Engineer

OpenID Connect

ett identitetslager ovanpå OAuth för inloggning

Vad det är

OpenID Connect är en standard som låter en applikation få veta vem en person är från en tjänst som just har loggat in personen. Specifikationen kallar sig själv ett enkelt identitetslager ovanpå OAuth 2.0, och det är precis så den är byggd: samma omdirigeringar, endpoints och tokens som i OAuth, plus en token till, en endpoint till och reglerna för hur de kontrolleras. I vardagligt tal förkortas den ofta till OIDC. Kärnspecifikationen blev slutgiltig i februari 2014. Den aktuella texten, där den andra omgången rättelser har arbetats in, är daterad den 15 december 2023 och förvaltas av en arbetsgrupp inom OpenID Foundation.

OAuth på egen hand svarar på frågan ”vad får den här applikationen göra?”. Applikationen får en åtkomsttoken som är avsedd för ett API, och det är inte meningen att applikationen själv ska läsa den. OpenID Connect ställer dessutom frågan ”vem loggade in?” och besvarar den med en ID-token: en JSON Web Token, adresserad till applikationen själv, som talar om vem personen är, vem som går i god för det, när token utfärdades och vilken applikation den är avsedd för. ID-tokens måste vara signerade. Specifikationen tillåter endast `alg: none` om svarstypen inte returnerar någon ID-token från auktoriseringsendpointen och klienten uttryckligen begärde `none` vid registreringen.

För den här uppgiften ger specifikationen OAuth-rollerna nya namn. En OpenID Provider (OP) är en auktoriseringsserver som autentiserar personer och utfärdar ID-tokens. En Relying Party (RP) är OAuth-klienten som behöver veta vem personen är, och personen själv kallas End-User. En begäran blir en OpenID Connect-begäran genom att `openid` finns med i scopet, och det värdet är obligatoriskt: saknas det är beteendet enligt specifikationens egna ord helt ospecificerat.

Andra scope-värden ber om profiluppgifter. `profile` täcker namn, smeknamn, bild, språkinställning och liknande claims, `email` ger `email` och `email_verified`, och `address` och `phone` gör det namnen säger. Applikationen kan hämta de här claims från leverantörens UserInfo-endpoint, ett helt vanligt OAuth-skyddat API som den anropar med åtkomsttoken.

Det finns tre sätt att köra det. Authorization code-flödet skickar bara en kortlivad kod genom webbläsaren och hämtar själva tokens med en HTTPS-begäran till token-endpointen. Implicit-flödet levererar tokens rakt i omdirigeringen, och hybridflödet blandar de två. En kompletterande specifikation, OpenID Connect Discovery 1.0, låter applikationen slå upp alla endpoints och signeringsnycklar utifrån en enda adress.

Använd begreppet i ett beslut

Använd ”Vad är OpenID Connect?” 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

Skälet att införa det är single sign-on. Med en identitetsleverantör framför många applikationer har människor en uppsättning inloggningsuppgifter, applikationerna lagrar inga av dem, och reglerna för hur man loggar in, från lösenordskrav till en extra faktor, bestäms på ett ställe, hos leverantören. Spärras ett konto hos leverantören kan det inte längre starta en ny session i någon av applikationerna som förlitar sig på den, och därmed blir offboarding ett enda steg i stället för en checklista.

Standarden täpper också till ett hål som team faller i när de bygger inloggning på ren OAuth. RFC 6749 varnar för att en publik klient i implicit-flödet inte kan se vilken klient en åtkomsttoken utfärdades till. En klient som utgår från att bara resursägaren kan ge den en giltig token går därför att lura: en angripare med en token som utfärdats till en annan applikation lämnar fram den och tas för personen bakom. En ID-token anger i sitt `aud`-claim vilken applikation den är till för, och applikationen kontrollerar det.

Eftersom det är en standard går arbetet att återanvända. Specifikationen utlovar profiluppgifter på ett interoperabelt sätt, så koden som loggar in människor via en leverantör kan logga in dem via en annan också. Med Discovery lär sig applikationen leverantörens endpoints och signeringsnycklar utifrån issuer-URL:en i stället för från inställningar som någon har skrivit av för hand.

Till sist ger OpenID Connect varje person en stabil identifierare. Specifikationen säger uttryckligen att paret `iss` (utfärdaren) och `sub` (subjektidentifieraren) är de enda claims som en applikation kan lita på som stabil identifierare för en person. Värdet i `sub` är unikt hos utfärdaren, återanvänds aldrig och är högst 255 ASCII-tecken långt. En leverantör kan till och med ge varje applikation ett eget `sub`, en så kallad parvis identifierare, så att applikationerna inte kan koppla ihop det en person gör utan tillstånd.

Så ser det ut i praktiken

Ta en webbapplikation som är registrerad hos en OpenID Provider på `https://server.example.com` med klient-ID:t `s6BhdRkqt3`; värdena nedan är hämtade från specifikationens egna exempel. Applikationen börjar med att hämta leverantörens konfiguration från den fasta adressen under utfärdaren:

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

Svaret är ett JSON-dokument. I förkortad 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 kräver att `issuer` i dokumentet är identiskt med den issuer-URL som applikationen satte framför den fasta sökvägen, och applikationen avbryter om de skiljer sig åt. Därefter skickar den personens webbläsare till

auktoriseringsendpointen med code-flödet, scopet `openid` och två slumpmässiga värden som den har sparat tillsammans med webbläsarsessionen: `state` för omdirigeringen och `nonce` för ID-token. En publik klient skulle dessutom skicka PKCE-parametrarna, som RFC 9700 gör obligatoriska för den; de är utelämnade här för att hålla raderna korta.

```
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
```

Leverantören loggar in personen och skickar tillbaka webbläsaren med en kod, som applikationen löser in vid token-endpointen precis som i vanlig OAuth. Skillnaden ligger i svaret, som nu har en `id_token` bredvid åtkomsttoken (förkortat här):

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

ID-token består av tre base64url-delar åtskilda av punkter. Den första avkodas till `{"alg":"RS256","kid":"1e9gdk7"}` : signeringsalgoritmen och ID:t för nyckeln i leverantörens publicerade nyckeluppsättning. Den mittersta delen innehåller själva claims, och i specifikationens exempel ser de ut så här:

```
{
  "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"
}
```

Innan applikationen litar på ett enda ord av detta validerar den token, och specifikationen räknar upp stegen. `iss` måste stämma exakt med leverantörens utfärdare. `aud` måste innehålla applikationens eget klient-ID och inga mottagare som den inte litar på. Signaturen måste gå att verifiera med leverantörens nycklar från `jwks_uri` och med algoritmen i headern; kom token, som här, direkt från token-endpointen över TLS kan en kontroll av serverns TLS-certifikat ersätta signaturkontrollen. Klockan måste vara före `exp`, och `nonce` måste vara det värde som applikationen själv skickade. Först då loggar den in personen, med `https://server.example.com` plus `24400320` som nyckel, och använder åtkomsttoken för att hämta e-post och profiluppgifter från UserInfo-endpointen om den behöver dem.

En leverantör för många system är lika mycket en organisatorisk förändring som en teknisk. I ett av projekten här på webbplatsen, behörighetshantering i M365 och SSO, samlades identiteterna och åtkomsten i M365, 1Password, Red Hat SSO och OKTA SSO. Principen om minsta behörighet upprätthölls, och onboarding och offboarding standardiserades, så att åtkomst återkallades snabbt och konsekvent i stället för att ligga kvar efter att någon hade gått vidare.

Var det går fel

E-postadressen används som kontonyckel. Det är det självklara valet och det felaktiga. Specifikationen säger att en utfärdare med tiden kan ge samma e-postadress till en annan person, och att `email`, `phone_number`, `preferred_username` och `name` inte får användas som unika identifierare. En applikation som slår upp människor på e-post kan ge en persons konto till nästa person som får adressen. Spara `iss` och `sub` tillsammans som nyckel, behåll e-posten som ett vanligt attribut, och kontrollera att `email_verified` är sant innan adressen används till något alls.

En åtkomsttoken tas som bevis på vem som loggade in. Vissa applikationer tar emot en åtkomsttoken, anropar ett API med den och behandlar svaret som en inloggning. Token kan ha utfärdats till en helt annan applikation, och då är det just det angrepp där någon utger sig för att vara någon annan som RFC 6749 varnar för. Logga in människor på ID-token, och först när ni har bekräftat att `aud` innehåller ert eget klient-ID.

ID-token avkodas men kontrolleras aldrig. Att läsa ut claims ur en token kräver en base64url-avkodning och ingen nyckel alls, och det är just det som gör det frestande. En token där signaturen, `iss`, `aud` eller `exp` inte kontrolleras kan

vara förfalskad, tillhöra en annan applikation eller ha gått ut. Använd ett bibliotek som underhålls, ge det utfärdaren och klient-ID:t, och låt det köra alla steg i specifikationens lista, inte bara de bekväma.

Nonce saknas, är fast eller jämförs aldrig. Utan den kan en ID-token som snappats upp från en inloggning spelas upp igen i en annan. RFC 9700 tillägger att nonce måste höra till en bestämd inloggning och vara bunden till webbläsaren som startade den. Specifikationen föreslår ett sätt att göra det: spara ett kryptografiskt slumpmässigt värde i en HttpOnly-sessionscookie, skicka dess hash som `nonce` och jämför de två när ID-token kommer tillbaka.

Signeringsnycklarna cachas för evigt. Leverantörer byter nycklar, och en applikation som håller fast vid gårdagens nyckeluppsättning avvisar varje token som är signerad med den nya. Specifikationen förväntar sig att leverantören publicerar nycklarna på `jwks_uri` och märker varje token med ett `kid`, och att applikationen hämtar nyckeluppsättningen på nytt när den stöter på ett `kid` som den inte känner igen. Cacha gärna nycklarna, men låt ett okänt `kid` vara en signal att hämta dem på nytt och inte ett skäl att avvisa inloggningen.

Två tenants på samma värd behandlas som en utfärdare. En leverantör kan köra flera utfärdare på samma värd, och sökvägen är en del av utfärdaridentifieraren: subjektet `1234` hos `https://example.com` och subjektet `1234` hos `https://example.com/sales` är två olika personer. Spara hela issuer-strängen, jämför den tecken för tecken, och korta aldrig ned den till ett värdnamn.

Läs mer

- OpenID Connect Core 1.0 — ID-token, de tre flödena, standardclaims och valideringsstegen, i utgåvan med rättelseomgång 2 från december 2023.
- OpenID Connect Discovery 1.0 — konfigurationsdokumentet på den fasta adressen och regeln att dess issuer måste vara identisk med den issuer-URL det hämtades under.
- RFC 6749: The OAuth 2.0 Authorization Framework — ramverket som OpenID Connect bygger på, med varningen i avsnitt 10.16 mot att använda åtkomsttokens för att identifiera personer.
- RFC 7519: JSON Web Token (JWT) — tokenformatet som alla ID-tokens använder.

- RFC 9700: Best Current Practice for OAuth 2.0 Security — aktuella rekommendationer om PKCE, nonce och omdirigering, som gäller för OpenID Connect lika mycket som för OAuth.

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 OpenID Connect?” 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/openid-connect/