SICHERHEITS- UND AUTHENTIFIZIERUNGSMUSTER · MODUL 1

Identitätsgrenzen in Wisej.NET

Lektionsleitfaden · Lab- und Prüfungsleitfaden

LearnWisej

Lektionsleitfaden

Die drei Identitäten einer Wisej.NET-Anfrage – Host-Principal, Anwendungs-Principal und effektives Windows-Token –, Application.User und der Lebenszyklus der Session sowie eine Ablehnung auf Ressourcenebene in der Service-Schicht, die auch den Entzug einer Rolle übersteht. 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 drei Identitäten einer Wisej.NET-Anfrage – Host-Principal, Anwendungs-Principal und effektives Windows-Token –, Application.User und der Lebenszyklus der Session sowie eine Ablehnung auf Ressourcenebene in der Service-Schicht, die auch den Entzug einer Rolle übersteht.

1. Das Problem im Produktivbetrieb

AccessOps bedient Mitarbeitende über einen Windows-Einstiegspunkt im Intranet und Partner über einen öffentlichen HTTPS-Einstiegspunkt, und beide erreichen dieselben vertraulichen Dokumente. Jede Anfrage trägt mehrere Identitäten gleichzeitig, und ein Fehler, der sie verwechselt, zeigt sich darin, dass ein Mandant den Datensatz eines anderen Mandanten liest oder ein entzogener Leser trotzdem noch exportieren kann. Dieses Modul trennt die Identitäten und verlegt die Entscheidung dorthin, wo sie sich beweisen lässt: in die Service-Schicht, bevor das Repository läuft.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Host-Principal der Anfrage	Was der Host für die Anfrage oder das WebSocket-Upgrade authentifiziert hat – `HttpContext.User` unter ASP.NET Core; anonym bei einem einfachen Kestrel-Lauf.
Anwendungs-Principal	Das Subjekt, das Ihre Anmeldung der Wisej.NET-Session zuweist, nachdem ein vertrauenswürdiger Verifier ein Konto zurückgegeben hat: Konto-ID, Mandant, Grant-ID.
effektive Windows-Identität	Das Token, das das Betriebssystem prüft, wenn der Prozess eine Datei öffnet – das Konto des Anwendungspools oder des Prozesses, sofern der Code nicht impersoniert.
Session-Override	Ein Wert, der in der aktuellen Session `Application.User` zugewiesen wurde; er hat Vorrang vor dem Principal der Anfrage, bis er wieder auf `null` gesetzt wird.
Grant	Ein widerrufbarer, ablaufender serverseitiger Eintrag, der festhält, dass ein Konto in einem Mandanten lesen darf; jede geschützte Operation liest seinen aktuellen Zustand.
Service-Schicht	Die eine Methode, die jeder Pfad – Klick, Download, Job in der Warteschlange – vor dem Repository durchläuft, sodass eine einzige Ablehnung alle abdeckt.
lesezählendes Repository	Ein Test-Repository, das seine Lesezugriffe zählt, sodass eine Ablehnung bei null Repository-Aufrufen bewiesen wird, statt aus einem leeren Bildschirm geschlossen zu werden.

3. Der rote Faden der Lektion

Drei Identitäten, und keine davon ist eine Berechtigung

Folgen Sie `CONTOSO\reader1` auf der Intranet-Route. IIS authentifiziert die Anfrage, der Host-Principal ist also eine Windows-Identität – doch wenn `DocumentService` eine Datei öffnet, prüft das Betriebssystem das Token des Anwendungspools, nicht das von reader1, sofern der Code nicht impersoniert (Modul 3). Ein föderierter Partner (Modul 5) hat einen gültigen Anwendungs-Principal und überhaupt kein Windows-Token. Der nächtliche Export hat keinen angemeldeten Benutzer, und trotzdem kann das Prozesskonto den gesamten Dokumentenordner lesen.

Halten Sie sie als drei getrennte Beobachtungen fest und lassen Sie nie eine für eine andere einspringen. Keine davon ist eine Autorisierungsentscheidung: Der Host-Principal sagt, wer sich verbunden hat, ein Rollen-Claim sagt, was bei der Anmeldung galt, das Prozess-Token sagt, was das Betriebssystem erlaubt. AccessOps muss bei jedem geschützten Lesezugriff eine engere Frage beantworten: Erlaubt der aktuelle Grant diesem Konto, diesen Datensatz in diesem Mandanten zu lesen – genau jetzt?

`Visible` und `Enabled` beschreiben den Bildschirm. Wisej.NET prüft zwar auf dem Server, dass ein Klick von einem aktivierten, sichtbaren Steuerelement kommt, aber Daten, die an eine ausgeblendete Grid-Spalte gebunden sind, erreichen trotzdem den Browser, und dasselbe Dokument ist über einen Handler, einen Download-Pfad und einen Job erreichbar. Die Regel gehört in den Service, den sie alle aufrufen.

Wie Application.User und IsAuthenticated aufgelöst werden

`Application.User` ist ein statischer `IPrincipal` mit einem zweistufigen Getter. Wurde in dieser Session ein Wert zugewiesen, gibt er diesen Session-Override zurück; andernfalls gibt er den Principal der Anfrage zurück – unter ASP.NET Core den `HttpContext.User`, der beim Eintreffen der Anfrage oder des WebSocket-Upgrades erfasst wurde: eine `WindowsIdentity` unter der IIS-Windows-Authentifizierung, ein nicht authentifizierter Principal bei einem Kestrel-Lauf auf dem Entwicklerrechner.
`Application.IsAuthenticated` ist schlicht `User?.Identity?.IsAuthenticated == true`: Es weiß nichts über Mandant, Rolle oder Besitz. `Application.User = null` entfernt den Override und fällt auf den Host-Principal zurück; wer einen Windows-Benutzer auf diese Weise „abmeldet“, lässt ihn also authentifiziert.

Weisen Sie den Override erst zu, nachdem ein vertrauenswürdiger Verifier ein Konto zurückgegeben hat; eine `ClaimsIdentity`, die aus einem eingetippten Benutzernamen gebaut wird, lässt den Browser seine Identität selbst wählen. Die Hilfsklasse `ApplicationPrincipal` des Labs baut Claims aus dem verifizierten Konto – unveränderliche Konto-ID, Mandant, Grant-ID, Rollen – unter dem Authentifizierungstyp `"Portal"`. Verwenden Sie diese ID als Schlüssel, nie einen Anzeigenamen oder eine E-Mail-Adresse.

```
// Läuft beim Laden einer Seite, nachdem der Test-Verifier ein Konto zurückgegeben hat
internal static void Establish(string accountId, string tenantId,
    string grantId, IEnumerable<string> roles)
{
    var claims = new List<Claim> {
        new Claim(ClaimTypes.NameIdentifier, accountId),
        new Claim("tenant", tenantId),
        new Claim("grant", grantId) };
    claims.AddRange(roles.Select(r => new Claim(ClaimTypes.Role, r)));

    Application.RefreshSessionId();   // zuerst die neue ID, beim Laden
    Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"));
}
```

`Application.UserIdentity` gibt es nur im Build für das .NET Framework (net48), wo es die Anmeldeidentität von System.Web ist; der net8.0-Build, den ein AccessOps unter .NET 10 referenziert, deklariert es nicht. Lesen Sie unter .NET 10 den Windows-Principal über `Application.User` und prüfen Sie, dass seine `Identity` eine `WindowsIdentity` ist, bevor Sie sich darauf verlassen.

Der Lebenszyklus der Session und Arbeit außerhalb davon

Das erste Laden über HTTP baut die Oberfläche auf, spätere Ereignisse kommen über den WebSocket, und jeder Übergang der Session ist ein eigenes Ereignis. `ApplicationStart` wird für eine neue Session nach `Program.Main` ausgelöst; `ApplicationRefresh` bei einem Neuladen im Browser oder einer URL-Änderung in derselben Session; `SessionTimeout` kündigt den Ablauf bei Inaktivität an (setzen Sie `Handled`, um den Standarddialog zur Verlängerung zu unterdrücken); `ApplicationExit` wird ausgelöst, wenn die Session beendet oder verworfen wird – das einzige Ereignis beim Ablauf. Nutzen Sie es zum Aufräumen, nie für den Entzug: Der gehört in den Grant-Speicher.

`Application.RefreshSessionId` vergibt eine neue Session-ID, ohne die Session zu verlieren, und schließt so die Session Fixation nach der Anmeldung – aber nur beim Laden einer Seite. Bei einem Klick über den WebSocket wechselt der Server die ID, der Browser übernimmt die neue nie, und seine nächste Anfrage

startet eine frische Session. Deshalb verifiziert AccessOps beim Klick, legt das Konto unter einem Pending-Schlüssel ab, ruft `Application.Reload()` auf und rotiert in `ApplicationRefresh` zuerst die ID und etabliert dann den Principal. Schwärzen Sie `Application.SessionId` in Traces.

Außerhalb der Anfrage versagt die umgebende Identität. `Application.StartTask` stellt die Session auf seinem Pool-Thread wieder her; ein einfaches `Task.Run` oder ein Timer-Callback hat keine, und `Application.Update` stellt eine nur aus einem gespeicherten `Application.Current` wieder her. Deshalb erhält der Hintergrundexport ein explizites Subjekt – Konto-ID, Mandant, Grant-ID, Datensatz-ID –, und `DocumentService` prüft den Grant erneut, wenn der Job startet. Wenn zwei Browserprofile dasselbe Konto zeigen, suchen Sie nach einem statischen Feld, das den Principal hält.

Die Ablehnung in der Service-Schicht

`DocumentService.Read(subject, tenant, recordId)` lädt den aktuellen Grant und lehnt ab, wenn er fehlt, entzogen oder abgelaufen ist oder kein Leserecht enthält, oder wenn sein Mandant vom angefragten abweicht; erst dann ruft er das Repository auf. Eine Rollenprüfung genügt nicht: Der Reader von Mandant A ist wirklich ein Reader, Datensatz 17 existiert in beiden Mandanten, und `IsInRole("Reader")` lässt den horizontalen Lesezugriff durch.

Der Beweis ist eine Zahl. In Fixture F02 fordert der Reader von Mandant A den Datensatz 17 von Mandant B an: Ablehnung, und der Lesezähler bewegt sich nicht. Entziehen Sie dann den Reader-Grant in der Admin-Konsole und klicken Sie auf Export auf dem Bildschirm, der den Button noch zeigt: Der Service liest den aktuellen Grant und lehnt ab, bevor eine Datei geöffnet wird. Jede Entscheidung schreibt ein Audit-Ereignis mit opakem Subjekt, Aktion, Klassifizierung, Entscheidung und Korrelations-ID – nie Cookies, Session-IDs oder Tokens.

Die eigenständige Konsole `PolicyHarness` (net8.0 im Pack; net10.0 funktioniert genauso) wendet die Regel ohne Wisej.NET, IIS oder Identitätsanbieter an: ein erlaubter Lesezugriff, dann Ablehnungen wegen Mandantengrenze, fehlender Rolle, unbekanntem, abgelaufenem und entzogenem Grant bei null Repository-Aufrufen. Das Harness beweist die Richtlinie überall; die Anwendung beweist, dass jeder Pfad sie verwendet.

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:

- ✓ Das Anwendungssubjekt, den Host-Principal der Anfrage und die effektive Windows-Identität einer AccessOps-Anfrage unterscheiden und erklären, warum keine davon für sich allein eine Autorisierungsentscheidung ist.

- ✓ Erklären, wie Application.User und IsAuthenticated aufgelöst werden, einschließlich des Session-Overrides und des Rückfalls auf den Host-Principal, und warum es Application.UserIdentity nur unter dem .NET Framework gibt.
- ✓ Den Lebenszyklus der Session nachverfolgen – ApplicationStart, ApplicationRefresh, SessionTimeout, ApplicationExit und Application.RefreshSessionId – und Arbeit, die außerhalb der Session läuft, ein explizites Subjekt übergeben.
- ✓ Eine Mandanten- und Datensatzprüfung in der Service-Schicht durchsetzen und beweisen, dass die Ablehnung erfolgt, bevor das Repository liest – auch nachdem die Rolle auf einem offenen Bildschirm entzogen wurde.

Lab- und Prüfungsleitfaden

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

> **Praktisches Lab**
>
> Erstellen Sie die AccessOps-Projektmappe mit den Mandanten A und B, die beide einen Datensatz 17 mit unterschiedlichem Inhalt enthalten, einem Reader-Konto in Mandant A und einem nicht zugewiesenen Konto. Bauen Sie einen DocumentService, der den aktuellen Grant, den Mandanten und den Datensatz prüft, bevor er ein Repository aufruft, das seine Lesezugriffe zählt, etablieren Sie den Session-Principal mit ApplicationPrincipal aus einem vertrauenswürdigen Test-Verifier und ergänzen Sie eine Diagnoseansicht (fern von öffentlichen Routen und vor dem Paketieren entfernt), die den Authentifizierungstyp, das Authentifiziert-Flag, ein geschwärztes Subjekt und die effektive Betriebssystem-Identität zeigt. Verfolgen Sie den Start, einen Klick, ein Neuladen im Browser und einen Hintergrundjob in zwei Browserprofilen, fordern Sie als Reader von Mandant A den Datensatz 17 von Mandant B an, entziehen Sie den Reader-Grant zwischen dem Laden der Seite und Export auf einem bereits geöffneten Bildschirm und führen Sie das eigenständige PolicyHarness aus, um dieselben Erlaubnis- und Ablehnungsentscheidungen ohne Wisej, IIS oder Identitätsanbieter zu zeigen.

Ausgangspunkt

Eine neue Wisej.NET-Webanwendung namens AccessOps auf `net10.0` mit dem Paket `Wisej-4` 4.1.7, ausgeführt unter Kestrel auf Ihrem eigenen Rechner, dazu das Konsolenprojekt `PolicyHarness` aus dem Pack in derselben Projektmappe. Dieses Modul braucht kein IIS, keine Windows-Domäne und keinen Identitätsanbieter: Jeder Schritt läuft lokal, und die „effektive Betriebssystem-Identität", die Sie festhalten, ist einfach das Konto, unter dem der `dotnet`-Prozess läuft.

Vorgeschlagenes Vorgehen

- ✓ Legen Sie zuerst die Fixtures an: Datensatz 17 in Mandant A und in Mandant B mit unterschiedlichem Inhalt, einen Reader-Grant für Mandant A und ein Konto ohne Grant sowie ein Repository, dessen `ReadCount` Sie auf dem Bildschirm anzeigen können.
- ✓ Schreiben Sie `DocumentService.Read(subject, tenant, recordId)` so, dass er den aktuellen Grant lädt und einen fehlenden, entzogenen oder abgelaufenen Grant, ein fehlendes Leserecht und einen abweichenden Mandanten ablehnt, bevor er das Repository berührt.
- ✓ Verbinden Sie die Anmeldung mit einem Trainings-Verifier, der nie ausgeliefert wird: Verifizieren Sie beim Klick, legen Sie das Konto ab, rufen Sie `Application.Reload()` auf und rufen Sie beim Laden `ApplicationPrincipal.Establish` auf, das die Session-ID rotiert, bevor es `Application.User` setzt.
- ✓ Ergänzen Sie die Diagnoseansicht hinter einem Kompilierschalter, der nur im Debug-Build gilt, und füllen Sie dann die vierspaltige Matrix für Start, Klick, Neuladen und Hintergrundjob in zwei Browserprofilen aus, einschließlich eines Jobs, der mit `Application.StartTask` gestartet wird, und eines mit `Task.Run` gestarteten, der ein explizites Subjekt erhält.
- ✓ Führen Sie die Ablehnungen aus – Datensatz 17 von Mandant B als Reader von Mandant A, dann Export nach dem Entzug des Grants auf einem offenen Bildschirm –, prüfen Sie, dass sich

`ReadCount` nicht bewegt hat, führen Sie `PolicyHarness` aus und bestätigen Sie, dass der veröffentlichte Build keine Diagnoseansicht enthält.

Geforderte Ergebnisse

- ✓ AccessOps-Projektmappe mit Fixtures für die Mandanten A und B und einem lesezählenden Repository
- ✓ Ressourcen-Gate DocumentService mit Prüfungen von Grant, Mandant und Datensatz
- ✓ Diagnoseansicht für Identitäten, aus dem paketierten Build entfernt
- ✓ Vierspaltige Identitätsmatrix für Anfrage, Session, Operation und Ergebnis
- ✓ PolicyHarness-Lauf mit einem erlaubten Lesezugriff und zwei Ablehnungen bei null Repository-Aufrufen

Wiederholungsfragen

- ✓ Ihre Diagnoseansicht zeigt für beide Browserprofile dieselbe effektive Betriebssystem-Identität, während sich die Anwendungssubjekte unterscheiden. Was sagt Ihnen das darüber, welche Identität das Dateisystem prüfen wird, und was würde sich unter IIS mit Windows-Authentifizierung ändern?
- ✓ Nach dem Entzug des Reader-Grants steht der Export-Button noch auf dem offenen Bildschirm, und `Application.User` trägt noch den Rollen-Claim Reader. Welcher einzelne Lesezugriff in Ihrem Code bewirkt die Ablehnung, und warum ist der Rollen-Claim nicht dieser Lesezugriff?
- ✓ Ihr Hintergrundexport läuft eine Stunde nach dem Klick über `Task.Run`. Was genau erhält er, was prüft er beim Start erneut, und was würde schiefgehen, wenn er stattdessen `Application.User` läse?

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.