記事
Toshihiko Minamoto · 2021年8月11日 17m read

Open Exchange

OAuth 認証と InterSystes IRIS: トラストプロトコルを使いこなす

不在時に、セキュリティとプライバシーを維持しながら、コンピューターを相互に信頼させるにはどうすればよいでしょうか？

「ドライマルティーニを」と彼は言った。「1 杯。深いシャンパングラスで。」
「承知いたしました。」
「気が変わった。 ゴードンを 3、ヴォッカを 1、キナリレを半量。
キンキンに冷えるまでよくシェイクしてから、大きめの薄いレモンピールを 1 つ加えてくれ。 わかったかい？」
「お承りいたしました。」 バーテンダーはその考えが気に入ったようだった。
イアン・フレミング著『カジノ・ロワイヤル』（1953 年）より

OAuth は、ユーザーログイン情報を伴うサービスを「運用中」のデータベースから、物理的にも地理的にも分離する上で役立ちます。 このように分離すると、ID
データの保護が強化され、必要であれば、諸国のデータ保護法の要件に準拠しやすくしてくれます。

OAuth を使用すると、ユーザーは、最小限の個人データをさまざまなサービスやアプリケーションに「公開」しながら、一度に複数のデバイスから安全に作業することができるようになります。 また、サービスのユーザーに関する「過剰な」データを操作しなくてよくなります（データはパーソナル化されていない形態で処理することができます）。

InterSystems IRIS を使用する場合、OAuth と OIDC サービスを自律的かつサードパーティのソフトウェア製品と連携してテストし、デプロイするための既成の完全なツールセットを利用できます。

OAuth 2.0 と Open ID Connect

OAuth と Open ID Connect（OIDC または
OpenID）は、アクセスと識別をデリゲートするためにオープンプロトコルを汎用的に組み合わせたもので、21 世紀現在、人気を得ているようです。
大規模な使用において、これより優れたオプションはまだ誰も思いついていません。 HTTP(S)
プロトコル上にとどまり、JWT（JSON Web Token）コンテナ
を使用するため、特にフロントエンドのエンジニアに人気があります。

OpenID は OAuth を使用して機能しています。実際、OpenID は OAuth のラッパーです。 OpenID を電子識別システムの認証と作成に使用するオープンスタンダードとして使用することは、開発者にとって目新しい事ではありません。 2019 年には、公開から 14 周年を迎えました（バージョン 3）。
Webとモバイル開発、そしてエンタープライズシステムで人気があります。

そのパートナーである OAuth オープンスタンダードはアクセスをデリゲートする役割を担っており、12 年目を迎えています。関連する RFC 5849 標準が登場してからは 9 年です。
この記事の目的により、プロトコルの最新バージョンである OAuth 2.0 と最新の RFC 6749
を使用することにしましょう。 （OAuth 2.0 は、その前身の OAuth 1.0 とは互換していません。）

厳密に言えば、OAuth はプロトコルではなく、ソフトウェアシステムにアクセス権制限アーキテクチャを実装する際に、ユーザー識別操作を分離して別のトラステッドサーバーに転送するための一連のルール（スキーム）です。

OAuth は特定のユーザーについて何も言及できないことに注意してください！ ユーザーが誰であるか、ユーザーがどこにいるのか、またユーザーが現在コンピューターを使用しているかどうかさえも、知ることはできません。ただし、OAuth を使用すれば、事前に発行されたアクセストークンを使用して、ユーザーが参加することなくシステムと対話することが可能であり、 これは重要なポイントです（詳細は、OAuth サイトにある「User Authentication with OAuth 2.0」をご覧ください）。

User-Managed Access（UMA）プロトコルも OAuth に基づくプロトコルです。 OAuth、OIDC、および UMA を合わせて使用することで、次のような分野で保護された ID
とアクセス管理（IdM、IAM）システムを実装することができます。

- 医療分野における患者の HEART（Health Relationship Trust）個人データプロファイルの使用。
- 製造会社および貿易会社向けの顧客 ID＆アクセス管理（CIAM）プラットフォーム。
- OAuth 2.0 Internet of Things (IoT) Client Credentials Grant による、IoT（モノのインターネット）システムにおけるスマートデバイス向けデジタル証明書のパーソナル化。

API エコノミーの新しいアクセス制御ベン図

何よりも、個人データをシステムのほかの部分と同じ場所に保存してはいけません。
認証と認可は物理的に分離する必要があります。 そして、ID と認証を各個人に与えることが理想と言えます。
自分で保管せずに、 所有者のデバイスを信頼するのです。

信頼と認証

ユーザーの個人データを自分のアプリや作業データベースと組み合わさったストレージ場所に保存するのはベストプラクティスではありません。
言い換えれば、このサービスを提供できる信頼のある人を選ぶようにする必要があります。

このサービスは、次の項目で構成されます。

- ユーザー
- クライアントアプリ
- 識別サービス
- リソースサーバー

アクションは、ユーザーのコンピューターの Web ブラウザで実行されます。
ユーザーには識別サービスが備わったアカウントがあり、
クライアントアプリは、識別サービスと相互インターフェースとの契約に署名済みです。
リソースサーバーは、識別サービスを信頼して、識別できた人にアクセスキーを発行します。

ユーザーはクライアント Web アプリを実行して、リソースを要求します。
クライアントアプリは、アクセス権が必要なそのリソースへのキーを提示する必要があります。
ユーザーにキーがない場合、クライアントアプリはリソースサーバーへのキーを発行するために契約している識別サービスに接続します（ユーザーを識別サービスに転送します）。

識別サービスは、どのようなキーが必要かを問い合わせます。

ユーザーは、リソースにアクセスするためのパスワードを入力します。 この時点でユーザー認証が行われ、ユーザーの身元が確認されると、リソースへのキーが提供され（ユーザーをクライアントアプリに戻します）、ユーザーがリソースを利用できるようになります。

認可サービスの実装

InterSystems IRIS
プラットフォームでは、必要に応じてさまざまなプラットフォームからのサービスをアセンブルできます。
次はその例です。

1. デモクライアントが登録された OAuth サーバーを構成して起動します。
2. デモ OAuth クライアントを OAuth サーバーと Web リソースに関連付けて構成します。
3. OAuth を使用できるクライアントアプリを開発します。 Java、Python、C#、または Node JS を使用できます。 以下の方に、ObjectScript でのアプリケーションコードの例を示しています。

OAuth にはさまざまな設定があるため、チェックリストが役立ちます。 例を見ていきましょう。 IRIS 管理ポータルに移動し、［システム管理］>［セキュリティ］>［OAuth 2.0］>［サーバー］の順に選択します。

各項目には設定行の名前とコロン、そして必要であればその後に例または説明が含まれます。
別の方法として、Daniel Kutac の 3 部構成になっている「InterSystems IRIS Open Authorization Framework (OAuth 2.0)の実装 - パート1」、パート2、そしてパート3
に記載されているスクリーンショットのヒントを参考にしてください。

次のスクリーンショットはすべて、例として提示されています。
独自のアプリケーションを作成する際は、別のオプションを選択する必要があるでしょう。

［一般設定］タブで、次のように設定してください。

- 説明: 構成の説明を入力します。「認証サーバー」など。
- ジェネレーターのエンドポイント（以降「EPG」）のホスト名： サーバーの DNS 名。
- サポートされている許可の種類（少なくとも 1 つを選択）:
    - 認可コード
    - 暗黙
    - アカウントの詳細: リソース、所有者、パスワード
    - クライアントアカウントの詳細
- SSL/TLS 構成: oauthserver

［スコープ］タブで、次を設定します。

- サポートされているスコープを追加: この例では「scope1」です。

［間隔］タブで、次を設定します。

- アクセスキー間隔: 3600
- 認可コードの間隔: 60
- キー更新の間隔: 86400
- セッション中断間隔: 86400
- クライアントキー（クライアントシークレット）の有効期間: 0

［JWT 設定］タブで、次を設定します。

- 入力アルゴリズム: RS512
- キー管理アルゴリズム: RSA-OAEP
- コンテンツ暗号化アルゴリズム: A256CBC-HS512

［カスタマイズ］タブで、次を設定します。

- 識別クラス: %OAuth2.Server.Authenticate
- ユーザークラスの確認: %OAuth2.Server.Validate
- セッションサービスクラス: OAuth2.Server.Session
- キーの生成クラス: %OAuth2.Server.JWT
- カスタムネームスペース: %SYS
- カスタマイズロール（少なくとも 1 つ選択）: %DBIRISSYS および %Manager

では、変更内容を保存します。

次のステップでは、OAuth サーバーにクライアントを登録します。
［顧客の説明］ボタンをクリックして、［顧客の説明を作成］をクリックします。

［一般設定］タブで、次の情報を入力します。

- 名前: OAuthClient
- 説明: 簡単な説明を入力します。
- クライアントタイプ: 機密
- リダイレクト URL: oauthclient から識別した後に、アプリに戻るポイントのアドレス。
- サポートされている付与の種類：
    - 認可コード: はい
    - 暗黙
    - アカウントの詳細: リソース、所有者、パスワード
    - クライアントアカウントの詳細
    - JWT 認可
- サポートされているレスポンスタイプ: 次のすべてを選択してください。
    - コード
    - idtoken
    - idtoken キー
    - トークン
- 認可タイプ: シンプル

［クライアントアカウントの詳細］タブは自動的に入力されますが、クライアントの正しい情報であるかを確認してください。
［クライアント情報］ タブには次の項目があります。

- 認可画面:
    - クライアント名
    - ロゴの URL
    - クライアントのホームページ URL
    - ポリシーの URL
    - 利用規約の URL

では、［システム管理］>［セキュリティ］>［OAuth 2.0］>［クライアント］の順に移動して、OAuth サーバークライアントにバインディングを構成します。

サーバーの説明の作成:

- ジェネレーターのエンドポイント: 一般的なサーバーのパラメーターから取得されます（上記を参照）。
- SSL/TLS 構成: 事前構成済みのリストから選択します。
- 認可サーバー:
    - 認可エンドポイント: EPG + /authorize
    - キーエンドポイント: EPG + /token
    - ユーザーエンドポイント: EPG + /userinfo
    - キーのセルフテストエンドポイント: EPG + /revocation
    - キーの終了エンドポイント: EPG + /introspection
- JSON Web Token（JWT）設定:
    - 動的登録以外のほかのソース: URL から JWKS を選択します。
    - URL: EPG + /jwks

このリストから、たとえばサーバーが OAuth-client にユーザーに関するさまざまな情報を提供できることがわかります（scopessupported および claimssupported）。 また、アプリケーションを実装するときは、共有する準備ができているデータが何であるかをユーザーに尋ねても何の価値もありません。 以下の例では、scope1 の許可のみを要求します。

では、構成を保存しましょう。

SSL 構成に関するエラーがある場合は、［設定］>［システム管理］>［セキュリティ］>［SSL/TSL 構成］に移動して、構成を削除してください。

これで OAuth クライアントをセットアップする準備が整いました。
［システム管理］>［セキュリティ］>［OAuth 2.0］>［クライアント］>［クライアント構成］>［クライアント構成を作成］に移動します。
［一般］タブで、次を設定します。

- アプリケーション名: OAuthClient
- クライアント名: OAuthClient
- 説明: 説明を入力します。
- 有効: はい
- クライアントタイプ: 機密
- SSL/TCL 構成: oauthclient を選択します。
- クライアントリダイレクト URL: サーバーの DNS 名
- 必要な許可の種類:
    - 認可コード: はい
    - 暗黙
    - アカウントの詳細: リソース、所有者、パスワード
    - クライアントアカウントの詳細
    - JWT 認可
- 認可タイプ: シンプル

［クライアント情報］タブで、次を設定します。

- 認可画面:
    - ロゴの URL
    - クライアントのホームページ URL
    - ポリシーの URL
    - 利用規約の URL
- デフォルトのボリューム: サーバーに以前に指定したものが取得されます（scope1 など）。
- 連絡先メールアドレス: カンマ区切りでアドレスを入力します。

- デフォルトの最大経過時間（分）: 最大認可経過時間または省略できます。

［JWT 設定］タブで、次を設定します。

- JSON Web Token（JWT）設定
- X509 アカウントの詳細から JWT 設定を作成する
- IDToken アルゴリズム:
  - 署名: RS256
  - 暗号化: A256CBC
  - キー: RSA-OAEP
- Userinfo アルゴリズム
- アクセストークンアルゴリズム
- クエリアルゴリズム

［クライアントログイン情報］タブで、次を設定します。

- クライアント ID: クライアントがサーバーに登録された際に発行された ID（上記を参照）。
- 発効されたクライアント ID: 入力されません
- クライアントシークレット:
  クライアントがサーバーに登録された際に発行されたシークレット（上記を参照）。
- クライアントシークレットの有効期限: 入力されません
- クライアント登録 URI: 入力されません

構成を保存しましょう。

OAuth 認可を使用した Web アプリ

OAuth は、インタラクション参加体（サーバー、クライアント、Web アプリケーション、ユーザーのブラウザ、リソースサーバー）間の通信チャネルが何らかの形で保護されていることに依存しています。 この役割は SSL/TLS プロトコルが果たしているのがほとんどですが、 OAuth は、保護されていないチャネルでも機能します。 そのため、たとえばサーバー Keycloak はデフォルトで HTTP プロトコルを使用して、保護なしで実行します。 調整と調整時のデバッグが単純化されます。 サービスを実際に使用する際、OAuth のチャネル保護は、厳重な要件に含まれるべきであり、Keycloak ドキュメントに記述されている必要があります。 InterSystems IRIS の開発者は、OAuth に関するより厳密なアプローチに従っており、SSL/TSL の使用を要件としています。 単純化できる唯一の方法は、自己署名証明書を使用するか、組み込みの IRIS サービス PKI（［システム管理］>>［セキュリティ］>>［公開鍵システム］を利用することです。

ユーザーの認可の検証は、UAuth サーバーに登録されているアプリケーションの名前と OAuth クライアントスコープの 2 つのパラメータを明示的に示すことで行えます。

```
Parameter OAUTH2APPNAME = "OAuthClient";
set isAuthorized = ##class(%SYS.OAuth2.AccessToken).IsAuthorized(
..#OAUTH2APPNAME,
.sessionId,
"scope1",
.accessToken,
.idtoken,
.responseProperties,
.error)
```

認可がない場合に備え、ユーザー ID をリクエストして、アプリケーションを操作する許可を取得するためのリンクを準備しておきます。 ここでは、OAuth サーバーに登録されているアプリケーションの名前を指定して、OAuth

クライアントと要求されるボリューム（スコープ）を入力するだけでなく、ユーザーを返す Web アプリケーションのポイントへのバックリンクも指定する必要があります。

```
Parameter OAUTH2CLIENTREDIRECTURI = "https://52773b-76230063.labs.learning.intersyste
ms.com/oauthclient/"
set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
..#OAUTH2APPNAME,
"scope1",
..#OAUTH2CLIENTREDIRECTURI,
.properties,
.isAuthorized,
.sc)
```

IRIS を使用して、ユーザーを IRIS OAuth サーバーに登録しましょう。
たとえば、ユーザーに名前とパスワードを設定するだけで十分です。
受信した参照の下でユーザーを転送すると、サーバーはユーザーを識別する手続きを実行し、Web アプリケーションのアカウントデータによって、操作許可が照会されます。また、%SYS フィールドのグローバル OAuth2.Server.Session で自身に結果を保持します。

3. 認可されたユーザーのデータを示します。
   手続きが正常に完了したら、アクセストークンなどがあります。 それを取得しましょう。

   set valid = ##class(%SYS.OAuth2.Validation).ValidateJWT(.#OAUTH2APPNAME, accessToken, "scope1", .aud, .JWTJsonObject, .securityParameters, .sc)

以下に、完全に動作する OAuth の例のコードを示します。

```
Class OAuthClient.REST Extends %CSP.REST
{
Parameter OAUTH2APPNAME = "OAuthClient";
Parameter OAUTH2CLIENTREDIRECTURI = "https://52773b-76230063.labs.learning.intersyste
ms.com/oauthclient/";
// to keep sessionId
Parameter UseSession As Integer = 1;
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
    &lt;Routes>
        &lt;Route Method="GET" Url = "/" Call = "Do" />
    &lt;/Routes>
}
ClassMethod Do() As %Status
{
    // Check for accessToken
    set isAuthorized = ##class(%SYS.OAuth2.AccessToken).IsAuthorized(
        ..#OAUTH2APPNAME,
        .sessionId,
        "scope1",
        .accessToken,
        .idtoken,
        .responseProperties,
        .error)
    // to show accessToken
    if isAuthorized {
```

```
     set valid = ##class(%SYS.OAuth2.Validation).ValidateJWT(
         ..#OAUTH2APPNAME,
         accessToken,
         "scope1",
         .aud,
         .JWTJsonObject,
         .securityParameters,
         .sc
     )
     &html&lt; Hello!&lt;br> >
         w "You access token = ", JWTJsonObject.%ToJSON()
     &html&lt; &lt;/html> >
     quit $$$OK
    }
    // perform the process of user and client identification and get accessToken
    set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
        ..#OAUTH2APPNAME,
        "scope1",
        ..#OAUTH2CLIENTREDIRECTURI,
        .properties,
        .isAuthorized,
        .sc)
    if $$$ISERR(sc) {
         w "error handling here"
         quit $$$OK
    }
    // url magic correction: change slashes in the query parameter to its code
    set urlBase = $PIECE(url, "?")
    set urlQuery = $PIECE(url, "?", 2)
    set urlQuery = $REPLACE(urlQuery, "/", "%2F")
    set url = urlBase _ "?" _ urlQuery
    &html&lt;
     &lt;html>
         &lt;h1>Authorization in IRIS via OAuth2&lt;/h1>
         &lt;a href = "#(url)#">Authorization in &lt;b>IRIS&lt;/b>&lt;/a>
     &lt;/html>
    >
    quit $$$OK
}
}
```

コードの作業コピーは、InterSystems GitHub
リポジトリ（https://github.com/intersystems-community/iris-oauth-example）にもあります。

必要に応じて、OAuth サーバーと OAuth
クライアントに高度なデバッグメッセージモードを有効にしてください。これらは、%SYS エリアの ISCLOG
グローバルに記述されます。

```
set ^%ISCLOG = 5
set ^%ISCLOG("Category", "OAuth2") = 5
set ^%ISCLOG("Category", "OAuth2Server") = 5
```

詳細については、「IRIS、OAuth 2.0 と OpenID Connect の使用」ドキュメントをご覧ください。

まとめ

これまで見てきたように、すべての OAuth
機能には簡単にアクセスでき、完全に使用できる状態になっています。
必要に応じて、ハンドラークラスとユーザーインターフェースを独自のものに置き換えることができます。
OAuth
サーバーとクライアントの設定は、管理ポータルを使う代わりに、構成ファイルで構成することも可能です。

#JSON #OAuth2 #セキュリティ #チュートリアル #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/oauth-%E8%AA%8D%E8%A8%BC%E3%81%A8-intersystes-iris-%E3%83%88%E3%83%A9%E3%82%B9%E3%83%88%E3%83%97%E3%83%AD%E3%83%88%E3%82%B3%E3%83%AB%E3%82%92%E4%BD%BF%E3%81%84%E3%81%93%E3%81%AA%E3%81%99