SICHERHEITS- UND AUTHENTIFIZIERUNGSMUSTER · MODUL 3

Windows-Impersonation und Delegierung

Lektionsleitfaden · Lab- und Prüfungsleitfaden

LearnWisej

Lektionsleitfaden

Identität der Anfrage gegenüber effektivem Token, begrenzte Impersonation unter dem .NET Framework und mit WindowsIdentity.RunImpersonated unter ASP.NET Core, die Wisej.NET-Einstellung impersonate mit ihren Hooks ImpersonationBegin und ImpersonationEnd, das Zurücksetzen des Tokens und der zweite Kerberos-Hop. 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

Identität der Anfrage gegenüber effektivem Token, begrenzte Impersonation unter dem .NET Framework und mit WindowsIdentity.RunImpersonated unter ASP.NET Core, die Wisej.NET-Einstellung impersonate mit ihren Hooks ImpersonationBegin und ImpersonationEnd, das Zurücksetzen des Tokens und der zweite Kerberos-Hop.

1. Das Problem im Produktivbetrieb

AccessOps legt vertrauliche Dokumente auf einem Windows-Dateispeicher ab, dessen ACLs für Menschen geschrieben wurden, nicht für den Webserver. Die Intranet-Route kennt die Mitarbeiterin oder den Mitarbeiter bereits, also ist es verlockend, überall zu impersonieren und Windows entscheiden zu lassen. Das macht aus einem einzigen Dateizugriff einen Identitätswechsel für den ganzen Prozess und scheitert auf verwirrende Weise, sobald die Datei auf einem zweiten Server liegt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Identität der Anfrage	Die `WindowsIdentity`, die die IIS-Windows-Authentifizierung an die HTTP-Anfrage angehängt hat; sie sagt, wer aufgerufen hat, öffnet für sich allein aber nichts.
effektives Token	Das Token, das der Thread vorlegt, wenn Windows ein Objekt öffnet, gelesen mit `WindowsIdentity.GetCurrent()`; normalerweise die Pool-Identität, der Benutzer nur innerhalb eines Impersonation-Bereichs.
begrenzte Impersonation	Eine einzelne, abgegrenzte Operation unter dem Token des Benutzers ausführen und am Ende zurücksetzen, auch wenn sie eine Ausnahme wirft.
Einstellung `impersonate`	Der Schalter in Default.json hinter `Configuration.Impersonate` (Standard `false`), der Wisej.NET seine Impersonation-Ereignisse rund um Anfrage-Threads und Tasks aus `Application.StartTask` auslösen lässt.
`ImpersonationBegin` / `ImpersonationEnd`	Prozessweite Ereignisse vom Typ `HandledEventHandler`; unter dem .NET Framework impersoniert die Standardbehandlung und setzt später zurück, und das Setzen von `Handled` unterdrückt dieses Standardverhalten.
ACL-Nachweis	Der Beleg, dass der Zugriff aus den Gruppen des Tokens gegen eine echte ACL entstand: eine Datei, die nur der Pool lesen kann, eine, die nur ein benannter Lab-Benutzer lesen kann.
zweiter Hop	Der Webserver verwendet die Identität des Benutzers gegenüber einem anderen Rechner weiter; das braucht Kerberos-Delegierung an ein erlaubtes Ziel und eine ACL im Backend, die dem Benutzer Zugriff gewährt.

3. Der rote Faden der Lektion

Wer aufruft, ist nicht, wer die Datei öffnet

Eine per Windows authentifizierte Anfrage trägt zwei Fakten, die sich leicht vermischen. Die Identität der Anfrage ist das, was der Host authentifiziert hat: Unter ASP.NET Core erreicht sie die Wisej.NET-Session als `Application.User`, dessen `Identity` unter der IIS-Windows-Authentifizierung eine `WindowsIdentity` ist. Das effektive Token ist das, was der ausführende Thread dem Betriebssystem vorlegt, und bei einer normalen Anfrage ist das weiterhin die Identität des Anwendungspools. Windows fragt beim Öffnen einer Datei nie nach einem Namen; es vergleicht die Benutzer-SID und die Gruppen-SIDs des Tokens mit der ACL der Datei. Das AccessOps-Anwendungssubjekt aus Modul 1 ist ein dritter, eigener Fakt.

Deshalb beweist das Lab den Zugriff mit ACLs, nicht mit Namen: ein Ordner, den nur der Anwendungspool lesen kann, einer, den nur ein benannter Lab-Benutzer der Domäne lesen kann. Vor der Impersonation muss der Lesezugriff auf den Benutzerordner scheitern und der auf den Pool-Ordner gelingen; innerhalb des Bereichs kehren sich die Ergebnisse um. Lesen beide Identitäten beide Dateien, prüfen Sie geerbte ACLs und die Mitgliedschaft in der lokalen Administratorengruppe. Halten Sie `WindowsIdentity.GetCurrent()` mit `Name`, `User`-SID und `ImpersonationLevel` fest und geben Sie die zurückgelieferte Identität mit Dispose frei.

.NET Framework, die Einstellung impersonate und die Thread-Hooks

Unter dem .NET Framework klingen vier Schalter ähnlich und tun Verschiedenes: der IIS-Rollendienst Windows-Authentifizierung, das System.Web-Element `<authentication mode="Windows" />`, das System.Web-Element `<identity impersonate>` und der Wisej.NET-Schlüssel `impersonate` in Default.json. Das Lab lässt die System.Web-Impersonation ausgeschaltet und verwendet `using (identity.Impersonate()) { … }` um einen einzigen synchronen Dateizugriff, mit der Identität aus `Application.UserIdentity` (der Anmeldeidentität der Anfrage, die es nur im Build für das .NET Framework gibt).

Mit `"impersonate": true` – einmal gelesen und für den ganzen Prozess zwischengespeichert – löst Wisej.NET zu Beginn jedes Anfrage-Threads und jedes mit `Application.StartTask` gestarteten Tasks `ImpersonationBegin` aus und an dessen Ende `ImpersonationEnd`. Unter dem .NET Framework ruft Begin `UserIdentity.Impersonate()` auf und End gibt diesen Kontext frei, sofern kein Handler `e.Handled = true` setzt. Die Ereignisse werden rund um den Anfrage-Thread ausgelöst: `ImpersonationBegin` vor `ThreadBegin` und `ImpersonationEnd` vor `ThreadEnd`. (Ältere Dokumentation sagt das Gegenteil.) Es sind statische, prozessweite Ereignisse: Hängen Sie sie einmal beim Anwendungsstart an, und wenn Sie sie nur beobachten, lassen Sie `Handled` auf false, damit das Framework aufräumt, was es begonnen hat. Ein einfaches `Task.Run` bekommt nichts davon.

Das Lab vergleicht die Einstellung aus und an für einen Klick und den Rumpf eines `Application.StartTask`, mit aktiviertem WebSocket und im reinen HTTP-Modus, jeweils in einer frischen Session.

Ein abgegrenzter RunImpersonated-Lesezugriff unter ASP.NET Core

Unter .NET 8 und .NET 10 werden die beiden Ereignisse weiterhin ausgelöst, wenn `impersonate` true ist, aber Wisej.NET tut standardmäßig nichts: Modernes .NET hat kein `WindowsIdentity.Impersonate()` und keinen Impersonation-Kontext, der zwischen Begin und End offen bleiben könnte. Das unterstützte Muster ist ein Bereich, der sein eigenes Zurücksetzen besitzt. AccessOps kapselt es in einem `ScopedDocumentReader`, der eine Dokument-ID erhält, nie einen Pfad:

```
[SupportedOSPlatform("windows")]
public byte[] Read(HttpContext context, int documentId)
{
    if (context.User.Identity is not WindowsIdentity identity || !identity.IsAuthenticated)
        throw new UnauthorizedAccessException();
    // zuerst die Richtlinie: Grant, Mandant und Datensatz, dann ein kanonischer Pfad unter
dem Stammordner
    string path = _catalog.ResolveApprovedPath(documentId);

    return WindowsIdentity.RunImpersonated(identity.AccessToken, () ⇒
    {
        using (var current = WindowsIdentity.GetCurrent()) _trace.Record("inside", current);
        return File.ReadAllBytes(path);
    });
}
```

Der Browser wählt nie einen Pfad; der Katalog ordnet die ID erst dann einer Datei unter einem einzigen freigegebenen Stammordner zu, wenn die Entscheidung der Anwendung den Lesezugriff erlaubt. `RunImpersonated` stellt das vorherige Token wieder her, wenn der Delegat zurückkehrt oder eine Ausnahme wirft – genau das beweist Fixture F03: Fordern Sie ein freigegebenes Dokument an, dessen Datei fehlt, lassen Sie die `FileNotFoundException` entweichen, und die Zeile „danach“ muss wieder die Pool-Identität zeigen. Der Fall der verweigerten ACL verwendet eine existierende Datei, damit sich „nicht gefunden“ nie als „nicht erlaubt“ tarnt. Halten Sie den Bereich synchron; verwenden Sie `RunImpersonatedAsync` nur, wenn die erwartete Arbeit innerhalb der Anfrage abgeschlossen wird, und speichern Sie das Token nie im Session-Zustand und übergeben Sie es nie an langlebige Arbeit.

Führen Sie den Reader dort aus, wo das Token der Anfrage lebt: in einem autorisierten Endpunkt, der vor `app.UseWisej()` registriert ist, oder in Code, der den aktuellen `HttpContext` über `IHttpContextAccessor` erreicht. Ein Wisej-Ereignis über den WebSocket sieht den Principal, der beim Öffnen des Sockets erfasst wurde, und die Lebensdauer dieses Tokens gehört dem Host; das Lab hält diesen Fall deshalb fest, statt sich darauf zu verlassen.

Den zweiten Hop einordnen

Ein Lesezugriff, der in einem lokalen NTFS-Ordner funktioniert und gegen `\\files01\confidential` oder eine integrierte SQL-Anmeldung scheitert, ist meist kein Impersonation-Fehler. Stellen Sie drei Fragen. Erster Hop: Hat der Browser IIS überhaupt mit Kerberos erreicht? Ein Negotiate-Header allein beweist das nicht; `klist` auf dem Client sollte ein Ticket für den HTTP-SPN der Site zeigen, und NTLM lässt sich nicht delegieren. Delegierung: Darf das Konto des Anwendungspools per eingeschränkter Delegierung an genau diesen Ziel-SPN delegieren, und ist das Benutzerkonto nicht als sensibel markiert? Autorisierung im Backend: Gewährt die Freigabe-, NTFS- oder SQL-Berechtigung diesem Benutzer tatsächlich Zugriff? Fixture F04 ist genau dieser Fall; ändern Sie den Code erst, wenn alle drei Antworten vorliegen.

Dann entscheiden Sie. Delegierter Benutzerzugriff lässt die ACL im Backend maßgeblich bleiben, braucht aber eingeschränkte Delegierung, konfiguriert für jedes Ziel, nie uneingeschränkte. Eine eng berechtigte Dienstidentität verlegt die Entscheidung in AccessOps: Der Service prüft Grant, Mandant und Datensatz, und ein Audit-Eintrag nennt die Person, in deren Auftrag er gelesen hat. Der Entscheidungseintrag hält fest, welchen Weg AccessOps nimmt und warum.

Seien Sie ehrlich, woher jeder Nachweis stammt. Impersonierte Lesezugriffe, ACL-Ergebnisse, der Vergleich unter dem .NET Framework und der zweite Hop brauchen IIS mit Windows-Authentifizierung und Domänenkonten: Sie werden nur in der Lab-Umgebung geprüft. Auf einem Entwicklerrechner beweist das PolicyHarness, was zuerst kommt: die Entscheidung der Anwendung, die den Lesezugriff erlauben muss, bevor sich ein Impersonation-Bereich öffnet.

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 Unterschied zwischen der an eine Anfrage angehängten Windows-Identität und dem bei einer Dateioperation verwendeten effektiven Token erklären und den Zugriff mit einer ACL statt mit einem Benutzernamen beweisen.
- ✓ Begrenzte Impersonation unter dem .NET Framework konfigurieren und die Wisej.NET-Einstellung impersonate, ihre Ereignisse ImpersonationBegin und ImpersonationEnd sowie Application.StartTask über HTTP- und WebSocket-Ereignisse hinweg vergleichen.
- ✓ Einen abgegrenzten Lesezugriff mit WindowsIdentity.RunImpersonated unter ASP.NET Core mit einem vom Server gewählten kanonischen Pfad implementieren und prüfen, dass das Token nach einer Ausnahme zurückgesetzt wird.
- ✓ Einen Fehler beim zweiten Hop einordnen – SPN, Ziel der eingeschränkten Delegierung oder ACL im Backend – und zwischen delegiertem Benutzerzugriff und einer eingeschränkten Dienstidentität mit ausdrücklichem Audit entscheiden.

Lab- und Prüfungsleitfaden

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

Praktisches Lab

Ergänzen Sie AccessOps um einen Reader für vertrauliche Dokumente mit zwei Ordnern, einem, den nur der Anwendungspool lesen kann, und einem, den nur ein benannter Lab-Benutzer der Domäne lesen kann, sowie um einen ScopedDocumentReader auf Basis der Hilfsklasse WindowsScopedRead (ASP.NET Core) oder Net48ScopedRead (.NET Framework), der eine über eine serverseitige ID gewählte Datei öffnet. Halten Sie das Anwendungssubjekt, die Identität der Anfrage und die effektive Identität vor, während und nach dem begrenzten Lesezugriff fest, erzwingen Sie eine Ausnahme bei einer fehlenden freigegebenen Fixture und zeigen Sie, dass die normale Identität zurückkehrt, vergleichen Sie die Wisej.NET-Einstellung impersonate aus und an für einen Klick und ein Application.StartTask im HTTP- und im WebSocket-Modus unter dem .NET Framework, und ordnen Sie, wo ein zweiter Server existiert, einen Fehler bei entferntem SMB oder integriertem SQL nach erstem Hop, Delegierung und Autorisierung im Backend ein. Impersonierte Lesezugriffe, ACL-Ergebnisse und der zweite Hop werden nur in der Lab-Umgebung mit IIS-Windows-Authentifizierung und Domänenkonten geprüft; auf einem Entwicklerrechner beweist das PolicyHarness die Entscheidung der Anwendung, die den Lesezugriff erlauben muss, bevor sich ein Impersonation-Bereich öffnet.

Ausgangspunkt

Öffnen Sie AccessOps in dem Zustand, in dem Modul 2 es hinterlassen hat: die Intranet-Site unter IIS mit Windows-Authentifizierung, die Ressourcenrichtlinie aus Modul 1 hinter den Dokument-Services und ein grünes PolicyHarness. Legen Sie in der Lab-Umgebung die beiden Fixture-Ordner unter einem freigegebenen Stammordner an und gewähren Sie dem Pool-Konto und dem Lab-Benutzer der Domäne ihre getrennten Leserechte; auf einem Entwicklerrechner arbeiten Sie im PolicyHarness und im Code des Readers, bis Sie diese Umgebung erreichen.

Vorgeschlagenes Vorgehen

- ✓ Ergänzen Sie einen Dokumentkatalog, der serverseitige Dokument-IDs kanonischen Pfaden unter dem freigegebenen Stammordner zuordnet, und lassen Sie `ScopedDocumentReader` die Ressourcenrichtlinie aufrufen, bevor er irgendeinen Pfad auflöst.
- ✓ Kapseln Sie den Lesezugriff in `WindowsIdentity.RunImpersonated` (oder `using (identity.Impersonate())` unter dem .NET Framework) und halten Sie `WindowsIdentity.GetCurrent()` vorher, währenddessen und danach fest, neben dem Anwendungssubjekt und der Identität der Anfrage.
- ✓ Lesen Sie die Pool-Fixture und die Benutzer-Fixture außerhalb und innerhalb des Bereichs, fordern Sie dann ein freigegebenes Dokument an, dessen Datei fehlt, und bestätigen Sie, dass die Zeile „danach“ wieder die Pool-Identität zeigt (F03).
- ✓ Setzen Sie in der .NET-Framework-Sandbox `"impersonate": true`, hängen Sie einmal beim Start protokollierende Handler an `ImpersonationBegin` und `ImpersonationEnd` und wiederholen Sie die Beobachtungen für einen Klick und ein `Application.StartTask` mit WebSocket an und aus.

- ✓ Wo ein Dateiserver oder SQL Server verfügbar ist, reproduzieren Sie F04 und halten Sie ersten Hop, Delegierungsziel und ACL im Backend getrennt fest, bevor Sie die Migrationsentscheidung schreiben.

Geforderte Ergebnisse

- ✓ ScopedDocumentReader mit einem vom Server aufgelösten kanonischen Pfad
- ✓ Matrix aus Identität und Ergebnis vorher, währenddessen und danach
- ✓ Trace des Zurücksetzens nach erzwungenem Fehler
- ✓ Vergleich der Einstellung impersonate und von StartTask über HTTP und WebSocket
- ✓ Einordnung des zweiten Hops und Migrationsentscheidung für die entfernte Ressource

Wiederholungsfragen

- ✓ Ihr Trace zeigt innerhalb des Bereichs den Namen des Lab-Benutzers, und trotzdem öffnen sich beide Fixture-Dateien in jeder Zeile. Was sagt Ihnen das über die Nachweise, und was prüfen Sie zuerst?
- ✓ Unter .NET 10 protokolliert der Handler für `ImpersonationBegin` bei jedem Klick. Warum läuft der Dateizugriff trotzdem als Pool, und was müsste sich ändern, damit es nicht so ist?
- ✓ Die entfernte Freigabe verweigert den Zugriff nur über AccessOps, während derselbe Benutzer sie von seinem Desktop aus öffnet. Welche der drei Prüfungen für den zweiten Hop schließt das aus, und welche zwei bleiben?

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.