MODÈLES DE SÉCURITÉ ET D'AUTHENTIFICATION · MODULE 6

Clés d'accès avec l'extension WebAuthn

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

LearnWisej

Guide de la leçon

La conception de la partie de confiance WebAuthn, l'enrôlement à partir d'un compte vérifié, les cérémonies d'enregistrement et d'authentification avec l'extension WebAuthn de Wisej.NET, la validation côté serveur de chaque assertion, un magasin d'identifiants sûr face à la concurrence et la récupération de compte. 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

La conception de la partie de confiance WebAuthn, l'enrôlement à partir d'un compte vérifié, les cérémonies d'enregistrement et d'authentification avec l'extension WebAuthn de Wisej.NET, la validation côté serveur de chaque assertion, un magasin d'identifiants sûr face à la concurrence et la récupération de compte.

1. Le problème en production

Les partenaires d'AccessOps se connectent déjà avec un compte local (Module 4) ou avec leur propre fournisseur d'identité (Module 5). Une clé d'accès ajoute un identifiant qui ne peut être ni hameçonné ni rejoué sur un autre site, mais seulement si le serveur vérifie tout ce que le navigateur lui renvoie. L'extension WebAuthn de Wisej.NET exécute les cérémonies du navigateur et vérifie une signature ; le challenge, l'origine, la partie de confiance, le propriétaire, le compteur et le magasin d'identifiants relèvent du code d'AccessOps, et chacun est un endroit où une connexion peut être acceptée à tort.

2. Concepts clés

Concept	Pourquoi c'est important
ID de la partie de confiance	Le domaine, sans schéma ni port, auquel un identifiant est rattaché ; le changer plus tard oblige chaque utilisateur à s'enregistrer de nouveau.
liste d'autorisation des origines	Les valeurs exactes de schéma, d'hôte et de port acceptées dans `ClientData.Origin` ; tout le reste est refusé, même avec une signature valide.
ID d'identifiant et clé publique	Ce que le serveur stocke pour chaque identifiant ; la clé privée et tout échantillon biométrique ne quittent jamais l'authentificateur.
user handle	Un identifiant de compte opaque et stable, envoyé comme `id` de l'entité utilisateur lors de l'enregistrement et renvoyé sous la forme `UserHandle`.
challenge à usage unique	Des octets aléatoires propres à chaque cérémonie, liés à son type, à son compte et à son expiration, comparés avec `ClientData.Challenge` et consommés pour qu'une réponse capturée ne puisse pas servir deux fois.
politique d'assertion	Les vérifications qui entourent `WebAuthn.Validate`, lequel prouve seulement que la signature correspond à une clé publique stockée.
commit compare-and-swap	Une seule mise à jour conditionnelle qui n'applique une connexion vérifiée que si la ligne a toujours la version qui a été lue, et renvoie Updated, Conflict ou NotFound.

3. Le fil de la leçon

Une partie de confiance, des origines exactes, et ce que le serveur conserve

Un identifiant WebAuthn appartient à un ID de partie de confiance, un simple domaine comme `partners.accessops.example`. Une origine est plus étroite : `https://partners.accessops.example`, schéma, hôte et port compris. AccessOps s'enregistre avec un RP ID égal à l'hôte de connexion des partenaires et tient une liste d'autorisation d'origines exactes, pas un motif. Pour le lab, le contexte sécurisé documenté est `https://localhost` sur le port du lab avec le RP ID `"localhost"` ; partout ailleurs, WebAuthn refuse le HTTP simple. Le choix de l'hôte compte avec cette extension : `WebAuthn.GetAsync` ne transmet aucun RP ID au navigateur, qui utilise donc le domaine de la page elle-même, et un identifiant enregistré pour un domaine parent peut ne pas être proposé sur un sous-hôte. Enregistrez-vous et connectez-vous sur le même hôte.

Le serveur conserve, pour chaque identifiant : l'ID d'identifiant, la clé publique sérialisée (`JSON.Stringify(publicKey)`, restaurée avec `new PublicKey(json)`), l'algorithme, le RP ID, le compte et son user handle, le dernier compteur de signatures, l'indicateur `BackupEligibility`, un libellé et des horodatages. AccessOps génère le user handle, jamais à partir d'une adresse e-mail, et le garde en ASCII : le client envoie l'`id` à raison d'un octet par caractère et décode `UserHandle` en UTF-8. L'application ne reçoit jamais de clé privée, d'empreinte digitale ni d'image de visage.

Deux cérémonies depuis du C# côté serveur

Le package est `Wisej-4-WebAuthn` 4.1.7, espace de noms `Wisej.Ext.WebAuthn`. Il s'agit d'une seule classe statique avec quatre méthodes : ni composants, ni événements, ni magasin, ni gestion des challenges. `IsUserVerifyingPlatformAuthenticatorAvailableAsync` signale Windows Hello, Touch ID ou Face ID ; une clé de sécurité peut encore fonctionner quand la méthode renvoie false. L'enregistrement passe par `CreateAsync` avec sept arguments obligatoires : un challenge en base64 standard d'au moins 16 octets aléatoires, un `RelyingParty`, un `PublicKeyCredentialUserEntity`, les algorithmes, un `AuthenticatorSelectionCriteria`, un délai en millisecondes et un `AttestationConveyancePreference` (`None` dans AccessOps). Ne proposez que `COSEAlgorithmIdentifier.ES256` et `RS256` : `Validate` renvoie false pour les autres courbes ECDSA et lève une exception pour EdDSA. La manière dont ce package sérialise la valeur d'attachement n'est pas vérifiée ; AccessOps ne s'y fie donc pas pour restreindre les authentificateurs.

L'enrôlement ne démarre qu'à partir d'une session de compte vérifiée : le partenaire s'est connecté récemment via le Module 4 ou le Module 5 et l'octroi actuel est valide. `CreateAsync` n'effectue aucune vérification d'attestation et ne remplit ni les indicateurs de l'authentificateur ni le hachage du RP ID ; la réponse prouve donc peu de chose sur la personne, c'est la session qui le prouve. AccessOps compare le challenge, vérifie que `ClientData.Type` vaut `"webauthn.create"` et que l'origine figure dans la liste d'autorisation, puis stocke `AuthenticatorData.PublicKey`. Les deux appels asynchrones lèvent une `Exception` portant le message du navigateur en cas d'annulation, de délai dépassé ou de non-correspondance ; ils se trouvent donc dans un `try` / `catch`, et l'écran affiche un texte fixe avec `allowHtml: false` : `AlertBox` et `MessageBox` le mettent à true par défaut.

Valider chaque assertion sur le serveur

La connexion appelle `GetAsync` avec un nouveau challenge. Passer `null` pour les identifiants autorisés laisse le partenaire choisir une clé d'accès détectable, et `UserHandle` désigne alors le compte ; un formulaire qui demande d'abord le compte passe plutôt la liste de `PublicKeyCredentialDescriptor` de ce compte. La réponse n'indique pas quel identifiant a signé ; AccessOps essaie donc les clés stockées pour ce user handle sous ce RP et vérifie lui-même tout le reste :

```
var r = await WebAuthn.GetAsync(ticket.Base64, null, 60000);
bool fresh = _challenges.TryConsume(ticket.Id, r.ClientData.Challenge);
var flags = AuthenticatorFlags.Read(r.AuthenticatorData.Base64); // 4.1.7: UserVerified
reads a reserved bit
var cred = _store.ForUserHandle(Rp.Id, r.UserHandle).FirstOrDefault(c ⇒
    WebAuthn.Validate(c.Key, r.AuthenticatorData.Base64,
                      r.ClientData.Base64, Es256Signature.ForValidate(r.Signature)));
bool ok = fresh && cred ≠ null
    && r.ClientData.Type == "webauthn.get"
    && AllowedOrigins.Contains(r.ClientData.Origin)
    && r.AuthenticatorData.RPIDHash.SequenceEqual(RpIdHash)
    && flags.UserPresent && flags.UserVerified
    && CounterPolicy.Accepts(cred.SignCount, r.SignatureCounter);
if (!ok) { _audit.PasskeyDenied(ticket.Id); return; }
var outcome = _store.CommitUse(cred, r.SignatureCounter); // Updated | Conflict | NotFound
```

Le challenge est consommé en premier : une tentative échouée le brûle donc aussi. `GetAsync` n'envoie aucune option de vérification de l'utilisateur ; c'est la valeur par défaut du navigateur qui s'applique, et la vérification de l'utilisateur doit être contrôlée plutôt que supposée. Deux défauts de l'extension 4.1.7 décident de la manière. Son script client renseigne `AuthenticatorData.UserVerified` à partir du bit 1 de l'octet de drapeaux, un bit réservé toujours à zéro, si bien que la propriété reste false même lorsqu'une invite Windows Hello a vérifié la personne ; AccessOps lit donc lui-même l'octet de drapeaux dans `AuthenticatorData.Base64` (l'octet 32 des données d'authentificateur signées : bit 0 utilisateur présent, bit 2 utilisateur vérifié, bit 3 sauvegarde possible) et n'utilise la propriété que comme contre-vérification de la présence. Le second défaut se trouve dans `WebAuthn.Validate` : la méthode lit la signature ES256 encodée en DER à des positions fixes, de sorte qu'une signature authentique dont la valeur `r` ou `s` fait moins de 32 octets, environ une sur 128, échoue à la vérification ; AccessOps ré-encode les deux entiers sur exactement 33 octets avant d'appeler Validate, ce qui ne change ni les valeurs ni ce que prouve la signature, et une signature malformée échoue toujours. Les deux contournements tiennent dans deux petites classes, `AuthenticatorFlags` et `Es256Signature`, et doivent être retirés dès qu'une extension corrigée sera publiée : les deux défauts ont été signalés le 10 novembre 2026 et sont corrigés dans le code source de l'extension ; le correctif arrivera avec une version du package postérieure à la 4.1.7. La politique du compteur est écrite noir sur blanc : de nombreux authentificateurs renvoient toujours zéro, si bien que zéro suivi de zéro est accepté, tandis qu'un compteur non nul qui n'augmente pas est refusé et consigné comme anomalie. L'extension expose `BackupEligibility` mais aucun indicateur d'état de sauvegarde ; AccessOps consigne donc l'éligibilité et ne prend aucune décision qui dépende du fait qu'une clé d'accès soit actuellement synchronisée. Ce n'est que lorsque le commit renvoie Updated que le transfert du Module 4 s'exécute : mettre le compte vérifié en attente, recharger, appeler `Application.RefreshSessionId()` au chargement, définir `Application.User` et créer exactement un octroi de connexion, que chaque lecture protégée continue de vérifier.

Un magasin d'identifiants qui rend la révocation définitive

Le `PasskeyCredentialStore` est une classe propre à AccessOps, posée sur une seule table avec une contrainte d'unicité sur le RP ID et l'ID d'identifiant, si bien qu'un identifiant ne peut être enrôlé ni deux fois ni sous un second compte. Son commit est une seule instruction conditionnelle : mettre à jour le compteur et la date de dernière utilisation là où le RP, l'ID d'identifiant, le propriétaire, l'identité de la ligne et la version correspondent tous à ce que le vérificateur a lu. Une ligne affectée, c'est Updated. Zéro ligne est classé, jamais réessayé à l'aveugle : NotFound si la ligne a disparu, sinon Conflict. Un identifiant supprimé pendant qu'une connexion était en cours reste supprimé, et un identifiant ré-enrôlé avec le même ID reçoit une nouvelle identité de ligne, si bien qu'une mise à jour périmée ne peut pas s'y appliquer. Lire, comparer en C# puis écrire dans une seconde étape non gardée, c'est exactement la situation de concurrence à éviter.

La page de revue liste, renomme et supprime uniquement les identifiants de l'utilisateur connecté, en prenant le propriétaire dans le principal de session, jamais dans une cellule de grille. La récupération après la perte d'un appareil est un plan écrit : une seconde clé d'accès enrôlée, ou une connexion via le Module 4 ou le Module 5 plus une étape de vérification récente avant l'enrôlement d'un nouvel identifiant, la suppression de l'identifiant perdu, et chaque changement écrit dans l'historique du compte que le support peut auditer. La récupération ne contourne jamais la règle de la session vérifiée.

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 :

- ✓ Choisir l'ID de la partie de confiance et les origines autorisées exactes pour AccessOps, et expliquer ce que le serveur stocke — ID d'identifiant, clé publique, user handle — et ce qu'il ne reçoit jamais.
- ✓ Exécuter l'enregistrement et l'authentification par clé d'accès depuis du C# côté serveur avec l'extension WebAuthn de Wisej.NET, en ne démarrant l'enrôlement qu'à partir d'une session de compte vérifiée.
- ✓ Valider chaque assertion sur le serveur — challenge à usage unique, origine, RP ID, présence et vérification de l'utilisateur, propriétaire de l'identifiant et signature — et appliquer une politique documentée pour le compteur et la sauvegarde.
- ✓ Construire un magasin d'identifiants dont les mises à jour utilisent un compare-and-swap atomique, pour qu'un identifiant supprimé ou révoqué le reste, et planifier la récupération après la perte d'un appareil.

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 les clés d'accès à AccessOps avec l'extension Wisej-4-WebAuthn, sur un nom d'hôte HTTPS stable ou dans un contexte sécurisé localhost documenté : vérifiez la prise en charge par le navigateur, connectez-vous en tant que compte A via le flux du Module 4 ou du Module 5 et enrôlez un identifiant doté d'un libellé avec CreateAsync, puis déconnectez-vous et authentifiez-vous avec GetAsync et Validate côté serveur, mappez le compte local et revérifiez l'octroi actuel avant d'afficher des enregistrements. Stockez les identifiants dans un PasskeyCredentialStore indexé par RP ID et ID d'identifiant, avec une contrainte d'unicité et un commit compare-and-swap qui renvoie Updated, Conflict ou NotFound, ajoutez une page de revue des identifiants où un utilisateur connecté renomme ou supprime uniquement ses propres identifiants, et rejetez, sans nouvel octroi, l'identifiant du compte A présenté pour le compte B, une origine non approuvée, un challenge rejoué et une invite annulée, puis lisez un identifiant deux fois dans un test d'intégration du magasin et montrez l'échec d'une mise à jour périmée et d'une suppression avant commit.

Point de départ

Ouvrez la solution AccessOps cumulative du Module 5 : la route partenaires dispose déjà de la connexion locale avec son `GrantStore`, de la fédération OpenID Connect et du transfert `ApplicationPrincipal` qui fait tourner l'ID de session lors d'un chargement de page. Ajoutez le package `Wisej-4-WebAuthn` 4.1.7 au projet AccessOps et utilisez les deux fixtures partenaires, le compte A et le compte B, dans le même tenant.

Approche suggérée

- ✓ Fixez d'abord la partie de confiance : notez le RP ID, les origines autorisées exactes et le contexte sécurisé (l'hôte HTTPS, ou `https://localhost` avec le RP ID `"localhost"`), et réalisez l'enregistrement et la connexion sur ce même hôte.
- ✓ Construisez la table du `PasskeyCredentialStore` et son commit conditionnel avant toute interface, et écrivez le test d'intégration du magasin qui lit une ligne deux fois, effectue un commit, puis tente la mise à jour périmée et la suppression avant commit.
- ✓ Ajoutez l'enrôlement uniquement à l'écran du compte connecté : émettez un challenge à usage unique lié au compte, appelez `WebAuthn.CreateAsync` avec ES256 et RS256, vérifiez le challenge, le type et l'origine, et stockez l'identifiant avec son libellé.
- ✓ Ajoutez « Sign in with a passkey » à la page de connexion des partenaires : `WebAuthn.GetAsync`, consommation du challenge, recherche de la clé du propriétaire avec `WebAuthn.Validate` (en passant la signature ES256 par `Es256Signature.ForValidate`), vérification du type, de l'origine, du hachage du RP ID, de la présence et de la vérification de l'utilisateur d'après l'octet de drapeaux signé (`AuthenticatorFlags.Read`, et non la propriété `UserVerified`) et du compteur, commit, puis transfert de l'octroi du Module 4.
- ✓ Exécutez les cinq résultats et les deux situations de concurrence du magasin, ajoutez la page de revue des identifiants, et rédigez le plan de récupération après perte d'un appareil, en indiquant

qui approuve un ré-enrôlement et où il est audité.

Livrables attendus

- ✓ Enrôlement et connexion par clé d'accès avec l'extension WebAuthn
- ✓ Validation des assertions côté serveur avec des challenges à usage unique
- ✓ PasskeyCredentialStore avec des commits compare-and-swap atomiques
- ✓ Cinq résultats : une connexion valide et quatre rejets, avec les tests de concurrence du magasin
- ✓ Page de revue des identifiants et plan de récupération de compte

Questions de révision

- ✓ `WebAuthn.Validate` a renvoyé true pour une réponse dont le challenge avait déjà été consommé. Quelle vérification la refuse, et de quoi un attaquant aurait-il eu besoin pour produire cette réponse ?
- ✓ Un partenaire supprime la clé d'accès d'un ordinateur portable perdu pendant qu'une connexion avec cette clé attend le commit. Décrivez ce que renvoie la mise à jour conditionnelle et pourquoi aucun octroi n'est créé.
- ✓ Votre RP ID est le domaine parent et la page de connexion passe sur un sous-hôte. Pourquoi le `GetAsync` de cette extension ne trouve-t-il pas l'identifiant, et que changeriez-vous ?

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.