usage — hvordan en klient viser fram et tilgangstoken, og hvordan et API sier at det mangler, har utløpt eller gir for lite tilgang.
- The OAuth 2.1 Authorization Framework — arbeidsgruppens utkast som samler disse dokumentene i ett, fortsatt et Internet-Draft i oktober 2026.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

**Se hva det koster** Ta kontakt →

# Gjør begrepene praktiske

Spør hvordan «Hva er OAuth?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/oauth/