Engineer

OAuth

✒ en måde at give en app begrænset adgang uden adgangskoden

Hvad det er

OAuth er en protokol, der lader et program bruge en persons data hos en anden tjeneste, med personens samtykke og inden for grænser, personen selv kan se, uden at programmet nogensinde får personens adgangskode at vide. Den udgave, der bruges i dag, er OAuth 2.0, som IETF udgav i oktober 2012 som RFC 6749, og den står lige så tit skrevet OAuth 2.0 eller OAuth2 som bare OAuth. Den afløste OAuth 1.0, og de to versioner er ikke kompatible.

Specifikationen arbejder med fire roller. Ressourceejeren er den, der kan give adgang, som regel en person. Ressourceserveren har de beskyttede data og svarer på forespørgsler, der har et token med. Klienten er programmet, der vil have adgang: en webapp, en mobilapp, et script, en AI-agent. Autorisationsserveren logger ejeren ind, beder om samtykke og udsteder tokens. Et lille produkt kan køre ressourceserveren og autorisationsserveren som ét system; i en virksomhed sidder der oftere én identitetsudbyder foran mange API'er.

Det, klienten får, er et adgangstoken (access token): en tekststreng, der står for et bestemt omfang af adgang i en bestemt periode. Klienten sender tokenet med hver forespørgsel til ressourceserveren. RFC'en kalder strengen normalt uigennemsigtig for klienten, som bærer den uden at skulle læse den. Autorisationsserveren kan også udstede et refresh-token, som klienten bytter til et nyt adgangstoken, når det gamle udløber, uden at spørge personen igen.

Sit første token får klienten gennem en grant, en tilladelsestype. RFC 6749 definerer fire: authorization code, implicit, resource owner password credentials og client credentials. Client credentials er til et program, der handler på egne vegne, helt uden en person. Authorization code er den, der tæller for alt andet, fordi de gældende sikkerhedsanbefalinger har pensioneret implicit og password.

Hos IETF samles anbefalingerne, sammen med Proof Key for Code Exchange (PKCE) og reglerne for bearer tokens, i ét OAuth 2.1-dokument. I det sekstende udkast, dateret 3. september 2026, er det stadig et Internet-Draft, altså et arbejde i gang og ikke en standard.

OAuth handler om autorisation, ikke om autentificering. Et token fortæller API'et, hvad klienten må; det fortæller ikke klienten noget pålideligt om, hvem personen er. At logge folk ind oven på OAuth er opgaven for OpenID Connect, som tilføjer et token, der netop er beregnet til, at klienten læser det.

Brug begrebet i en beslutning

Brug »Hvad er OAuth?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

RFC 6749 indleder med det problem, den skulle gøre en ende på. Skulle et tredjepartsprogram have adgang til dine data, gav du det din adgangskode. Programmet måtte så gemme den, typisk i klartekst. Det nåede alt det, adgangskoden nåede, så længe det ville. Ville du lukke for ét program, måtte du skifte adgangskoden, og så lukkede du samtidig for alle de andre. Og blev ét af dem kompromitteret, fulgte adgangskoden og alt bag den med. OAuth fjerner hvert af de problemer: programmet har aldrig adgangskoden, tokenet har et omfang og et udløb, og adgangen kan trækkes tilbage fra én klient uden at røre resten.

Et team møder OAuth fra begge sider. Ejer I et API, lader OAuth partnere, integrationer og kundernes egne værktøjer kalde det på en brugers vegne, uden at nogen gemmer andres adgangskoder. Bruger I et, er OAuth det, kalendere, postkasser, regnskabssystemer og fillagre stiller til rådighed, så klienthalvdelen er kode, jeres team skriver og holder ved lige. AI-agenter, der handler for en person, sidder på samme plads som enhver anden klient: de skal have et token, der er begrænset til det, personen har sagt ja til, og aldrig personens loginoplysninger.

Scopes gør princippet om mindste privilegium til noget, en server kan håndhæve. Et token, der lader en printtjeneste læse fotos, bør ikke lade den slette dem, og et token udstedt til ét API bør afvises af et andet. IETF's sikkerhedsvejledning fra 2025, RFC 9700, beder om netop det: et adgangstokens rettigheder holdt på det minimum, brugsscenariet kræver, og tokens begrænset til den ressourceserver, de blev udstedt til.

Sådan ser det ud i praksis

RFC'ens eget eksempel er en printtjeneste, der vil hente fotos fra en fotodelingstjeneste. Her er den udveksling som authorization code-flowet med PKCE, den form, de gældende anbefalinger beder alle klienter om at bruge. Allerførst finder klienten på en tilfældig code verifier på 43 til 128 tegn og holder den for sig selv. Den hasher verifieren med SHA-256 og sender resultatet, base64url-kodet, som code challenge med personens browser til autorisationsserveren:

```
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
```

Autorisationsserveren logger personen ind, viser, hvad printtjenesten beder om, og sender browseren tilbage med en kortlivet autorisationskode, når personen har godkendt. Klienten tjekker, at `state` er den værdi, den selv sendte:

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

Koden er intet værd alene. Klienten indløser den med en HTTPS-forespørgsel til autorisationsserverens token-endpoint og sender denne gang 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 den challenge, den gemte sammen med koden, og først når de to stemmer overens, returnerer den tokens, mærket sådan, at mellemliggende cacher lader dem være:

```
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 kald til foto-API'et. En time senere er tokenet udløbet, API'et svarer `401 Unauthorized` med `error="invalid_token"` i sin `WWW-Authenticate` -header, og klienten bruger refresh-tokenet på at hente et nyt. Beder klienten om noget uden for det tildelte scope, lyder svaret i stedet `403 Forbidden` med `insufficient_scope`. Verifieren beskytter mod misbrug af en opsnappet kode: en kode, der kopieres fra enheden under overførslen — det angreb, PKCE oprindeligt blev skrevet imod — kan ikke indløses af nogen, der mangler verifieren. Selve verifieren blev hverken sendt i autorisationsforespørgslen eller i browserens omdirigering.

Den samme delegering er det, der lader software indgå i agenters arbejde. På sitets side om at gøre eksisterende software til værktøjer, AI-agenter kan kalde, giver OAuth den delegerede adgang, så en agent handler på vegne af en bruger, uden at en adgangskode skifter hænder, og kun inden for brugerens rettigheder.

Hvor det går galt

En løs kontrol af redirect-URI'en sender koden til en anden. En autorisationsserver, der godtager enhver redirect-URI, som begynder med et registreret præfiks eller passer på et wildcard, kan lokkes til at levere en kode til en side, angriberen styrer. RFC 9700 kræver, at autorisationsservere

sammenligner redirect-URI'er med de registrerede ved præcis strengsammenligning, med én undtagelse for portnumre på localhost til native apps, og forbyder både klienter og servere at køre åbne redirectors, der sender browseren videre til en vilkårlig URL fra en forespørgselsparameter.

Klienten tager imod et svar, den aldrig har bedt om. Godtager klienten den kode, der nu engang lander på dens redirect-URI, kan en angriber skubbe sin egen kode gennem et offers browser, så offeret ender med at arbejde i angriberens konto. Det er cross-site request forgery rettet mod omdirigeringen. En klient kan lade PKCE beskytte sig, når den har sikret sig, at serveren understøtter PKCE; ellers sender den en engangsværdi i `state`, bundet til personens browsersession, og afviser et svar, der kommer tilbage uden den.

Tokens rejser i URL'er. Implicit-flowet lægger adgangstokenet direkte i omdirigeringen, hvor browserhistorikken og `Referer`-headere kan bære det steder hen, det aldrig var tiltænkt, og der er intet, der binder det til den klient, der bad om det. RFC 9700 siger, at klienter ikke bør bruge det, og OAuth 2.1-udkastet har det slet ikke med. Det samme gælder et API, der tager imod et token som forespørgselsparameter: ifølge RFC 6750 bør bearer tokens ikke sendes i sidens URL'er. Brug authorization code-flowet, og send tokens i `Authorization`-headeren.

Programmet beder alligevel om adgangskoden. Password-flowet, resource owner password credentials, lader en klient indsamle en persons brugernavn og adgangskode og bytte dem til et token, hvilket ophæver selve grunden til, at OAuth findes. RFC 9700 slår fast, at det ikke må bruges: det udsætter loginoplysningerne for klienten, vænner folk til at taste deres adgangskode andre steder end hos autorisationsserveren, og det er slet ikke lavet til tofaktorgodkendelse og login i flere trin. Send folk til autorisationsserveren med authorization code-flowet, og brug client credentials, når der slet ikke er en person involveret.

Et stjålet token virker for den, der har det. RFC 6750 definerer et bearer token som et token, enhver, der har det i sin besiddelse, kan bruge uden yderligere bevis. Hold adgangstokens kortlivede, begræns hvert af dem til det API, det blev udstedt til, og bind dem til klienten med mutual TLS (RFC 8705) eller DPoP (RFC 9449), hvor indsatsen retfærdiggør det. Refresh-tokens til offentlige klienter, såsom mobil- og browserapps, skal enten være bundet til klienten eller roteres: hver fornyelse giver et nyt token og ugyldiggør det gamle. Bliver en stjålet kopi

også brugt, ender den ene part med at vise et ugyldiggjort token frem, serveren tilbagekalder det aktive, og angrebet stopper, mod at den rigtige klient må bede personen om en ny tilladelse.

Læs mere

- RFC 6749: The OAuth 2.0 Authorization Framework — de fire roller, de fire grant-typer og endepunkterne, som de blev udgivet i oktober 2012.
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients — code verifier og code challenge, med det S256-eksempel, der er brugt ovenfor.
- RFC 9700: Best Current Practice for OAuth 2.0 Security — vejledningen fra januar 2025: præcis sammenligning af redirect-URI'er, PKCE for alle klienter og farvel til implicit- og password-flowet.
- RFC 6750: Bearer Token Usage — hvordan en klient viser et adgangstoken frem, og hvordan et API siger, at det mangler, er udløbet eller giver for lidt adgang.
- The OAuth 2.1 Authorization Framework — arbejdsgruppens udkast, der samler dokumenterne i ét, stadig et Internet-Draft i oktober 2026.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er OAuth?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/oauth/