PATRONES DE SEGURIDAD Y AUTENTICACIÓN · MÓDULO 6

Passkeys con la extensión WebAuthn

Guía de la lección · Guía del laboratorio / examen

LearnWisej

Guía de la lección

El diseño de la parte de confianza de WebAuthn, el registro a partir de una cuenta verificada, las ceremonias de registro y de autenticación con la extensión WebAuthn de Wisej.NET, la validación en el servidor de cada aserción, un almacén de credenciales seguro frente a la concurrencia y la recuperación de la cuenta. Qué significa esto para un desarrollador que protege los caminos de inicio de sesión, identidad y autorización de una aplicación Wisej.NET en producción — y cómo leer este módulo.

Enfoque de producción

El diseño de la parte de confianza de WebAuthn, el registro a partir de una cuenta verificada, las ceremonias de registro y de autenticación con la extensión WebAuthn de Wisej.NET, la validación en el servidor de cada aserción, un almacén de credenciales seguro frente a la concurrencia y la recuperación de la cuenta.

1. El problema en producción

Los socios de AccessOps ya inician sesión con una cuenta local (Módulo 4) o con su propio proveedor de identidad (Módulo 5). Una passkey añade una credencial que no se puede capturar por phishing ni reutilizar en otro sitio, pero solo si el servidor comprueba todo lo que le devuelve el navegador. La extensión WebAuthn de Wisej.NET ejecuta las ceremonias del navegador y verifica una firma; el desafío, el origen, la parte de confianza, el propietario, el contador y el almacén de credenciales son código de AccessOps, y cada uno es un punto en el que se puede aceptar por error un inicio de sesión.

2. Conceptos clave

Concepto	Por qué importa
ID de la parte de confianza	El dominio, sin esquema ni puerto, al que se limita una credencial; cambiarlo más tarde obliga a todos los usuarios a registrarse de nuevo.
lista de orígenes permitidos	Los valores exactos de esquema, host y puerto que se aceptan en `ClientData.Origin`; cualquier otro se rechaza aunque la firma sea válida.
ID de credencial y clave pública	Lo que el servidor guarda por cada credencial; la clave privada y cualquier muestra biométrica nunca salen del autenticador.
identificador de usuario	Un identificador de cuenta opaco y estable que se envía como `id` de la entidad de usuario en el registro y vuelve como `UserHandle`.
desafío de un solo uso	Bytes aleatorios por ceremonia, ligados a su tipo, a la cuenta y a una caducidad, comparados con `ClientData.Challenge` y consumidos para que una respuesta capturada no pueda usarse dos veces.
política de aserción	Las comprobaciones alrededor de `WebAuthn.Validate`, que solo demuestra que la firma coincide con una clave pública guardada.
confirmación con comparar e intercambiar	Una única actualización condicional que aplica un inicio de sesión verificado solo si la fila sigue teniendo la versión que se leyó, y devuelve Updated, Conflict o NotFound.

3. Desarrollo de la lección

Una parte de confianza, orígenes exactos y lo que guarda el servidor

Una credencial WebAuthn pertenece a un ID de parte de confianza, un dominio sin más, como `partners.accessops.example`. Un origen es más estrecho: `https://partners.accessops.example`, con esquema, host y puerto incluidos. AccessOps registra con un ID de RP igual al host de inicio de sesión de socios y mantiene una lista de orígenes exactos, no un patrón. Para el laboratorio, el contexto seguro documentado es `https://localhost` en el puerto del laboratorio con el ID de RP `"localhost"`; WebAuthn rechaza HTTP sin cifrar en cualquier otro lugar. La elección del host importa en esta extensión: `WebAuthn.GetAsync` no pasa ningún ID de RP al navegador, así que el navegador usa el dominio de la propia página, y una credencial registrada para un dominio padre puede no ofrecerse en un subhost. Registra e inicia sesión en el mismo host.

El servidor guarda, por cada credencial: el ID de credencial, la clave pública serializada (`JSON.Stringify(publicKey)`, restaurada con `new PublicKey(json)`), el algoritmo, el ID de RP, la cuenta y su identificador de usuario, el último contador de firmas, el indicador `BackupEligibility`, una etiqueta y las marcas de tiempo. AccessOps genera el identificador de usuario, nunca una dirección de correo, y lo mantiene en ASCII: el cliente envía el `id` a un byte por carácter y descodifica `UserHandle` como UTF-8. La aplicación nunca recibe una clave privada, una huella dactilar ni una imagen de la cara.

Dos ceremonias desde C# en el servidor

El paquete es `Wisej-4-WebAuthn` 4.1.7, espacio de nombres `Wisej.Ext.WebAuthn`. Es una única clase estática con cuatro métodos: sin componentes, eventos, almacén ni gestión de desafíos. `IsUserVerifyingPlatformAuthenticatorAvailableAsync` informa de Windows Hello, Touch ID o Face ID; una llave de seguridad puede seguir funcionando cuando devuelve false. El registro es `CreateAsync` con siete argumentos obligatorios: un desafío en base64 estándar de al menos 16 bytes aleatorios, un `RelyingParty`, un `PublicKeyCredentialUserEntity`, los algoritmos, un `AuthenticatorSelectionCriteria`, un tiempo de espera en milisegundos y un `AttestationConveyancePreference` (`None` en AccessOps). Ofrece solo `COSEAlgorithmIdentifier.ES256` y `RS256`: `Validate` devuelve false para las demás curvas ECDSA y lanza una excepción con EdDSA. No está verificado cómo serializa este paquete el valor de vinculación del autenticador, así que AccessOps no depende de él para restringir autenticadores.

El registro solo empieza desde una sesión de cuenta verificada: el socio inició sesión hace poco mediante el Módulo 4 o el Módulo 5 y la concesión vigente es válida. `CreateAsync` no realiza ninguna comprobación de atestación ni rellena los indicadores del autenticador ni el hash del ID de RP, así que la respuesta demuestra poco sobre la persona; lo demuestra la sesión. AccessOps compara el desafío, comprueba que `ClientData.Type` es `"webauthn.create"` y que el origen está en la lista de permitidos, y después guarda `AuthenticatorData.PublicKey`. Ambas llamadas asíncronas lanzan una `Exception` con el mensaje del navegador al cancelar, al agotar el tiempo o ante una discrepancia, así que van dentro de `try` / `catch` y la pantalla muestra un texto fijo con `allowHtml: false`: `AlertBox` y `MessageBox` lo ponen a true de forma predeterminada.

Valida cada aserción en el servidor

El inicio de sesión llama a `GetAsync` con un desafío nuevo. Pasar `null` como credenciales permitidas deja que el socio elija una passkey detectable, y entonces `UserHandle` nombra la cuenta; un formulario que pide primero la cuenta pasa en su lugar la lista de `PublicKeyCredentialDescriptor` de esa cuenta. La respuesta no dice qué credencial firmó, así que AccessOps prueba las claves guardadas para ese identificador de usuario bajo esta RP y comprueba todo lo demás por sí mismo:

```
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
```

El desafío se consume primero, así que un intento fallido también lo gasta. `GetAsync` no envía ninguna opción de verificación del usuario, así que se aplica el valor predeterminado del navegador y la verificación del usuario debe comprobarse en lugar de suponerse. Dos defectos de la extensión 4.1.7 deciden cómo. Su script de cliente asigna `AuthenticatorData.UserVerified` a partir del bit 1 del byte de indicadores, un bit reservado que siempre vale cero, por lo que la propiedad es false incluso cuando un aviso de Windows Hello verificó a la persona; por eso AccessOps lee el byte de indicadores por su cuenta desde `AuthenticatorData.Base64` (el byte 32 de los datos firmados del autenticador: bit 0 usuario presente, bit 2 usuario verificado, bit 3 apto para copia de seguridad) y usa la propiedad solo como comprobación cruzada de la presencia. El segundo defecto está en `WebAuthn.Validate`: lee la firma ES256 codificada en DER por desplazamientos fijos, así que una firma auténtica cuyo valor `r` o `s` sea más corto que 32 bytes, aproximadamente una de cada 128, no supera la verificación; AccessOps vuelve a codificar ambos enteros en exactamente 33 bytes antes de llamar a Validate, lo que no cambia los valores ni lo que la firma demuestra, y una firma malformada sigue fallando. Ambas soluciones provisionales viven en dos clases pequeñas, `AuthenticatorFlags` y `Es256Signature`, y deben retirarse cuando se publique una extensión corregida: ambos defectos se notificaron el 10 de noviembre de 2026 y ya están corregidos en el código fuente de la extensión; la corrección llegará en una versión del paquete posterior a la 4.1.7. La política del contador está escrita: muchos autenticadores siempre informan de cero, así que se acepta cero seguido de cero, mientras que un contador distinto de cero que no aumenta se rechaza y se registra como anomalía. La extensión expone `BackupEligibility` pero ningún indicador de estado de copia de seguridad, así que AccessOps registra la elegibilidad y no toma ninguna decisión que dependa de si una passkey está sincronizada en ese momento. Solo cuando la confirmación devuelve Updated se ejecuta la entrega del Módulo 4: aparcar la cuenta verificada, recargar, llamar a `Application.RefreshSessionId()` en la carga, asignar `Application.User` y crear exactamente una concesión de inicio de sesión que cada lectura protegida sigue comprobando.

Un almacén de credenciales que mantiene definitiva la revocación

El `PasskeyCredentialStore` es una clase propia de AccessOps sobre una tabla con una restricción única sobre el ID de RP y el ID de credencial, de modo que una credencial no puede registrarse dos veces ni bajo una segunda cuenta. Su confirmación es una única sentencia condicional: actualizar el contador y la hora del último uso donde la RP, el ID de credencial, el propietario, la identidad de la fila y la versión coincidan con lo que leyó el verificador. Una fila afectada es Updated. Cero filas se clasifica, nunca se reintenta a ciegas: NotFound si la fila ya no existe y, si no, Conflict. Una credencial eliminada mientras había un inicio de sesión en curso sigue eliminada, y una que se vuelve a registrar con el mismo ID recibe una nueva identidad de fila, así que una actualización obsoleta no puede caer sobre ella. Leer, comparar en C# y escribir en un segundo paso sin protección es exactamente la carrera que hay que evitar.

La página de revisión lista, renombra y elimina solo las credenciales propias del usuario con sesión iniciada, tomando el propietario del principal de la sesión, nunca de una celda de la cuadrícula. La recuperación tras perder un dispositivo es un plan escrito: una segunda passkey registrada, o un inicio de sesión del Módulo 4 o del Módulo 5 más un paso de verificación reciente antes de registrar una credencial nueva, la eliminación de la credencial perdida y cada cambio escrito en el historial de la cuenta que soporte puede auditar. La recuperación nunca se salta la regla de la sesión verificada.

Regla de seguridad

Mantén separadas las identidades y decide en el servidor. El principal del host, el principal de la aplicación Wisej.NET y el token efectivo de Windows son tres hechos distintos, y ninguno de ellos es un permiso. Cada lectura protegida comprueba la concesión vigente, el tenant y el registro en el límite del servicio antes de que se ejecute el repositorio, y un botón oculto, un claim de rol o una pestaña con la sesión cerrada no demuestran nada hasta que esa comprobación deniega.

Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Elegir el ID de la parte de confianza y los orígenes permitidos exactos para AccessOps, y explicar lo que guarda el servidor —ID de credencial, clave pública, identificador de usuario— y lo que nunca recibe.
- ✓ Ejecutar el registro y la autenticación con passkeys desde C# en el servidor con la extensión WebAuthn de Wisej.NET, empezando el registro solo desde una sesión de cuenta verificada.
- ✓ Validar cada aserción en el servidor —desafío de un solo uso, origen, ID de RP, presencia y verificación del usuario, propietario de la credencial y firma— y aplicar una política documentada de contador y de copia de seguridad.
- ✓ Construir un almacén de credenciales cuyas actualizaciones usen una operación atómica de comparar e intercambiar para que una credencial eliminada o revocada siga revocada, y planificar la recuperación ante la pérdida de un dispositivo.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.

Laboratorio práctico

Añade passkeys a AccessOps con la extensión Wisej-4-WebAuthn en un nombre de host HTTPS estable o en un contexto seguro localhost documentado: comprueba la compatibilidad del navegador, inicia sesión como la cuenta A mediante el flujo del Módulo 4 o del Módulo 5 y registra una credencial con etiqueta con CreateAsync; después cierra la sesión y autentícate con GetAsync y Validate en el servidor, asigna la cuenta local y vuelve a comprobar la concesión vigente antes de mostrar registros. Guarda las credenciales en un PasskeyCredentialStore con clave por ID de RP e ID de credencial, con una restricción única y una confirmación con comparar e intercambiar que devuelve Updated, Conflict o NotFound, añade una página de revisión de credenciales en la que un usuario con sesión iniciada renombre o elimine solo sus propias credenciales, y rechaza la credencial de la cuenta A para la cuenta B, un origen no aprobado, un desafío reutilizado y un aviso cancelado sin crear ninguna concesión nueva; después lee una credencial dos veces en una prueba de integración del almacén y muestra que fallan una actualización obsoleta y una eliminación antes de la confirmación.

Punto de partida

Abre la solución acumulativa de AccessOps del Módulo 5: la ruta de socios ya tiene el inicio de sesión local con su `GrantStore`, la federación de OpenID Connect y la entrega de `ApplicationPrincipal` que rota el ID de sesión en una carga de página. Añade el paquete `Wisej-4-WebAuthn` 4.1.7 al proyecto AccessOps y usa los dos fixtures de socios, la cuenta A y la cuenta B, en el mismo tenant.

Enfoque sugerido

- ✓ Fija primero la parte de confianza: anota el ID de RP, los orígenes permitidos exactos y el contexto seguro (el host HTTPS, o `https://localhost` con el ID de RP `"localhost"`), y ejecuta el registro y el inicio de sesión en ese mismo host.
- ✓ Construye la tabla de `PasskeyCredentialStore` y su confirmación condicional antes que cualquier interfaz, y escribe la prueba de integración del almacén que lee una fila dos veces, confirma una vez e intenta después la actualización obsoleta y la eliminación antes de la confirmación.
- ✓ Añade el registro solo a la pantalla de la cuenta con sesión iniciada: emite un desafío de un solo uso ligado a la cuenta, llama a `WebAuthn.CreateAsync` con ES256 y RS256, comprueba el desafío, el tipo y el origen, y guarda la credencial con su etiqueta.
- ✓ Añade "Sign in with a passkey" a la página de inicio de sesión de socios: `WebAuthn.GetAsync`, consume el desafío, encuentra la clave del propietario con `WebAuthn.Validate` (pasando la firma ES256 por `Es256Signature.ForValidate`), comprueba el tipo, el origen, el hash del ID de RP, la presencia y la verificación del usuario a partir del byte de indicadores firmado (`AuthenticatorFlags.Read`, no la propiedad `UserVerified`) y el contador, confirma y ejecuta después la entrega de la concesión del Módulo 4.

- ✓ Ejecuta los cinco resultados y las dos carreras del almacén, añade la página de revisión de credenciales y escribe el plan de recuperación ante la pérdida de un dispositivo, con quién aprueba un nuevo registro y dónde se audita.

Entregables requeridos

- ✓ Registro e inicio de sesión con passkeys mediante la extensión WebAuthn
- ✓ Validación de aserciones en el servidor con desafíos de un solo uso
- ✓ PasskeyCredentialStore con confirmaciones atómicas de comparar e intercambiar
- ✓ Cinco resultados: un inicio de sesión válido y cuatro rechazos, con pruebas de carrera del almacén
- ✓ Página de revisión de credenciales y plan de recuperación de la cuenta

Preguntas de repaso

- ✓ `WebAuthn.Validate` devolvió true para una respuesta cuyo desafío ya se había consumido. ¿Qué comprobación la rechaza, y qué habría necesitado un atacante para producir esa respuesta?
- ✓ Un socio elimina la passkey de un portátil perdido mientras un inicio de sesión con ella espera la confirmación. Explica paso a paso qué devuelve la actualización condicional y por qué no se crea ninguna concesión.
- ✓ Tu ID de RP es el dominio padre y la página de inicio de sesión se traslada a un subhost. ¿Por qué el `GetAsync` de esta extensión no encuentra la credencial, y qué cambiarías?

Regla de seguridad

Mantén separadas las identidades y decide en el servidor. El principal del host, el principal de la aplicación Wisej.NET y el token efectivo de Windows son tres hechos distintos, y ninguno de ellos es un permiso. Cada lectura protegida comprueba la concesión vigente, el tenant y el registro en el límite del servicio antes de que se ejecute el repositorio, y un botón oculto, un claim de rol o una pestaña con la sesión cerrada no demuestran nada hasta que esa comprobación deniega.