PATRONES DE SEGURIDAD Y AUTENTICACIÓN · MÓDULO 1

Límites de identidad en Wisej.NET

Guía de la lección · Guía del laboratorio / examen

LearnWisej

Guía de la lección

Las tres identidades de una petición Wisej.NET —principal del host, principal de la aplicación y token efectivo de Windows—, Application.User y el ciclo de vida de la sesión, y una denegación a nivel de recurso en el límite del servicio que resiste la revocación del rol. 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

Las tres identidades de una petición Wisej.NET —principal del host, principal de la aplicación y token efectivo de Windows—, Application.User y el ciclo de vida de la sesión, y una denegación a nivel de recurso en el límite del servicio que resiste la revocación del rol.

1. El problema en producción

AccessOps atiende al personal a través de un punto de entrada Windows en la intranet y a los socios a través de un punto de entrada HTTPS público, y ambos llegan a los mismos documentos confidenciales. Cada petición lleva varias identidades a la vez, y un fallo que las confunde aparece como un tenant que lee el registro de otro tenant, o como un lector revocado que todavía puede exportar. Este módulo separa las identidades y pone la decisión donde se puede demostrar: en el límite del servicio, antes de que se ejecute el repositorio.

2. Conceptos clave

Concepto	Por qué importa
principal de la petición del host	Lo que el host autenticó para la petición o para la actualización a WebSocket: `HttpContext.User` en ASP.NET Core; anónimo en una ejecución simple con Kestrel.
principal de la aplicación	El sujeto que tu inicio de sesión asigna a la sesión Wisej.NET después de que un verificador de confianza devuelve una cuenta: ID de cuenta, tenant, ID de concesión.
identidad efectiva de Windows	El token que comprueba el sistema operativo cuando el proceso abre un archivo: la cuenta del grupo de aplicaciones o del proceso, salvo que el código suplante.
sustitución de sesión	Un valor asignado a `Application.User` en la sesión actual; prevalece sobre el principal de la petición hasta que se vuelve a poner a `null`.
concesión	Un registro del lado del servidor, revocable y con caducidad, que indica que una cuenta puede leer en un tenant; cada operación protegida lee su estado actual.
límite del servicio	El único método que cruza cada camino —clic, descarga, trabajo en cola— antes del repositorio, de modo que una sola denegación los cubre todos.
repositorio que cuenta lecturas	Un repositorio de prueba que cuenta sus lecturas, para que una denegación se demuestre con cero llamadas al repositorio en lugar de deducirse de una pantalla vacía.

3. Desarrollo de la lección

Tres identidades, y ninguna es un permiso

Sigue a `CONTOSO\reader1` por la ruta de la intranet. IIS autentica la petición, así que el principal del host es una identidad de Windows; sin embargo, cuando `DocumentService` abre un archivo, el sistema operativo comprueba el token del grupo de aplicaciones, no el de reader1, salvo que el código suplante (Módulo 3). Un socio federado (Módulo 5) tiene un principal de la aplicación válido y ningún token de Windows. La exportación nocturna no tiene ningún usuario con sesión iniciada y, aun así, la cuenta del proceso puede leer toda la carpeta de documentos.

Anótalas como tres observaciones separadas y no dejes nunca que una sustituya a otra. Ninguna es una decisión de autorización: el principal del host dice quién se conectó, un claim de rol dice lo que era cierto al iniciar sesión, el token del proceso dice lo que permite el sistema operativo. AccessOps debe responder en cada lectura protegida a una pregunta más estrecha: ¿permite la concesión vigente que esta cuenta lea este registro en este tenant, ahora mismo?

`Visible` y `Enabled` describen la pantalla. Wisej.NET sí verifica en el servidor que un clic procede de un control habilitado y visible, pero los datos enlazados a una columna oculta de la cuadrícula siguen llegando al navegador, y el mismo documento es accesible desde un controlador, una ruta de descarga y un trabajo. La regla pertenece al servicio al que todos ellos llaman.

Cómo se resuelven Application.User e IsAuthenticated

`Application.User` es un `IPrincipal` estático con un getter de dos pasos. Si se asignó un valor en esta sesión, devuelve esa sustitución de sesión; si no, devuelve el principal de la petición: en ASP.NET Core, el `HttpContext.User` capturado cuando llegó la petición o la actualización a WebSocket, es decir, un `WindowsIdentity` con la autenticación de Windows de IIS y un principal no autenticado en una ejecución de desarrollo con Kestrel. `Application.IsAuthenticated` es simplemente `User?.Identity?.IsAuthenticated == true`: no sabe nada de tenant, rol ni propiedad. Asignar `Application.User = null` elimina la sustitución y vuelve al principal del host, así que "cerrar la sesión" de un usuario de Windows de esa forma lo deja autenticado.

Asigna la sustitución solo después de que un verificador de confianza devuelva una cuenta; un `ClaimsIdentity` construido a partir de un nombre de usuario tecleado deja que el navegador elija su identidad. El ayudante `ApplicationPrincipal` del laboratorio construye los claims a partir de la cuenta verificada —ID de cuenta inmutable, tenant, ID de concesión, roles— con el tipo de autenticación `"Portal"`. Usa ese ID como clave, nunca un nombre visible ni un correo electrónico.

```
// Runs on a page load, after the test verifier returned an account
internal static void Establish(string accountId, string tenantId,
    string grantId, IEnumerable<string> roles)
{
    var claims = new List<Claim> {
        new Claim(ClaimTypes.NameIdentifier, accountId),
        new Claim("tenant", tenantId),
        new Claim("grant", grantId) };
    claims.AddRange(roles.Select(r => new Claim(ClaimTypes.Role, r)));

    Application.RefreshSessionId();   // new ID first, on the load
    Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"));
}
```

`Application.UserIdentity` solo existe en la compilación para .NET Framework (net48), donde es la identidad de inicio de sesión de System.Web; la compilación net8.0 a la que hace referencia un AccessOps en .NET 10 no la declara. En .NET 10, lee el principal de Windows a través de `Application.User` y comprueba que su `Identity` es un `WindowsIdentity` antes de confiar en él.

El ciclo de vida de la sesión y el trabajo fuera de ella

La primera carga HTTP construye la interfaz, los eventos posteriores llegan por el WebSocket y cada transición de la sesión es un evento propio. `ApplicationStart` se dispara para una sesión nueva después de `Program.Main`; `ApplicationRefresh` se dispara al recargar el navegador o al cambiar la URL en la misma sesión; `SessionTimeout` anuncia la caducidad por inactividad (establece `Handled` para suprimir el diálogo de prórroga predeterminado); `ApplicationExit` se dispara cuando la sesión se termina o se elimina, y es el único evento que se produce al caducar. Úsalo para limpiar, nunca como revocación: eso vive en el almacén de concesiones.

`Application.RefreshSessionId` emite un nuevo ID de sesión sin perder la sesión, lo que cierra la fijación de sesión tras el inicio de sesión, pero solo en una carga de página. Desde un clic por WebSocket, el servidor cambia el ID, el navegador nunca adopta el nuevo y su siguiente petición inicia una sesión nueva.

Por eso AccessOps verifica en el clic, aparca la cuenta bajo una clave pendiente, llama a `Application.Reload()` y, en `ApplicationRefresh`, rota primero el ID y después establece el principal. Oculta `Application.SessionId` en las trazas.

Fuera de la petición, la identidad ambiental falla. `Application.StartTask` restaura la sesión en su hilo del grupo; un `Task.Run` directo o la devolución de llamada de un temporizador no tienen ninguna, y `Application.Update` solo restaura una a partir de un `Application.Current` guardado. Así que la exportación en segundo plano recibe un sujeto explícito —ID de cuenta, tenant, ID de concesión, ID de registro— y `DocumentService` vuelve a comprobar la concesión cuando arranca el trabajo. Si dos perfiles del navegador muestran la misma cuenta, busca un campo estático que guarde el principal.

La denegación en el límite del servicio

`DocumentService.Read(subject, tenant, recordId)` carga la concesión vigente y deniega cuando falta, está revocada, ha caducado o no incluye acceso de lectura, o cuando su tenant es distinto del solicitado; solo entonces llama al repositorio. Una comprobación de rol no basta: el Reader del tenant A es realmente un Reader, el registro 17 existe en ambos tenants, e `IsInRole("Reader")` deja pasar la lectura horizontal.

La prueba es un número. En el fixture F02, el Reader del tenant A solicita el registro 17 del tenant B: denegación, y el contador de lecturas no se mueve. Después revoca la concesión Reader desde la consola de administración y haz clic en Export en la pantalla que todavía muestra el botón: el servicio lee la concesión vigente y deniega antes de que se abra ningún archivo. Cada decisión escribe un evento de auditoría con un sujeto opaco, la acción, la clasificación, la decisión y el ID de correlación, nunca cookies, IDs de sesión ni tokens.

La consola autónoma `PolicyHarness` (net8.0 en el paquete; net10.0 funciona igual) aplica la regla sin Wisej.NET, IIS ni proveedor de identidad: una lectura permitida y, después, denegaciones por límite de tenant, rol ausente, cuenta desconocida, concesión caducada y concesión revocada, con cero llamadas al repositorio. El harness demuestra la política en cualquier lugar; la aplicación demuestra que todos los caminos la usan.

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:

- ✓ Distinguir el sujeto de la aplicación, el principal de la petición del host y la identidad efectiva de Windows de una petición de AccessOps, y explicar por qué ninguno de ellos es por sí solo una decisión de autorización.

- ✓ Explicar cómo se resuelven Application.User e IsAuthenticated, incluidas la sustitución de sesión y la vuelta al principal del host, y por qué Application.UserIdentity solo existe en .NET Framework.
- ✓ Seguir el ciclo de vida de la sesión —ApplicationStart, ApplicationRefresh, SessionTimeout, ApplicationExit y Application.RefreshSessionId— y pasar un sujeto explícito al trabajo que se ejecuta fuera de la sesión.
- ✓ Aplicar una comprobación de tenant y de registro en el límite del servicio y demostrar que la denegación ocurre antes de que el repositorio lea, incluso después de revocar el rol en una pantalla abierta.

Guía del laboratorio / examen

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

Laboratorio práctico

Crea la solución AccessOps con los tenants A y B, cada uno con un registro 17 de contenido distinto, una cuenta Reader en el tenant A y una cuenta sin asignar. Construye un DocumentService que compruebe la concesión vigente, el tenant y el registro antes de llamar a un repositorio que cuenta sus lecturas, establece el principal de la sesión a partir de un verificador de prueba de confianza con ApplicationPrincipal y añade una vista de diagnóstico (fuera de las rutas públicas y eliminada antes de empaquetar) que muestre el tipo de autenticación, el indicador de autenticado, un sujeto ocultado y la identidad efectiva del sistema operativo. Sigue el arranque, un clic, una recarga del navegador y un trabajo en segundo plano en dos perfiles del navegador, solicita el registro 17 del tenant B como Reader del tenant A, revoca la concesión Reader entre la carga de la página y Export en una pantalla ya abierta, y ejecuta el PolicyHarness autónomo para mostrar las mismas decisiones de permiso y denegación sin Wisej, IIS ni proveedor de identidad.

Punto de partida

Una nueva aplicación web Wisej.NET llamada AccessOps en `net10.0` con el paquete `Wisej-4` 4.1.7, ejecutada con Kestrel en tu propia máquina, más el proyecto de consola `PolicyHarness` del paquete añadido a la misma solución. Este módulo no necesita IIS, dominio Windows ni proveedor de identidad: todos los pasos se ejecutan en local, y la "identidad efectiva del sistema operativo" que anotas es simplemente la cuenta con la que se ejecuta el proceso `dotnet`.

Enfoque sugerido

- ✓ Prepara primero los fixtures: el registro 17 en el tenant A y en el tenant B con contenidos distintos, una concesión Reader para el tenant A y una cuenta sin concesión, y un repositorio cuyo `ReadCount` puedas mostrar en pantalla.
- ✓ Escribe `DocumentService.Read(subject, tenant, recordId)` de modo que cargue la concesión vigente y rechace una concesión ausente, revocada o caducada, la falta del derecho de lectura y una discrepancia de tenant antes de tocar el repositorio.
- ✓ Conecta el inicio de sesión a un verificador de formación que nunca se publica: verifica en el clic, aparca la cuenta, llama a `Application.Reload()` y, en la carga, llama a `ApplicationPrincipal.Establish`, que rota el ID de sesión antes de asignar `Application.User`.
- ✓ Añade la vista de diagnóstico detrás de un conmutador de compilación solo para depuración y rellena después la matriz de cuatro columnas para el arranque, un clic, una recarga y un trabajo en segundo plano en dos perfiles del navegador, incluido un trabajo iniciado con `Application.StartTask` y otro iniciado con `Task.Run` que recibe un sujeto explícito.
- ✓ Ejecuta las denegaciones —el registro 17 del tenant B como Reader del tenant A y, después, Export tras revocar la concesión en una pantalla abierta—, comprueba que `ReadCount` no se

movió, ejecuta `PolicyHarness` y confirma que la compilación publicada no contiene ninguna vista de diagnóstico.

Entregables requeridos

- ✓ Solución AccessOps con fixtures de los tenants A y B y un repositorio que cuenta lecturas
- ✓ Control de acceso a recursos DocumentService con comprobaciones de concesión, tenant y registro
- ✓ Vista de diagnóstico de identidad, eliminada de la compilación empaquetada
- ✓ Matriz de identidades de cuatro columnas para petición, sesión, operación y resultado
- ✓ Ejecución del PolicyHarness con una lectura permitida y dos denegaciones con cero llamadas al repositorio

Preguntas de repaso

- ✓ Tu vista de diagnóstico muestra la misma identidad efectiva del sistema operativo para ambos perfiles del navegador, mientras que los sujetos de la aplicación son distintos. ¿Qué te dice eso sobre la identidad que comprobará el sistema de archivos, y qué cambiaría en IIS con la autenticación de Windows?
- ✓ Después de revocar la concesión Reader, el botón Export sigue en la pantalla abierta y `Application.User` sigue llevando el claim del rol Reader. ¿Qué lectura concreta de tu código provoca la denegación, y por qué esa lectura no es el claim de rol?
- ✓ Tu exportación en segundo plano se ejecuta en `Task.Run` una hora después del clic. ¿Qué recibe exactamente, qué vuelve a comprobar al arrancar y qué saldría mal si leyera `Application.User` en su lugar?

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.