SICHERHEITS- UND AUTHENTIFIZIERUNGSMUSTER · MODUL 5

Föderation mit OpenID Connect

Lektionsleitfaden · Lab- und Prüfungsleitfaden

LearnWisej

Lektionsleitfaden

OAuth gegenüber OpenID Connect, der Authorization Code Flow mit PKCE über die Cookie- und OpenID-Connect-Handler von ASP.NET Core, registriert in Startup.cs vor app.UseWisej(), Token-Validierung, Zuordnung von Issuer und Subject zu einem Konto in der Wisej.NET-Session, lokale Anmeldung als Rückfall und Abmeldung aus beiden Sessions. Was das für Entwicklerinnen und Entwickler bedeutet, die die Anmelde-, Identitäts- und Autorisierungspfade einer Wisej.NET-Anwendung im Produktivbetrieb absichern – und wie Sie dieses Modul lesen.

> ◎ **Fokus: Produktivbetrieb**
>
> OAuth gegenüber OpenID Connect, der Authorization Code Flow mit PKCE über die Cookie- und OpenID-Connect-Handler von ASP.NET Core, registriert in Startup.cs vor app.UseWisej(), Token-Validierung, Zuordnung von Issuer und Subject zu einem Konto in der Wisej.NET-Session, lokale Anmeldung als Rückfall und Abmeldung aus beiden Sessions.

1. Das Problem im Produktivbetrieb

Partner von AccessOps haben bereits Konten in ihrer eigenen Organisation, mit eigener Multi-Faktor-Richtlinie und eigenem Austrittsprozess; die Partner-Route delegiert die Authentifizierung deshalb an einen OpenID-Connect-Anbieter, statt ein zweites Passwort zu führen. Die Identität reist nun durch den Browser des Benutzers, wo jeder Parameter gefälscht, wiederverwendet oder vertauscht werden kann. Wisej.NET 4.1.7 hat keine Föderationskomponente; AccessOps bindet deshalb die Standard-Handler von ASP.NET Core in seinen Host ein und übergibt deren verifiziertes Ergebnis an die Wisej.NET-Session.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
OAuth gegenüber OpenID Connect	OAuth autorisiert einen Client, eine geschützte API aufzurufen; OpenID Connect ergänzt eine Identitätsschicht, ein ID-Token, das dem Client sagt, wer sich angemeldet hat.
Authorization Code + PKCE	Der Browser trägt nur einen kurzlebigen Code; der Server löst ihn im Back Channel mit einem Code Verifier ein, der den Server nie verlassen hat – ein gestohlener Code ist also nutzlos.
State und Nonce	Der State bindet den Callback an den Versuch im Browser, der ihn gestartet hat; die Nonce bindet das ID-Token an dieselbe Anfrage, sodass sich keines von beiden in eine andere Anmeldung einschleusen lässt.
ID-Token und Access Token	Das ID-Token ist ein Identitätsnachweis für AccessOps selbst; das Access Token gehört dem Resource Server, für den es ausgestellt wurde, und beweist nichts darüber, wer an der Tastatur sitzt.
Issuer + Subject	Das validierte Paar aus `iss` und `sub` ist der einzige stabile externe Kontoschlüssel; E-Mail-Adressen und Anzeigenamen können neu vergeben werden.
exakter Callback, festgelegte Authority	Der Anbieter sendet Codes nur an eine registrierte Redirect-URI, und der Handler lädt Schlüssel und Endpunkte nur von einer konfigurierten Authority – keines von beiden lässt sich umleiten.
Übergabe-Handle	Ein einmaliger Zufallswert in einem HttpOnly-Cookie, mit dem das Laden der Wisej.NET-Seite eine validierte Identität abholt; der Browser hält nie ein Token.

3. Der rote Faden der Lektion

Ein Code, zwei Nachweise: der Flow mit PKCE, State und Nonce

Ein Partner klickt auf *Sign in with your work account*. Die AccessOps-Seite ruft `IExternalSignIn.Begin()` auf, das den Tab zum gewöhnlichen ASP.NET-Core-Endpunkt `/account/signin` navigiert; dieser Endpunkt stellt eine Challenge für `OpenIdConnectDefaults.AuthenticationScheme`. Der Handler erzeugt nun drei Geheimnisse: einen zufälligen State, geschützt und abgesichert durch ein Korrelations-Cookie in diesem Browser; eine Nonce in einem Nonce-Cookie; und einen PKCE-Code-Verifier, dessen SHA-256-Hash als S256-Challenge zum Anbieter reist. Der Partner meldet sich beim Anbieter an; AccessOps sieht das Passwort nie.

Der Anbieter schickt den Browser mit einem Code und dem State zu `/signin-oidc` zurück. Der Handler prüft zuerst, dass sich der State entschlüsseln lässt und zum Korrelations-Cookie *dieses* Browsers passt. Dann löst er den Code über den Back Channel mit dem Client Secret und dem ursprünglichen Verifier am Token-Endpunkt ein und beweist damit, dass der Einlösende den Versuch gestartet hat. Schließlich validiert er das ID-Token: Signatur gegen die veröffentlichten Schlüssel des Anbieters, Issuer, Audience, Gültigkeitsdauer und Nonce. Erst dann gibt es eine Identität.

Halten Sie die beiden Tokens auseinander. Das ID-Token beantwortet, *wer sich bei welchem Anbieter für diesen Client angemeldet hat*. Ein Access Token ist an einen Resource Server adressiert, etwa eine Verzeichnis-API des Partners, und AccessOps sendet es nur dorthin, vom Server aus. Jemanden anhand eines dekodierten Access Tokens anzumelden überspringt jede der obigen Prüfungen.

Startup.cs: die Handler vor app.UseWisej()

Eine Wisej.NET-4-Anwendung ist eine ASP.NET-Core-Anwendung; die Föderation wird daher in `Startup.cs` mit dem Paket `Microsoft.AspNetCore.Authentication.OpenIdConnect` 10.0.14 konfiguriert: `AddAccessOpsOpenIdConnect()` registriert die Handler, und `MapAccessOpsAccountEndpoints()` ruft `app.UseAuthentication()` auf und bildet die Konto-Endpunkte ab, alles vor `app.UseWisej()`, sodass `/signin-oidc` und `/signout-callback-oidc` von der Authentifizierungs-Middleware beantwortet werden.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie()
    .AddOpenIdConnect(o ⇒
    {
        o.Authority = oidc["Authority"];        // festgelegt, Discovery über HTTPS
        o.ClientId = oidc["ClientId"];
        o.ClientSecret = oidc["ClientSecret"];  // Oidc__ClientSecret zur Laufzeit
        o.ResponseType = "code";
        o.UsePkce = true;
        o.SaveTokens = false;                   // Tokens bleiben auf dem Server
        o.Scope.Clear(); o.Scope.Add("openid"); o.Scope.Add("profile");
        o.Events = new OpenIdConnectEvents { OnTokenValidated = HandOffToWisej };
    });
// app.UseAuthentication(); Endpunkte; dann app.UseWisej();
```

`Authority` kommt aus der Konfiguration und ist der einzige Ort, von dem Discovery und Signaturschlüssel geholt werden; `RequireHttpsMetadata` bleibt außerhalb des Labs eingeschaltet. Die Registrierung beim Anbieter führt genau einen HTTPS-Callback, der auf den `CallbackPath` des Handlers endet; hinter einem Proxy, der TLS terminiert, muss der kanonische Origin daher erhalten bleiben (`app.UseForwardedHeaders()`), sonst baut der Handler eine `http://`-Redirect-URI, die der Anbieter ablehnt. Das Client Secret kommt als `Oidc__ClientSecret` aus der Umgebung oder aus den User Secrets, die nur geladen werden, wenn `GenerateAssemblyInfo` true ist (die Wisej-Vorlage setzt es auf false). Lassen Sie `MapInboundClaims` ausgeschaltet, damit `iss` und `sub` ihre Protokollnamen behalten, und stimmen Sie `ResponseMode` auf den Anbieter ab; der Standard ist `form_post`.

Von einem validierten Token zu einem Wisej.NET-Principal

Die Callback-Anfrage hat keine Wisej.NET-Session, und `Application.RefreshSessionId()` erreicht den Browser nur beim Laden einer Seite. `OnTokenValidated` meldet deshalb niemanden an: Es baut aus dem validierten Issuer und Subject eine `ExternalIdentity`, legt sie unter einem zufälligen Handle in `ExternalHandoffStore` ab, schreibt dieses Handle in das HttpOnly-Cookie `AccessOps.Handoff`, ruft `HandleResponse()` auf, damit keine Cookie-Session des Hosts entsteht, und leitet zur Startseite weiter.

Beim Laden dieser Seite, in `Program.Main` oder `Application.ApplicationRefresh`, liest `WisejExternalSignIn.TryCompleteOnPageLoad()` das Handle mit `Application.Cookies[ … ]`, lässt das

Cookie ablaufen und löst das Handle einmal ein. `ExternalIdentityMapper` schlägt Issuer und Subject in `external_logins` nach, nie die E-Mail-Adresse, weist ein deaktiviertes Konto oder einen von AccessOps nicht genehmigten Mandanten ab und liefert das lokale Konto mit Mandant und Rollen aus den AccessOps-Daten. Dann stellt der `GrantStore` aus Modul 4 genau einen Anmelde-Grant aus, und `ApplicationPrincipal.Establish( ... )` setzt `Application.User` auf einen `ClaimsPrincipal` mit Konto, Mandant und Grant und rotiert die Session-ID. Nichts im Token ist zu einer Berechtigung geworden.

Das Passwortformular aus Modul 4 bleibt unter *Service accounts only*: `LoginService` akzeptiert benannte Dienstkonten wie `svc.reports` und gibt jedem interaktiven Konto den generischen Fehler, sodass der Rückfall nie ein zweiter Weg hinein ist.

Ablehnungen vor dem Grant, und zwei Abmeldungen

Jeder Fehler muss stoppen, bevor `GrantStore` läuft. Ein Callback mit dem State eines anderen Versuchs (Fixture F06) lässt sich nicht entschlüsseln, und der Handler löst `OnRemoteFailure` aus; AccessOps protokolliert einen Grund aus einer Allowlist, nie die Query des Callbacks, und leitet zur Anmeldeansicht weiter. Ein wiederholter Callback (F07) findet sein Korrelations-Cookie bereits verbraucht und seinen Code bereits eingelöst, und ein wiederholtes Handle findet nichts im Speicher; der Versuch hat also insgesamt weiterhin einen Grant. Eine gültige Identität aus einem nicht genehmigten Mandanten (F08) besteht die Protokollprüfungen und wird vom Mapper ohne Principal abgewiesen. Eine Verweigerung durch den Anbieter kommt über `OnAccessDenied` oder `OnRemoteFailure` und landet auf der Anmeldeansicht mit einem Weg zum erneuten Versuch.

Die Abmeldung beendet zwei Sessions, und jede wird getrennt geprüft. Lokal entzieht AccessOps den Grant, sodass ein aus einem zweiten offenen Tab wiederholter Export abgelehnt wird, und beendet dann die Wisej.NET-Session. Danach ruft der Browser `/account/signout` auf, das `Results.SignOut` für das Cookie- und das OpenID-Connect-Schema zurückgibt; der Handler schickt ihn zum End-Session-Endpunkt des Anbieters und über `/signout-callback-oidc` zurück. Der Nachweis, dass das funktioniert hat, ist die nächste Anmeldung, die erneut nach Anmeldedaten fragt. Keines von beiden beweist, dass Refresh Tokens widerrufen wurden; prüfen Sie den Widerruf, wo angeboten, als drittes Ergebnis.

Sicherheitsregel

Halten Sie die Identitäten auseinander und entscheiden Sie auf dem Server. Der Host-Principal, der Wisej.NET-Anwendungs-Principal und das effektive Windows-Token sind drei verschiedene Fakten, und keines davon ist eine Berechtigung. Jeder geschützte Lesezugriff prüft den aktuellen Grant, den Mandanten und den Datensatz in der Service-Schicht, bevor das Repository läuft, und ein ausgeblendeter Button, ein Rollen-Claim oder ein abgemeldeter Tab beweist nichts, solange diese Prüfung nicht ablehnt.

Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Den OpenID-Connect-Authorization-Code-Flow mit PKCE, State und Nonce erklären und wofür jeweils das ID-Token und das Access Token da sind.

- ✓ Die Cookie- und OpenID-Connect-Handler von ASP.NET Core in Startup.cs vor app.UseWisej() konfigurieren, mit einem exakt registrierten Callback, minimalen Scopes, einer festgelegten Authority und zur Laufzeit geladenen Secrets.
- ✓ Den validierten Issuer und das Subject einem lokalen AccessOps-Konto und Mandanten zuordnen, den Wisej.NET-Session-Principal aus vertrauenswürdigen Daten etablieren und eine lokale Anmeldung als Rückfall für Dienstkonten behalten.
- ✓ Einen gefälschten State, einen wiederholten Callback und einen nicht genehmigten Mandanten abweisen, bevor ein Grant entsteht, und die Abmeldung aus der lokalen und der Anbieter-Session als getrennte, geprüfte Ergebnisse durchführen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.

Praktisches Lab

Ergänzen Sie AccessOps um Partner-Föderation: Registrieren Sie die Anwendung bei einem Test-Anbieter für OpenID Connect und einem Test-Mandanten, konfigurieren Sie die Cookie- und OpenID-Connect-Handler in Startup.cs vor app.UseWisej() für den Authorization Code Flow mit PKCE, einem exakten HTTPS-Callback, dem über den Proxy erhaltenen kanonischen Origin und dem zur Laufzeit geladenen Client Secret, und übergeben Sie die validierte Identität an die Wisej.NET-Session, in der ein Konto-Mapper Issuer und Subject nachschlägt, Mandant und Rollen aus den AccessOps-Daten zuweist und genau einen Anmelde-Grant erzeugt. Behalten Sie die lokale Anmeldung aus Modul 4 als Rückfall für Dienstkonten, führen Sie eine erfolgreiche Anmeldung aus, verwenden Sie dann die Wegwerf-Fixtures, um den State zu verändern, einen verbrauchten Callback zu wiederholen, sich aus einem nicht genehmigten Mandanten anzumelden und eine Verweigerung durch den Anbieter zu simulieren, und zeigen Sie, dass in keinem der vier Fälle ein Grant der Anwendung entsteht; schließen Sie mit einer Abmeldung ab, die den lokalen Grant entzieht und die Session beim Anbieter beendet.

Ausgangspunkt

Öffnen Sie AccessOps in dem Zustand nach Modul 4: die Anmeldeseite der Partner-Route mit ihrem Passwortformular, der `GrantStore`, den jeder Export prüft, und `ApplicationPrincipal`, das den Session-Principal beim Laden einer Seite etabliert. Verwenden Sie eine Registrierung bei einem Anbieter außerhalb des Produktivbetriebs und einen Wegwerf-Test-Mandanten (ein lokaler Test-Identitätsanbieter genügt) sowie die reservierten Fixtures F06, F07 und F08 für die Negativfälle.

Vorgeschlagenes Vorgehen

- ✓ Registrieren Sie AccessOps beim Test-Anbieter mit genau einem HTTPS-Callback, der auf `/signin-oidc` endet, nur den Scopes `openid profile email` und einem Client Secret, das Sie als `Oidc__ClientSecret` in der Umgebung oder in den User Secrets setzen, nie in `appsettings.json`.
- ✓ Schreiben Sie `AddAccessOpsOpenIdConnect()` in `Startup.cs`: Cookie-Schema plus OpenID Connect mit `ResponseType` code, `UsePkce`, `SaveTokens` false und einer festgelegten `Authority`; bilden Sie `/account/signin` und `/account/signout` ab und rufen Sie `app.UseAuthentication()` vor `app.UseWisej()` auf.
- ✓ Legen Sie in `OnTokenValidated` Issuer und Subject in `ExternalHandoffStore` ab, setzen Sie das HttpOnly-Cookie `AccessOps.Handoff` und rufen Sie `HandleResponse()` auf; lassen Sie beim Laden der Seite `TryCompleteOnPageLoad()` es einmal einlösen, `ExternalIdentityMapper` ausführen, einen Grant erzeugen und `ApplicationPrincipal.Establish` aufrufen.
- ✓ Führen Sie den Erfolgspfad aus, dann F06 (State aus einem anderen Versuch), F07 (derselbe Callback zweimal), F08 (ein nicht genehmigter Mandant) und eine Verweigerung durch den Anbieter; zählen Sie nach jedem Fall die Grants und protokollieren Sie nur eine geschwärzte Korrelation und den Grund aus der Allowlist.

- ✓ Melden Sie sich ab, wiederholen Sie einen Export aus einem zweiten offenen Tab, melden Sie sich dann erneut an und halten Sie fest, ob der Anbieter nach Anmeldedaten fragt; zeichnen Sie aus Ihren Beobachtungen das bereinigte Sequenzdiagramm.

Geforderte Ergebnisse

- ✓ Konfiguration der Cookie- und OpenID-Connect-Handler in Startup.cs mit Secrets außerhalb des Quellcodes
- ✓ Konto-Mapper für Issuer und Subject, der pro Anmeldung einen Grant erzeugt
- ✓ Lokale Anmeldung als Rückfall für Dienstkonten
- ✓ Fünf Transaktionsergebnisse: ein Erfolg und vier Ablehnungen ohne Grant
- ✓ Bereinigtes Sequenzdiagramm und Abmeldung, die beide Sessions abdeckt

Wiederholungsfragen

- ✓ Ein Angreifer kopiert die Callback-URL eines Partners samt Code und State und öffnet sie wenige Sekunden später im eigenen Browser. Welche zwei unabhängigen Prüfungen weisen sie ab, und was würde AccessOps noch schützen, wenn der Code noch nicht eingelöst worden wäre?
- ✓ Die E-Mail-Adresse eines Partners ändert sich bei seinem Anbieter, und eine Neueinstellung dort erhält die alte Adresse. Was tut Ihr Mapper bei der nächsten Anmeldung jeder der beiden Personen, und warum würde ein nach E-Mail-Adresse schlüsselnder Mapper falsch entscheiden?
- ✓ Nach der Abmeldung wird der nächste Benutzer an einem gemeinsam genutzten Rechner ohne Passwort direkt wieder angemeldet. Welches Ihrer beiden Abmeldeergebnisse ist gescheitert, und welcher Nachweis hätte es vor dem Release aufgedeckt?

Sicherheitsregel

Halten Sie die Identitäten auseinander und entscheiden Sie auf dem Server. Der Host-Principal, der Wisej.NET-Anwendungs-Principal und das effektive Windows-Token sind drei verschiedene Fakten, und keines davon ist eine Berechtigung. Jeder geschützte Lesezugriff prüft den aktuellen Grant, den Mandanten und den Datensatz in der Service-Schicht, bevor das Repository läuft, und ein ausgeblendeter Button, ein Rollen-Claim oder ein abgemeldeter Tab beweist nichts, solange diese Prüfung nicht ablehnt.