Engineer

OAuth

begrenzter App-Zugriff ohne Weitergabe des Passworts

Was es ist

OAuth ist ein Protokoll, mit dem eine Anwendung auf Daten einer Person bei einem anderen Dienst zugreifen kann: mit deren Zustimmung und innerhalb von Grenzen, die diese Person einsehen kann, ohne je deren Passwort zu erfahren. Verwendet wird OAuth 2.0, das die IETF im Oktober 2012 als RFC 6749 veröffentlichte. OAuth 2.0 und OAuth2 liest man ebenso häufig wie schlicht OAuth. Es ersetzte OAuth 1.0. Die beiden sind nicht kompatibel.

Die Spezifikation nennt vier Rollen. Der Ressourceninhaber kann Zugriff gewähren und ist gewöhnlich eine Person. Der Ressourcenserver hält die geschützten Daten und beantwortet Anfragen mit Token. Der Client ist die Anwendung, die Zugriff möchte: eine Web- oder Mobil-App, ein Skript oder ein KI-Agent. Der Autorisierungsserver meldet den Inhaber an, fragt nach seiner Zustimmung und stellt Token aus. Ein kleines Produkt kann Ressourcenserver und Autorisierungsserver als ein System betreiben. Ein Unternehmen hat häufiger einen Identitätsanbieter vor vielen APIs.

Der Client erhält ein Zugriffstoken: eine Zeichenfolge, die für einen bestimmten Umfang und eine bestimmte Gültigkeitsdauer des Zugriffs steht. Er sendet es bei jeder Anfrage an den Ressourcenserver. Der RFC nennt die Zeichenfolge für den Client gewöhnlich opak: Er trägt sie mit, soll sie aber nicht lesen. Der Autorisierungsserver kann zusätzlich ein Aktualisierungstoken ausgeben. Dieses tauscht der Client gegen ein neues Zugriffstoken, wenn das alte abläuft, ohne die Person erneut zu fragen.

Sein erstes Token erhält ein Client über einen Grant. RFC 6749 definiert vier: Authorization Code, Implicit, Resource Owner Password Credentials und Client Credentials. Client Credentials dient einem Programm, das in eigenem Namen handelt, ohne beteiligte Person. Für alles andere zählt der Authorization-Code-

Grant, weil die aktuellen Sicherheitsempfehlungen den Implicit- und den Passwort-Grant ausgemustert haben. Diese Empfehlungen werden zusammen mit Proof Key for Code Exchange (PKCE) und den Bearer-Token-Regeln bei der IETF in einem gemeinsamen OAuth-2.1-Dokument gebündelt. In seinem sechzehnten Entwurf vom 3. September 2026 ist es weiterhin ein Internet-Draft, also ein noch in Arbeit befindlicher Entwurf und kein Standard.

OAuth regelt die Autorisierung, nicht die Authentifizierung. Ein Token sagt der API, was der Client tun darf. Dem Client sagt es nichts Verlässliches darüber, wer die Person ist. Menschen auf Grundlage von OAuth anzumelden ist Aufgabe von OpenID Connect. Es ergänzt ein Token, das der Client lesen soll.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist OAuth?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

RFC 6749 beginnt mit dem Problem, das er beseitigen sollte. Um einer fremden Anwendung Zugriff auf eigene Daten zu geben, gab man ihr sein Passwort. Die Anwendung musste es speichern, meist im Klartext. Sie erreichte alles, was das Passwort erreichte, so lange sie wollte. Eine Anwendung auszuschließen ging nur durch eine Passwortänderung, die alle anderen ebenfalls aussperrte. Wurde eine von ihnen kompromittiert, gingen das Passwort und alles dahinter mit verloren. OAuth beseitigt jeden dieser Punkte: Die Anwendung hält nie das Passwort, das Token trägt Umfang und Ablaufzeit, und einem Client kann der Zugriff entzogen werden, ohne die übrigen anzutasten.

Ein Team begegnet OAuth auf beiden Seiten. Mit einer eigenen API erlaubt es Partnern, Integrationen und kundeneigenen Werkzeugen, diese im Namen eines Nutzers aufzurufen, ohne fremde Passwörter zu speichern. Wer eine API nutzt, trifft OAuth bei Kalendern, Postfächern, Buchhaltungssystemen und Dateispeichern. Die Client-Seite ist deshalb Code, den das Team schreibt und

funktionsfähig hält. KI-Agenten, die für eine Person handeln, nehmen dieselbe Rolle ein wie jeder andere Client: Sie sollten ein Token nur mit den von ihr erlaubten Rechten halten, nie ihre Zugangsdaten.

Scopes machen das Prinzip geringster Rechte serverseitig durchsetzbar. Ein Token, mit dem ein Druckdienst Fotos lesen darf, sollte ihm nicht erlauben, sie zu löschen. Ein für eine API ausgestelltes Token sollte eine andere zurückweisen. Genau das verlangen die Sicherheitsempfehlungen der IETF von 2025, RFC 9700: Rechte eines Zugriffstokens auf das für den Anwendungsfall nötige Minimum beschränken und Token an den Ressourcenserver binden, für den sie ausgestellt wurden.

Wie es in der Praxis aussieht

Das Beispiel des RFC ist ein Druckdienst, der Fotos von einer Foto-Website möchte. Hier folgt dieser Austausch als Authorization-Code-Grant mit PKCE, wie es die aktuellen Empfehlungen von jedem Client verlangen. Zuerst erzeugt der Client einen zufälligen Code-Verifier mit 43 bis 128 Zeichen und behält ihn für sich. Er hasht ihn mit SHA-256 und leitet den Browser der Person mit dem base64url-kodierten Ergebnis, der Code-Challenge, an den Autorisierungsserver weiter:

```
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
```

Der Autorisierungsserver meldet die Person an, zeigt die vom Druckdienst angefragten Rechte und schickt nach ihrer Zustimmung den Browser mit einem kurzlebigen Autorisierungscode zurück. Der Client prüft, ob `state` dem von ihm gesendeten Wert entspricht:

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

Der Code ist allein wertlos. Der Client löst ihn mit einer HTTPS-Anfrage am Token-Endpunkt des Autorisierungsservers ein und sendet diesmal den Verifier selbst:

```
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
```

Der Server hasht den Verifier, vergleicht ihn mit der neben dem Code gespeicherten Challenge und liefert die Token nur bei Übereinstimmung zurück. Sie sind so markiert, dass kein Cache sie behält:

```
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"
}
```

Von nun an sendet der Client bei jedem Aufruf der Foto-API `Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA` . Eine Stunde später ist das Token abgelaufen. Die API antwortet mit `401 Unauthorized` und `error="invalid_token"` im Header `WWW-Authenticate` , und der Client tauscht das Aktualisierungstoken gegen ein neues Zugriffstoken. Eine Anfrage außerhalb des gewährten Umfangs erhält stattdessen `403 Forbidden` und `insufficient_scope` . Der Verifier schützt vor dem Missbrauch eines abgefangenen Codes: Ein beim Transport vom Gerät kopierter Code, der ursprüngliche Anlass für PKCE, lässt sich ohne Verifier nicht einlösen. Der Verifier selbst wurde weder in der Autorisierungsanfrage noch in der Browserweiterleitung gesendet.

Dieselbe Delegation ermöglicht Software die Teilnahme an Agentenabläufen. Auf der Seite dieser Website über Werkzeuge aus vorhandener Unternehmenssoftware, die KI-Agenten aufrufen können liefert OAuth den delegierten Zugriff: Ein Agent handelt für einen Nutzer, ohne dass ein Passwort weitergegeben wird, und nur innerhalb der Rechte dieses Nutzers.

Wo es schiefgeht

Eine lockere Prüfung der Weiterleitung schickt den Code an jemand anderen. Akzeptiert ein Autorisierungsserver jede Weiterleitungs-URI mit einem registrierten Präfix oder einem Platzhaltermuster, lässt er sich zur Zustellung eines Codes an eine Angreiferseite bringen. RFC 9700 verlangt den exakten Zeichenfolgenvergleich der Weiterleitungs-URIs mit den registrierten. Die einzige Ausnahme sind localhost-Portnummern für native Apps. Clients wie Server dürfen außerdem keine offenen Weiterleiter betreiben, die den Browser an jede in einem Abfrageparameter gefundene URL schicken.

Der Callback akzeptiert eine nie angefragte Antwort. Nimmt der Client jeden an seiner Weiterleitungs-URI eintreffenden Code an, kann ein Angreifer seinen eigenen durch den Browser eines Opfers schleusen. Das Opfer arbeitet dann im Konto des Angreifers. Es ist Cross-Site Request Forgery gegen die Weiterleitung. Ein Client darf sich zum Schutz auf PKCE verlassen, sobald er die PKCE-Unterstützung des Servers bestätigt hat. Andernfalls sendet er einen einmaligen, an die Browsersitzung gebundenen Wert `state` und weist jede Antwort zurück, die ihn nicht mitbringt.

Token reisen in URLs. Der Implicit-Grant steckt das Zugriffstoken direkt in die Weiterleitung. Browserhistorie und `Referer` -Header können es dadurch an unbeabsichtigte Orte tragen. Nichts bindet es an den anfragenden Client. RFC 9700 sagt, Clients sollten das Verfahren nicht verwenden. Der OAuth-2.1-Entwurf lässt es weg. Dasselbe gilt für eine API, die Token als Abfrageparameter annimmt: Laut RFC 6750 sollten Seiten-URLs niemals Bearer-Token enthalten. Verwenden Sie den Code-Grant und senden Sie Token im Header `Authorization` .

Die App fragt trotzdem nach dem Passwort. Der Resource-Owner-Password-Credentials-Grant lässt einen Client Nutzername und Passwort einer Person einsammeln und gegen ein Token tauschen. Das unterläuft den Zweck von OAuth. RFC 9700 sagt, dass er nicht verwendet werden darf: Er legt dem Client die Zugangsdaten offen, gewöhnt Menschen an Passworteingaben außerhalb des Autorisierungsservers und war nie für Zwei-Faktor-Authentifizierung oder

mehrstufige Anmeldungen gedacht. Leiten Sie Menschen mit dem Code-Grant zum Autorisierungsserver und verwenden Sie Client Credentials, wenn überhaupt keine Person beteiligt ist.

Ein gestohlenes Token funktioniert für jeden Besitzer. RFC 6750 definiert ein Bearer-Token so, dass jeder Beteiligte es verwenden kann, der es besitzt. Der Besitz ist der ganze Nachweis. Halten Sie Zugriffstoken kurzlebig, beschränken Sie jedes auf seine vorgesehene API und binden Sie Token bei entsprechend hohem Risiko mit gegenseitigem TLS (RFC 8705) oder DPoP (RFC 9449) an den Client. Aktualisierungstoken öffentlicher Clients, etwa Mobil- und Browser-Apps, müssen entweder an den Client gebunden oder rotiert werden: Jede Aktualisierung liefert ein neues Token und entwertet das alte. Wird auch eine gestohlene Kopie verwendet, legt eine Seite schließlich ein entwertetes Token vor. Der Server widerruft das aktive, und der Angriff endet. Der Preis ist, dass der echte Client die Person um einen neuen Grant bitten muss.

Weiterführende Lektüre

- RFC 6749: Das OAuth-2.0-Autorisierungsframework — die vier Rollen, vier Grant-Arten und Endpunkte in der Fassung vom Oktober 2012.
- RFC 7636: Proof Key for Code Exchange für öffentliche OAuth-Clients — Code-Verifier und Challenge mit dem oben verwendeten S256-Beispiel.
- RFC 9700: Aktuelle bewährte Sicherheitspraktiken für OAuth 2.0 — die Empfehlungen vom Januar 2025: exakter Weiterleitungsabgleich, PKCE für jeden Client und weder Implicit- noch Passwort-Grant.
- RFC 6750: Verwendung von Bearer-Token — wie ein Client ein Zugriffstoken vorlegt und eine API ein fehlendes, abgelaufenes oder zu eng begrenztes meldet.
- Das OAuth-2.1-Autorisierungsframework — der Arbeitsgruppenentwurf, der die Dokumente vereint und im Oktober 2026 weiter ein Internet-Draft ist.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist OAuth?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/oauth/