MODÈLES DE SÉCURITÉ ET D'AUTHENTIFICATION · MODULE 3

# Emprunt d'identité Windows et délégation

Guide de la leçon · Guide du lab / de l'examen

LearnWisej

## Guide de la leçon

Identité de la requête contre jeton effectif, emprunt d'identité délimité sur .NET Framework et avec WindowsIdentity.RunImpersonated sur ASP.NET Core, le paramètre impersonate de Wisej.NET et ses points d'accroche ImpersonationBegin et ImpersonationEnd, le rétablissement du jeton, et le second saut Kerberos. Ce que cela signifie pour un développeur qui sécurise les chemins de connexion, d'identité et d'autorisation d'une application Wisej.NET en production — et comment aborder ce module.

**Objectif en production**

Identité de la requête contre jeton effectif, emprunt d'identité délimité sur .NET Framework et avec WindowsIdentity.RunImpersonated sur ASP.NET Core, le paramètre impersonate de Wisej.NET et ses points d'accroche ImpersonationBegin et ImpersonationEnd, le rétablissement du jeton, et le second saut Kerberos.

## 1. Le problème en production

AccessOps conserve les documents confidentiels sur un stockage de fichiers Windows dont les ACL ont été écrites pour des personnes, pas pour le serveur web. La route intranet connaît déjà le membre du personnel ; il est donc tentant d'emprunter son identité partout et de laisser Windows décider. Cela transforme une simple lecture de fichier en changement d'identité à l'échelle du processus, et échoue de façon déroutante dès que le fichier se trouve sur un second serveur.

## 2. Concepts clés

| Concept | Pourquoi c'est important |
|---|---|
| identité de la requête | La `WindowsIdentity` que l'authentification Windows d'IIS a attachée à la requête HTTP ; elle dit qui a appelé, mais à elle seule elle n'ouvre rien. |
| jeton effectif | Le jeton que le thread présente quand Windows ouvre un objet, lu avec `WindowsIdentity.GetCurrent()` ; normalement l'identité du pool ; l'utilisateur seulement à l'intérieur d'une portée d'emprunt d'identité. |
| emprunt d'identité délimité | Exécuter une seule opération bornée sous le jeton de l'utilisateur et revenir en arrière à la fin, même quand elle lève une exception. |
| paramètre `impersonate` | Le commutateur de Default.json derrière `Configuration.Impersonate` (par défaut `false`) qui fait lever par Wisej.NET ses événements d'emprunt d'identité autour des threads de requête et des tâches `Application.StartTask`. |
| `ImpersonationBegin` / `ImpersonationEnd` | Des événements `HandledEventHandler` à l'échelle du processus ; sur .NET Framework, le traitement par défaut emprunte l'identité puis la rétablit, et définir `Handled` supprime ce comportement par défaut. |
| preuve par ACL | La preuve que l'accès provient des groupes du jeton face à une vraie ACL : un fichier que seul le pool peut lire, un autre que seul un utilisateur de lab nommé peut lire. |
| second saut | Le serveur web qui réutilise l'identité de l'utilisateur auprès d'une autre machine, ce qui exige une délégation Kerberos vers une cible autorisée et une ACL du backend qui accorde l'accès à l'utilisateur. |

## 3. Le fil de la leçon

### Qui appelle n'est pas qui ouvre le fichier

Une requête authentifiée par Windows porte deux faits faciles à confondre. L'identité de la requête est ce que l'hôte a authentifié : sur ASP.NET Core, elle atteint la session Wisej.NET sous la forme d'`Application.User`, dont l'`Identity` est une `WindowsIdentity` sous l'authentification Windows d'IIS. Le jeton effectif est ce que le thread en cours présente au système d'exploitation, et lors d'une requête normale, c'est toujours l'identité du pool d'applications. Windows ne demande jamais de nom quand il ouvre un fichier ; il compare le SID utilisateur et les SID de groupe du jeton avec l'ACL du fichier. Le sujet applicatif d'AccessOps du Module 1 est un troisième fait, distinct.

C'est pourquoi le lab prouve l'accès par des ACL, pas par des noms : un dossier que seul le pool d'applications peut lire, un autre que seul un utilisateur de lab du domaine nommé peut lire. Avant l'emprunt d'identité, la lecture réservée à l'utilisateur doit échouer et la lecture réservée au pool réussir ; à l'intérieur de la portée, les résultats s'inversent. Si les deux identités lisent les deux fichiers, vérifiez les ACL héritées et l'appartenance au groupe des administrateurs locaux. Consignez `WindowsIdentity.GetCurrent()` avec son `Name`, son SID `User` et son `ImpersonationLevel`, et libérez l'identité qu'il renvoie.

### .NET Framework, le paramètre impersonate et les points d'accroche des threads

Sur .NET Framework, quatre commutateurs se ressemblent et font des choses différentes : le service de rôle Authentification Windows d'IIS, le `<authentication mode="Windows" />` de System.Web, le `<identity impersonate>` de System.Web, et la clé `impersonate` de Wisej.NET dans Default.json. Le lab laisse désactivé l'emprunt d'identité de System.Web et utilise `using (identity.Impersonate()) { … }` autour d'une seule lecture de fichier synchrone, avec l'identité tirée d'`Application.UserIdentity` (l'identité de connexion de la requête, qui n'existe que dans le build .NET Framework).

Avec `"impersonate": true`, lu une seule fois puis mis en cache pour tout le processus, Wisej.NET lève `ImpersonationBegin` au début de chaque thread de requête et de chaque tâche lancée avec `Application.StartTask`, et `ImpersonationEnd` à leur fin. Sur .NET Framework, sauf si un gestionnaire définit `e.Handled = true`, Begin appelle `UserIdentity.Impersonate()` et End libère ce contexte. Les événements sont levés autour du thread de requête : `ImpersonationBegin` se déclenche avant `ThreadBegin`, et `ImpersonationEnd` avant `ThreadEnd`. (Une documentation plus ancienne affirme le contraire.) Ce sont des événements statiques, à l'échelle du processus : attachez-les une seule fois au démarrage de l'application et, si vous ne faites que les observer, laissez `Handled` à false pour que le framework nettoie ce qu'il a commencé. Un `Task.Run` brut ne bénéficie de rien de tout cela.

Le lab compare le paramètre désactivé et activé pour un clic et pour le corps d'un `Application.StartTask`, avec WebSocket activé et en mode HTTP seul, chaque fois dans une nouvelle session.

### Une lecture RunImpersonated bornée sur ASP.NET Core

Sur .NET 8 et .NET 10, les deux événements se déclenchent encore quand `impersonate` vaut true, mais Wisej.NET ne fait rien par défaut : le .NET moderne n'a ni `WindowsIdentity.Impersonate()` ni contexte d'emprunt d'identité à garder ouvert entre un Begin et un End. Le modèle pris en charge est une portée qui gère elle-même son rétablissement. AccessOps l'enveloppe dans un `ScopedDocumentReader` qui reçoit un ID de document, jamais un chemin :

```
[SupportedOSPlatform("windows")]
public byte[] Read(HttpContext context, int documentId)
{
    if (context.User.Identity is not WindowsIdentity identity || !identity.IsAuthenticated)
        throw new UnauthorizedAccessException();
    // policy first: grant, tenant and record, then a canonical path under the root
    string path = _catalog.ResolveApprovedPath(documentId);

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

Le navigateur ne choisit jamais de chemin ; le catalogue fait correspondre l'ID à un fichier sous une racine approuvée unique, et seulement après que la décision applicative a autorisé la lecture. `RunImpersonated` rétablit le jeton précédent quand le délégué se termine ou lève une exception, et c'est ce que prouve la

fixture F03 : demandez un document approuvé dont le fichier est manquant, laissez s'échapper la `FileNotFoundException`, et la ligne « après » doit de nouveau montrer l'identité du pool. Le cas d'ACL refusée utilise un fichier qui existe, afin que « introuvable » ne se fasse jamais passer pour « non autorisé ». Gardez la portée synchrone ; n'utilisez `RunImpersonatedAsync` que lorsque le travail attendu se termine à l'intérieur de la requête, et ne stockez jamais le jeton dans l'état de session ni ne le confiez à un travail de longue durée.

Exécutez le lecteur là où le jeton de la requête est vivant : un endpoint autorisé mappé avant `app.UseWisej()`, ou du code qui atteint le `HttpContext` courant via `IHttpContextAccessor`. Un événement Wisej sur WebSocket voit le principal capturé à l'ouverture du socket, et c'est l'hôte qui gère la durée de vie de ce jeton ; le lab consigne donc ce cas plutôt que de s'y fier.

### Classer le second saut

Une lecture qui fonctionne sur un dossier NTFS local et échoue contre `\\files01\confidential` ou une connexion SQL intégrée n'est généralement pas un bogue d'emprunt d'identité. Posez trois questions. Premier saut : le navigateur a-t-il seulement atteint IIS avec Kerberos ? Un en-tête Negotiate seul ne le prouve pas ; `klist` sur le client doit montrer un ticket pour le SPN HTTP du site, et NTLM ne peut pas être délégué. Délégation : le compte du pool d'applications est-il autorisé à déléguer, par délégation contrainte, vers ce SPN cible précis, et le compte utilisateur n'est-il pas marqué comme sensible ? Autorisation du backend : la permission du partage, NTFS ou SQL accorde-t-elle réellement l'accès à cet utilisateur ? La fixture F04 est ce cas-là ; ne modifiez le code qu'une fois les trois réponses obtenues.

Décidez ensuite. L'accès utilisateur délégué garde l'ACL du backend comme autorité, mais exige une délégation contrainte configurée pour chaque cible, jamais une délégation non contrainte. Une identité de service aux privilèges restreints déplace la décision dans AccessOps : le service vérifie l'octroi, le tenant et l'enregistrement, et une entrée d'audit nomme la personne pour le compte de laquelle il a lu. Le relevé de décision indique laquelle des deux AccessOps utilise, et pourquoi.

Soyez honnête sur l'origine de chaque preuve. Les lectures sous emprunt d'identité, les résultats d'ACL, la comparaison .NET Framework et le second saut exigent IIS avec l'authentification Windows et des comptes de domaine : ils ne sont vérifiés que dans l'environnement de lab. Sur une machine de développement, le PolicyHarness prouve ce qui vient en premier : la décision applicative qui doit autoriser la lecture avant qu'aucune portée d'emprunt d'identité ne s'ouvre.

**Réflexe de sécurité**

Gardez les identités séparées et décidez sur le serveur. Le principal de l'hôte, le principal applicatif Wisej.NET et le jeton Windows effectif sont trois faits différents, et aucun n'est une permission. Chaque lecture protégée vérifie l'octroi actuel, le tenant et l'enregistrement à la couche service avant que le repository ne s'exécute, et un bouton masqué, un claim de rôle ou un onglet déconnecté ne prouvent rien tant que cette vérification n'a pas refusé.

**Objectifs d'apprentissage**

À la fin de ce module, vous saurez :

- ✓ Expliquer la différence entre l'identité Windows attachée à une requête et le jeton effectif utilisé lors d'une opération sur un fichier, et prouver l'accès par une ACL plutôt que par un nom d'utilisateur.
- ✓ Configurer un emprunt d'identité délimité sur .NET Framework et comparer le paramètre impersonate de Wisej.NET, ses événements ImpersonationBegin et ImpersonationEnd et Application.StartTask entre événements HTTP et WebSocket.
- ✓ Implémenter une lecture WindowsIdentity.RunImpersonated bornée sur ASP.NET Core avec un chemin canonique choisi par le serveur, et vérifier que le jeton est rétabli après une exception.
- ✓ Classer un échec au second saut — SPN, cible de délégation contrainte ou ACL du backend — et choisir entre un accès utilisateur délégué et une identité de service restreinte avec un audit explicite.

## Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.

**Lab pratique**

Ajoutez à AccessOps un lecteur de documents confidentiels avec deux dossiers, l'un lisible seulement par le pool d'applications et l'autre seulement par un utilisateur de lab du domaine nommé, ainsi qu'un ScopedDocumentReader construit à partir de l'utilitaire WindowsScopedRead (ASP.NET Core) ou Net48ScopedRead (.NET Framework), qui ouvre un fichier choisi par un ID côté serveur. Consignez le sujet applicatif, l'identité de la requête et l'identité effective avant, pendant et après la lecture délimitée, provoquez une exception sur une fixture approuvée manquante et montrez que l'identité ordinaire reprend, comparez le paramètre impersonate de Wisej.NET désactivé et activé pour un clic et un Application.StartTask en modes HTTP et WebSocket sur .NET Framework, et, si un second serveur existe, classez un échec SMB distant ou SQL intégré selon le premier saut, la délégation et l'autorisation du backend. Les lectures sous emprunt d'identité, les résultats d'ACL et le second saut ne sont vérifiés que dans l'environnement de lab avec l'authentification Windows d'IIS et des comptes de domaine ; sur une machine de développement, le PolicyHarness prouve la décision applicative qui doit autoriser la lecture avant qu'aucune portée d'emprunt d'identité ne s'ouvre.

### Point de départ

Ouvrez AccessOps dans l'état où l'a laissé le Module 2 : le site intranet sur IIS avec l'authentification Windows, la politique de ressources du Module 1 derrière les services de documents, et le PolicyHarness au vert. Dans l'environnement de lab, créez les deux dossiers de fixtures sous une racine approuvée unique et accordez au compte du pool et à l'utilisateur de lab du domaine leurs droits de lecture distincts ; sur une machine de développement, travaillez dans le PolicyHarness et dans le code du lecteur jusqu'à ce que vous disposiez de cet environnement.

### Approche suggérée

- ✓ Ajoutez un catalogue de documents qui fait correspondre des ID de document côté serveur à des chemins canoniques sous la racine approuvée, et faites en sorte que `ScopedDocumentReader` appelle la politique de ressources avant de résoudre le moindre chemin.
- ✓ Enveloppez la lecture dans `WindowsIdentity.RunImpersonated` (ou `using (identity.Impersonate())` sur .NET Framework) et consignez `WindowsIdentity.GetCurrent()` avant, pendant et après, à côté du sujet applicatif et de l'identité de la requête.
- ✓ Lisez la fixture réservée au pool et celle réservée à l'utilisateur hors de la portée puis à l'intérieur, demandez ensuite un document approuvé dont le fichier est manquant et confirmez que la ligne « après » montre de nouveau l'identité du pool (F03).
- ✓ Dans le bac à sable .NET Framework, définissez `"impersonate": true`, attachez une seule fois au démarrage des gestionnaires de journalisation à `ImpersonationBegin` et `ImpersonationEnd`, et répétez les observations pour un clic et un `Application.StartTask`, avec WebSocket activé puis désactivé.

- ✓ Si un serveur de fichiers ou un SQL Server est disponible, reproduisez F04 et consignez séparément le premier saut, la cible de délégation et l'ACL du backend avant de rédiger la décision de migration.

## Livrables attendus

- ✓ ScopedDocumentReader avec un chemin canonique résolu par le serveur
- ✓ Matrice des identités et des résultats avant, pendant et après
- ✓ Trace de rétablissement après échec provoqué
- ✓ Comparaison du paramètre impersonate et de StartTask entre HTTP et WebSocket
- ✓ Classement du second saut et décision de migration pour la ressource distante

## Questions de révision

- ✓ Votre trace montre le nom de l'utilisateur de lab à l'intérieur de la portée, et pourtant les deux fichiers de fixtures s'ouvrent dans chaque ligne. Qu'est-ce que cela vous apprend sur la preuve, et que vérifiez-vous en premier ?
- ✓ Sur .NET 10, le gestionnaire `ImpersonationBegin` écrit dans le journal à chaque clic. Pourquoi la lecture du fichier s'exécute-t-elle toujours sous le pool, et que faudrait-il changer pour qu'il en soit autrement ?
- ✓ Le partage distant ne refuse l'accès qu'à travers AccessOps, alors que le même utilisateur l'ouvre depuis son poste. Laquelle des trois vérifications du second saut cela écarte-t-il, et quelles deux restent ?

**Réflexe de sécurité**

Gardez les identités séparées et décidez sur le serveur. Le principal de l'hôte, le principal applicatif Wisej.NET et le jeton Windows effectif sont trois faits différents, et aucun n'est une permission. Chaque lecture protégée vérifie l'octroi actuel, le tenant et l'enregistrement à la couche service avant que le repository ne s'exécute, et un bouton masqué, un claim de rôle ou un onglet déconnecté ne prouvent rien tant que cette vérification n'a pas refusé.