MODÈLES DE SÉCURITÉ ET D'AUTHENTIFICATION · MODULE 1

Frontières d'identité dans Wisej.NET

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

LearnWisej

Guide de la leçon

Les trois identités d'une requête Wisej.NET — principal de l'hôte, principal applicatif et jeton Windows effectif —, Application.User et le cycle de vie de la session, et un refus au niveau de la ressource, appliqué à la couche service, qui résiste à la révocation d'un rôle. 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

Les trois identités d'une requête Wisej.NET — principal de l'hôte, principal applicatif et jeton Windows effectif —, Application.User et le cycle de vie de la session, et un refus au niveau de la ressource, appliqué à la couche service, qui résiste à la révocation d'un rôle.

1. Le problème en production

AccessOps sert le personnel par un point d'entrée Windows intranet et les partenaires par un point d'entrée HTTPS public, et tous deux atteignent les mêmes documents confidentiels. Chaque requête porte plusieurs identités à la fois, et un bogue qui les confond se manifeste par un tenant qui lit l'enregistrement d'un autre tenant, ou par un lecteur révoqué qui peut encore exporter. Ce module sépare ces identités et place la décision là où elle peut être prouvée : à la couche service, avant que le repository ne s'exécute.

2. Concepts clés

Concept	Pourquoi c'est important
principal de requête de l'hôte	Ce que l'hôte a authentifié pour la requête ou pour l'upgrade WebSocket — `HttpContext.User` sur ASP.NET Core ; anonyme lors d'une simple exécution sous Kestrel.
principal applicatif	Le sujet que votre connexion affecte à la session Wisej.NET après qu'un vérificateur de confiance a renvoyé un compte : ID de compte, tenant, ID d'octroi.
identité Windows effective	Le jeton que le système d'exploitation vérifie quand le processus ouvre un fichier — le compte du pool d'applications ou du processus, sauf si le code emprunte une identité.
substitution de session	Une valeur affectée à `Application.User` dans la session courante ; elle l'emporte sur le principal de requête jusqu'à ce qu'elle soit remise à `null`.
octroi	Un enregistrement côté serveur, révocable et à durée limitée, indiquant qu'un compte peut lire dans un tenant donné ; chaque opération protégée lit son état actuel.
couche service	La méthode unique que tout chemin — clic, téléchargement, tâche en file d'attente — traverse avant le repository, si bien qu'un seul refus les couvre tous.
repository compteur de lectures	Un repository de test qui compte ses lectures, de sorte qu'un refus est prouvé à zéro appel au repository plutôt que déduit d'un écran vide.

3. Le fil de la leçon

Trois identités, et aucune n'est une permission

Suivez `CONTOSO\reader1` sur la route intranet. IIS authentifie la requête, donc le principal de l'hôte est une identité Windows ; pourtant, quand `DocumentService` ouvre un fichier, le système d'exploitation vérifie le jeton du pool d'applications, pas celui de reader1, sauf si le code emprunte une identité (Module 3). Un partenaire fédéré (Module 5) a un principal applicatif valide et aucun jeton Windows. L'export nocturne n'a aucun utilisateur connecté, et pourtant le compte du processus peut lire tout le dossier des documents.

Consignez-les comme trois observations distinctes et ne laissez jamais l'une remplacer l'autre. Aucune n'est une décision d'autorisation : le principal de l'hôte dit qui s'est connecté, un claim de rôle dit ce qui était vrai à la connexion, le jeton du processus dit ce que le système d'exploitation permet. AccessOps doit répondre à une question plus étroite à chaque lecture protégée : l'octroi actuel permet-il à ce compte de lire cet enregistrement dans ce tenant, maintenant ?

`Visible` et `Enabled` décrivent l'écran. Wisej.NET vérifie bien sur le serveur qu'un clic provient d'un contrôle activé et visible, mais les données liées à une colonne de grille masquée atteignent quand même le navigateur, et le même document est accessible depuis un handler, un chemin de téléchargement et une tâche. La règle a sa place dans le service qu'ils appellent tous.

Comment Application.User et IsAuthenticated se résolvent

`Application.User` est un `IPrincipal` statique dont le getter procède en deux temps. Si une valeur a été affectée dans cette session, il renvoie cette substitution de session ; sinon, il renvoie le principal de

requête — sur ASP.NET Core, le `HttpContext.User` capturé à l'arrivée de la requête ou de l'upgrade WebSocket : une `WindowsIdentity` sous l'authentification Windows d'IIS, un principal non authentifié lors d'une exécution Kestrel de développement. `Application.IsAuthenticated` vaut simplement `User?.Identity?.IsAuthenticated == true` : il ne sait rien du tenant, du rôle ni de la propriété. Définir `Application.User = null` supprime la substitution et revient au principal de l'hôte : « déconnecter » ainsi un utilisateur Windows le laisse donc authentifié.

N'affectez la substitution qu'après qu'un vérificateur de confiance a renvoyé un compte ; une `ClaimsIdentity` construite à partir d'un nom d'utilisateur saisi laisse le navigateur choisir son identité. L'utilitaire `ApplicationPrincipal` du lab construit les claims à partir du compte vérifié — ID de compte immuable, tenant, ID d'octroi, rôles — sous le type d'authentification `"Portal"`. Indexez sur cet ID, jamais sur un nom d'affichage ni sur une adresse e-mail.

```
// Runs on a page load, after the test verifier returned an account
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();   // new ID first, on the load
    Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"));
}
```

`Application.UserIdentity` n'existe que dans le build .NET Framework (net48), où il s'agit de l'identité de connexion System.Web ; le build net8.0 qu'un AccessOps en .NET 10 référence ne le déclare pas. Sur .NET 10, lisez le principal Windows via `Application.User` et vérifiez que son `Identity` est une `WindowsIdentity` avant de vous y fier.

Le cycle de vie de la session et le travail hors de celle-ci

Le premier chargement HTTP construit l'interface, les événements suivants arrivent par le WebSocket, et chaque transition de session est un événement à part. `ApplicationStart` se déclenche pour une nouvelle session après `Program.Main` ; `ApplicationRefresh` se déclenche lors d'un rechargement du navigateur ou d'un changement d'URL dans la même session ; `SessionTimeout` annonce l'expiration pour inactivité (définissez `Handled` pour supprimer la boîte de dialogue de prolongation par défaut) ; `ApplicationExit` se déclenche quand la session est terminée ou libérée, et c'est le seul événement levé à l'expiration. Utilisez-le pour le nettoyage, jamais comme révocation : celle-ci vit dans le magasin d'octrois.

`Application.RefreshSessionId` émet un nouvel ID de session sans perdre la session, ce qui ferme la fixation après la connexion — mais seulement lors d'un chargement de page. Depuis un clic WebSocket, le serveur change d'ID, le navigateur n'adopte jamais le nouveau, et sa requête suivante démarre une nouvelle session. AccessOps vérifie donc au clic, met le compte en attente sous une clé, appelle `Application.Reload()`, puis, dans `ApplicationRefresh`, fait d'abord tourner l'ID avant d'établir le principal. Masquez `Application.SessionId` dans les traces.

Hors de la requête, l'identité ambiante fait défaut. `Application.StartTask` restaure la session sur son thread du pool ; un `Task.Run` brut ou un callback de timer n'en a aucune, et `Application.Update` n'en restaure une qu'à partir d'un `Application.Current` mémorisé. L'export en arrière-plan reçoit donc un sujet explicite — ID de compte, tenant, ID d'octroi, ID d'enregistrement — et `DocumentService` revérifie l'octroi au démarrage de la tâche. Si deux profils de navigateur affichent le même compte, cherchez un champ statique qui contient le principal.

Le refus à la couche service

`DocumentService.Read(subject, tenant, recordId)` charge l'octroi actuel et refuse lorsqu'il est absent, révoqué, expiré ou dépourvu du droit de lecture, ou lorsque son tenant diffère de celui demandé ; c'est seulement ensuite qu'il appelle le repository. Une vérification de rôle ne suffit pas : le Reader du tenant A est bel et bien un Reader, l'enregistrement 17 existe dans les deux tenants, et `IsInRole("Reader")` laisse passer la lecture horizontale.

La preuve est un nombre. Dans la fixture F02, le Reader du tenant A demande l'enregistrement 17 du tenant B : refus, et le compteur de lectures ne bouge pas. Révoquez ensuite l'octroi Reader depuis la console d'administration et cliquez sur Export sur l'écran qui affiche encore le bouton : le service lit l'octroi actuel et refuse avant qu'aucun fichier ne s'ouvre. Chaque décision écrit un événement d'audit avec un sujet opaque, l'action, la classification, la décision et un ID de corrélation — jamais de cookies, d'ID de session ni de jetons.

La console autonome `PolicyHarness` (net8.0 dans le pack ; net10.0 fonctionne de la même manière) applique la règle sans Wisej.NET, IIS ni fournisseur d'identité : une lecture autorisée, puis des refus pour franchissement de tenant, rôle manquant, compte inconnu, octroi expiré et octroi révoqué, à zéro appel au repository. Le harness prouve la politique n'importe où ; l'application prouve que chaque chemin l'utilise.

> **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 :

- ✓ Distinguer le sujet applicatif, le principal de requête de l'hôte et l'identité Windows effective d'une requête AccessOps, et expliquer pourquoi aucun d'eux n'est, à lui seul, une décision d'autorisation.
- ✓ Expliquer comment Application.User et IsAuthenticated se résolvent, y compris la substitution de session et le repli sur le principal de l'hôte, et pourquoi Application.UserIdentity n'existe que sur .NET Framework.

- ✓ Retracer le cycle de vie de la session — ApplicationStart, ApplicationRefresh, SessionTimeout, ApplicationExit et Application.RefreshSessionId — et transmettre un sujet explicite au travail qui s'exécute hors de la session.
- ✓ Appliquer une vérification du tenant et de l'enregistrement à la couche service et prouver que le refus intervient avant toute lecture du repository, même après la révocation du rôle sur un écran ouvert.

Guide du lab / de l'examen

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

Lab pratique

Créez la solution AccessOps avec les tenants A et B contenant chacun un enregistrement 17 au contenu distinct, un compte Reader dans le tenant A et un compte non affecté. Construisez un DocumentService qui vérifie l'octroi actuel, le tenant et l'enregistrement avant d'appeler un repository qui compte ses lectures, établissez le principal de session à partir d'un vérificateur de test de confiance avec ApplicationPrincipal, et ajoutez une vue de diagnostic (tenue à l'écart des routes publiques et retirée avant le packaging) qui affiche le type d'authentification, l'indicateur d'authentification, un sujet masqué et l'identité effective du système d'exploitation. Retracez le démarrage, un clic, un rafraîchissement du navigateur et une tâche en arrière-plan dans deux profils de navigateur, demandez l'enregistrement 17 du tenant B en tant que Reader du tenant A, révoquez l'octroi Reader entre le chargement de la page et Export sur un écran déjà ouvert, et exécutez le PolicyHarness autonome pour montrer les mêmes décisions d'autorisation et de refus sans Wisej, IIS ni fournisseur d'identité.

Point de départ

Une nouvelle application web Wisej.NET nommée AccessOps sur `net10.0` avec le package `Wisej-4` 4.1.7, exécutée sous Kestrel sur votre propre machine, plus le projet console `PolicyHarness` du pack ajouté à la même solution. Ce module ne nécessite ni IIS, ni domaine Windows, ni fournisseur d'identité : chaque étape s'exécute localement, et l'« identité effective du système d'exploitation » que vous consignez est simplement le compte sous lequel s'exécute le processus `dotnet`.

Approche suggérée

- ✓ Créez d'abord les fixtures : l'enregistrement 17 dans le tenant A et dans le tenant B avec des contenus différents, un octroi Reader pour le tenant A et un compte sans octroi, ainsi qu'un repository dont vous pouvez afficher le `ReadCount` à l'écran.
- ✓ Écrivez `DocumentService.Read(subject, tenant, recordId)` de sorte qu'il charge l'octroi actuel et refuse un octroi absent, révoqué ou expiré, un droit de lecture manquant et une discordance de tenant avant de toucher au repository.
- ✓ Reliez la connexion à un vérificateur d'entraînement qui n'est jamais livré : vérifiez au clic, mettez le compte en attente, appelez `Application.Reload()`, puis, au chargement, appelez `ApplicationPrincipal.Establish`, qui fait tourner l'ID de session avant de définir `Application.User`.
- ✓ Ajoutez la vue de diagnostic derrière un commutateur de compilation réservé au débogage, puis remplissez la matrice à quatre colonnes pour le démarrage, un clic, un rafraîchissement et une tâche en arrière-plan dans deux profils de navigateur, y compris une tâche lancée avec `Application.StartTask` et une autre lancée avec `Task.Run` qui reçoit un sujet explicite.
- ✓ Exécutez les refus — l'enregistrement 17 du tenant B en tant que Reader du tenant A, puis Export après révocation de l'octroi sur un écran ouvert —, vérifiez que `ReadCount` n'a pas

bougé, exécutez `PolicyHarness`, et confirmez que le build publié ne contient aucune vue de diagnostic.

Livrables attendus

- ✓ Solution AccessOps avec les fixtures des tenants A et B et un repository compteur de lectures
- ✓ Garde de ressource DocumentService avec vérification de l'octroi, du tenant et de l'enregistrement
- ✓ Vue de diagnostic des identités, retirée du build packagé
- ✓ Matrice d'identités à quatre colonnes pour la requête, la session, l'opération et le résultat
- ✓ Exécution du PolicyHarness avec une lecture autorisée et deux refus à zéro appel au repository

Questions de révision

- ✓ Votre vue de diagnostic affiche la même identité effective du système d'exploitation pour les deux profils de navigateur, alors que les sujets applicatifs diffèrent. Qu'est-ce que cela vous apprend sur l'identité que le système de fichiers vérifiera, et qu'est-ce qui changerait sur IIS avec l'authentification Windows ?
- ✓ Après la révocation de l'octroi Reader, le bouton Export est toujours sur l'écran ouvert et `Application.User` porte toujours le claim de rôle Reader. Quelle lecture unique dans votre code provoque le refus, et pourquoi le claim de rôle n'est-il pas cette lecture ?
- ✓ Votre export en arrière-plan s'exécute sur `Task.Run` une heure après le clic. Que reçoit-il exactement, que revérifie-t-il au démarrage, et que se passerait-il s'il lisait `Application.User` à la place ?

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.