PATTERN DI SICUREZZA E AUTENTICAZIONE · MODULO 6

Passkey con l'estensione WebAuthn

Guida della lezione · Guida al lab / esame

LearnWisej

Guida della lezione

La progettazione della relying party WebAuthn, la registrazione a partire da un account verificato, le cerimonie di registrazione e di autenticazione con l'estensione WebAuthn di Wisej.NET, la convalida lato server di ogni assertion, un archivio delle credenziali sicuro rispetto alla concorrenza e il recupero dell'account. Che cosa significa per uno sviluppatore che mette in sicurezza i percorsi di accesso, identità e autorizzazione di un'app Wisej.NET di produzione — e come affrontare questo modulo.

Focus di produzione

La progettazione della relying party WebAuthn, la registrazione a partire da un account verificato, le cerimonie di registrazione e di autenticazione con l'estensione WebAuthn di Wisej.NET, la convalida lato server di ogni assertion, un archivio delle credenziali sicuro rispetto alla concorrenza e il recupero dell'account.

1. Il problema in produzione

I partner di AccessOps accedono già con un account locale (Modulo 4) o con il proprio identity provider (Modulo 5). Una passkey aggiunge una credenziale che non può essere sottratta con il phishing né riprodotta su un altro sito, ma solo se il server controlla tutto ciò che il browser gli restituisce. L'estensione WebAuthn di Wisej.NET esegue le cerimonie nel browser e verifica una firma; challenge, origine, relying party, proprietario, contatore e archivio delle credenziali sono codice di AccessOps, e ciascuno è un punto in cui un accesso può essere accettato per errore.

2. Concetti chiave

Concetto	Perché è importante
ID della relying party	Il dominio, senza schema né porta, a cui è legata una credenziale; cambiarlo in seguito costringe ogni utente a registrarsi di nuovo.
allowlist delle origini	I valori esatti di schema, host e porta accettati in `ClientData.Origin`; qualsiasi altro viene rifiutato anche con una firma valida.
ID della credenziale e chiave pubblica	Ciò che il server memorizza per ogni credenziale; la chiave privata e qualsiasi campione biometrico non lasciano mai l'autenticatore.
user handle	Un identificatore dell'account opaco e stabile, inviato come `id` della user entity alla registrazione e restituito come `UserHandle`.
challenge monouso	Byte casuali per ogni cerimonia, legati al suo tipo, all'account e a una scadenza, confrontati con `ClientData.Challenge` e consumati così che una risposta catturata non possa essere usata due volte.
policy delle assertion	I controlli attorno a `WebAuthn.Validate`, che dimostra soltanto che la firma corrisponde a una chiave pubblica memorizzata.
commit compare-and-swap	Un unico aggiornamento condizionale che applica un accesso verificato solo se la riga ha ancora la versione che è stata letta, restituendo Updated, Conflict o NotFound.

3. Il filo della lezione

Una relying party, origini esatte e ciò che il server conserva

Una credenziale WebAuthn appartiene a un ID di relying party, un dominio nudo come `partners.accessops.example`. Un'origine è più ristretta: `https://partners.accessops.example`, schema, host e porta compresi. AccessOps si registra con un RP ID uguale all'host di accesso dei partner e mantiene una allowlist di origini esatte, non un pattern. Per il lab, il contesto sicuro documentato è `https://localhost` sulla porta del lab con RP ID `"localhost"`; WebAuthn rifiuta il semplice HTTP in qualsiasi altro caso. La scelta dell'host conta in questa estensione: `WebAuthn.GetAsync` non passa alcun RP ID al browser, quindi il browser usa il dominio della pagina stessa, e una credenziale registrata per un dominio padre potrebbe non essere proposta su un sotto-host. Registrati e accedi sullo stesso host.

Per ogni credenziale il server conserva: l'ID della credenziale, la chiave pubblica serializzata (`JSON.Stringify(publicKey)`, ripristinata con `new PublicKey(json)`), l'algoritmo, l'RP ID, l'account e il suo user handle, l'ultimo contatore di firma, il flag `BackupEligibility`, un'etichetta e i timestamp. È AccessOps a generare lo user handle, mai un indirizzo e-mail, e lo mantiene ASCII: il client invia l'`id` un byte per carattere e decodifica `UserHandle` come UTF-8. L'applicazione non riceve mai una chiave privata, un'impronta digitale o l'immagine di un volto.

Due cerimonie da C# lato server

Il pacchetto è `Wisej-4-WebAuthn` 4.1.7, namespace `Wisej.Ext.WebAuthn`. È un'unica classe statica con quattro metodi: niente componenti, eventi, archivio o gestione dei challenge. `IsUserVerifyingPlatformAuthenticatorAvailableAsync` segnala Windows Hello, Touch ID o Face ID; una chiave di sicurezza può funzionare anche quando restituisce false. La registrazione è `CreateAsync` con sette argomenti obbligatori: un challenge in base64 standard di almeno 16 byte casuali, una `RelyingParty`, una `PublicKeyCredentialUserEntity`, gli algoritmi, un `AuthenticatorSelectionCriteria`, un timeout in millisecondi e una `AttestationConveyancePreference` (`None` in AccessOps). Offri solo `COSEAlgorithmIdentifier.ES256` e `RS256`: `Validate` restituisce false per le altre curve ECDSA e lancia un'eccezione per EdDSA. Il modo in cui questo pacchetto serializza il valore dell'attachment non è verificato, quindi AccessOps non ci fa affidamento per limitare gli autenticatori.

La registrazione parte solo da una sessione con un account verificato: il partner ha effettuato l'accesso di recente tramite il Modulo 4 o il Modulo 5 e il grant corrente è valido. `CreateAsync` non esegue alcun controllo di attestation e non compila i flag dell'autenticatore né l'hash dell'RP ID, quindi la risposta dimostra poco sulla persona; è la sessione a farlo. AccessOps confronta il challenge, verifica che `ClientData.Type` sia `"webauthn.create"` e che l'origine sia nella allowlist, poi memorizza `AuthenticatorData.PublicKey`. Entrambe le chiamate asincrone lanciano una `Exception` con il messaggio del browser in caso di annullamento, timeout o mancata corrispondenza, quindi stanno in un `try` / `catch` e la schermata mostra un testo fisso con `allowHtml: false`: `AlertBox` e `MessageBox` lo impostano a true per impostazione predefinita.

Convalida ogni assertion sul server

L'accesso chiama `GetAsync` con un nuovo challenge. Passare `null` come credenziali consentite permette al partner di scegliere una passkey rilevabile, e `UserHandle` indica poi l'account; un modulo che chiede prima l'account passa invece l'elenco di `PublicKeyCredentialDescriptor` di quell'account. La risposta non dice quale credenziale ha firmato, quindi AccessOps prova le chiavi memorizzate per quello user handle sotto questa RP e controlla da sé tutto il resto:

```
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
```

Il challenge viene consumato per primo, quindi anche un tentativo fallito lo brucia. `GetAsync` non invia alcuna opzione di verifica dell'utente, quindi si applica il valore predefinito del browser e la verifica dell'utente va controllata invece che data per scontata. Due difetti dell'estensione 4.1.7 decidono come. Il suo script client imposta `AuthenticatorData.UserVerified` dal bit 1 del byte dei flag, un bit riservato che vale sempre zero, perciò la proprietà è false anche quando una richiesta di Windows Hello ha verificato la persona; AccessOps legge quindi da sé il byte dei flag da `AuthenticatorData.Base64` (il byte 32 dei dati firmati dell'authenticator: bit 0 utente presente, bit 2 utente verificato, bit 3 idoneo al backup) e usa la proprietà solo come controprova della presenza. Il secondo difetto è in `WebAuthn.Validate`: legge la firma ES256 codificata DER a offset fissi, così una firma autentica il cui valore `r` o `s` è più corto di 32 byte, circa una su 128, non supera la verifica; AccessOps ricodifica entrambi gli interi in esattamente 33 byte prima di chiamare Validate, il che non cambia né i valori né ciò che la firma dimostra, e una firma malformata continua a fallire. Entrambe le soluzioni temporanee stanno in due piccole classi, `AuthenticatorFlags` ed `Es256Signature`, e vanno rimosse quando uscirà un'estensione corretta: entrambi i difetti sono stati segnalati il 10 novembre 2026 e sono già corretti nel codice sorgente dell'estensione; la correzione arriverà con una versione del pacchetto successiva alla 4.1.7. La policy del contatore è scritta nero su bianco: molti autenticatori riportano sempre zero, quindi zero seguito da zero viene accettato, mentre un contatore diverso da zero che non aumenta viene rifiutato e registrato come anomalia. L'estensione espone `BackupEligibility` ma nessun flag sullo stato del backup, quindi AccessOps registra l'idoneità e non prende alcuna decisione che dipenda dal fatto che una passkey sia sincronizzata in quel momento. Solo quando il commit restituisce Updated viene eseguito il passaggio del Modulo 4: parcheggia l'account verificato, ricarica, chiama `Application.RefreshSessionId()` al caricamento, imposta `Application.User` e crea esattamente un grant di login, che ogni lettura protetta continua a controllare.

Un archivio delle credenziali che rende definitiva la revoca

Il `PasskeyCredentialStore` è una classe di AccessOps su un'unica tabella con un vincolo di unicità su RP ID e ID della credenziale, così che una credenziale non possa essere registrata due volte o sotto un secondo account. Il suo commit è un'unica istruzione condizionale: aggiorna il contatore e l'ora dell'ultimo utilizzo dove RP, ID della credenziale, proprietario, identità della riga e versione corrispondono tutti a ciò che il verificatore ha letto. Una riga interessata significa Updated. Zero righe vengono classificate, mai ritentate alla cieca: NotFound se la riga non c'è più, altrimenti Conflict. Una credenziale eliminata mentre un accesso era in corso resta eliminata, e una registrata di nuovo con lo stesso ID riceve una nuova identità di riga, così che un aggiornamento obsoleto non possa finirci sopra. Leggere, confrontare in C# e scrivere in un secondo passo non protetto è esattamente la race condition da evitare.

La pagina di revisione elenca, rinomina ed elimina solo le credenziali dell'utente che ha effettuato l'accesso, prendendo il proprietario dal principal della sessione, mai da una cella della griglia. Il recupero dopo la perdita di un dispositivo è un piano scritto: una seconda passkey registrata, oppure un accesso tramite il Modulo 4 o il Modulo 5 più un passo di verifica recente prima che venga registrata una nuova credenziale, la credenziale persa eliminata e ogni modifica scritta nello storico dell'account che il supporto può verificare. Il recupero non aggira mai la regola della sessione verificata.

Regola di sicurezza

Tieni separate le identità e decidi sul server. Il principal dell'host, il principal applicativo di Wisej.NET e il token Windows effettivo sono tre fatti diversi, e nessuno di essi è un permesso. Ogni lettura protetta controlla il grant corrente, il tenant e il record al confine del servizio prima che il repository venga eseguito, e un pulsante nascosto, un claim di ruolo o una scheda disconnessa non dimostrano nulla finché quel controllo non rifiuta.

Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Scegliere l'ID della relying party e le origini esatte consentite per AccessOps, e spiegare che cosa memorizza il server — ID della credenziale, chiave pubblica, user handle — e che cosa non riceve mai.
- ✓ Eseguire la registrazione e l'autenticazione con passkey da C# lato server con l'estensione WebAuthn di Wisej.NET, avviando la registrazione solo da una sessione con un account verificato.
- ✓ Convalidare ogni assertion sul server — challenge monouso, origine, RP ID, presenza e verifica dell'utente, proprietario della credenziale e firma — e applicare una policy documentata su contatore e backup.
- ✓ Costruire un archivio delle credenziali i cui aggiornamenti usano un compare-and-swap atomico, così che una credenziale eliminata o revocata resti revocata, e pianificare il recupero in caso di perdita del dispositivo.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.

Lab pratico

Aggiungi le passkey ad AccessOps con l'estensione Wisej-4-WebAuthn su un nome host HTTPS stabile o su un contesto sicuro localhost documentato: verifica il supporto del browser, accedi come account A tramite il flusso del Modulo 4 o del Modulo 5 e registra una credenziale con etichetta con CreateAsync, poi disconnettiti e autenticati con GetAsync e Validate lato server, mappa l'account locale e ricontrolla il grant corrente prima di mostrare i record. Memorizza le credenziali in un PasskeyCredentialStore con chiave su RP ID e ID della credenziale, con un vincolo di unicità e un commit compare-and-swap che restituisce Updated, Conflict o NotFound, aggiungi una pagina di revisione delle credenziali in cui un utente connesso rinomina o elimina solo le proprie credenziali, e rifiuta la credenziale dell'account A per l'account B, un'origine non approvata, un challenge riprodotto e un prompt annullato senza alcun nuovo grant, poi leggi una credenziale due volte in un test di integrazione dell'archivio e mostra che un aggiornamento obsoleto e un'eliminazione prima del commit falliscono.

Punto di partenza

Apri la soluzione AccessOps cumulativa del Modulo 5: la route dei partner ha già l'accesso locale con il suo `GrantStore`, la federazione OpenID Connect e il passaggio tramite `ApplicationPrincipal` che ruota l'ID di sessione al caricamento di una pagina. Aggiungi il pacchetto `Wisej-4-WebAuthn` 4.1.7 al progetto AccessOps e usa le due fixture dei partner, account A e account B, nello stesso tenant.

Approccio suggerito

- ✓ Fissa prima la relying party: scrivi l'RP ID, le origini esatte consentite e il contesto sicuro (l'host HTTPS, oppure `https://localhost` con RP ID `"localhost"`), ed esegui registrazione e accesso su quello stesso host.
- ✓ Costruisci la tabella di `PasskeyCredentialStore` e il suo commit condizionale prima di qualsiasi interfaccia, e scrivi il test di integrazione dell'archivio che legge una riga due volte, esegue un commit e poi tenta l'aggiornamento obsoleto e l'eliminazione prima del commit.
- ✓ Aggiungi la registrazione solo alla schermata dell'account connesso: emetti un challenge monouso legato all'account, chiama `WebAuthn.CreateAsync` con ES256 e RS256, controlla challenge, tipo e origine, e memorizza la credenziale con etichetta.
- ✓ Aggiungi "Sign in with a passkey" alla pagina di accesso dei partner: `WebAuthn.GetAsync`, consuma il challenge, trova la chiave del proprietario con `WebAuthn.Validate` (passando la firma ES256 attraverso `Es256Signature.ForValidate`), controlla tipo, origine, hash dell'RP ID, presenza e verifica dell'utente dal byte dei flag firmato (`AuthenticatorFlags.Read`, non la proprietà `UserVerified`) e contatore, esegui il commit, poi avvia il passaggio del grant del Modulo 4.
- ✓ Esegui i cinque esiti e le due race condition dell'archivio, aggiungi la pagina di revisione delle credenziali e scrivi il piano di recupero per la perdita del dispositivo, indicando chi approva una nuova registrazione e dove viene registrata nell'audit.

Deliverable richiesti

- ✓ Registrazione e accesso con passkey tramite l'estensione WebAuthn
- ✓ Convalida delle assertion lato server con challenge monouso
- ✓ PasskeyCredentialStore con commit compare-and-swap atomici
- ✓ Cinque esiti: un accesso valido e quattro rifiuti, con i test delle race condition dell'archivio
- ✓ Pagina di revisione delle credenziali e piano di recupero dell'account

Domande di autovalutazione

- ✓ `WebAuthn.Validate` ha restituito true per una risposta il cui challenge era già stato consumato. Quale controllo la rifiuta, e che cosa avrebbe dovuto avere un attaccante per produrre quella risposta?
- ✓ Un partner elimina la passkey di un portatile smarrito mentre un accesso con quella passkey è in attesa del commit. Ripercorri che cosa restituisce l'aggiornamento condizionale e perché non viene creato alcun grant.
- ✓ Il tuo RP ID è il dominio padre e la pagina di accesso si sposta su un sotto-host. Perché il `GetAsync` di questa estensione non trova la credenziale, e che cosa cambieresti?

Regola di sicurezza

Tieni separate le identità e decidi sul server. Il principal dell'host, il principal applicativo di Wisej.NET e il token Windows effettivo sono tre fatti diversi, e nessuno di essi è un permesso. Ogni lettura protetta controlla il grant corrente, il tenant e il record al confine del servizio prima che il repository venga eseguito, e un pulsante nascosto, un claim di ruolo o una scheda disconnessa non dimostrano nulla finché quel controllo non rifiuta.