SICHERHEITS- UND AUTHENTIFIZIERUNGSMUSTER · MODUL 4

Eigene Anmeldung und Session-Sicherheit

Lektionsleitfaden · Lab- und Prüfungsleitfaden

LearnWisej

Lektionsleitfaden

Die Anmeldetransaktion selbst verantworten, den Anwendungs-Principal etablieren und die Session-ID rotieren, Host-Cookies nur über HTTP ausstellen, Abmeldung als Entzug eines serverseitigen Grants und das Schließen der Pfade für Request Forgery, WebSocket-Origin und Weiterleitungen. 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

Die Anmeldetransaktion selbst verantworten, den Anwendungs-Principal etablieren und die Session-ID rotieren, Host-Cookies nur über HTTP ausstellen, Abmeldung als Entzug eines serverseitigen Grants und das Schließen der Pfade für Request Forgery, WebSocket-Origin und Weiterleitungen.

1. Das Problem im Produktivbetrieb

Partner erreichen AccessOps über die öffentliche HTTPS-Route, wo es keine Windows-Challenge gibt, auf die man sich stützen könnte: Die Anwendung verantwortet die gesamte Anmeldung, vom Passwortfeld bis zu dem Moment, in dem die Konsole die Anmeldeansicht ersetzt. Jede Abkürzung wird zum Angriffspfad: eine Sperre, die verrät, welche Namen existieren, eine Session-ID, die die Anmeldung überlebt, oder eine Abmeldung, die einen Button ausblendet, während ein zweiter Tab weiter exportiert. Dieses Modul baut die Anmeldung für Partner und beweist, dass ein entzogener Grant überall abgelehnt wird.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Anmeldetransaktion	Die serverseitige Abfolge Drosselung → Verifizierung → Konto und Mandant auflösen → Richtlinie anwenden → Grant erzeugen → Principal etablieren, bei der jeder fehlgeschlagene Schritt den Versuch beendet, ohne Zustand zu hinterlassen.
Anmeldedrosselung	Zähler der jüngsten Fehlversuche pro Konto und pro Client-Adresse, die weitere Versuche für eine Sperrfrist abweisen; sie liegen im Anmeldedienst, weil sich ein Wisej-Klick nicht durch ASP.NET-Core-Middleware begrenzen lässt.
Generischer Fehler	Ein identischer Satz für falsches Passwort, unbekannten Namen, deaktiviertes Konto und Sperre, mit vergleichbarem Zeitverhalten, sodass der Bildschirm nie bestätigt, dass ein Konto existiert.
Anwendungs-Principal	Der `ClaimsPrincipal`, der `Application.User` für diese Wisej-Session zugewiesen wird; er ist Session-Zustand, kein Host-Cookie und kein Windows-Token.
Rotation der Session-ID	`Application.RefreshSessionId()` gibt der Session eine neue ID, ohne ihren Zustand zu verlieren, sodass eine ID, die vor der Anmeldung existierte, danach wertlos ist.
Authentifizierungs-Cookie des Hosts	Ein Cookie-Ticket von ASP.NET Core, ausgestellt mit `Secure` und `HttpOnly` von einem HTTPS-Endpunkt; nur nötig, wenn gewöhnliche HTTP-Endpunkte den Benutzer erkennen müssen.
Anmelde-Grant	Ein serverseitiger Eintrag mit opaker Zufalls-ID, Kontoschlüssel, Mandant, Ablaufzeit und Entzugsflag, den jede geschützte Operation prüft; ihn zu entziehen ist das, was Abmeldung bedeutet.

3. Der rote Faden der Lektion

Die Anmeldetransaktion selbst verantworten

Die AccessOps-Anmeldeansicht auf der anonymen öffentlichen Route tut in `btnSignIn_Click` genau eines: Sie liest die beiden Felder, leert das Passwortfeld und ruft `LoginService.SignIn` auf. Die Verifizierung der Anmeldedaten gehört in ein unterstütztes Identitätssystem – ASP.NET Core Identity, einen Verzeichnisdienst oder, für lokale Konten, eine Ableitung aus dem Framework wie PBKDF2 über `Rfc2898DeriveBytes.Pbkdf2` mit einem Salt pro Konto und einem Vergleich in konstanter Zeit. Ist der Benutzername unbekannt, verifiziert der Dienst trotzdem gegen Dummy-Anmeldedaten, damit „kein solcher Benutzer“ genauso viel Zeit kostet wie „falsches Passwort“.

Die Drosselung liegt im selben Dienst. Der AccessOps-`LoginThrottle` sperrt ein Konto für 15 Minuten nach fünf Fehlversuchen innerhalb eines 15-Minuten-Fensters und eine Adresse nach zwanzig, weil Credential Stuffing viele Namen von einem Ort aus probiert. Das Rate Limiting von ASP.NET Core wirkt auf Endpunkte neben `app.UseWisej()`, sieht aber keinen Wisej-Klick, der über den WebSocket der Session ankommt. Lesen Sie die Adresse aus `Application.UserHostAddress` und beachten Sie ihre Grenzen: Ein Browser kann sowohl als `::1` als auch als `127.0.0.1` erscheinen, und hinter einem Proxy kommt der

echte Client nur aus Forwarded-Headern, denen Sie vertrauen. Zähler im Prozessspeicher sind nur für einen einzelnen Server korrekt.

Jede Ablehnung – falsches Passwort, unbekannter Name, deaktiviertes Konto, Sperre – zeigt einen einzigen Satz: „That user name and password combination is not valid." Der wahre Grund geht nur in die Audit-Tabelle der Anmeldung. Das Passwort selbst geht nirgendwohin: nicht in `Application.Session`, nicht in eine Logmeldung, nicht in einen Ausnahmetext.

Den Principal erst etablieren, wenn die Richtlinie besteht

Ein verifiziertes Passwort beweist die Kenntnis eines Geheimnisses, nicht das Recht, AccessOps zu nutzen. Der Dienst löst das lokale Konto und seinen Mandanten auf, weist ein deaktiviertes Konto oder einen nicht genehmigten Mandanten mit demselben generischen Satz ab, erzeugt einen Anmelde-Grant und baut erst dann den Principal. Die Hilfsmethode `ApplicationPrincipal.Establish` aus dem Pack macht aus Konto, Mandant, Grant und Rollen Claims und weist `Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"))` zu. Dieser Principal ist Session-Zustand: Er stellt kein Cookie aus, ASP.NET-Core-Middleware sieht ihn bei anderen Anfragen nie, und er erzeugt kein Windows-Token.

Die Reihenfolge zählt, und unter Wisej.NET 4.1.7 auch die Art der Anfrage. Ein Aufruf von `Application.RefreshSessionId()` aus einem Klick – einem WebSocket-Ereignis – wechselt die ID auf dem Server, aber der Browser übernimmt sie nie; seine nächste Anfrage startet eine ganz neue Session, und die Anmeldung ist verloren. Beim Laden einer Seite aufgerufen, übernimmt der Browser die neue ID, die er im sessionStorage des Tabs hält. Deshalb verifiziert AccessOps beim Klick, legt das verifizierte Ergebnis unter einem Pending-Schlüssel ab, ruft `Application.Reload()` auf und schließt beim Laden ab:

```
// Program.Main: Application.ApplicationRefresh += (s, e) => OnPageLoad();
private static void OnPageLoad()
{
    if (Application.Session["PendingSignIn"] is not VerifiedSignIn p)
        return;

    Application.Session["PendingSignIn"] = null;
    Application.RefreshSessionId();          // zuerst die neue ID (HTTP-Laden)
    ApplicationPrincipal.Establish(p.AccountId, p.TenantId, p.GrantId, p.Roles);

    var login = Application.MainPage;
    Application.MainPage = new ConsolePage(); // ersetzt die Anmeldeansicht
    login?.Dispose();
}
```

Die Kopie von `ApplicationPrincipal` in diesem Kurs weist `Application.User` daher ohne Rotation zu; das Laden rotiert einmal, bevor die Identität existiert.

Cookies über HTTP, Abmeldung als Entzug

Ein WebSocket-Frame hat keine Antwort-Header; ein Klick kann also kein `Set-Cookie` mit `HttpOnly` liefern. `Application.Cookies` kann HttpOnly-Cookies nur während `Program.Main`, `ApplicationStart`, `ApplicationRefresh` oder bei `Application.IsWebSocket` gleich false schreiben, und `Application.Browser.CookieStorage` wird von JavaScript geschrieben, das nie HttpOnly setzen kann.

Wenn gewöhnliche HTTP-Endpunkte den Partner erkennen müssen, registriert AccessOps vor `app.UseWisej()` einen HTTPS-Endpunkt, der ein von der Session erzeugtes, einmal verwendbares, kurzlebiges Handle einlöst, `HttpContext.SignInAsync` mit dem Schema `"Cookies"` aufruft und den Cookie-Handler `Secure`, `HttpOnly` und `SameSite=Strict` setzen lässt. Dieses Ticket gehört nicht Wisej: Die Session-ID lebt im sessionStorage, und Wisej 4.1.7 schreibt ein eigenes HttpOnly-Cookie zur Validierung von Anfragen, das Sie in Ruhe lassen.

Abmeldung ist kein ausgeblendeter Button und kein gelöschtes Cookie. Der `GrantStore` hält opake, ablaufende Grants; `ExportService` prüft den Grant aus dem Claim `grant` des Principals, den Mandanten und den Datensatz, bevor das Repository läuft. Die Abmeldung entzieht zuerst den Grant, protokolliert im Audit, lässt das Host-Cookie über den Abmelde-Endpunkt ablaufen, leert die Identität der Session und ruft `Application.Exit()` auf. `Application.User` auf `null` zu setzen genügt nicht – der Getter fällt auf den Host-Principal zurück, und unter der IIS-Windows-Authentifizierung meldet ein Neuladen direkt wieder an. Ein zweiter Tab zeigt noch Export; sein nächster Klick wird bei null Repository-Lesezugriffen abgelehnt (Fixture F05). `CookieAuthenticationEvents.OnValidatePrincipal` wendet dieselbe Grant-Prüfung auf Anfragen an den Host an.

Die Cross-Site-Pfade schließen

Der Browser hängt das Host-Cookie an jede Anfrage an, auch an eine, die von einer anderen Site ausgelöst wurde; jeder zustandsändernde POST, der ihm vertraut – Anmeldung, Abmeldung, die Cookie-Übergabe –, validiert deshalb ein Antiforgery-Token. Die Wisej-Session-ID liegt im sessionStorage, und eine fremde Seite kann den Browser nicht dazu bringen, sie zu senden; der WebSocket-Handshake trägt aber trotzdem Cookies und fällt nicht unter CORS: Eine kleine Middleware vor `app.UseWisej()` weist ein Upgrade ab, dessen `Origin` kein AccessOps-Origin ist, und der Ingress wiederholt die Regel.

Fortsetzungs-URLs sind Eingaben. Eine `returnUrl` aus `Application.QueryString` oder aus der Query eines Endpunkts wird nur akzeptiert, wenn sie ein lokaler Pfad ist, der nicht mit `//` oder `/\` beginnt, oder ein Eintrag in einer ausdrücklichen Allowlist auf dem Server; alles andere wird zu `/`. Leiten Sie Schema und Host erst ab, nachdem `app.UseForwardedHeaders()` für bekannte Proxys konfiguriert ist. Fixture F11 – ein POST oder Upgrade von einem fremden Origin – muss ohne jede Zustandsänderung enden.

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:

- ✓ Eine Anmeldung implementieren, die Anmeldedaten mit einem unterstützten Identitätssystem verifiziert, Versuche drosselt, generische Fehler zurückgibt und Passwörter aus Logs und Session-Zustand heraushält.

- ✓ Den Wisej.NET-Anwendungs-Principal erst etablieren, wenn die Richtlinien für Konto und Mandant bestanden sind, die Session-ID mit Application.RefreshSessionId rotieren und die Anmeldeansicht ersetzen.
- ✓ Host-Cookies mit Secure und HttpOnly von einem eigenen HTTPS-Endpunkt statt aus einem WebSocket-Ereignis ausstellen und ablaufen lassen und die Abmeldung als Entzug eines serverseitigen Grants behandeln, den jede geschützte Operation prüft.
- ✓ Cross-Site-Pfade mit Antiforgery bei zustandsändernden POSTs, Origin-Prüfungen für den WebSocket und einer Allowlist für Fortsetzungs-URLs schließen.

Lab- und Prüfungsleitfaden

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

> **Praktisches Lab**
>
> Geben Sie der Partner-Route von AccessOps eine eigene Anmeldung: eine Anmeldeansicht auf der anonymen öffentlichen Route, die eine Fixture-Identität über einen unterstützten Verifier oder einen Trainings-Mock verifiziert, der nie ausgeliefert wird, falsche Anmeldedaten und ein deaktiviertes Konto mit derselben generischen Meldung abweist, das lokale Konto und den Mandanten auflöst, den Principal mit ApplicationPrincipal etabliert und die Session-ID rotiert. Stützen Sie sie auf einen GrantStore mit opaken, ablaufenden, widerrufbaren Anmelde-Grants, die der Export-Service vor jedem Lesezugriff prüft, ergänzen Sie einen HTTPS-Endpunkt für An- und Abmeldung für das Host-Cookie, öffnen Sie dann zwei Browserprofile, entziehen Sie den Grant bei geöffnetem zweitem Tab, wiederholen Sie den Export in beiden Tabs und nach einem Neuladen und testen Sie einen gefälschten POST, ein WebSocket-Upgrade von einem fremden Origin und eine externe Fortsetzungs-URL.

Ausgangspunkt

Öffnen Sie die AccessOps-Projektmappe aus Modul 1 mit ihren Fixtures für die Mandanten A und B, dem lesezählenden Repository und dem Gate `DocumentService`, und ergänzen Sie eine `LoginPage` auf der anonymen öffentlichen Route. Führen Sie sie unter einem HTTPS-Hostnamen (oder `https://localhost`) aus und halten Sie zwei Browserprofile für den Entzugstest bereit.

Vorgeschlagenes Vorgehen

- ✓ Schreiben Sie `LoginService.SignIn` mit `LoginThrottle`, einer Prüfung gegen Dummy-Anmeldedaten für unbekannte Namen und einem einzigen generischen Fehlersatz; halten Sie die Ergebnisse in der Audit-Tabelle fest und bestätigen Sie, dass kein Passwortwert ein Log oder `Application.Session` erreicht.
- ✓ Ergänzen Sie den `GrantStore` (opake Zufalls-ID, Konto, Mandant, Ablaufzeit, Entzugsflag) und lassen Sie `ExportService` Grant, Mandant und Datensatz prüfen, bevor das Repository läuft.
- ✓ Legen Sie die verifizierte Anmeldung beim Klick ab, rufen Sie `Application.Reload()` auf und rufen Sie in `ApplicationRefresh` erst `Application.RefreshSessionId()`, dann `ApplicationPrincipal.Establish` auf und tauschen Sie dann die Anmeldeansicht gegen die Konsole; protokollieren Sie alte und neue ID als booleschen Wert „rotiert“.
- ✓ Registrieren Sie die HTTPS-Endpunkte für Cookie-Übergabe und Abmeldung vor `app.UseWisej()` mit Antiforgery, einer `Origin`-Prüfung für WebSocket-Upgrades und einer Prüfung auf lokale Pfade für `returnUrl`; erfassen Sie die Attribute von `Set-Cookie` mit geschwärztem Wert.
- ✓ Entziehen Sie den Grant in Tab 1, wiederholen Sie Export in Tab 2 und nach einem Neuladen, führen Sie dann die Tests für gefälschten POST, Upgrade vom falschen Origin und externe Fortsetzungs-URL aus und zeichnen Sie das Übergangsdiagramm der Anmeldezustände.

Geforderte Ergebnisse

- ✓ Anmeldeansicht mit generischen Fehlern, Drosselung und Behandlung deaktivierter Konten
- ✓ Etablierung des Principals mit Rotation der Session-ID
- ✓ HTTPS-Cookie-Endpunkt mit geschwärzten Set-Cookie-Attributen
- ✓ GrantStore mit über offene Tabs hinweg durchgesetztem Entzug
- ✓ Übergangsdiagramm der Anmeldezustände und Tests für gefälschten POST, falschen Origin und Fortsetzungs-URL

Wiederholungsfragen

- ✓ Ihr erster Entwurf rief `Application.RefreshSessionId()` in `btnSignIn_Click` auf, und die Konsole erschien und verschwand beim nächsten Klick wieder. Was hat der Browser mit der neuen ID gemacht, und warum verhält sich das Laden einer Seite anders?
- ✓ Nach der Abmeldung zeigt Tab 2 noch den Export-Button und hält noch einen Principal mit Rollen-Claim. Welche einzelne Prüfung lässt seinen nächsten Export scheitern, und warum würde das bloße Löschen des Host-Cookies das nicht erreichen?
- ✓ Die Wisej-Session-ID steckt in keinem Cookie, und trotzdem braucht AccessOps Antiforgery und eine Origin-Prüfung. Welche Anfragen tragen umgebende Anmeldedaten, die eine andere Site auslösen könnte?

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.