SICHERHEITS- UND AUTHENTIFIZIERUNGSMUSTER · MODUL 6

Passkeys mit der Erweiterung WebAuthn

Lektionsleitfaden · Lab- und Prüfungsleitfaden

LearnWisej

Lektionsleitfaden

Das Design der WebAuthn-Relying-Party, die Registrierung aus einem verifizierten Konto heraus, Registrierungs- und Authentifizierungszeremonien mit der Wisej.NET-Erweiterung WebAuthn, die serverseitige Validierung jeder Assertion, ein nebenläufigkeitssicherer Credential-Speicher und die Kontowiederherstellung. Was das für Entwicklerinnen und Entwickler bedeutet, die die Anmelde-, Identitäts- und Autorisierungspfade einer Wisej.NET-Anwendung im Produktivbetrieb absichern – und wie Sie dieses Modul lesen.

Fokus: Produktivbetrieb

Das Design der WebAuthn-Relying-Party, die Registrierung aus einem verifizierten Konto heraus, Registrierungs- und Authentifizierungszeremonien mit der Wisej.NET-Erweiterung WebAuthn, die serverseitige Validierung jeder Assertion, ein nebenläufigkeitssicherer Credential-Speicher und die Kontowiederherstellung.

1. Das Problem im Produktivbetrieb

Partner von AccessOps melden sich bereits mit einem lokalen Konto (Modul 4) oder über ihren eigenen Identitätsanbieter (Modul 5) an. Ein Passkey ergänzt ein Credential, das sich weder phishen noch auf einer anderen Site wiederverwenden lässt – aber nur, wenn der Server alles prüft, was der Browser zurückgibt. Die Wisej.NET-Erweiterung WebAuthn führt die Zeremonien im Browser aus und prüft eine Signatur; Challenge, Origin, Relying Party, Besitzer, Zähler und Credential-Speicher sind AccessOps-Code, und jedes davon ist eine Stelle, an der eine Anmeldung fälschlich akzeptiert werden kann.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Relying-Party-ID	Die Domäne ohne Schema und Port, an die ein Credential gebunden ist; sie später zu ändern zwingt jeden Benutzer zu einer neuen Registrierung.
Origin-Allowlist	Die exakten Werte aus Schema, Host und Port, die in `ClientData.Origin` akzeptiert werden; alles andere wird abgewiesen, selbst mit gültiger Signatur.
Credential-ID und öffentlicher Schlüssel	Was der Server pro Credential speichert; der private Schlüssel und jede biometrische Probe verlassen den Authenticator nie.
User Handle	Eine opake, stabile Kontokennung, die bei der Registrierung als `id` der Benutzerentität gesendet und als `UserHandle` zurückgegeben wird.
einmalige Challenge	Zufällige Bytes pro Zeremonie, gebunden an Typ, Konto und Ablaufzeit, verglichen mit `ClientData.Challenge` und verbraucht, sodass eine abgefangene Antwort nicht zweimal verwendbar ist.
Assertion-Richtlinie	Die Prüfungen rund um `WebAuthn.Validate`, das nur beweist, dass die Signatur zu einem gespeicherten öffentlichen Schlüssel passt.
Compare-and-Swap-Commit	Ein einziges bedingtes Update, das eine verifizierte Anmeldung nur übernimmt, wenn die Zeile noch die gelesene Version hat, und Updated, Conflict oder NotFound zurückgibt.

3. Der rote Faden der Lektion

Eine Relying Party, exakte Origins und was der Server behält

Ein WebAuthn-Credential gehört zu einer Relying-Party-ID, einer bloßen Domäne wie `partners.accessops.example`. Ein Origin ist enger gefasst: `https://partners.accessops.example`, einschließlich Schema, Host und Port. AccessOps registriert mit einer RP-ID, die dem Host der Partner-Anmeldung entspricht, und führt eine Allowlist exakter Origins, kein Muster. Für das Lab ist der dokumentierte sichere Kontext `https://localhost` auf dem Lab-Port mit der RP-ID `"localhost"`; überall sonst lehnt WebAuthn einfaches HTTP ab. Die Wahl des Hosts ist bei dieser Erweiterung wichtig: `WebAuthn.GetAsync` übergibt dem Browser keine RP-ID, der Browser verwendet also die Domäne der Seite selbst, und ein für eine übergeordnete Domäne registriertes Credential wird auf einem Sub-Host womöglich nicht angeboten. Registrieren und melden Sie sich auf demselben Host an.

Der Server behält pro Credential: die Credential-ID, den serialisierten öffentlichen Schlüssel (`JSON.Stringify(publicKey)`, wiederhergestellt mit `new PublicKey(json)`), den Algorithmus, die RP-ID, das Konto und sein User Handle, den letzten Signaturzähler, das Flag `BackupEligibility`, eine Bezeichnung und Zeitstempel. AccessOps erzeugt das User Handle selbst, nie eine E-Mail-Adresse, und hält es in ASCII: Der Client sendet die `id` mit einem Byte pro Zeichen und dekodiert `UserHandle` als UTF-8. Die Anwendung erhält nie einen privaten Schlüssel, einen Fingerabdruck oder ein Gesichtsbild.

Zwei Zeremonien aus serverseitigem C#

Das Paket ist `Wisej-4-WebAuthn` 4.1.7, Namespace `Wisej.Ext.WebAuthn`. Es ist eine einzige statische Klasse mit vier Methoden: keine Komponenten, keine Ereignisse, kein Speicher, keine Verwaltung von Challenges. `IsUserVerifyingPlatformAuthenticatorAvailableAsync` meldet Windows Hello, Touch ID oder Face ID; ein Sicherheitsschlüssel kann trotzdem funktionieren, wenn es false liefert. Die Registrierung ist `CreateAsync` mit sieben Pflichtargumenten: eine Standard-Base64-Challenge aus mindestens 16 Zufallsbytes, eine `RelyingParty`, eine `PublicKeyCredentialUserEntity`, die Algorithmen, ein `AuthenticatorSelectionCriteria`, ein Timeout in Millisekunden und eine `AttestationConveyancePreference` (`None` in AccessOps). Bieten Sie nur `COSEAlgorithmIdentifier.ES256` und `RS256` an: `Validate` liefert für die anderen ECDSA-Kurven false und wirft bei EdDSA eine Ausnahme. Wie dieses Paket den Attachment-Wert serialisiert, ist nicht geprüft; AccessOps verlässt sich deshalb nicht darauf, um Authenticators einzuschränken.

Die Registrierung beginnt nur aus einer verifizierten Konto-Session: Der Partner hat sich kürzlich über Modul 4 oder Modul 5 angemeldet, und der aktuelle Grant besteht. `CreateAsync` führt keine Attestation-Prüfung durch und befüllt weder die Authenticator-Flags noch den RP-ID-Hash; die Antwort beweist also wenig über die Person – das tut die Session. AccessOps vergleicht die Challenge, prüft, dass `ClientData.Type` gleich `"webauthn.create"` ist und der Origin auf der Allowlist steht, und speichert dann `AuthenticatorData.PublicKey`. Beide asynchronen Aufrufe werfen bei Abbruch, Timeout oder Abweichung eine `Exception` mit der Meldung des Browsers; sie stehen deshalb in `try` / `catch`, und der Bildschirm zeigt festen Text mit `allowHtml: false`: `AlertBox` und `MessageBox` setzen es standardmäßig auf true.

Jede Assertion auf dem Server validieren

Die Anmeldung ruft `GetAsync` mit einer neuen Challenge auf. Wird für die erlaubten Credentials `null` übergeben, kann der Partner einen auffindbaren Passkey wählen, und `UserHandle` benennt dann das Konto; ein Formular, das zuerst das Konto erfragt, übergibt stattdessen die Liste der `PublicKeyCredentialDescriptor` dieses Kontos. Die Antwort sagt nicht, welches Credential signiert hat; AccessOps probiert deshalb die für dieses User Handle unter dieser RP gespeicherten Schlüssel und prüft alles andere selbst:

```
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
```

Die Challenge wird zuerst verbraucht, sodass auch ein gescheiterter Versuch sie verbrennt. `GetAsync` sendet keine Option für die Benutzerverifizierung, es gilt also der Standard des Browsers, und die Verifizierung des Benutzers muss geprüft statt angenommen werden. Zwei Defekte der Erweiterung in Version 4.1.7 bestimmen, wie. Ihr Client-Skript setzt `AuthenticatorData.UserVerified` aus Bit 1 des Flags-Bytes, einem reservierten Bit, das immer null ist; die Eigenschaft bleibt also false, selbst wenn eine Windows-Hello-Abfrage die Person verifiziert hat. AccessOps liest das Flags-Byte deshalb selbst aus `AuthenticatorData.Base64` (Byte 32 der signierten Authenticator-Daten: Bit 0 Benutzer anwesend, Bit 2 Benutzer verifiziert, Bit 3 sicherungsfähig) und nutzt die Eigenschaft nur als Gegenprobe für die Anwesenheit. Der zweite Defekt steckt in `WebAuthn.Validate`: Die Methode liest die DER-codierte ES256-Signatur über feste Offsets, sodass eine echte Signatur, deren Wert `r` oder `s` kürzer als 32 Bytes ist, etwa jede 128., die Prüfung nicht besteht. AccessOps codiert beide Ganzzahlen vor dem Aufruf von Validate auf genau 33 Bytes um; die Werte ändern sich nicht, also ändert sich nichts an dem, was die Signatur beweist, und eine fehlerhafte Signatur scheitert weiterhin. Beide Umgehungen liegen in zwei kleinen Klassen, `AuthenticatorFlags` und `Es256Signature`, und sollten entfernt werden, sobald eine korrigierte Erweiterung erscheint: Beide Defekte wurden am 10. November 2026 gemeldet und im Quellcode der Erweiterung behoben; die Korrektur erscheint mit einer Paketversion nach 4.1.7. Die Zählerrichtlinie ist schriftlich festgelegt: Viele Authenticators melden immer null, daher wird null gefolgt von null akzeptiert, während ein Zähler ungleich null, der nicht steigt, abgewiesen und als Anomalie festgehalten wird. Die Erweiterung stellt `BackupEligibility` bereit, aber kein Flag für den Backup-Zustand; AccessOps hält die Eignung fest und trifft keine Entscheidung, die davon abhängt, ob ein Passkey gerade synchronisiert ist. Erst wenn der Commit Updated liefert, läuft die Übergabe aus Modul 4: verifiziertes Konto ablegen, neu laden, beim Laden `Application.RefreshSessionId()` aufrufen, `Application.User` setzen und genau einen Anmelde-Grant erzeugen, den jeder geschützte Lesezugriff weiterhin prüft.

Ein Credential-Speicher, der den Entzug endgültig hält

Der `PasskeyCredentialStore` ist eine eigene Klasse von AccessOps über einer Tabelle mit einem Unique Constraint auf RP-ID und Credential-ID, sodass ein Credential weder zweimal noch unter einem zweiten Konto registriert werden kann. Sein Commit ist eine einzige bedingte Anweisung: Zähler und Zeitpunkt der letzten Nutzung aktualisieren, wo RP, Credential-ID, Besitzer, Zeilenidentität und Version mit dem übereinstimmen, was der Verifier gelesen hat. Eine betroffene Zeile bedeutet Updated. Null Zeilen werden eingeordnet, nie blind wiederholt: NotFound, wenn die Zeile weg ist, sonst Conflict. Ein Credential, das während einer laufenden Anmeldung gelöscht wurde, bleibt gelöscht, und eines, das mit derselben ID neu registriert wurde, erhält eine neue Zeilenidentität, sodass ein veraltetes Update nicht darauf landen kann. In C# zu lesen, zu vergleichen und in einem zweiten, ungeschützten Schritt zu schreiben ist genau die Race Condition, die es zu vermeiden gilt.

Die Übersichtsseite listet, benennt um und löscht nur die eigenen Credentials des angemeldeten Benutzers und nimmt den Besitzer aus dem Session-Principal, nie aus einer Grid-Zelle. Die Wiederherstellung nach einem verlorenen Gerät ist ein schriftlicher Plan: ein zweiter registrierter Passkey oder eine Anmeldung über Modul 4 oder Modul 5 plus ein Schritt zur erneuten Verifizierung, bevor ein neues Credential registriert wird; das verlorene Credential wird gelöscht und jede Änderung in die Kontohistorie geschrieben, die der Support prüfen kann. Die Wiederherstellung umgeht nie die Regel der verifizierten Session.

Sicherheitsregel

Halten Sie die Identitäten auseinander und entscheiden Sie auf dem Server. Der Host-Principal, der Wisej.NET-Anwendungs-Principal und das effektive Windows-Token sind drei verschiedene Fakten, und keines davon ist eine Berechtigung. Jeder geschützte Lesezugriff prüft den aktuellen Grant, den Mandanten und den Datensatz in der Service-Schicht, bevor das Repository läuft, und ein ausgeblendeter Button, ein Rollen-Claim oder ein abgemeldeter Tab beweist nichts, solange diese Prüfung nicht ablehnt.

Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Die Relying-Party-ID und die exakten erlaubten Origins für AccessOps wählen und erklären, was der Server speichert – Credential-ID, öffentlicher Schlüssel, User Handle – und was er nie erhält.
- ✓ Registrierung und Authentifizierung mit Passkeys aus serverseitigem C# mit der Wisej.NET-Erweiterung WebAuthn ausführen und die Registrierung nur aus einer verifizierten Konto-Session beginnen.
- ✓ Jede Assertion auf dem Server validieren – einmalige Challenge, Origin, RP-ID, Anwesenheit und Verifizierung des Benutzers, Besitzer des Credentials und Signatur – und eine dokumentierte Richtlinie für Zähler und Backup anwenden.
- ✓ Einen Credential-Speicher bauen, dessen Updates einen atomaren Compare-and-Swap verwenden, sodass ein gelöschtes oder entzogenes Credential entzogen bleibt, und die Wiederherstellung nach Geräteverlust planen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.

Praktisches Lab

Ergänzen Sie AccessOps mit der Erweiterung Wisej-4-WebAuthn um Passkeys, auf einem stabilen HTTPS-Hostnamen oder in einem dokumentierten sicheren Kontext auf localhost: Prüfen Sie die Browserunterstützung, melden Sie sich als Konto A über den Ablauf aus Modul 4 oder Modul 5 an und registrieren Sie mit CreateAsync ein Credential mit Bezeichnung, melden Sie sich dann ab und authentifizieren Sie sich mit GetAsync und serverseitigem Validate, ordnen Sie das lokale Konto zu und prüfen Sie den aktuellen Grant erneut, bevor Datensätze angezeigt werden. Speichern Sie Credentials in einem PasskeyCredentialStore mit dem Schlüssel aus RP-ID und Credential-ID, einem Unique Constraint und einem Compare-and-Swap-Commit, der Updated, Conflict oder NotFound zurückgibt, ergänzen Sie eine Übersichtsseite, auf der ein angemeldeter Benutzer nur seine eigenen Credentials umbenennt oder löscht, und weisen Sie das Credential von Konto A für Konto B, einen nicht genehmigten Origin, eine wiederholte Challenge und eine abgebrochene Abfrage ohne neuen Grant ab; lesen Sie dann in einem Integrationstest des Speichers ein Credential zweimal und zeigen Sie, dass ein veraltetes Update und ein Löschen vor dem Commit scheitern.

Ausgangspunkt

Öffnen Sie die kumulative AccessOps-Projektmappe aus Modul 5: Die Partner-Route hat bereits die lokale Anmeldung mit ihrem `GrantStore`, die OpenID-Connect-Föderation und die Übergabe über `ApplicationPrincipal`, die die Session-ID beim Laden einer Seite rotiert. Fügen Sie dem AccessOps-Projekt das Paket `Wisej-4-WebAuthn` 4.1.7 hinzu und verwenden Sie die beiden Partner-Fixtures, Konto A und Konto B, im selben Mandanten.

Vorgeschlagenes Vorgehen

- ✓ Legen Sie zuerst die Relying Party fest: Notieren Sie die RP-ID, die exakten erlaubten Origins und den sicheren Kontext (den HTTPS-Host oder `https://localhost` mit der RP-ID `"localhost"`) und führen Sie Registrierung und Anmeldung auf genau diesem Host aus.
- ✓ Bauen Sie die Tabelle von `PasskeyCredentialStore` und ihren bedingten Commit vor jeder Oberfläche, und schreiben Sie den Integrationstest des Speichers, der eine Zeile zweimal liest, einmal committet und dann das veraltete Update und das Löschen vor dem Commit versucht.
- ✓ Ergänzen Sie die Registrierung nur auf dem Bildschirm des angemeldeten Kontos: Stellen Sie eine an das Konto gebundene einmalige Challenge aus, rufen Sie `WebAuthn.CreateAsync` mit ES256 und RS256 auf, prüfen Sie Challenge, Typ und Origin und speichern Sie das Credential mit Bezeichnung.
- ✓ Ergänzen Sie „Sign in with a passkey“ auf der Anmeldeseite für Partner: `WebAuthn.GetAsync`, die Challenge verbrauchen, den Schlüssel des Besitzers mit `WebAuthn.Validate` finden (die ES256-Signatur zuvor durch `Es256Signature.ForValidate` geführt), Typ, Origin, RP-ID-Hash, Anwesenheit und Verifizierung des Benutzers aus dem signierten Flags-Byte (`AuthenticatorFlags.Read`, nicht die Eigenschaft `UserVerified`) sowie den Zähler prüfen, committen und dann die Grant-Übergabe aus Modul 4 ausführen.

- ✓ Führen Sie die fünf Ergebnisse und die beiden Race Conditions im Speicher aus, ergänzen Sie die Übersichtsseite für Credentials und schreiben Sie den Wiederherstellungsplan für Geräteverlust, mit der Angabe, wer eine erneute Registrierung genehmigt und wo sie protokolliert wird.

Geforderte Ergebnisse

- ✓ Registrierung und Anmeldung mit Passkeys über die Erweiterung WebAuthn
- ✓ Serverseitige Validierung von Assertions mit einmaligen Challenges
- ✓ PasskeyCredentialStore mit atomaren Compare-and-Swap-Commits
- ✓ Fünf Ergebnisse: eine gültige Anmeldung und vier Ablehnungen, mit Race-Tests für den Speicher
- ✓ Übersichtsseite für Credentials und Plan zur Kontowiederherstellung

Wiederholungsfragen

- ✓ `WebAuthn.Validate` hat für eine Antwort, deren Challenge bereits verbraucht war, true geliefert. Welche Prüfung weist sie ab, und was hätte ein Angreifer gebraucht, um diese Antwort zu erzeugen?
- ✓ Ein Partner löscht den Passkey eines verlorenen Laptops, während eine Anmeldung damit auf den Commit wartet. Gehen Sie durch, was das bedingte Update zurückgibt und warum kein Grant entsteht.
- ✓ Ihre RP-ID ist die übergeordnete Domäne, und die Anmeldeseite zieht auf einen Sub-Host um. Warum findet `GetAsync` dieser Erweiterung das Credential nicht, und was würden Sie ändern?

Sicherheitsregel

Halten Sie die Identitäten auseinander und entscheiden Sie auf dem Server. Der Host-Principal, der Wisej.NET-Anwendungs-Principal und das effektive Windows-Token sind drei verschiedene Fakten, und keines davon ist eine Berechtigung. Jeder geschützte Lesezugriff prüft den aktuellen Grant, den Mandanten und den Datensatz in der Service-Schicht, bevor das Repository läuft, und ein ausgeblendeter Button, ein Rollen-Claim oder ein abgemeldeter Tab beweist nichts, solange diese Prüfung nicht ablehnt.