Engineer

OpenID Connect

et identitetslag oppå OAuth for å logge inn brukere

Hva det er

OpenID Connect er en standard som lar en applikasjon få vite hvem en person er, fra en tjeneste som nettopp har logget personen inn. Spesifikasjonen kaller seg selv et enkelt identitetslag oppå OAuth 2.0, og det er akkurat slik den er bygget: de samme omdirigeringene, endepunktene og tokenene som i OAuth, pluss ett token til, ett endepunkt til og reglene for hvordan de kontrolleres. I dagligtale blir den ofte forkortet til OIDC. Kjernespesifikasjonen ble endelig i februar 2014. Den gjeldende teksten, der andre runde med rettelser er tatt inn, er datert 15. desember 2023 og vedlikeholdes av en arbeidsgruppe i OpenID Foundation.

OAuth alene svarer på spørsmålet «hva har denne applikasjonen lov til?». Applikasjonen får et tilgangstoken som er ment for et API, og det er ikke meningen at applikasjonen selv skal lese det. OpenID Connect stiller i tillegg spørsmålet «hvem logget inn?» og besvarer det med et ID-token: et JSON Web Token, adressert til applikasjonen selv, som forteller hvem personen er, hvem som går god for det, når tokenet ble utstedt, og hvilken applikasjon det er ment for. ID-tokener skal være signert. Spesifikasjonen tillater bare `alg: none` hvis responstypen ikke returnerer noe ID-token fra autorisasjonsendepunktet, og klienten uttrykkelig ba om `none` ved registrering.

Til denne jobben gir spesifikasjonen OAuth-rollene nye navn. En OpenID Provider (OP) er en autorisasjonsserver som autentiserer personer og utsteder ID-tokener. En Relying Party (RP) er OAuth-klienten som trenger å vite hvem personen er, og personen selv kalles End-User. En forespørsel blir en OpenID Connect-forespørsel ved å ha `openid` med i scopet, og den verdien er obligatorisk: mangler den, er oppførselen etter spesifikasjonens egne ord helt uspesifisert.

Andre scope-verdier ber om profildata. `profile` dekker navn, kallenavn, bilde, språkinnstilling og lignende claims, `email` gir `email` og `email_verified`, og `address` og `phone` gjør det navnene sier. Applikasjonen kan hente disse claimene fra leverandørens UserInfo-endepunkt, et helt vanlig OAuth-beskyttet API som den kaller med tilgangstokenet.

Det finnes tre måter å kjøre det på. Authorization code-flyten sender bare en kortlivet kode gjennom nettleseren og henter selve tokenene med en HTTPS-forespørsel til tokenendepunktet. Implicit-flyten leverer tokenene rett i omdirigeringen, og hybrid-flyten blander de to. En tilhørende spesifikasjon, OpenID Connect Discovery 1.0, lar applikasjonen slå opp alle endepunkter og signeringsnøkler ut fra én adresse.

Bruk begrepet i en beslutning

Bruk «Hva er OpenID Connect?» 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

Grunnen til å ta det i bruk er single sign-on. Med én identitetsleverandør foran mange applikasjoner har folk ett sett med innloggingsopplysninger, applikasjonene lagrer ingen av dem, og reglene for hvordan man logger inn, fra krav til passord til en ekstra faktor, settes ett sted, hos leverandøren. Sperrer man en konto hos leverandøren, kan den ikke lenger starte en ny økt i noen av applikasjonene som lener seg på den, og dermed blir offboarding ett steg og ikke en sjekkliste.

Standarden tetter også et hull som team faller i når de bygger innlogging på ren OAuth. RFC 6749 advarer om at en offentlig klient i implicit-flyten ikke kan se hvilken klient et tilgangstoken ble utstedt til. En klient som går ut fra at bare ressurseieren kan gi den et gyldig token, kan derfor lures: En angriper med et token utstedt til en annen applikasjon leverer det inn og blir tatt for personen bak. Et ID-token oppgir i `aud`-claimet hvilken applikasjon det er til, og applikasjonen kontrollerer det.

Fordi det er en standard, kan arbeidet gjenbrukes. Spesifikasjonen lover profilopplysninger på en interoperabel måte, så koden som logger folk inn via én leverandør, kan logge dem inn via en annen også. Med Discovery lærer applikasjonen leverandørens endepunkter og signeringsnøkler å kjenne ut fra issuer-URL-en i stedet for fra innstillinger noen har skrevet av for hånd.

Til slutt gir OpenID Connect hver person en stabil identifikator. Spesifikasjonen sier klart at paret `iss` (utstederen) og `sub` (subjektidentifikatoren) er de eneste claimene en applikasjon kan stole på som stabil identifikator for en person. `sub` er unik hos utstederen, blir aldri gjenbrukt og er på høyst 255 ASCII-tegn. En leverandør kan til og med gi hver applikasjon sin egen `sub`, en såkalt parvis identifikator, slik at applikasjonene ikke kan koble sammen det en person gjør, uten tillatelse.

Slik ser det ut i praksis

Ta en webapplikasjon som er registrert hos en OpenID Provider på `https://server.example.com` med klient-ID-en `s6BhdRkqt3`; verdiene nedenfor er hentet fra spesifikasjonens egne eksempler. Applikasjonen begynner med å hente leverandørens konfigurasjon fra den faste adressen under utstederen:

```
GET /.well-known/openid-configuration HTTP/1.1
Host: server.example.com
```

Svaret er et JSON-dokument. I forkortet form:

```
{
  "issuer": "https://server.example.com",
  "authorization_endpoint": "https://server.example.com/connect/authorize",
  "token_endpoint": "https://server.example.com/connect/token",
  "userinfo_endpoint": "https://server.example.com/connect/userinfo",
  "jwks_uri": "https://server.example.com/jwks.json",
  "scopes_supported": ["openid", "profile", "email", "address", "phone"],
  "id_token_signing_alg_values_supported": ["RS256", "ES256", "HS256"]
}
```

Discovery krever at `issuer` i dokumentet er identisk med issuer-URL-en som applikasjonen satte foran den faste stien, og applikasjonen stopper hvis de to er ulike. Deretter sender den personens nettleser til autorisasjonsendepunktet med

code-flyten, scopet `openid` og to tilfeldige verdier den har lagret sammen med nettleserøkten: `state` for omdirigeringen og `nonce` for ID-tokenet. En offentlig klient ville også sendt PKCE-parameterne, som RFC 9700 gjør obligatoriske for den; de er utelatt her for å holde linjene korte.

```
GET /connect/authorize?response_type=code&client_id=s6BhdRkqt3
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &scope=openid%20profile%20email
    &state=af0ifjsldkj&nonce=n-0S6_WzA2Mj HTTP/1.1
Host: server.example.com
```

Leverandøren logger personen inn og sender nettleseren tilbake med en kode, som applikasjonen løser inn ved token-endepunktet akkurat som i vanlig OAuth. Forskjellen ligger i svaret, som nå har et `id_token` ved siden av tilgangstokenet (forkortet her):

```
{
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "refresh_token": "8xLOxBtZp8",
  "expires_in": 3600,
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc..."
}
```

ID-tokenet består av tre base64url-deler, skilt med punktum. Den første dekodes til `{"alg":"RS256","kid":"1e9gdk7"}` : signeringsalgoritmen og ID-en til nøkkelen i leverandørens publiserte nøkkelsett. Den midterste delen inneholder selve claimene, og i spesifikasjonens eksempel ser de slik ut:

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969,
  "acr": "urn:mace:incommon:iap:silver"
}
```

Før applikasjonen stoler på ett eneste ord av dette, validerer den tokenet, og spesifikasjonen lister opp stegene. `iss` må stemme nøyaktig med leverandørens issuer. `aud` må inneholde applikasjonens egen klient-ID og ingen mottakere den ikke stoler på. Signaturen må kunne verifiseres med leverandørens nøkler fra `jwks_uri` og med algoritmen i headeren; kom tokenet, som her, direkte fra token-endepunktet over TLS, kan en kontroll av serverens TLS-sertifikat tre inn i stedet for signaturkontrollen. Klokken må være før `exp`, og `nonce` må være verdien applikasjonen selv sendte. Først da logger den personen inn, med `https://server.example.com` pluss `24400320` som nøkkel, og bruker tilgangstokenet til å hente e-post og profilopplysninger fra UserInfo-endepunktet hvis den trenger dem.

Én leverandør for mange systemer er like mye en organisatorisk endring som en teknisk. I ett av prosjektene her på nettstedet, tilgangsstyring på tvers av M365 og SSO, ble identitetene og tilgangene samlet i M365, 1Password, Red Hat SSO og OKTA SSO. Minste privilegium ble håndhevet, og onboarding og offboarding ble standardisert, slik at tilganger ble trukket tilbake raskt og konsekvent i stedet for å bli hengende igjen etter at noen hadde gått videre.

Hvor det går galt

E-postadressen brukes som kontonøkkel. Det er det opplagte valget og det gale. Spesifikasjonen sier at en utsteder over tid kan gi den samme e-postadressen til en annen person, og at `email`, `phone_number`, `preferred_username` og `name` ikke skal brukes som unike identifikatorer. En applikasjon som slår opp folk på e-post, kan gi én persons konto til den neste som får adressen. Lagre `iss` og `sub` sammen som nøkkel, behold e-posten som en vanlig egenskap, og kontroller at `email_verified` er sann før adressen brukes til noe som helst.

Et tilgangstoken blir tatt som bevis på hvem som logget inn. Noen applikasjoner tar imot et tilgangstoken, kaller et API med det og behandler svaret som en innlogging. Tokenet kan være utstedt til en helt annen applikasjon, og da er det nettopp angrepet der noen utgir seg for å være en annen, som RFC 6749 advarer mot. Logg folk inn på ID-tokenet, og først når dere har bekreftet at `aud` inneholder deres egen klient-ID.

ID-tokenet dekodes, men kontrolleres aldri. Å lese claimene ut av et token krever en base64url-dekoding og ingen nøkkel i det hele tatt, og det er nettopp det som gjør det fristende. Et token der signaturen, `iss`, `aud` eller `exp` ikke

blir kontrollert, kan være forfalsket, høre til en annen applikasjon eller ha utløpt. Bruk et bibliotek som blir vedlikeholdt, gi det utstederen og klient-ID-en, og la det kjøre alle stegene i spesifikasjonens liste, ikke bare de lettvinte.

Nonce mangler, er fast eller blir aldri sammenlignet. Uten den kan et ID-token som er snappet opp fra én innlogging, spilles av igjen i en annen. RFC 9700 legger til at nonce må høre til én bestemt innlogging og være bundet til nettleseren som startet den. Spesifikasjonen foreslår én måte å gjøre det på: Lagre en kryptografisk tilfeldig verdi i en HttpOnly-øktcookie, send hashen av den som `nonce`, og sammenlign de to når ID-tokenet kommer tilbake.

Signeringsnøklene caches for alltid. Leverandører bytter nøkler, og en applikasjon som holder fast ved gårsdagens nøkkelsett, avviser alle tokener som er signert med den nye. Spesifikasjonen forventer at leverandøren publiserer nøklene på `jwks_uri` og merker hvert token med en `kid`, og at applikasjonen henter nøkkelsettet på nytt når den møter en `kid` den ikke kjenner. Cache gjerne nøklene, men la en ukjent `kid` være et signal om å hente dem på nytt og ikke en grunn til å avvise innloggingen.

To tenanter på samme vert blir behandlet som én utsteder. En leverandør kan kjøre flere utstedere på samme vert, og stien er en del av utstederidentifikatoren: Subjektet `1234` hos `https://example.com` og subjektet `1234` hos `https://example.com/sales` er to forskjellige personer. Lagre hele issuer-strengen, sammenlign den tegn for tegn, og kutt den aldri ned til et vertsnavn.

Les mer

- OpenID Connect Core 1.0 — ID-tokenet, de tre flytene, standardclaimene og valideringsstegene, i utgaven med rettelsessett 2 fra desember 2023.
- OpenID Connect Discovery 1.0 — konfigurasjonsdokumentet på den faste adressen og regelen om at issuer i det må være identisk med issuer-URL-en det ble hentet under.
- RFC 6749: The OAuth 2.0 Authorization Framework — rammeverket OpenID Connect bygger på, med advarselen i avsnitt 10.16 mot å bruke tilgangstokener til å identifisere personer.
- RFC 7519: JSON Web Token (JWT) — tokenformatet som alle ID-tokener bruker.
- RFC 9700: Best Current Practice for OAuth 2.0 Security — gjeldende anbefalinger om PKCE, nonce og omdirigering, som gjelder for OpenID Connect like mye som for OAuth.

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 OpenID Connect?» 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/openid-connect/