MODÈLES DE SÉCURITÉ ET D'AUTHENTIFICATION · MODULE 5

Fédération avec OpenID Connect

Guide de la leçon · Guide du lab / de l'examen

LearnWisej

Guide de la leçon

OAuth face à OpenID Connect, le flux authorization code avec PKCE via les handlers de cookies et OpenID Connect d'ASP.NET Core enregistrés dans Startup.cs avant app.UseWisej(), la validation des jetons, le mappage du compte par émetteur et sujet dans la session Wisej.NET, le repli sur la connexion locale et la déconnexion des deux sessions. Ce que cela signifie pour un développeur qui sécurise les chemins de connexion, d'identité et d'autorisation d'une application Wisej.NET en production — et comment aborder ce module.

Objectif en production

OAuth face à OpenID Connect, le flux authorization code avec PKCE via les handlers de cookies et OpenID Connect d'ASP.NET Core enregistrés dans Startup.cs avant app.UseWisej(), la validation des jetons, le mappage du compte par émetteur et sujet dans la session Wisej.NET, le repli sur la connexion locale et la déconnexion des deux sessions.

1. Le problème en production

Les partenaires d'AccessOps ont déjà des comptes dans leur propre organisation, avec sa propre politique multifacteur et son propre processus de départ ; la route partenaires délègue donc l'authentification à un fournisseur OpenID Connect au lieu de gérer un second mot de passe. L'identité transite désormais par le navigateur de l'utilisateur, où chaque paramètre peut être forgé, rejoué ou échangé. Wisej.NET 4.1.7 n'a pas de composant de fédération ; AccessOps branche donc les handlers standard d'ASP.NET Core dans son hôte et transmet leur résultat vérifié à la session Wisej.NET.

2. Concepts clés

Concept	Pourquoi c'est important
OAuth face à OpenID Connect	OAuth autorise un client à appeler une API protégée ; OpenID Connect ajoute une couche d'identité, un ID token qui indique au client qui s'est connecté.
authorization code + PKCE	Le navigateur ne transporte qu'un code de courte durée ; le serveur l'échange par le canal arrière avec un code verifier qui n'a jamais quitté le serveur, si bien qu'un code volé est inutile.
state et nonce	Le state rattache le callback à la tentative du navigateur qui l'a lancée ; le nonce rattache l'ID token à cette même requête, si bien qu'aucun des deux ne peut être rejoué dans une autre connexion.
ID token et access token	L'ID token est une preuve d'identité destinée à AccessOps lui-même ; l'access token appartient au serveur de ressources pour lequel il a été émis et ne prouve rien sur la personne au clavier.
émetteur + sujet	La paire validée `iss` et `sub` est la seule clé de compte externe stable ; adresses e-mail et noms d'affichage peuvent être réattribués.
callback exact, autorité épinglée	Le fournisseur n'envoie les codes qu'à une URI de redirection enregistrée, et le handler ne charge clés et endpoints que depuis une seule autorité configurée, si bien qu'aucun des deux ne peut être détourné.
handle de transfert	Une valeur aléatoire à usage unique dans un cookie HttpOnly, qui permet au chargement de la page Wisej.NET de récupérer une identité validée ; le navigateur ne détient jamais de jeton.

3. Le fil de la leçon

Un code, deux preuves : le flux avec PKCE, state et nonce

Un partenaire clique sur *Sign in with your work account*. La page AccessOps appelle `IExternalSignIn.Begin()`, qui fait naviguer l'onglet vers l'endpoint ASP.NET Core ordinaire `/account/signin` ; cet endpoint émet un challenge pour `OpenIdConnectDefaults.AuthenticationScheme`. Le handler crée alors trois secrets : un state aléatoire, protégé et adossé à un cookie de corrélation dans ce navigateur ; un nonce, conservé dans un cookie de nonce ; et un code verifier PKCE dont le hachage SHA-256 part vers le fournisseur comme challenge S256. Le partenaire se connecte chez le fournisseur ; AccessOps ne voit jamais le mot de passe.

Le fournisseur renvoie le navigateur vers `/signin-oidc` avec un code et le state. Le handler vérifie d'abord que le state se déprotège et correspond au cookie de corrélation de *ce* navigateur. Il échange ensuite le code au token endpoint par le canal arrière, avec le secret client et le verifier d'origine, ce qui prouve que celui qui échange le code est bien celui qui a lancé la tentative. Enfin, il valide l'ID token : signature contre les clés publiées du fournisseur, émetteur, audience, durée de validité et nonce. Ce n'est qu'alors qu'il existe une identité.

Gardez les deux jetons séparés. L'ID token répond à la question *qui s'est connecté, chez quel fournisseur, pour ce client*. Un access token est adressé à un serveur de ressources, par exemple une API d'annuaire partenaire, et AccessOps ne l'envoie qu'à celui-ci, depuis le serveur. Connecter quelqu'un à partir d'un access token décodé saute toutes les vérifications ci-dessus.

Startup.cs : les handlers avant app.UseWisej()

Une application Wisej.NET 4 est une application ASP.NET Core ; la fédération se configure donc dans `Startup.cs` avec le package `Microsoft.AspNetCore.Authentication.OpenIdConnect` 10.0.14 : `AddAccessOpsOpenIdConnect()` enregistre les handlers, et `MapAccessOpsAccountEndpoints()` appelle `app.UseAuthentication()` et mappe les endpoints de compte, le tout avant `app.UseWisej()`, pour que `/signin-oidc` et `/signout-callback-oidc` soient traités par le middleware d'authentification.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie()
    .AddOpenIdConnect(o ⇒
    {
        o.Authority = oidc["Authority"];        // pinned, HTTPS discovery
        o.ClientId = oidc["ClientId"];
        o.ClientSecret = oidc["ClientSecret"];  // Oidc__ClientSecret at runtime
        o.ResponseType = "code";
        o.UsePkce = true;
        o.SaveTokens = false;                   // tokens stay on the server
        o.Scope.Clear(); o.Scope.Add("openid"); o.Scope.Add("profile");
        o.Events = new OpenIdConnectEvents { OnTokenValidated = HandOffToWisej };
    });
// app.UseAuthentication(); endpoints; then app.UseWisej();
```

`Authority` provient de la configuration et c'est le seul endroit d'où sont récupérées la découverte et les clés de signature ; `RequireHttpsMetadata` reste activé en dehors du lab. L'enregistrement auprès du fournisseur liste un seul callback HTTPS exact se terminant par le `CallbackPath` du handler ; derrière un proxy qui termine TLS, l'origine canonique doit donc être préservée (`app.UseForwardedHeaders()`), faute de quoi le handler construit une URI de redirection `http://` que le fournisseur refuse. Le secret client arrive sous la forme `Oidc__ClientSecret` depuis l'environnement ou les user secrets, qui ne se chargent que lorsque `GenerateAssemblyInfo` vaut true (le modèle Wisej le met à false). Laissez `MapInboundClaims` désactivé pour que `iss` et `sub` gardent leurs noms de protocole, et alignez `ResponseMode` sur le fournisseur ; la valeur par défaut est `form_post`.

D'un jeton validé à un principal Wisej.NET

La requête de callback n'a pas de session Wisej.NET, et `Application.RefreshSessionId()` n'atteint le navigateur que lors d'un chargement de page. `OnTokenValidated` ne connecte donc personne : il construit une `ExternalIdentity` à partir de l'émetteur et du sujet validés, la met en attente dans `ExternalHandoffStore` sous un handle aléatoire, écrit ce handle dans le cookie HttpOnly `AccessOps.Handoff`, appelle `HandleResponse()` pour qu'aucune session de cookie de l'hôte ne soit créée, et redirige vers la page de démarrage.

Lors de ce chargement de page, dans `Program.Main` ou `Application.ApplicationRefresh`, `WisejExternalSignIn.TryCompleteOnPageLoad()` lit le handle avec `Application.Cookies[ … ]`, fait

expirer le cookie et échange le handle une seule fois. `ExternalIdentityMapper` recherche l'émetteur et le sujet dans `external_logins`, jamais par adresse e-mail, refuse un compte désactivé ou un tenant qu'AccessOps n'a pas approuvé, et renvoie le compte local avec son tenant et ses rôles tirés des données d'AccessOps. Le `GrantStore` du Module 4 émet alors exactement un octroi de connexion, et `ApplicationPrincipal.Establish( ... )` définit `Application.User` sur un `ClaimsPrincipal` portant le compte, le tenant et l'octroi, et fait tourner l'ID de session. Rien dans le jeton n'est devenu une permission.

Le formulaire de mot de passe du Module 4 reste disponible sous *Service accounts only* : `LoginService` accepte des comptes de service nommés comme `svc.reports` et renvoie l'échec générique à tout compte interactif, si bien que le repli n'est jamais une seconde porte d'entrée.

Les rejets avant l'octroi, et deux déconnexions

Chaque échec doit s'arrêter avant que `GrantStore` ne s'exécute. Un callback portant le state d'une autre tentative (fixture F06) ne se déprotège pas, et le handler lève `OnRemoteFailure` ; AccessOps journalise une raison issue d'une liste d'autorisation, jamais la query du callback, et redirige vers la vue de connexion. Un callback rejoué (F07) trouve son cookie de corrélation déjà consommé et son code déjà échangé, et un handle rejoué ne trouve rien dans le magasin ; la tentative n'a donc toujours qu'un seul octroi au total. Une identité valide issue d'un tenant non approuvé (F08) passe les vérifications du protocole et est refusée par le mappeur, sans principal. Un refus du fournisseur arrive par `OnAccessDenied` ou `OnRemoteFailure` et aboutit sur la vue de connexion avec un chemin pour réessayer.

La déconnexion met fin à deux sessions, et chacune est vérifiée séparément. Localement, AccessOps révoque l'octroi, si bien qu'un export réessayé depuis un second onglet ouvert est refusé, puis met fin à la session Wisej.NET. Le navigateur visite ensuite `/account/signout`, qui renvoie `Results.SignOut` pour les schémas de cookies et OpenID Connect ; le handler l'envoie vers l'end-session endpoint du fournisseur puis le ramène par `/signout-callback-oidc`. La preuve que cela a fonctionné, c'est que la connexion suivante redemande des identifiants. Aucune des deux ne prouve que les refresh tokens ont été révoqués ; vérifiez la révocation, là où elle est proposée, comme un troisième résultat.

Réflexe de sécurité

Gardez les identités séparées et décidez sur le serveur. Le principal de l'hôte, le principal applicatif Wisej.NET et le jeton Windows effectif sont trois faits différents, et aucun n'est une permission. Chaque lecture protégée vérifie l'octroi actuel, le tenant et l'enregistrement à la couche service avant que le repository ne s'exécute, et un bouton masqué, un claim de rôle ou un onglet déconnecté ne prouvent rien tant que cette vérification n'a pas refusé.

Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Expliquer le flux authorization code d'OpenID Connect avec PKCE, state et nonce, et à quoi servent respectivement l'ID token et l'access token.
- ✓ Configurer les handlers de cookies et OpenID Connect d'ASP.NET Core dans Startup.cs avant app.UseWisej(), avec un callback enregistré exact, des scopes minimaux, une autorité épinglée et

des secrets chargés à l'exécution.

- ✓ Mapper l'émetteur et le sujet validés vers un compte et un tenant AccessOps locaux, établir le principal de session Wisej.NET à partir de données de confiance, et conserver un repli sur la connexion locale pour les comptes de service.
- ✓ Rejeter un state forgé, un callback rejoué et un tenant non approuvé avant la création de tout octroi, et se déconnecter de la session locale et de la session du fournisseur comme deux résultats distincts et vérifiés.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.

Lab pratique

Ajoutez la fédération des partenaires à AccessOps : enregistrez l'application auprès d'un fournisseur OpenID Connect de test et d'un tenant de test, configurez les handlers de cookies et OpenID Connect dans Startup.cs avant app.UseWisej() pour le flux authorization code avec PKCE, un callback HTTPS exact, l'origine canonique préservée à travers le proxy et le secret client chargé à l'exécution, et transmettez l'identité validée à la session Wisej.NET, où un mappeur de comptes recherche l'émetteur et le sujet, affecte le tenant et les rôles à partir des données d'AccessOps et crée exactement un octroi de connexion. Gardez la connexion locale du Module 4 comme repli pour les comptes de service, réalisez une connexion réussie, puis utilisez les fixtures jetables pour modifier le state, rejouer un callback déjà consommé, vous connecter depuis un tenant non approuvé et simuler un refus du fournisseur, en montrant qu'aucun octroi applicatif n'est créé dans aucun des quatre cas, et terminez par une déconnexion qui révoque l'octroi local et met fin à la session du fournisseur.

Point de départ

Ouvrez AccessOps tel que vous l'avez laissé après le Module 4 : la page de connexion de la route partenaires avec son formulaire de mot de passe, le `GrantStore` que vérifie chaque export, et `ApplicationPrincipal` qui établit le principal de session lors d'un chargement de page. Utilisez un enregistrement de fournisseur hors production et un tenant de test jetable (un fournisseur d'identité de test local convient), ainsi que les fixtures réservées F06, F07 et F08 pour les cas négatifs.

Approche suggérée

- ✓ Enregistrez AccessOps auprès du fournisseur de test avec un seul callback HTTPS exact se terminant par `/signin-oidc`, uniquement les scopes `openid profile email`, et un secret client que vous définissez sous la forme `Oidc__ClientSecret` dans l'environnement ou les user secrets, jamais dans `appsettings.json`.
- ✓ Écrivez `AddAccessOpsOpenIdConnect()` dans `Startup.cs` : schéma de cookies plus OpenID Connect avec `ResponseType` code, `UsePkce`, `SaveTokens` à false et une `Authority` épinglée ; mappez `/account/signin` et `/account/signout` et appelez `app.UseAuthentication()` avant `app.UseWisej()`.
- ✓ Dans `OnTokenValidated`, mettez l'émetteur et le sujet en attente dans `ExternalHandoffStore`, définissez le cookie HttpOnly `AccessOps.Handoff` et appelez `HandleResponse()` ; au chargement de la page, laissez `TryCompleteOnPageLoad()` l'échanger une seule fois, exécuter `ExternalIdentityMapper`, créer un octroi et appeler `ApplicationPrincipal.Establish`.
- ✓ Exécutez le chemin de réussite, puis F06 (state d'une autre tentative), F07 (le même callback deux fois), F08 (un tenant non approuvé) et un refus du fournisseur, en comptant les octrois après chacun et en ne journalisant qu'une corrélation masquée et la raison issue de la liste d'autorisation.

- ✓ Déconnectez-vous, réessayez un export depuis un second onglet ouvert, puis reconnectez-vous et consignez si le fournisseur redemande des identifiants ; dessinez le diagramme de séquence épuré à partir de ce que vous avez observé.

Livrables attendus

- ✓ Configuration des handlers de cookies et OpenID Connect dans Startup.cs, secrets hors du code source
- ✓ Mappeur de comptes par émetteur et sujet créant un octroi par connexion
- ✓ Repli sur la connexion locale pour les comptes de service
- ✓ Cinq résultats de transaction : une réussite et quatre rejets sans octroi
- ✓ Diagramme de séquence épuré et déconnexion couvrant les deux sessions

Questions de révision

- ✓ Un attaquant copie l'URL de callback d'un partenaire, code et state compris, et l'ouvre dans son propre navigateur quelques secondes plus tard. Quelles deux vérifications indépendantes la refusent, et qu'est-ce qui protégerait encore AccessOps si le code n'avait pas encore été échangé ?
- ✓ L'adresse e-mail d'un partenaire change chez son fournisseur, et une nouvelle recrue y reçoit l'ancienne adresse. Que fait votre mappeur à la prochaine connexion de chacun, et pourquoi un mappeur indexé sur l'adresse e-mail se tromperait-il ?
- ✓ Après la déconnexion, l'utilisateur suivant sur une machine partagée est reconnecté directement sans mot de passe. Lequel de vos deux résultats de déconnexion a échoué, et quelle preuve l'aurait détecté avant la release ?

Réflexe de sécurité

Gardez les identités séparées et décidez sur le serveur. Le principal de l'hôte, le principal applicatif Wisej.NET et le jeton Windows effectif sont trois faits différents, et aucun n'est une permission. Chaque lecture protégée vérifie l'octroi actuel, le tenant et l'enregistrement à la couche service avant que le repository ne s'exécute, et un bouton masqué, un claim de rôle ou un onglet déconnecté ne prouvent rien tant que cette vérification n'a pas refusé.