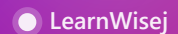


Audit, secrets et configuration

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



Guide de la leçon

Pistes d'audit pour les commandes sensibles, identifiants de corrélation, gestion des secrets, configuration par environnement, journalisation sans fuite de données. Ce que cela implique pour un développeur qui sécurise une application Wisej.NET de production — et comment aborder ce module.



Enjeux en production

Pistes d'audit pour les commandes sensibles, identifiants de corrélation, gestion des secrets, configuration par environnement, journalisation sans fuite de données.

1. Le problème en production

AccessOps authentifie désormais les personnes, résout leurs permissions et protège ses documents — et rien de tout cela n'aide le matin où quelqu'un demande qui a approuvé la demande 4471, depuis quelle adresse, et si quelqu'un a été refusé avant. Les cinq modules précédents ont décidé de ce qui est autorisé ; celui-ci décide de ce que l'on pourra reconstituer après coup. Il comble aussi la brèche qui transforme un petit incident en incident majeur : un mot de passe de base de données ou un secret client OIDC dans un fichier commité il y a deux ans.

2. Concepts clés

Concept	Pourquoi c'est important
piste d'audit	Un enregistrement métier de chaque commande sensible — qui a agi, sur quoi, quand et d'où — conservé pendant une durée de rétention définie, parce que quelqu'un devra en répondre.
stockage en ajout seul	Une table d'audit dans laquelle le compte de l'application ne peut faire que des <code>INSERT</code> . Si le code qui écrit l'historique peut aussi le réécrire, l'historique ne prouve rien.
preuve d'altération	Une séquence sans trou associée à une chaîne de hachage, où le hachage de chaque ligne couvre ses propres champs et le hachage précédent, de sorte qu'une ligne modifiée rompt la chaîne.
identifiant de corrélation	Un identifiant créé au début d'une commande et répété sur chaque ligne de log et chaque ligne d'audit qu'elle provoque, pour qu'un simple clic puisse être reconstitué plus tard.
scope de journalisation	Un bloc de propriétés ambiant — <code>ILogger.BeginScope</code> ou un enrichir Serilog — qui attache l'identifiant de corrélation et l'acteur à tout ce qui est journalisé à l'intérieur.
stockage de secrets	L'endroit où un identifiant vit en dehors du build : user secrets en local, variables d'environnement dans un conteneur, coffre managé en production — ou une identité managée, sans rien stocker du tout.
masquage	Une règle sur ce qui n'atteint jamais une destination de log — mots de passe, jetons, cookies, chaînes de connexion et données personnelles — appliquée par la journalisation structurée, pas par la mémoire.

3. Déroulé de la leçon

L'enregistrement qui survit à la dispute

Deux choses différentes sont appelées « journalisation ». Une ligne de log est un diagnostic, et peut être échantillonnée ou supprimée par rotation au bout d'une semaine. Une ligne d'audit est une preuve : elle a une durée de rétention, un propriétaire et un lecteur qui n'est pas développeur. AccessOps a besoin d'un `AuditService` comme point d'entrée unique pour la seconde catégorie, appelé par chaque commande sensible — approuver une demande, accorder ou révoquer une permission, télécharger un document confidentiel, et chaque tentative de connexion du Module 2. Rien d'autre n'écrit dans la table d'audit.

Une ligne répond à six questions : l'identifiant stable de l'acteur, le nom de l'action (`Approval.Approve` , `Document.Download`), le type et l'id de la cible, l'horodatage UTC issu de l'horloge du serveur, l'adresse client et l'user agent, et l'identifiant de corrélation. Consignez aussi le résultat, car un refus est souvent plus intéressant qu'un succès. Dans Wisej.NET, l'adresse réseau du client et l'user agent du navigateur proviennent du contexte applicatif propre à la session, et non de ce que la page envoie : capturez-les donc une fois, lors de la création du `SessionContext` après la connexion. Prenez aussi l'acteur dans ce contexte vérifié — jamais dans un contrôle ou dans un paramètre fourni par l'appelant.

Rendez ensuite la ligne difficile à effacer. Le compte de base de données de l'application reçoit `INSERT` et rien d'autre, pour qu'un bug ou une instruction injectée ne puisse pas effacer discrètement un mardi

entier. Donnez à chaque ligne une séquence sans trou et un hachage de ses propres champs et du hachage de la ligne précédente ; un traitement planifié recalcule la chaîne et nomme la première ligne qui ne correspond plus. C'est une *preuve* d'altération, pas une prévention — elle rend une modification visible plutôt qu'impossible. Enfin, écrivez la ligne d'audit dans la même transaction que la modification métier, pour qu'une approbation sans son enregistrement soit impossible.

```
using (_log.BeginScope(new Dictionary<string, object>
    { ["CorrelationId"] = ctx.CorrelationId, ["UserId"] = ctx.UserId }))
using (var tx = _db.BeginTransaction())
{
    _approvals.Approve(requestId, ctx);           // the business change
    _audit.Record(AuditEntry.For(ctx,             // actor, address, correlation
        "Approval.Approve", "ApprovalRequest", requestId));
    tx.Commit();                                 // both, or neither
}
```

Un identifiant, une histoire

L'identifiant de corrélation est créé là où commence une commande utilisateur — en tête d'un gestionnaire d'événements léger, avant tout appel de service. Un `Guid` suffit ; si vous utilisez déjà `System.Diagnostics.Activity`, prenez l'identifiant de trace de l'activité courante pour que l'identifiant corresponde à tout ce qui parle le format de trace W3C. Placez-le sur le contexte propre à la commande, qui porte aussi l'acteur et l'adresse client, et passez ce contexte à chaque service. N'allez pas rechercher la session depuis l'intérieur d'un service : un traitement en arrière-plan lancé avec `Application.StartTask` peut survivre à la session qui l'a démarré.

Attacher l'identifiant aux lignes de log n'est pas le travail de l'appelant. Ouvrez un scope de journalisation autour de la commande — `ILogger.BeginScope` avec un dictionnaire de propriétés, ou le `LogContext` de Serilog avec un enricher — et chaque ligne écrite à l'intérieur porte automatiquement l'identifiant de corrélation et l'acteur, y compris les lignes des bibliothèques sous-jacentes. Utilisez la journalisation structurée pour que l'identifiant soit une propriété nommée dans la destination, ce qui fait de « montrez-moi tout pour la corrélation 7f3c » une requête plutôt qu'un grep. Puis rendez-le visible : lorsqu'une commande échoue, terminez la phrase destinée à l'utilisateur par l'identifiant, en guise de référence.

Les secrets appartiennent à l'environnement, pas au dépôt

AccessOps détient désormais trois identifiants : la chaîne de connexion à la base de données, le secret client OIDC du Module 4, et la clé de signature des liens de partage à expiration du Module 5. Aucun n'a sa place dans un fichier suivi par le contrôle de version. En développement, utilisez les user secrets .NET — le projet reçoit un `UserSecretsId` et les valeurs vivent hors du dossier de la solution, si bien qu'un clone du dépôt ne contient aucun identifiant. En déploiement, fournissez-les par variables d'environnement ou lisez-les depuis un coffre managé via un fournisseur de configuration. Mieux encore, utilisez une identité managée : l'application s'authentifie en son propre nom et reçoit un jeton de courte durée, il n'y a donc aucun secret stocké susceptible de fuiter.

Cela fonctionne parce que la configuration ASP.NET Core est organisée en couches et que les sources ultérieures l'emportent : `appsettings.json`, puis la surcouche d'environnement, puis les user secrets en développement, puis les variables d'environnement, puis un fournisseur de coffre. Les fichiers commités

contiennent la structure et des valeurs par défaut inoffensives ; les valeurs de type secret arrivent par environnement. Le `Default.json` propre à Wisej.NET configure le shell — thème, délai d'expiration de session, fenêtre de démarrage — et n'est pas plus un stockage de secrets que ne l'est `Web.config`.

Un secret que vous ne pouvez pas faire tourner sereinement est un secret que vous ferez mal tourner à 3 heures du matin : la procédure de rotation est donc ici un livrable. Consignez, pour chaque secret, le propriétaire, le stockage, l'expiration et les étapes de remplacement. Prévoyez deux identifiants valides simultanément — un second secret client, un second compte de base de données — pour que la rotation soit un redémarrage progressif plutôt qu'une interruption de service. Et une fois qu'une valeur a été committée, supprimer la ligne ne sert à rien : faites-la tourner et révoquez l'ancienne.

Journaliser l'incident, pas la personne

Les logs sont copiés dans des index de recherche, expédiés à des services tiers, ouverts sur des portables de support et conservés bien plus longtemps que prévu. Ils doivent contenir assez d'informations pour reconstituer un incident, et rien qui soit préjudiciable à lui seul. Ne journalisez jamais un mot de passe (pas même un mot de passe erroné), un jeton, un en-tête d'autorisation, un cookie, une chaîne de connexion ou le contenu d'un document confidentiel.

Les données personnelles exigent la même discipline, pour une autre raison : une ligne d'audit est un enregistrement légitime doté d'une politique de rétention, tandis qu'une ligne de debug contenant l'adresse personnelle d'un employé en est une copie incontrôlée. Journalisez l'identifiant stable de l'utilisateur plutôt que son nom et son e-mail, et l'id du document plutôt qu'un nom de fichier qui peut lui-même être sensible. La journalisation structurée aide, car vous journalisez des propriétés nommées issues d'une liste d'autorisation au lieu de sérialiser tout un objet requête en espérant que tout ira bien.

- Ne jamais journaliser : mots de passe, jetons, cookies, chaînes de connexion, contenu des documents.
- Journaliser par référence : id utilisateur, id de document, id de demande — pas de noms ni de données personnelles en texte libre.
- L'imposer au niveau de la destination : une politique de masquage, plus un test vérifiant qu'un marqueur de secret connu n'atteint jamais la sortie.

Bouclez la boucle par une revue plutôt que par une règle : parcourez de bout en bout la sortie de log d'une vraie approbation et d'un vrai téléchargement, et marquez chaque champ comme nécessaire, masqué ou supprimé.



Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Consigner qui a fait quoi, quand et d'où pour chaque commande sensible, avec une piste d'audit qui révèle toute altération.
- ✓ Faire circuler un identifiant de corrélation dans les logs pour qu'une action utilisateur puisse être tracée à travers les services.
- ✓ Garder les secrets hors du contrôle de version grâce à la configuration d'environnement, aux stockages de secrets et aux identités managées.
- ✓ Journaliser suffisamment pour enquêter sur un incident sans journaliser de mots de passe, de jetons ni de données personnelles.

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 l'audit et la gestion des secrets à AccessOps : un `AuditService` qui enregistre les approbations, les modifications de permissions et les téléchargements de documents avec l'utilisateur, l'heure, l'adresse et l'identifiant de corrélation, une table d'audit en ajout seul, un identifiant de corrélation créé pour chaque commande et écrit sur chaque ligne de log, les chaînes de connexion et le secret client OIDC déplacés dans la configuration d'environnement avec une procédure de rotation documentée, et une revue des logs qui masque les champs sensibles.

Point de départ

Vous ouvrez la solution AccessOps telle qu'elle se présente à la fin du Module 5 : un `SessionContext` vérifié créé à la connexion, un `PermissionService` et une garde d'autorisation dans `DocumentService` et `ApprovalService`, la connexion OIDC, et la fonctionnalité de documents avec sa chaîne d'upload et ses liens de partage signés. L'écran d'approbation, l'éditeur de permissions d'administration et la liste des documents fonctionnent tous — ils ne laissent simplement aucune trace de ce qu'ils ont fait, et la chaîne de connexion à la base de données ainsi que le secret client OIDC se trouvent toujours dans un fichier de configuration suivi par le contrôle de version.

Approche suggérée

- ✓ Créez la table d'audit avec une séquence sans trou, l'acteur, l'action, le type et l'id de la cible, l'horodatage UTC, l'adresse client, l'user agent, le résultat, l'identifiant de corrélation et une colonne de hachage, puis accordez au compte de base de données de l'application uniquement `INSERT` — pas d' `UPDATE`, pas de `DELETE`.
- ✓ Écrivez `AuditService` comme unique rédacteur de cette table, en prenant l'acteur et les informations client dans le `SessionContext` vérifié, en calculant le hachage de chaque ligne sur ses propres champs et le hachage de la ligne précédente, et en inscrivant l'insertion dans la même transaction que la modification métier.
- ✓ Ajoutez un contexte propre à chaque commande qui crée un identifiant de corrélation en tête de chaque gestionnaire d'événements, passez-le à `ApprovalService`, à `DocumentService` et à l'éditeur de permissions, et ouvrez un scope de journalisation pour que chaque ligne écrite pendant la commande porte l'identifiant et l'acteur.
- ✓ Sortez la chaîne de connexion, le secret client OIDC et la clé de signature des liens de partage des fichiers de configuration suivis, vers les user secrets en développement et les variables d'environnement en déploiement, puis rédigez la procédure de rotation avec les propriétaires, les dates d'expiration et le chevauchement de deux identifiants valides.
- ✓ Appelez `AuditService` depuis l'approbation, l'octroi et la révocation de permissions, et le téléchargement de documents — en consignait les refus comme les succès — puis exécutez une approbation et un téléchargement et passez en revue chaque ligne de log produite à la

recherche de mots de passe, de jetons, d'en-têtes, de chaînes de connexion et de données personnelles.

Livrables attendus

- ✓ AuditService et table d'audit en ajout seul
- ✓ Identifiant de corrélation dans chaque commande et chaque ligne de log
- ✓ Secrets déplacés dans la configuration d'environnement
- ✓ Procédure de rotation des secrets
- ✓ Revue du masquage des logs

Questions de révision

- ✓ Votre table d'audit est en insertion seule pour l'application, et chaque ligne est chaînée par hachage. Nommez une action qu'un attaquant disposant du propre compte de base de données de l'application peut encore mener contre la piste, et expliquez précisément comment la vérification la ferait apparaître.
- ✓ Une approbation commence sur l'écran d'approbation et se termine dans une tâche en arrière-plan lancée avec `Application.StartTask` après la déconnexion de l'utilisateur. D'où cette tâche tire-t-elle son identifiant de corrélation et son acteur, et qu'est-ce qui casse si elle les lit dans la session ?
- ✓ Un développeur retire le secret client OIDC du fichier de configuration suivi et pousse la modification. Pourquoi le secret est-il toujours compromis, et que faut-il faire pour que l'incident soit réellement clos ?



Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.