PATRONES DE SEGURIDAD Y AUTENTICACIÓN · MÓDULO 5

Federación con OpenID Connect

Guía de la lección · Guía del laboratorio / examen

LearnWisej

Guía de la lección

OAuth frente a OpenID Connect, el flujo de código de autorización con PKCE mediante los controladores de cookies y de OpenID Connect de ASP.NET Core registrados en Startup.cs antes de app.UseWisej(), la validación de tokens, la asignación del emisor y el sujeto a una cuenta en la sesión Wisej.NET, el inicio de sesión local como alternativa y el cierre de ambas sesiones. Qué significa esto para un desarrollador que protege los caminos de inicio de sesión, identidad y autorización de una aplicación Wisej.NET en producción — y cómo leer este módulo.

Enfoque de producción

OAuth frente a OpenID Connect, el flujo de código de autorización con PKCE mediante los controladores de cookies y de OpenID Connect de ASP.NET Core registrados en Startup.cs antes de app.UseWisej(), la validación de tokens, la asignación del emisor y el sujeto a una cuenta en la sesión Wisej.NET, el inicio de sesión local como alternativa y el cierre de ambas sesiones.

1. El problema en producción

Los socios de AccessOps ya tienen cuentas en su propia organización, con su propia directiva de autenticación multifactor y su propio proceso de bajas, así que la ruta de socios delega la autenticación en un proveedor de OpenID Connect en lugar de mantener una segunda contraseña. La identidad viaja ahora a través del navegador del usuario, donde cada parámetro puede falsificarse, reutilizarse o intercambiarse. Wisej.NET 4.1.7 no tiene ningún componente de federación, así que AccessOps conecta los controladores estándar de ASP.NET Core en su host y entrega su resultado verificado a la sesión Wisej.NET.

2. Conceptos clave

Concepto	Por qué importa
OAuth frente a OpenID Connect	OAuth autoriza a un cliente a llamar a una API protegida; OpenID Connect añade una capa de identidad, un token de ID que le dice al cliente quién inició sesión.
código de autorización + PKCE	El navegador solo lleva un código de vida corta; el servidor lo canjea por el canal trasero con un verificador de código que nunca salió del servidor, así que un código robado no sirve de nada.
state y nonce	El state ata el callback al intento del navegador que lo inició; el nonce ata el token de ID a esa misma petición, así que ninguno puede reutilizarse en otro inicio de sesión.
token de ID y token de acceso	El token de ID es evidencia de identidad para el propio AccessOps; el token de acceso pertenece al servidor de recursos para el que se emitió y no demuestra nada sobre quién está ante el teclado.
emisor + sujeto	El par validado `iss` y `sub` es la única clave estable de cuenta externa; las direcciones de correo y los nombres visibles pueden reasignarse.
callback exacto, autoridad fijada	El proveedor envía códigos solo a una URI de redirección registrada, y el controlador carga claves y endpoints solo de una autoridad configurada, así que ninguno de los dos puede redirigirse.
identificador de entrega	Un valor aleatorio de un solo uso en una cookie HttpOnly que permite a la carga de página de Wisej.NET recoger una identidad validada; el navegador nunca tiene un token.

3. Desarrollo de la lección

Un código, dos pruebas: el flujo con PKCE, state y nonce

Un socio hace clic en *Sign in with your work account*. La página de AccessOps llama a `IExternalSignIn.Begin()`, que lleva la pestaña al endpoint ordinario de ASP.NET Core `/account/signin`; ese endpoint emite un desafío para `OpenIdConnectDefaults.AuthenticationScheme`. El controlador crea entonces tres secretos: un state aleatorio, protegido y respaldado por una cookie de correlación en este navegador; un nonce, guardado en una cookie de nonce; y un verificador de código PKCE cuyo hash SHA-256 viaja al proveedor como desafío S256. El socio inicia sesión en el proveedor; AccessOps nunca ve la contraseña.

El proveedor devuelve el navegador a `/signin-oidc` con un código y el state. El controlador comprueba primero que el state se desprotege y coincide con la cookie de correlación de *este* navegador. Después canjea el código en el endpoint de tokens por el canal trasero con el secreto del cliente y el verificador original, lo que demuestra que quien canjea inició el intento. Por último valida el token de ID: la firma frente a las claves publicadas del proveedor, el emisor, la audiencia, la vigencia y el nonce. Solo entonces hay una identidad.

Mantén separados los dos tokens. El token de ID responde a *quién inició sesión, en qué proveedor y para este cliente*. Un token de acceso va dirigido a un servidor de recursos, como la API de directorio de un

socio, y AccessOps solo lo envía allí, desde el servidor. Iniciar la sesión de alguien a partir de un token de acceso descodificado se salta todas las comprobaciones anteriores.

Startup.cs: los controladores antes de app.UseWisej()

Una aplicación Wisej.NET 4 es una aplicación ASP.NET Core, así que la federación se configura en `Startup.cs` con el paquete `Microsoft.AspNetCore.Authentication.OpenIdConnect` 10.0.14: `AddAccessOpsOpenIdConnect()` registra los controladores, y `MapAccessOpsAccountEndpoints()` llama a `app.UseAuthentication()` y asigna los endpoints de cuenta, todo antes de `app.UseWisej()`, de modo que `/signin-oidc` y `/signout-callback-oidc` los responde el middleware de autenticación.

```
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` procede de la configuración y es el único lugar del que se obtienen el documento de descubrimiento y las claves de firma; `RequireHttpsMetadata` permanece activado fuera del laboratorio. El registro en el proveedor incluye un único callback HTTPS exacto que termina en el `CallbackPath` del controlador, así que detrás de un proxy que termina TLS el origen canónico debe conservarse (`app.UseForwardedHeaders()`), o el controlador construirá una URI de redirección `http://` que el proveedor rechazará. El secreto del cliente llega como `Oidc__ClientSecret` desde el entorno o desde los secretos de usuario, que solo se cargan cuando `GenerateAssemblyInfo` es true (la plantilla de Wisej lo pone a false). Mantén desactivado `MapInboundClaims` para que `iss` y `sub` conserven sus nombres de protocolo, y ajusta `ResponseMode` al proveedor; el valor predeterminado es `form_post`.

De un token validado a un principal de Wisej.NET

La petición del callback no tiene sesión Wisej.NET, y `Application.RefreshSessionId()` solo llega al navegador en una carga de página. Por eso `OnTokenValidated` no inicia la sesión de nadie: construye una `ExternalIdentity` a partir del emisor y el sujeto validados, la aparca en `ExternalHandoffStore` bajo un identificador aleatorio, escribe ese identificador en la cookie HttpOnly `AccessOps.Handoff`, llama a `HandleResponse()` para que no se cree ninguna sesión de cookie del host y redirige a la página de inicio.

En esa carga de página, en `Program.Main` o en `Application.ApplicationRefresh`, `WisejExternalSignIn.TryCompleteOnPageLoad()` lee el identificador con `Application.Cookies[...]`, hace caducar la cookie y canjea el identificador una sola vez. `ExternalIdentityMapper` busca el emisor y el sujeto en `external_logins`, nunca por correo, rechaza una cuenta deshabilitada o un tenant que AccessOps no ha aprobado, y devuelve la cuenta local con su tenant y sus roles a partir de los datos de

AccessOps. Después, el `GrantStore` del Módulo 4 emite exactamente una concesión de inicio de sesión, y `ApplicationPrincipal.Establish( ... )` asigna a `Application.User` un `ClaimsPrincipal` con la cuenta, el tenant y la concesión, y rota el ID de sesión. Nada del token se convirtió en un permiso.

El formulario de contraseña del Módulo 4 permanece bajo *Service accounts only*: `LoginService` acepta cuentas de servicio con nombre como `svc.reports` y da a cualquier cuenta interactiva el fallo genérico, así que la alternativa nunca es una segunda puerta de entrada.

Rechazos antes de la concesión, y dos cierres de sesión

Cada fallo debe detenerse antes de que se ejecute `GrantStore`. Un callback que lleva el state de otro intento (fixture F06) no se puede desproteger, y el controlador produce `OnRemoteFailure`; AccessOps registra un motivo de una lista de permitidos, nunca la consulta del callback, y redirige a la vista de inicio de sesión. Un callback reutilizado (F07) encuentra su cookie de correlación ya consumida y su código ya canjeado, y un identificador reutilizado no encuentra nada en el almacén, así que el intento sigue teniendo una sola concesión en total. Una identidad válida de un tenant no aprobado (F08) supera las comprobaciones del protocolo y el asignador la rechaza sin principal. Una denegación del proveedor llega a través de `OnAccessDenied` u `OnRemoteFailure` y termina en la vista de inicio de sesión con un camino para reintentar.

El cierre de sesión termina dos sesiones, y cada una se verifica por separado. En local, AccessOps revoca la concesión, de modo que una exportación reintentada desde una segunda pestaña abierta se deniega, y después termina la sesión Wisej.NET. El navegador visita entonces `/account/signout`, que devuelve `Results.SignOut` para los esquemas de cookies y de OpenID Connect; el controlador lo envía al endpoint de fin de sesión del proveedor y de vuelta a través de `/signout-callback-oidc`. La prueba de que funcionó es que el siguiente inicio de sesión vuelva a pedir credenciales. Ninguno de los dos demuestra que se revocaran los tokens de actualización; comprueba la revocación, donde se ofrezca, como un tercer resultado.

Regla de seguridad

Mantén separadas las identidades y decide en el servidor. El principal del host, el principal de la aplicación Wisej.NET y el token efectivo de Windows son tres hechos distintos, y ninguno de ellos es un permiso. Cada lectura protegida comprueba la concesión vigente, el tenant y el registro en el límite del servicio antes de que se ejecute el repositorio, y un botón oculto, un claim de rol o una pestaña con la sesión cerrada no demuestran nada hasta que esa comprobación deniega.

Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Explicar el flujo de código de autorización de OpenID Connect con PKCE, state y nonce, y para qué sirven el token de ID y el token de acceso.
- ✓ Configurar los controladores de cookies y de OpenID Connect de ASP.NET Core en Startup.cs antes de app.UseWisej() con un callback registrado exacto, ámbitos mínimos, una autoridad fijada y secretos cargados en tiempo de ejecución.

- ✓ Asignar el emisor y el sujeto validados a una cuenta y un tenant locales de AccessOps, establecer el principal de la sesión Wisej.NET a partir de datos de confianza y mantener el inicio de sesión local como alternativa para las cuentas de servicio.
- ✓ Rechazar un state falsificado, un callback reutilizado y un tenant no aprobado antes de crear ninguna concesión, y cerrar la sesión local y la del proveedor como resultados separados y verificados.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.

Laboratorio práctico

Añade la federación de socios a AccessOps: registra la aplicación en un proveedor de OpenID Connect de prueba con su tenant, configura los controladores de cookies y de OpenID Connect en Startup.cs antes de app.UseWisej() para el flujo de código de autorización con PKCE, un callback HTTPS exacto, el origen canónico conservado a través del proxy y el secreto del cliente cargado en tiempo de ejecución, y entrega la identidad validada a la sesión Wisej.NET, donde un asignador de cuentas busca el emisor y el sujeto, asigna el tenant y los roles a partir de los datos de AccessOps y crea exactamente una concesión de inicio de sesión. Mantén el inicio de sesión local del Módulo 4 como alternativa para las cuentas de servicio, ejecuta un inicio de sesión correcto y usa después los fixtures desechables para cambiar el state, reutilizar un callback ya consumido, iniciar sesión desde un tenant no aprobado y simular una denegación del proveedor, mostrando que en ninguno de los cuatro se crea una concesión de la aplicación, y termina con un cierre de sesión que revoque la concesión local y finalice la sesión del proveedor.

Punto de partida

Abre AccessOps tal como lo dejaste tras el Módulo 4: la página de inicio de sesión de la ruta de socios con su formulario de contraseña, el `GrantStore` que comprueba cada exportación y `ApplicationPrincipal` estableciendo el principal de la sesión en una carga de página. Usa un registro de proveedor que no sea de producción y un tenant de prueba desechable (un proveedor de identidad de prueba local sirve), y los fixtures reservados F06, F07 y F08 para los casos negativos.

Enfoque sugerido

- ✓ Registra AccessOps en el proveedor de prueba con un único callback HTTPS exacto que termine en `/signin-oidc`, solo los ámbitos `openid profile email` y un secreto del cliente que establezcas como `Oidc__ClientSecret` en el entorno o en los secretos de usuario, nunca en `appsettings.json`.
- ✓ Escribe `AddAccessOpsOpenIdConnect()` en `Startup.cs`: el esquema de cookies más OpenID Connect con `ResponseType` code, `UsePkce`, `SaveTokens` false y una `Authority` fijada; asigna `/account/signin` y `/account/signout` y llama a `app.UseAuthentication()` antes de `app.UseWisej()`.
- ✓ En `OnTokenValidated`, aparca el emisor y el sujeto en `ExternalHandoffStore`, establece la cookie HttpOnly `AccessOps.Handoff` y llama a `HandleResponse()`; en la carga de página, deja que `TryCompleteOnPageLoad()` la canjee una sola vez, ejecute `ExternalIdentityMapper`, cree una concesión y llame a `ApplicationPrincipal.Establish`.
- ✓ Ejecuta el camino correcto y después F06 (state de otro intento), F07 (el mismo callback dos veces), F08 (un tenant no aprobado) y una denegación del proveedor, contando las concesiones tras cada uno y registrando solo una correlación ocultada y el motivo de la lista de permitidos.
- ✓ Cierra la sesión, reintenta una exportación desde una segunda pestaña abierta, vuelve a iniciar sesión y anota si el proveedor pide credenciales; dibuja el diagrama de secuencia saneado a

partir de lo que observaste.

Entregables requeridos

- ✓ Configuración de los controladores de cookies y de OpenID Connect en Startup.cs con los secretos fuera del código fuente
- ✓ Asignador de cuentas por emisor y sujeto que crea una concesión por inicio de sesión
- ✓ Inicio de sesión local como alternativa para las cuentas de servicio
- ✓ Cinco resultados de la transacción: un éxito y cuatro rechazos sin concesión
- ✓ Diagrama de secuencia saneado y cierre de sesión que cubre ambas sesiones

Preguntas de repaso

- ✓ Un atacante copia la URL del callback de un socio, con el código y el state incluidos, y la abre en su propio navegador unos segundos después. ¿Qué dos comprobaciones independientes la rechazan, y qué seguiría protegiendo a AccessOps si el código todavía no se hubiera canjeado?
- ✓ El correo de un socio cambia en su proveedor y una nueva incorporación de allí recibe la dirección antigua. ¿Qué hace tu asignador en el siguiente inicio de sesión de cada persona, y por qué se equivocaría un asignador basado en el correo?
- ✓ Tras cerrar la sesión, el siguiente usuario de un equipo compartido vuelve a entrar directamente sin contraseña. ¿Cuál de tus dos resultados de cierre de sesión falló, y qué evidencia lo habría detectado antes del release?

Regla de seguridad

Mantén separadas las identidades y decide en el servidor. El principal del host, el principal de la aplicación Wisej.NET y el token efectivo de Windows son tres hechos distintos, y ninguno de ellos es un permiso. Cada lectura protegida comprueba la concesión vigente, el tenant y el registro en el límite del servicio antes de que se ejecute el repositorio, y un botón oculto, un claim de rol o una pestaña con la sesión cerrada no demuestran nada hasta que esa comprobación deniega.