Engineer

OpenID Connect

OAuthの上にログイン用のID層を設ける標準

どのようなものか

OpenID Connectは、その人をログインさせたばかりのサービスから、アプリケーションがその人が誰なのかを知るための標準的な方法です。仕様は、OAuth 2.0の上の単純なID層と呼びます。実際、OAuthと同じリダイレクト、エンドポイント、トークンに、もう一つのトークンとエンドポイント、それらを検査する規則を足します。通常OIDCと略します。中核仕様は2014年2月に最終版となり、二度目の訂正を取り込んだ現在の本文は、2023年12月15日付です。OpenID Foundationの作業部会が維持しています。

OAuth単体が答えるのは、「このアプリケーションに、何を許すか」という問いです。API向けのアクセストークンを渡し、アプリケーション自身は、それを読む想定ではありません。OpenID Connectは、「誰がログインしたか」を加え、アプリケーション自身を受信先とするJSON Web Token、IDトークンで答えます。誰なのか、誰がその身元を保証するのか、いつ発行し、どのアプリケーションに向けたかを示します。IDトークンには署名が必須ですが、仕様は `alg: none` を例外として許します。ただし、認可エンドポイントからIDトークンを返さない応答型であり、かつクライアントが登録時に明示的に `none` を求めた場合に限ります。

仕様は、この用途に合わせてOAuthの役割を呼び替えます。OpenID Provider（OP）は、本人を認証し、IDトークンを出すOAuth認可サーバーです。Relying Party（RP）は、その人が誰かを知る必要のあるOAuthクライアントです。本人をEnd-User、エンドユーザーと呼びます。スコープに `openid` を入れると、OpenID Connectのリクエストになり、その値は必須です。「openidスコープ値がなければ、動作は完全に未規定である」と定めます。

さらに、プロフィール情報を求めるスコープ値があります。 `profile` は、名前、ニックネーム、画像、ロケールなどのクレームを求め、 `email` は、 `email` と `email_verified` を追加します。 `address` と `phone` は、名前どおりの情報です。ア

プリケーションは、プロバイダーのUserInfoエンドポイントから、これらを取得できます。アクセストークンで呼ぶ、通常のOAuth保護APIです。

動かし方は三つあります。認可コードフローは、短い有効期間のコードだけをブラウザーで運び、トークンエンドポイントへのHTTPSリクエストでトークンを取得します。インプリシットフローは、トークンを直接リダイレクトで返し、ハイブリッドフローは両方を組み合わせます。付属仕様のOpenID Connect Discovery 1.0を使うと、一つのアドレスから、すべてのエンドポイントと署名鍵を調べられます。

概念を判断に生かす

「OpenID Connectとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

採用する理由は、シングルサインオンです。多数のアプリケーションの前に一つのIDプロバイダーを置くと、本人が持つ資格情報は一組で、アプリケーションは保存しません。パスワードの規則から第二の認証要素まで、ログイン方法の方針を、プロバイダーで一度だけ定めます。プロバイダー側でアカウントを無効にすると、それに依存するどのアプリケーションでも、新しいセッションを始められません。そのため、利用終了時の処理は、作業の一覧でなく一段階で済みます。

素のOAuthでログインを実装するときに、陥りがちな穴も塞ぎます。RFC 6749は、インプリシットフローでは、パブリッククライアントが、アクセストークンの発行先クライアントを判別できないと警告します。有効なトークンを渡せるのはリソース所有者だけだと考えると、なりすましが可能になります。別のアプリケーション向けのトークンを持つ攻撃者が、それを提示して、そのトークンが表す本人だと認められてしまいます。IDトークンは、`aud` クレームで発行先のアプリケーションを示し、アプリケーションが検査します。

標準なので、実装を別の環境にも移せます。仕様は、相互運用できる形でプロフィール情報を提供することを約束します。一つのプロバイダーでログインするコードを、別のプロバイダーにも使えます。Discoveryがあれば、手でコピーした設定でなく、発行者URLから、エンドポイントと署名鍵を取得できます。

最後に、OpenID Connectは、各人に安定した識別子を与えます。仕様は、`iss`、つまりトークンの発行者と、`sub`、つまり対象主体の識別子の組だけが、本人の安定した識別子として頼れるクレームの組だと明示します。`sub`は、その発行者の中で一意で、再割り当てされず、最大255文字のASCII文字列です。プロバイダーは、アプリケーションごとに異なる`sub`を出す、ペアワイズ識別子も使えます。許可なく、その人の活動をアプリケーション間で関連付けられないようにするためです。

実際の活用例

OpenID Providerの`https://server.example.com`に、クライアントID`s6BhdRkqt3`で登録したウェブアプリケーションを考えます。以下の値は、仕様自身の例から取っています。まず、発行者URL配下の所定のアドレスから、プロバイダーの設定を読みます。

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

応答はJSON文書です。短くしたものは、次のとおりです。

```
{
  "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は、その文書の`issuer`が、アプリケーションが所定のパスの前に置いた発行者URLと、完全に一致することを求めます。違えば、処理を止めます。続いて、認可コードフローと`openid`スコープ、それにブラウザーセッションに結び付けて保存した二つの乱数値で、本人のブラウザーを認可エンドポイントへ送ります。リダイレクト用の`state`と、IDトークン用の`nonce`です。パブリッククライアントでは、RFC 9700が必須とするPKCEパラメーターも付けますが、ここでは行を短くするため省きます。

```
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
```

プロバイダーは、本人をログインさせ、コードを付けてブラウザーを戻します。アプリケーションは、通常のOAuthと同じように、トークンエンドポイントでコードをトークンに引き換えます。違いは応答にあり、アクセストークンに加え、`id_token`を含みます。ここでは短くしています。

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

IDトークンは、ドットで区切った三つのbase64url部分からなります。最初をデコードすると、`{"alg":"RS256","kid":"1e9gdk7"}`となり、署名アルゴリズムと、プロバイダーが公開する鍵集合内の鍵IDを示します。中央の部分にクレームがあり、仕様の例は次のとおりです。

```
{
  "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"
}
```

内容を一言でも信じる前に、アプリケーションはトークンを検証します。仕様は手順を列挙しています。`iss`は、プロバイダーの発行者と完全一致しなければなりません。`aud`は、自分のクライアントIDを含み、信頼しない受信対象を含んではいけません。署名は、ヘッダーのアルゴリズムを使い、`jwks_uri`から得たプロバイダーの鍵で検証できなければなりません。ただし、この例のように、TLSでトークンエンドポイントから直接受け取った場合は、サーバーのTLS証明

書の検査を、署名検証の代わりに使えます。現在時刻は `exp` より前で、 `nonce` は送った値と同じでなければなりません。そこで初めて、 `https://server.example.com` と `24400320` を組み合わせたキーで、本人をログインさせます。必要なら、アクセストークンでUserInfoエンドポイントからメールとプロフィールのクレームを取得します。

多数のシステムで一つのプロバイダーを使うのは、技術だけでなく、組織の変更でもあります。このサイトのM365とSSOにまたがるアクセス管理では、M365、1Password、Red Hat SSO、OKTA SSOのIDとアクセスを統合し、最小権限を強制しました。参加時と利用終了時の処理も標準化し、組織を離れた人のアクセスを速やかに取り消せるようにしました。

問題が起きるところ

メールアドレスを、アカウントのキーにします。 分かりやすい選択ですが、間違いです。仕様は、プロバイダーが時を経て同じメールアドレスを別人に再利用する可能性があり、 `email` 、 `phone_number` 、 `preferred_username` 、 `name` を、一意の識別子にしてはならないとします。メールで本人を探すアプリケーションは、そのアドレスの次の所有者に、以前の人のアカウントを渡しかねません。 `iss` と `sub` の組をキーにし、メールは属性として保存します。アドレスを何に使う場合も、先に `email_verified` がtrueか確かめます。

アクセストークンで、誰がログインしたかを証明しようとします。 アクセストークンを受け取り、それでAPIを呼んだ応答を、ログインと見なすアプリケーションがあります。全く別のアプリケーションに発行したトークンかもしれず、RFC 6749が警告するなりすましになります。IDトークンに基づいてログインさせ、必ず先に、その `aud` が自分のクライアントIDを指すか確かめます。

IDトークンをデコードしても、検証しません。 クレームを読むだけなら、base64urlでデコードするだけで、鍵は不要です。その手軽さが落とし穴です。署名、 `iss` 、 `aud` 、 `exp` を検査しないトークンは、偽造されていたり、別のアプリケーション向けだったり、期限切れだったりします。保守されているライブラリーに、発行者とクライアントIDを渡し、都合の良い検査だけを選ばず、仕様の一覧のすべてを実行させます。

nonceがないか、固定されているか、比較されません。 これがなければ、あるログインで取得されたIDトークンを、別のログインに再利用できてしまいます。RFC 9700は、nonceを一回のログイン専用にし、それを始めたブラウザーに結び

付けることも求めます。仕様が示す一例は、暗号学的な乱数値をHttpOnlyのセッションクッキーに保ち、そのハッシュを `nonce` として送り、IDトークンが戻ったら両者を照合する方法です。

署名鍵を、いつまでもキャッシュします。 プロバイダーは鍵を更新するため、昨日の鍵集合だけを持つアプリケーションは、新しい鍵で署名したすべてのトークンを拒否します。仕様は、プロバイダーが `jwks_uri` で鍵を公開し、各トークンに `kid` を付け、アプリケーションは、知らない `kid` に出会えば鍵集合を取得し直すことを想定します。鍵はキャッシュしても、未知の `kid` はログインを拒否する理由でなく、再取得の合図にします。

同じホストの二つのテナントを、一つの発行者として扱います。 プロバイダーは、一つのホストで複数の発行者を動かせます。パスも、発行者識別子の一部です。対象主体 `1234` が `https://example.com` にある場合と、対象主体 `1234` が `https://example.com/sales` にある場合は、別人です。発行者の文字列全体を保存し、完全一致で比べ、ホスト名だけに切り詰めてはいけません。

参考資料

- OpenID Connect Core 1.0 — 2023年12月の第2弾の訂正を取り込んだ版にある、IDトークン、三つのフロー、標準クレーム、検証手順です。
- OpenID Connect Discovery 1.0 — 所定の場所の設定文書と、取得元の発行者URLに文書の発行者が一致しなければならないという規則です。
- RFC 6749：OAuth 2.0認可フレームワーク — OpenID Connectの基になる枠組みで、第10.16節は、アクセストークンで本人を識別する危険を警告します。
- RFC 7519：JSON Web Token（JWT） — すべてのIDトークンが使う形式です。
- RFC 9700：OAuth 2.0の現在の最良セキュリティー実務 — OAuthと同じくOpenID Connectにも適用する、PKCE、nonce、リダイレクトの現在の指針です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「OpenID Connectとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/openid-connect/