PATRONES DE SEGURIDAD Y AUTENTICACIÓN · MÓDULO 4

Inicio de sesión propio y seguridad de la sesión

Guía de la lección · Guía del laboratorio / examen

LearnWisej

Guía de la lección

Hacerse cargo de la transacción de inicio de sesión, establecer el principal de la aplicación y rotar el ID de sesión, emitir cookies del host solo por HTTP, el cierre de sesión como revocación de una concesión del lado del servidor, y cerrar los caminos de falsificación de peticiones, de origen del WebSocket y de redirección. 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

Hacerse cargo de la transacción de inicio de sesión, establecer el principal de la aplicación y rotar el ID de sesión, emitir cookies del host solo por HTTP, el cierre de sesión como revocación de una concesión del lado del servidor, y cerrar los caminos de falsificación de peticiones, de origen del WebSocket y de redirección.

1. El problema en producción

Los socios llegan a AccessOps por la ruta HTTPS pública, donde no hay ningún desafío de Windows en el que apoyarse: la aplicación se encarga de todo el inicio de sesión, desde el cuadro de contraseña hasta el momento en que la consola sustituye a la vista de inicio de sesión. Cada atajo se convierte en una vía de ataque: un bloqueo que revela qué nombres existen, un ID de sesión que sobrevive al inicio de sesión o un cierre de sesión que oculta un botón mientras una segunda pestaña sigue exportando. Este módulo construye el inicio de sesión de socios y demuestra que una concesión revocada se deniega en todas partes.

2. Conceptos clave

Concepto	Por qué importa
Transacción de inicio de sesión	La secuencia del lado del servidor limitar → verificar → resolver cuenta y tenant → aplicar la política → crear la concesión → establecer el principal, en la que cualquier paso fallido termina el intento sin dejar estado detrás.
Limitador de inicios de sesión	Contadores de fallos recientes por cuenta y por dirección del cliente que rechazan nuevos intentos durante un periodo de bloqueo, mantenidos en el servicio de inicio de sesión porque un clic de Wisej no puede limitarse con el middleware de ASP.NET Core.
Fallo genérico	Una única frase idéntica para una contraseña errónea, un nombre desconocido, una cuenta deshabilitada y un bloqueo, con tiempos comparables, para que la pantalla nunca confirme que una cuenta existe.
Principal de la aplicación	El `ClaimsPrincipal` asignado a `Application.User` para esta sesión de Wisej; es estado de la sesión, no una cookie del host ni un token de Windows.
Rotación del ID de sesión	`Application.RefreshSessionId()` da a la sesión un ID nuevo sin perder su estado, de modo que un ID que existía antes del inicio de sesión no vale nada después.
Cookie de autenticación del host	Un vale de cookie de ASP.NET Core emitido con `Secure` y `HttpOnly` por un endpoint HTTPS, necesario solo cuando los endpoints HTTP ordinarios deben reconocer al usuario.
Concesión de inicio de sesión	Un registro del lado del servidor con un ID aleatorio opaco, la clave de la cuenta, el tenant, la caducidad y un indicador de revocación, comprobado por cada operación protegida; revocarlo es lo que significa cerrar la sesión.

3. Desarrollo de la lección

Hazte cargo de la transacción de inicio de sesión

La vista de inicio de sesión de AccessOps en la ruta pública anónima hace exactamente una cosa en `btnSignIn_Click`: lee los dos cuadros, vacía el cuadro de contraseña y llama a `LoginService.SignIn`. La verificación de credenciales corresponde a un sistema de identidad admitido —ASP.NET Core Identity, un directorio o, para cuentas locales, una derivación del framework como PBKDF2 mediante `Rfc2898DeriveBytes.Pbkdf2` con una sal por cuenta y una comparación de tiempo constante—. Cuando el nombre de usuario es desconocido, el servicio verifica igualmente contra una credencial ficticia para que "no existe ese usuario" cueste el mismo tiempo que "contraseña errónea".

La limitación vive en ese mismo servicio. El `LoginThrottle` de AccessOps bloquea una cuenta durante 15 minutos tras cinco fallos en una ventana de 15 minutos, y una dirección tras veinte, porque el relleno de credenciales prueba muchos nombres desde un mismo lugar. La limitación de velocidad de ASP.NET Core funciona en los endpoints junto a `app.UseWisej()` pero no puede ver un clic de Wisej, que llega por el WebSocket de la sesión. Lee la dirección de `Application.UserHostAddress` y recuerda sus límites: un mismo navegador puede aparecer como `::1` y como `127.0.0.1`, y detrás de un proxy el cliente real solo

llega por encabezados reenviados en los que confíes. Los contadores en la memoria del proceso solo son correctos para un único servidor.

Cada rechazo —contraseña errónea, nombre desconocido, cuenta deshabilitada, bloqueo— muestra una sola frase: "That user name and password combination is not valid." El motivo real solo va a la tabla de auditoría de inicios de sesión. La contraseña no va a ninguna parte: ni a `Application.Session`, ni a un mensaje de registro, ni al texto de una excepción.

Establece el principal solo después de que pase la política

Una contraseña verificada demuestra que se conoce un secreto, no el derecho a usar AccessOps. El servicio resuelve la cuenta local y su tenant, rechaza una cuenta deshabilitada o un tenant no aprobado con la misma frase genérica, crea una concesión de inicio de sesión y solo entonces construye el principal. El ayudante `ApplicationPrincipal.Establish` del paquete convierte la cuenta, el tenant, la concesión y los roles en claims y asigna `Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"))`. Ese principal es estado de la sesión: no emite ninguna cookie, el middleware de ASP.NET Core nunca lo ve en otras peticiones y no crea ningún token de Windows.

El orden importa y, en Wisej.NET 4.1.7, también el tipo de petición. Llamar a `Application.RefreshSessionId()` desde un clic —un evento WebSocket— cambia el ID en el servidor, pero el navegador nunca lo adopta; su siguiente petición inicia una sesión totalmente nueva y el inicio de sesión se pierde. Llamado durante una carga de página, el navegador adopta el ID nuevo, que guarda en el sessionStorage de la pestaña. Por eso AccessOps verifica en el clic, aparca el resultado verificado bajo una clave pendiente, llama a `Application.Reload()` y termina en la carga:

```
// Program.Main: Application.ApplicationRefresh += (s, e) => OnPageLoad();
private static void OnPageLoad()
{
    if (Application.Session["PendingSignIn"] is not VerifiedSignIn p)
        return;

    Application.Session["PendingSignIn"] = null;
    Application.RefreshSessionId();          // new ID first (HTTP load)
    ApplicationPrincipal.Establish(p.AccountId, p.TenantId, p.GrantId, p.Roles);

    var login = Application.MainPage;
    Application.MainPage = new ConsolePage(); // replace the login view
    login?.Dispose();
}
```

Por eso la copia del curso de `ApplicationPrincipal` asigna `Application.User` sin rotar; la carga rota una sola vez, antes de que exista la identidad.

Cookies por HTTP, cierre de sesión como revocación

Un marco WebSocket no tiene encabezados de respuesta, así que un clic no puede entregar un `Set-Cookie` con `HttpOnly`. `Application.Cookies` solo puede escribir cookies HttpOnly durante `Program.Main`, `ApplicationStart`, `ApplicationRefresh` o cuando `Application.IsWebSocket` es false, y `Application.Browser.CookieStorage` se escribe con JavaScript, que nunca puede establecer HttpOnly.

Cuando los endpoints HTTP ordinarios deben reconocer al socio, AccessOps asigna antes de `app.UseWisej()` un endpoint HTTPS que canjea un identificador de un solo uso y de vida corta creado por la sesión, llama a `HttpContext.SignInAsync` con el esquema `"Cookies"` y deja que el controlador de cookies emita `Secure`, `HttpOnly` y `SameSite=Strict`. Ese vale no es de Wisej: el ID de sesión vive en sessionStorage, y Wisej 4.1.7 escribe una única cookie HttpOnly propia de validación de peticiones que debes dejar como está.

Cerrar la sesión no es ocultar un botón ni borrar una cookie. El `GrantStore` guarda concesiones opacas con caducidad; `ExportService` comprueba la concesión del claim `grant` del principal, el tenant y el registro antes de que se ejecute el repositorio. El cierre de sesión revoca primero la concesión, audita, hace caducar la cookie del host mediante el endpoint de cierre de sesión, borra la identidad de la sesión y llama a `Application.Exit()`. Poner `Application.User` a `null` no basta: el getter vuelve al principal del host y, con la autenticación de Windows de IIS, una recarga vuelve a iniciar la sesión directamente. Una segunda pestaña sigue mostrando Export; su siguiente clic se deniega con cero lecturas del repositorio (fixture F05). `CookieAuthenticationEvents.OnValidatePrincipal` aplica la misma comprobación de concesión a las peticiones del host.

Cierra los caminos entre sitios

El navegador adjunta la cookie del host a cualquier petición, incluida una iniciada por otro sitio, así que cada POST que cambia el estado y confía en ella —inicio de sesión, cierre de sesión, entrega de la cookie— valida un token antifalsificación. El ID de sesión de Wisej está en sessionStorage, que una página ajena no puede hacer que el navegador envíe, pero el protocolo de enlace del WebSocket sigue llevando cookies y no está cubierto por CORS: un pequeño middleware antes de `app.UseWisej()` rechaza una actualización cuyo `Origin` no sea un origen de AccessOps, y el punto de entrada de red repite la regla.

Las URL de continuación son entrada del usuario. Un `returnUrl` procedente de `Application.QueryString` o de la consulta de un endpoint solo se acepta cuando es una ruta local que no empieza por `//` ni por `/\`, o una entrada de una lista de permitidos explícita en el servidor; cualquier otra cosa se convierte en `/`. Deriva el esquema y el host solo después de configurar `app.UseForwardedHeaders()` para proxies conocidos. El fixture F11 —un POST o una actualización desde otro origen— debe terminar sin ningún cambio de estado.

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:

- ✓ Implementar un inicio de sesión que verifique las credenciales con un sistema de identidad admitido, limite los intentos, devuelva fallos genéricos y mantenga las contraseñas fuera de los

registros y del estado de la sesión.

- ✓ Establecer el principal de la aplicación Wisej.NET solo después de que pasen las políticas de cuenta y de tenant, rotar el ID de sesión con Application.RefreshSessionId y sustituir la vista de inicio de sesión.
- ✓ Emitir y hacer caducar cookies del host Secure y HttpOnly desde un endpoint HTTPS dedicado en lugar de un evento WebSocket, y tratar el cierre de sesión como la revocación de una concesión del lado del servidor que comprueba cada operación protegida.
- ✓ Cerrar los caminos entre sitios con antifalsificación en los POST que cambian el estado, comprobaciones de origen del WebSocket y una lista de permitidos para las URL de continuación.

Guía del laboratorio / examen

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

Laboratorio práctico

Dale a la ruta de socios de AccessOps su propio inicio de sesión: una vista de inicio de sesión en la ruta pública anónima que verifique una identidad de fixture mediante un verificador admitido o un simulador de formación que nunca se publica, rechace una credencial errónea y una cuenta deshabilitada con el mismo mensaje genérico, resuelva la cuenta local y el tenant, establezca el principal con ApplicationPrincipal y rote el ID de sesión. Respáldala con un GrantStore de concesiones de inicio de sesión opacas, con caducidad y revocables que el servicio de exportación compruebe antes de cada lectura, añade un endpoint HTTPS de inicio y cierre de sesión para la cookie del host y, después, abre dos perfiles del navegador, revoca la concesión con la segunda pestaña abierta, reintenta la exportación en ambas pestañas y tras una recarga, y prueba un POST falsificado, una actualización a WebSocket desde otro origen y una URL de continuación externa.

Punto de partida

Abre la solución AccessOps del Módulo 1 con sus fixtures de los tenants A y B, el repositorio que cuenta lecturas y el control de acceso `DocumentService`, y añade una `LoginPage` en la ruta pública anónima. Ejecútala en un nombre de host HTTPS (o en `https://localhost`) y ten preparados dos perfiles del navegador para la prueba de revocación.

Enfoque sugerido

- ✓ Escribe `LoginService.SignIn` con `LoginThrottle`, una comprobación contra una credencial ficticia para los nombres desconocidos y una única frase genérica de fallo; registra los resultados en la tabla de auditoría y confirma que ningún valor de contraseña llega a un registro ni a `Application.Session`.
- ✓ Añade `GrantStore` (ID aleatorio opaco, cuenta, tenant, caducidad, indicador de revocación) y haz que `ExportService` compruebe la concesión, el tenant y el registro antes de que se ejecute el repositorio.
- ✓ Aparca el inicio de sesión verificado en el clic, llama a `Application.Reload()` y, en `ApplicationRefresh`, llama a `Application.RefreshSessionId()`, después a `ApplicationPrincipal.Establish` y después cambia la vista de inicio de sesión por la consola; registra el ID antiguo y el nuevo como un booleano "rotado".
- ✓ Asigna los endpoints HTTPS de entrega de la cookie y de cierre de sesión antes de `app.UseWisej()` con antifalsificación, una comprobación de `Origin` en las actualizaciones a WebSocket y una comprobación de ruta local para `returnUrl`; captura los atributos de `Set-Cookie` con el valor ocultado.
- ✓ Revoca la concesión desde la pestaña 1, reintenta Export en la pestaña 2 y tras una recarga, ejecuta después las pruebas de POST falsificado, actualización desde un origen erróneo y URL de continuación externa, y dibuja el diagrama de transiciones del estado de autenticación.

Entregables requeridos

- ✓ Vista de inicio de sesión con fallos genéricos, limitación y tratamiento de cuentas deshabilitadas
- ✓ Establecimiento del principal con rotación del ID de sesión
- ✓ Endpoint HTTPS de cookies con los atributos de Set-Cookie ocultados
- ✓ GrantStore con la revocación aplicada en todas las pestañas abiertas
- ✓ Diagrama de transiciones del estado de autenticación y pruebas de POST falsificado, origen erróneo y URL de continuación

Preguntas de repaso

- ✓ Tu primer borrador llamaba a `Application.RefreshSessionId()` dentro de `btnSignIn_Click` y la consola apareció y desapareció en el siguiente clic. ¿Qué hizo el navegador con el ID nuevo, y por qué una carga de página se comporta de otra forma?
- ✓ Tras cerrar la sesión, la pestaña 2 sigue mostrando el botón Export y sigue teniendo un principal con un claim de rol. ¿Qué comprobación concreta hace que falle su siguiente exportación, y por qué borrar solo la cookie del host no lo conseguiría?
- ✓ El ID de sesión de Wisej no está en una cookie y, aun así, AccessOps necesita antifalsificación y una comprobación de origen. ¿Qué peticiones llevan credenciales ambientales que otro sitio podría provocar?

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.