Engineer

OAuth

ett sätt att ge en app begränsad åtkomst utan lösenordet

Vad det är

OAuth är ett protokoll som låter en applikation använda data som en person har hos en annan tjänst, med personens medgivande och inom gränser som personen själv kan se, utan att applikationen någonsin får veta lösenordet. Den version som används i dag är OAuth 2.0, som IETF publicerade i oktober 2012 som RFC 6749, och den skrivs lika ofta OAuth 2.0 eller OAuth2 som bara OAuth. Den ersatte OAuth 1.0, och de två versionerna är inte kompatibla.

Specifikationen har fyra roller. Resursägaren är den som kan ge åtkomst, oftast en person. Resursservern har de skyddade uppgifterna och svarar på varje begäran som bär en token. Klienten är applikationen som vill ha åtkomst: en webbapp, en mobilapp, ett skript, en AI-agent. Auktoriseringsservern loggar in ägaren, ber om medgivande och utfärdar tokens. En liten produkt kan köra resursservern och auktoriseringsservern som ett och samma system; i ett företag står det oftare en identitetsleverantör framför många API:er.

Det klienten får är en åtkomsttoken (access token): en teckensträng som står för en viss omfattning av åtkomst under en viss tid. Klienten skickar den med varje begäran till resursservern. RFC:n beskriver strängen som i regel ogenomskinlig för klienten, som bär den utan att vara tänkt att läsa den. Auktoriseringsservern kan också lämna ut en refresh-token, som klienten byter mot en ny åtkomsttoken när den gamla går ut, utan att fråga personen igen.

Sin första token får klienten genom en grant, en typ av beviljande. RFC 6749 definierar fyra: authorization code, implicit, resource owner password credentials och client credentials. Client credentials är till för ett program som agerar för egen räkning, utan någon person inblandad. Authorization code är den som gäller för allt annat, eftersom dagens säkerhetsrekommendationer har pensionerat implicit och password. Hos IETF förs rekommendationerna,

tillsammans med Proof Key for Code Exchange (PKCE) och reglerna för bearer-tokens, samman i ett enda OAuth 2.1-dokument. I det sextonde utkastet, daterat den 3 september 2026, är det fortfarande ett Internet-Draft, alltså ett pågående arbete och inte en standard.

OAuth handlar om auktorisering, inte om autentisering. En token talar om för API:et vad klienten får göra; den säger inget tillförlitligt till klienten om vem personen är. Att logga in människor ovanpå OAuth är uppgiften för OpenID Connect, som lägger till en token som just är avsedd att läsas av klienten.

Använd begreppet i ett beslut

Använd ”Vad är OAuth?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

RFC 6749 inleds med problemet den skrevs för att göra slut på. Skulle en tredjepartsapplikation komma åt dina uppgifter, gav du den ditt lösenord. Applikationen måste då lagra lösenordet, ofta i klartext. Den nådde allt som lösenordet nådde, så länge den ville. Ville du stänga ute en enda applikation fick du byta lösenord, och då stängde du samtidigt ute alla andra. Och blev en av dem kompromitterad följde lösenordet och allt bakom det med. OAuth tar bort vart och ett av de problemen: applikationen har aldrig lösenordet, token har en omfattning och ett utgångsdatum, och åtkomsten kan dras tillbaka från en klient utan att resten påverkas.

Ett team möter OAuth från båda hållen. Äger ni ett API låter OAuth partner, integrationer och kundernas egna verktyg anropa det för en användares räkning, utan att någon lagrar någon annans lösenord. Använder ni ett är OAuth vad kalendrar, e-postlådor, ekonomisystem och fillagringstjänster erbjuder, så klienthalvan är kod som ert team skriver och håller vid liv. AI-agenter som agerar åt en person sitter på samma plats som vilken annan klient som helst: de ska ha en token som är begränsad till det personen har godkänt, aldrig personens inloggningsuppgifter.

Scopes gör principen om minsta behörighet till något en server kan upprätthålla. En token som låter en utskriftstjänst läsa bilder ska inte låta den radera dem, och en token som utfärdats för ett API ska avvisas av ett annat. IETF:s säkerhetsvägledning från 2025, RFC 9700, kräver just det: en åtkomsttokens rättigheter hållna till det minimum användningsfallet behöver, och tokens begränsade till den resursserver de utfärdades för.

Så ser det ut i praktiken

RFC:ns eget exempel är en utskriftstjänst som vill hämta bilder från en bilddelningstjänst. Här är det utbytet som authorization code-flödet med PKCE, den form dagens rekommendationer ber alla klienter använda. Allra först skapar klienten en slumpmässig code verifier på 43 till 128 tecken och behåller den för sig själv. Den hashar verifieraren med SHA-256, kodar resultatet med base64url och skickar det som code challenge när den leder personens webbläsare vidare till auktoriseringsservern:

```
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
```

Auktoriseringsservern loggar in personen, visar vad utskriftstjänsten ber om och skickar, när personen har godkänt, tillbaka webbläsaren med en kortlivad auktoriseringskod. Klienten kontrollerar att `state` är det värde den själv skickade:

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

Koden är värdelös i sig. Klienten löser in den med en HTTPS-begäran till auktoriseringsserverns token-endpoint och skickar den här gången själva verifieraren:

```
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
```

Servern hashar verifieraren, jämför med den challenge den sparade tillsammans med koden, och först när de stämmer överens returnerar den tokens, märkta så att ingen cache behåller dem:

```
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"
}
```

Härifrån skickar klienten `Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA` med varje anrop till bild-API:et. En timme senare har token gått ut, API:et svarar `401 Unauthorized` med `error="invalid_token"` i sitt `WWW-Authenticate`-huvud, och klienten använder sin refresh-token för att hämta en ny. Ber klienten om något utanför det beviljade scopet blir svaret i stället `403 Forbidden` med `insufficient_scope`. Verifieraren skyddar mot missbruk av en uppsnappad kod: en kod som kopieras från enheten under överföringen — den attack PKCE ursprungligen skrevs mot — kan inte lösas in av någon som saknar verifieraren. Själva verifieraren skickades varken i auktoriseringsbegäran eller i webbläsarens omdirigering.

Samma delegering är det som låter programvara delta i agenters arbete. På webbplatsens sida om att göra befintlig programvara till verktyg som AI-agenter kan anropa ger OAuth den delegerade åtkomsten, så att en agent agerar åt en användare utan att ett lösenord lämnas vidare, och bara inom den användarens rättigheter.

Var det går fel

En slapp kontroll av redirect-URI:n skickar koden till någon annan. En auktoriseringsserver som godtar varje redirect-URI som börjar med ett registrerat prefix eller matchar ett jokertecken kan luras att lämna en kod till en sida som angriparen styr. RFC 9700 kräver att auktoriseringsservrar jämför redirect-URI:er med de registrerade genom exakt strängjämförelse, med ett undantag för portnummer på localhost för native-appar, och förbjuder både klienter och servrar att exponera öppna omdirigerare som skickar webbläsaren vidare till en godtycklig URL ur en frågeparameter.

Klienten tar emot ett svar den aldrig bad om. Tar klienten emot vilken kod som helst som hamnar på dess redirect-URI kan en angripare trycka in sin egen kod genom ett offers webbläsare, och offret slutar med att arbeta i angriparens konto. Det är cross-site request forgery riktad mot omdirigeringen. En klient kan låta PKCE skydda den när den har försäkrat sig om att servern stöder PKCE; annars skickar den ett engångsvärde i `state`, bundet till personens webbläsarsession, och avvisar ett svar som kommer tillbaka utan det.

Tokens färdas i URL:er. Implicit-flödet lägger åtkomsttoken direkt i omdirigeringen, där webbläsarhistorik och `Referer`-huvuden kan bära den dit den aldrig borde komma, och ingenting binder den till klienten som bad om den. RFC 9700 säger att klienter inte bör använda det, och OAuth 2.1-utkastet tar inte med det alls. Detsamma gäller ett API som tar emot en token som frågeparameter: enligt RFC 6750 bör bearer-tokens inte skickas i sidans URL:er. Använd authorization code-flödet och skicka tokens i `Authorization`-huvudet.

Appen ber om lösenordet ändå. Password-flödet, resource owner password credentials, låter en klient samla in en persons användarnamn och lösenord och byta dem mot en token, vilket undergräver själva skälet till att OAuth finns. RFC 9700 slår fast att det inte får användas: det exponerar inloggningsuppgifterna för klienten, vänjer människor vid att skriva sitt lösenord på andra ställen än hos auktoriseringsservern, och det utformades aldrig för tvåfaktorsautentisering eller inloggningar i flera steg. Skicka människor till auktoriseringsservern med authorization code-flödet, och använd client credentials när ingen person alls är inblandad.

En stulen token fungerar för den som har den. RFC 6750 definierar en bearer-token som en token som vem som helst som har den i sin besittning kan använda, utan ytterligare bevis. Håll åtkomsttokens kortlivade, begränsa var och en till det

API den utfärdades för, och bind dem till klienten med ömsesidig TLS (RFC 8705) eller DPoP (RFC 9449) där mycket står på spel. Refresh-tokens för publika klienter, som mobil- och webbläsarappar, måste antingen vara bundna till klienten eller roteras: varje förnyelse ger en ny token och ogiltigförklarar den gamla. Används en stulen kopia också slutar den ena parten med att visa upp en ogiltigförklarad token, servern återkallar den aktiva, och attacken stoppas, till priset av att den riktiga klienten måste be personen om ett nytt medgivande.

Läs mer

- RFC 6749: The OAuth 2.0 Authorization Framework — de fyra rollerna, de fyra grant-typerna och slutpunkterna så som de publicerades i oktober 2012.
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients — code verifier och code challenge, med S256-exemplet som används ovan.
- RFC 9700: Best Current Practice for OAuth 2.0 Security — vägledningen från januari 2025: exakt jämförelse av redirect-URI:er, PKCE för alla klienter och inget implicit- eller password-flöde.
- RFC 6750: Bearer Token Usage — hur en klient visar upp en åtkomsttoken och hur ett API säger att den saknas, har gått ut eller ger för snäv åtkomst.
- The OAuth 2.1 Authorization Framework — arbetsgruppens utkast som för samman dessa dokument till ett, fortfarande ett Internet-Draft i oktober 2026.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är OAuth?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/oauth/