OAuth2.0 PKCEを UEでまじめに扱う

@UE Tokyo .dev #4 (2025.03) 2025/03/14

by みずあめ

About Me

名前: 佐藤 良

年齢: 18歳

所属: 筑波大学情報学群情報科学類1年

ついったー: @mizuameisgod

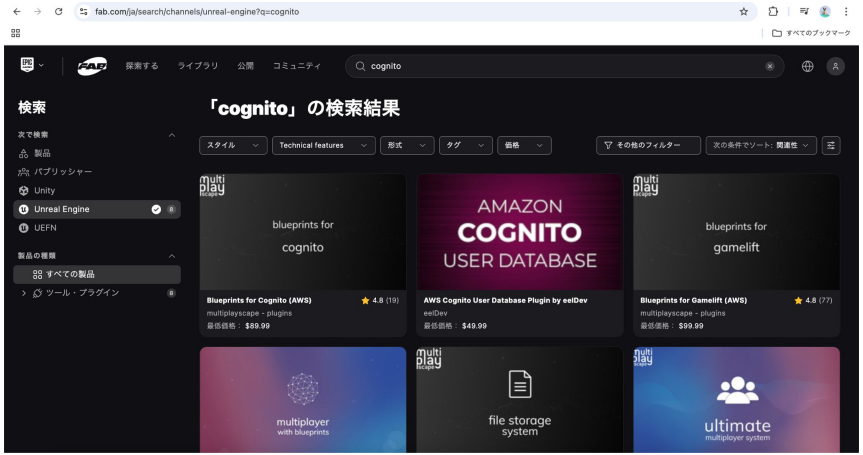
web: https://mizuame.works/

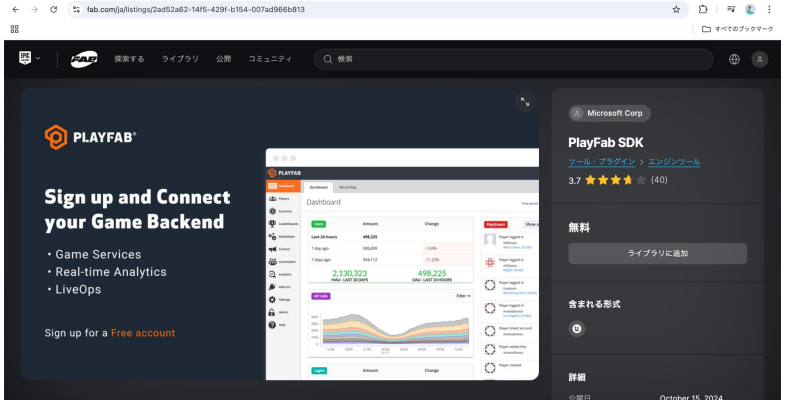
やってること: ゲームクライアント、インフラ、機械学習、言語モデル、セキュリティ、web、ネットワーク etc… 興味があることは割と何でもかじっています

アカウント認証したいですよね？

EOS、PlayFab、FireBase、Cognitoなど

fabを見ると、有料/無料問わずそれらに対応するプラグインが売られている





問題

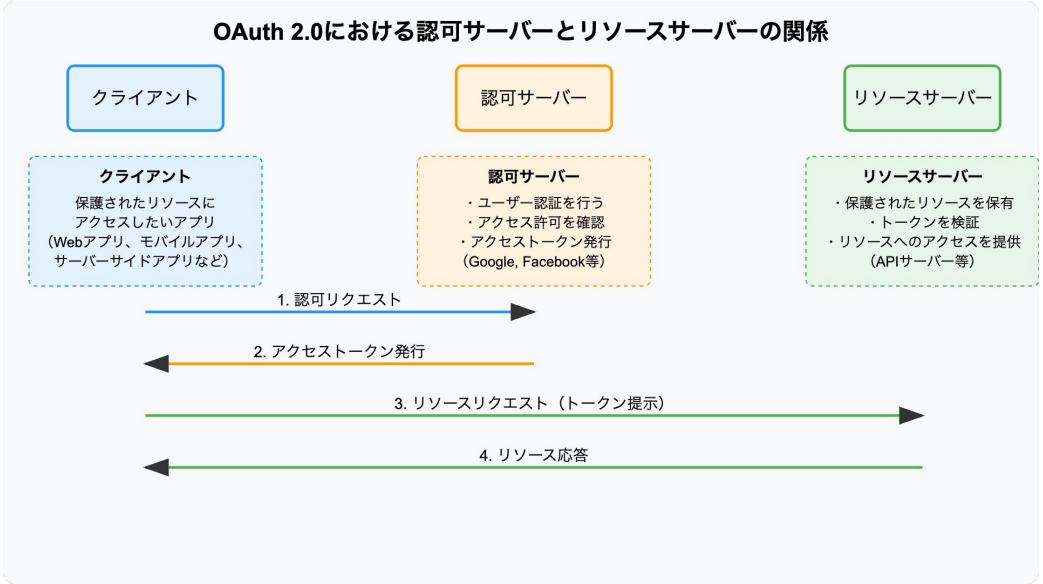
- プラグインが対応しなくなったら終焉
- ソーシャルログインめんどくさい
- C++でプラグイン作るにしてもSDKがアップデートされる度に対応しなければいけない可能性がある

実は: 別にゲーム内でログインさせる必要はない

→Webでログインさせ、何らかのトークンをアプリケーションに渡せれば良い

ここで登場するのが、OAuth2.0とPKCEと言う仕組みです。

ざっくりとOAuth

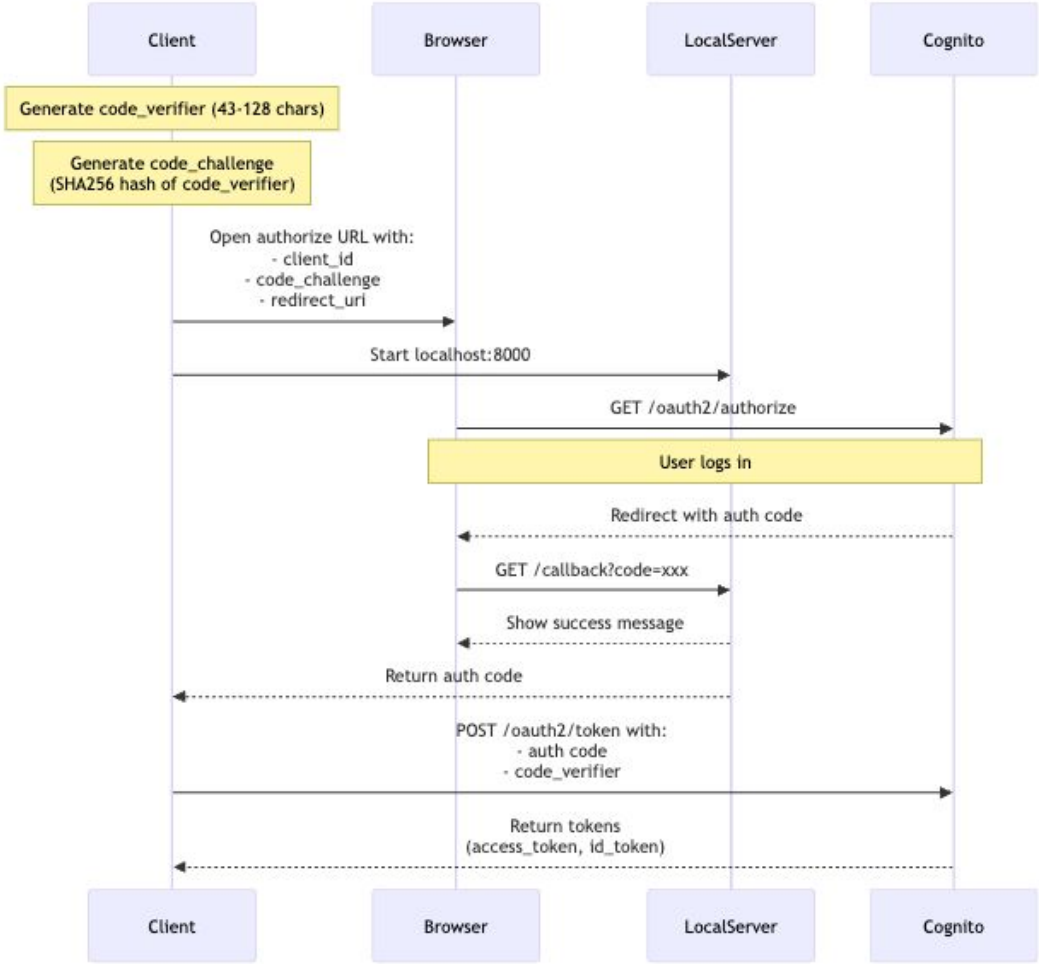


認可サーバーとリソースサーバに別れている

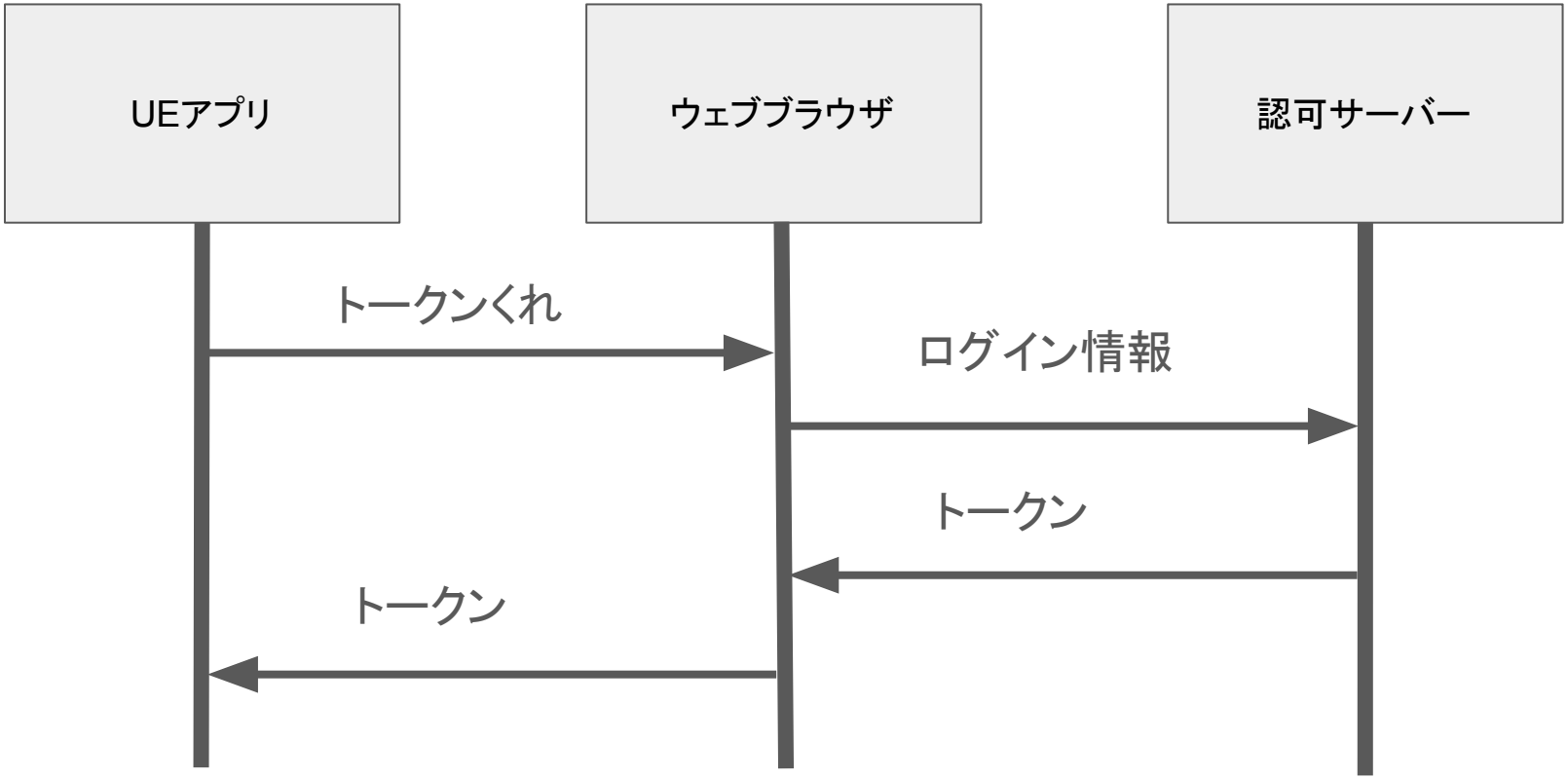
PKCEとは

EN: Proof Key for Code Exchange by OAuth Public Client

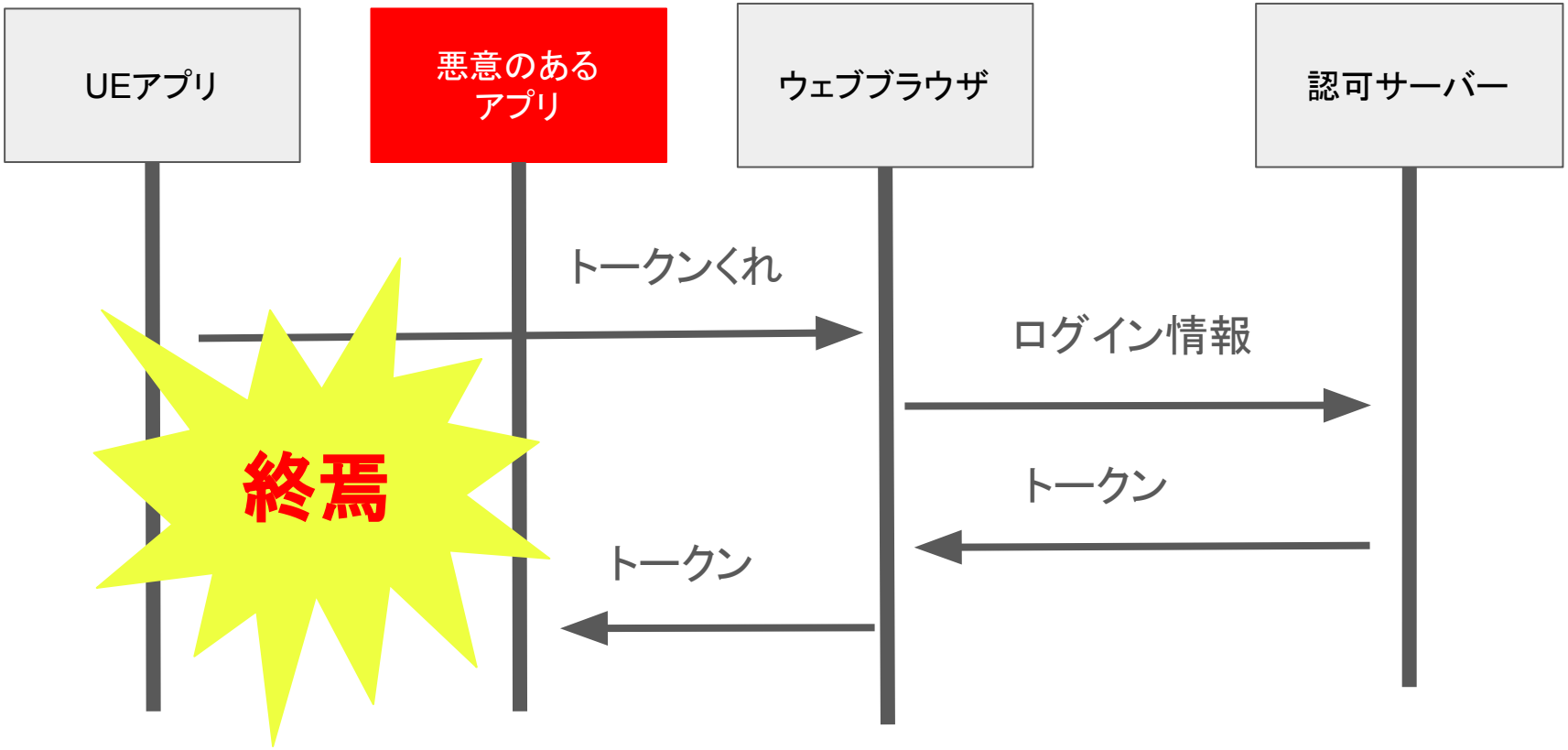
JA: OAuth公開クライアントによるコード交換のための検証キー


Client
Browser
LocalServer
Cognito
Generate code_verifier (43-128 chars)
Generate code_challenge
(SHA256 hash of code_verifier)
Open authorize URL with:
- client_id
- code_challenge
- redirect_uri
Start localhost:8000
GET /oauth2/authorize
User logs in
Redirect with auth code
GET /callback?code=xxx
Show success message
Return auth code
POST /oauth2/token with:
- auth code
- code_verifier
Return tokens
(access_token, id_token)
Client
Browser
LocalServer
Cognito

何を言っているんだ...?



脆弱性



思想

- 問題なのは、認証をしたいアプリケーションが、別のアプリケーションに認証を横取りされること。
- 横取りされるタイミングはWEBブラウザからのリダイレクト時
- では、真のアプリケーションしか知り得ない情報を、もう一回アプリケーション側で認証サーバーに送り、整合性を確認できれば良さそう

→一方向性関数を用いると良さそう

一方向性関数とは？

ただのハッシュ関数(今回はsha256)

- 出力された値を元に元の値を出すのが困難(原像計算困難性)
- 異なる2つの入力値から同じハッシュ値を生成することが困難(衝突発見困難性)
- ハッシュ値を元に、元となった値とは別の、ハッシュ値が一致する値を見つけるのが困難(第二原像計算困難性)

PKCEの実装

つまり、クライアント(UEアプリケーション)は

最初にブラウザで渡すハッシュ値(code_challenge)と

元の値(code_verifier)を生成する必要がある。

実は: 安全な乱数を生成するのはかなり難しいらしい

安全な乱数を作ろう(RFC 7636)

- code_verifier =予約されていない文字を使用した高エントロピー暗号ランダム文字列[A-Z] / [a-z] / [0-9] / ”-” / ”。” / [RFC3986]のセクション2.3の/ ”_” / ”〜“。最小長は43文字、最大長は128文字です。
- コードベリファイアには、値を推測するのを非実用的にするのに十分なエントロピーが必要です（SHOULD）

→　　void MutateSeed(){
_Seed = (_Seed * 196314165U) + 907633515U;
}

https://github.com/EpicGames/UnrealEngine/blob/0c544b150542b59fc87bdcf64caae09f4e3bc11c/Engine/Source/Runtime/Solaris/uLangCore/Public/uLang/Common/Misc/RandomStream.h

→BPでランダム系の処理を呼ぶと、多分、謎の線形合同法による乱数生成になってるし、そもそもシード値をどうやって決定するのかが問題になる

具体的な攻撃シナリオ

例えば、BPのRndom Int In Rangeを使い、シード値に現在時刻を使い、code_challengeとcode_verifierを生成したとする。

この値を元にLaunchURLをしてしまった場合、悪意のあるアプリケーションがLaunchURLを実行した時刻を元にある程度の元になったであろう大まかなシード値を計算し、これをRndom Int In Rangeに入れ再計算し、code_challengeに対してブルートフォースしてしまえば容易に元のcode_verifierを割ることができる。

これとauto codeジャックを組み合わせれば、トークンを盗むことが可能

疑問：真の乱数生成はどうやってるの？Linux編

本質的に、どんなに優秀なアルゴリズムであろうが、そもそものシード値の生成に問題を抱える。つまり、真のランダム性（エントロピー）をどこから得るかと言う話に帰着する。

Linuxの/dev/randomでは真の乱数を得ることができる。

CPUの熱、ノイズ、時間、ハードウェア、マウス/キーボードの入力

これらを組み合わせ、エントロピープールを作成している。

また一回使用されたものは使用できないため、エントロピープールには消費という概念がある。よって連続して真の乱数を生成するのができない時がある。

が、現実問題urandomで問題ないらしい(urandomでは使い回しがされる)

サーバーで生成しようか

cloudflare workersでjsの crypto.getRandomValues()を実行

→Crypto::getRandomValues()

→crypto::RandBytes()→base::RandBytes()

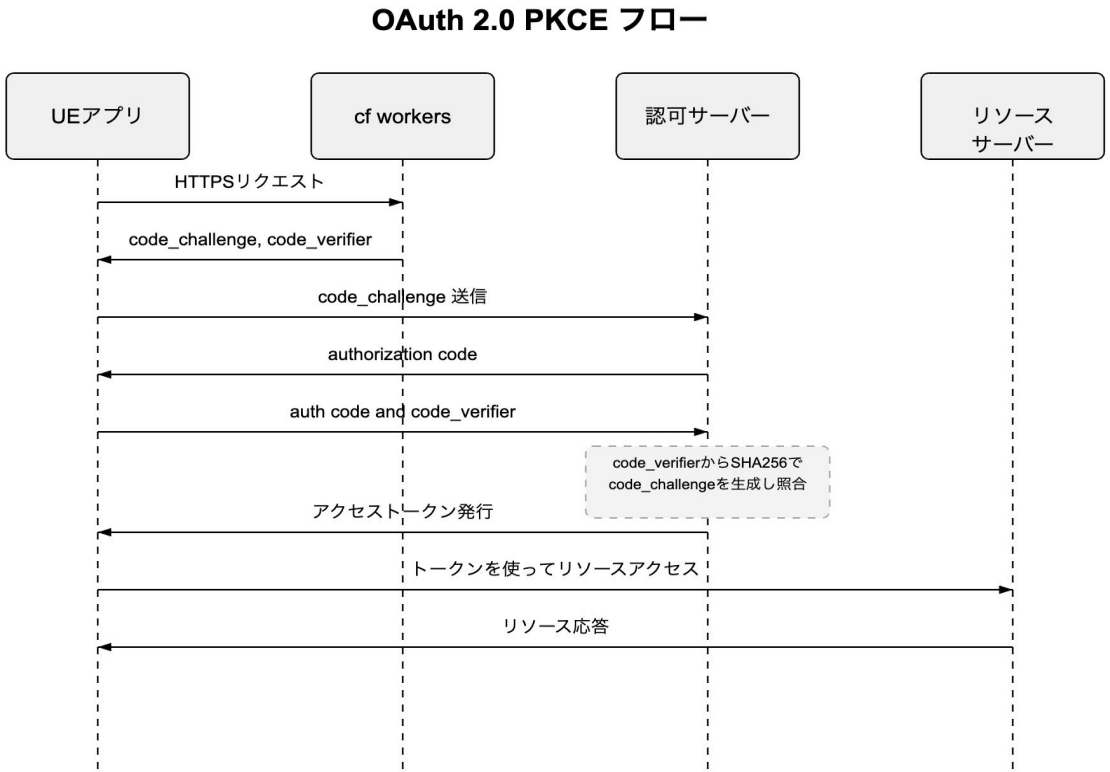
→RandBytesInternal()

→getrandom() {Linuxのシステムコール、urandomを使用}

httpsでgetすると取れるようにした。

{"code_verifier":"yCqQ3BxSzAvy35f0bfaJh1F13WZb06Z2Owg8qbzHu10","code_challenge":"AL7173Mu_Fcoe9lPhWEFyhaBhXfd4qAUvwrZlSkldfo","timestamp":1741678808328}

流れをざっと


OAuth 2.0 PKCE フロー
UEアプリ
cf workers
認可サーバー
リソース
サーバー
HTTPSリクエスト
code_challenge, code_verifier
code_challenge 送信
authorization code
auth code and code_verifier
code_verifierからSHA256で
code_challengeを生成し照合
アクセストークン発行
トークンを使ってリソースアクセス
リソース応答

具体的な実装

ブラウザを開くにはデフォルトであるLaunchURLを使えばいいが、

コードの受け取りをどうするのか問題になる。

今回はsimple http serverというプラグインを使用した。

https://www.fab.com/ja/listings/d95fcaab-6699-449a-a742-05564bc9959c

https://github.com/Kaboms/UE-Simple-Http-Server?tab=readme-ov-file

URL構築

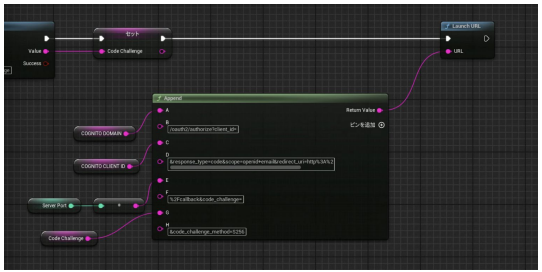
今回はIdPにAWS Cognitoを使用したため、以下のURLを構築する

https://**{cognito_domain}**/oauth2/authorize?client_id=**{cognito_client_id}**&response_type=code&scope=openid+email&redirect_uri=http://localhost:**{server_port}**/callback&code_challenge=**{code_challenge}**&code_challenge_method=S256

ドメイン、IDはAWS Consoleで取得できるもの

code_challengeはworkersからhttpsで取得したもの

server_portは自分で立てたsimple http serverのポート番号



リダイレクト先サーバー構築

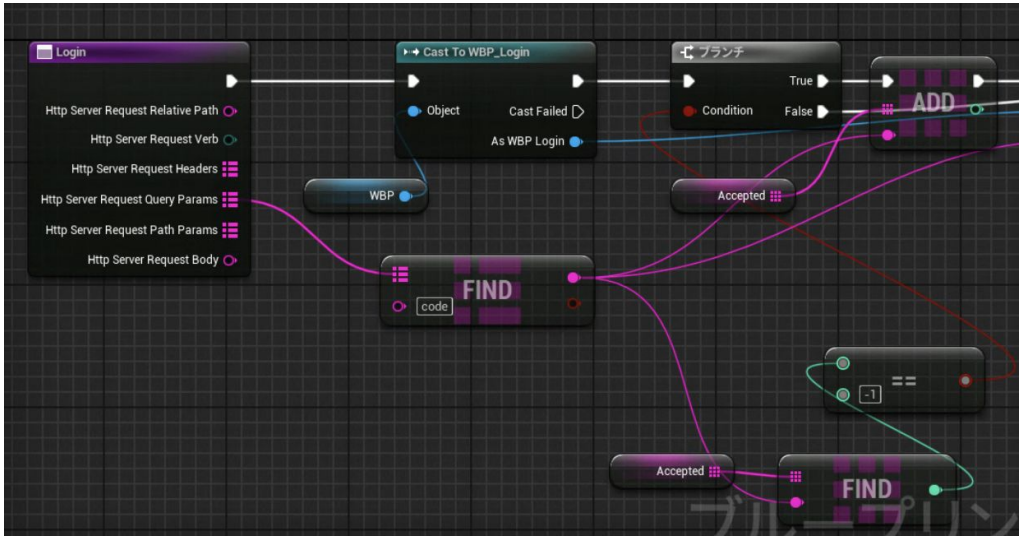
actorからsimple http server classを継承したBPを作り、カスタムイベントをバインドします

クエリパラメータからcodeをget

コールバックはgetリクエストとして来ます。

その際、?code=

URLクエリパラメーターでauth codeがくるので、これを獲得します。



exchange code

httpsリクエストを構築します。対象エンドポイントは

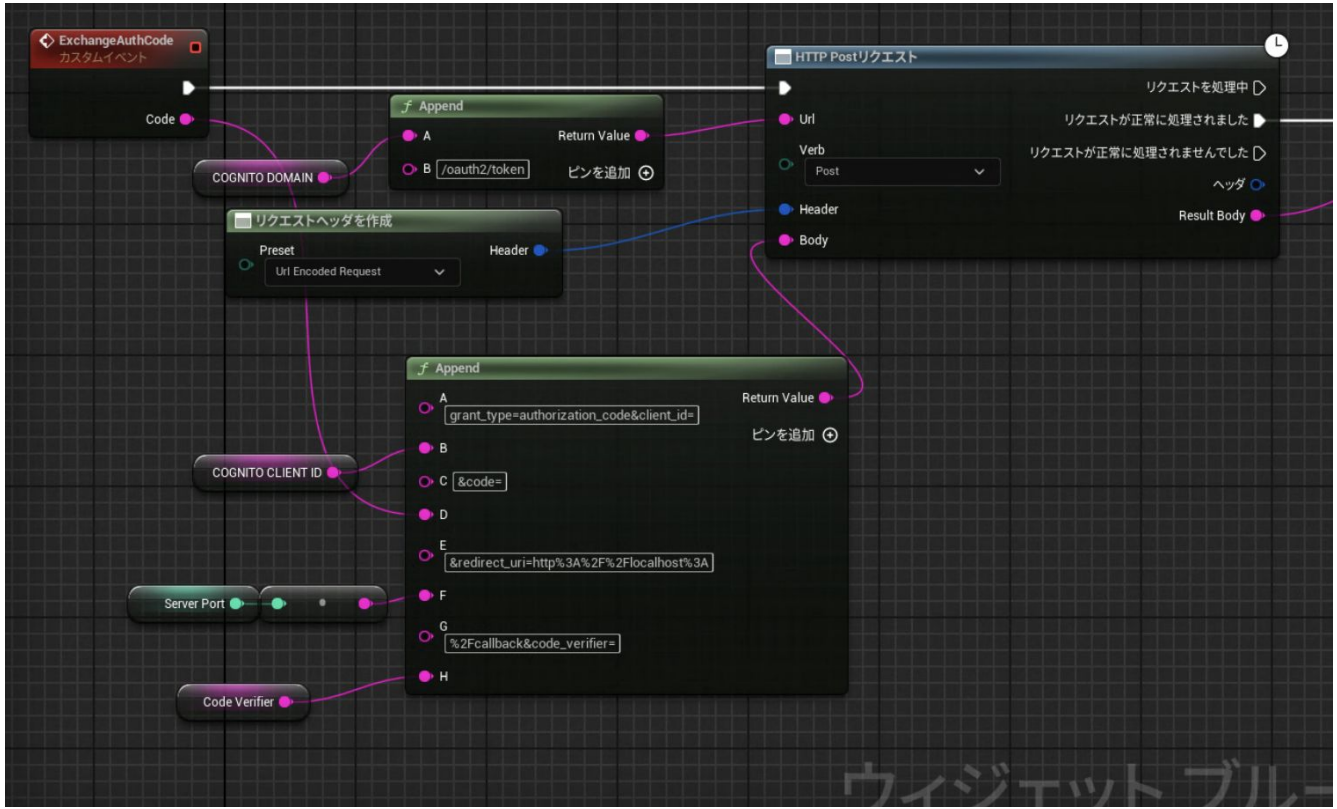
https://**{cognito_domain}**/oauth2/token

bodyは

grant_type=authorization_code&client_id=**{cognito_client_id}**&code=**{returned_code}**&redirect_uri=http://localhost:**{server_port}**/Fcallback&code_verifier=**{code_verifier}**

code_verifierは最初にworkersからhttpリクエストで受け取ったもの

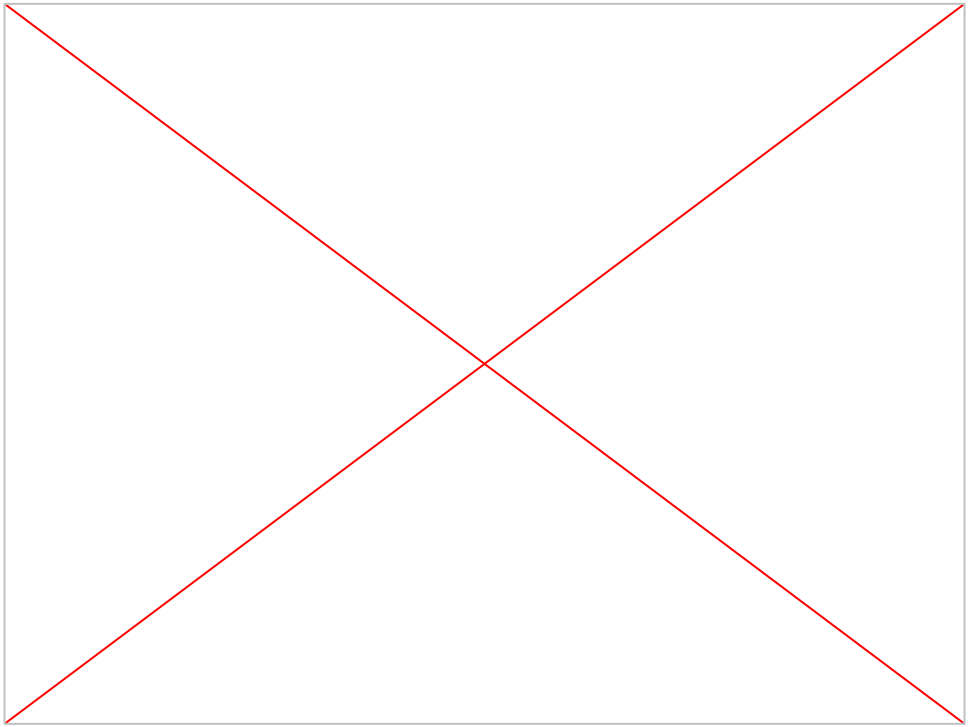
exchange code



実演 認証してみよう

auth codeのexhangeでもらった cognitoのid_token(JWT)を、

AWS Lambdaに投げ、メールアドレスを認証するデモです。



いかがでしたか？

cloudflare workersのコードを公開しています

https://github.com/mizuamedesu/gen-verifier/tree/master

JWTを検証後、メールアドレスを返すAWS Lambdaのコード(Node.js)を公開しています

https://github.com/mizuamedesu/cognito-lambda

又以前、今回の内容について雑に書いたブログがあるのでそちらもご覧下さい

https://mizuame.works/blog/2025-01-21/