PATRONES DE SEGURIDAD Y AUTENTICACIÓN · MÓDULO 3

Suplantación y delegación de Windows

Guía de la lección · Guía del laboratorio / examen

LearnWisej

Guía de la lección

Identidad de la petición frente a token efectivo, suplantación acotada en .NET Framework y con WindowsIdentity.RunImpersonated en ASP.NET Core, la opción impersonate de Wisej.NET y sus enganches ImpersonationBegin e ImpersonationEnd, la reversión del token y el segundo salto de Kerberos. 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

Identidad de la petición frente a token efectivo, suplantación acotada en .NET Framework y con WindowsIdentity.RunImpersonated en ASP.NET Core, la opción impersonate de Wisej.NET y sus enganches ImpersonationBegin e ImpersonationEnd, la reversión del token y el segundo salto de Kerberos.

1. El problema en producción

AccessOps guarda documentos confidenciales en un almacenamiento de archivos de Windows cuyas ACL se escribieron para personas, no para el servidor web. La ruta de la intranet ya conoce al empleado, así que resulta tentador suplantar en todas partes y dejar que Windows decida. Eso convierte la lectura de un archivo en un cambio de identidad para todo el proceso y falla de forma confusa en cuanto el archivo está en un segundo servidor.

2. Conceptos clave

Concepto	Por qué importa
identidad de la petición	El `WindowsIdentity` que la autenticación de Windows de IIS adjuntó a la petición HTTP; dice quién llamó, pero por sí sola no abre nada.
token efectivo	El token que presenta el hilo cuando Windows abre un objeto, leído con `WindowsIdentity.GetCurrent()`; normalmente la identidad del grupo; el usuario solo dentro de un ámbito de suplantación.
suplantación acotada	Ejecutar una operación delimitada con el token del usuario y revertir al final, incluso cuando lanza una excepción.
opción `impersonate`	El conmutador de Default.json detrás de `Configuration.Impersonate` (predeterminado `false`) que hace que Wisej.NET produzca sus eventos de suplantación alrededor de los hilos de petición y de las tareas de `Application.StartTask`.
`ImpersonationBegin` / `ImpersonationEnd`	Eventos `HandledEventHandler` de todo el proceso; en .NET Framework, el tratamiento predeterminado suplanta y después revierte, y establecer `Handled` suprime ese comportamiento.
prueba por ACL	Evidencia de que el acceso vino de los grupos del token frente a una ACL real: un archivo que solo puede leer el grupo, otro que solo puede leer un usuario de laboratorio concreto.
segundo salto	El servidor web que reutiliza la identidad del usuario contra otra máquina, lo que necesita delegación de Kerberos hacia un destino permitido y una ACL en el backend que conceda acceso al usuario.

3. Desarrollo de la lección

Quien llama no es quien abre el archivo

Una petición autenticada con Windows lleva dos hechos que es fácil fusionar. La identidad de la petición es lo que autenticó el host: en ASP.NET Core llega a la sesión Wisej.NET como `Application.User`, cuya `Identity` es un `WindowsIdentity` con la autenticación de Windows de IIS. El token efectivo es lo que el hilo en ejecución presenta al sistema operativo, y en una petición normal sigue siendo la identidad del grupo de aplicaciones. Windows nunca pide un nombre cuando abre un archivo; compara el SID de usuario y los SID de grupo del token con la ACL del archivo. El sujeto de la aplicación AccessOps del Módulo 1 es un tercer hecho, independiente.

Por eso el laboratorio demuestra el acceso con ACL, no con nombres: una carpeta que solo puede leer el grupo de aplicaciones y otra que solo puede leer un usuario de laboratorio concreto del dominio. Antes de suplantar, la lectura reservada al usuario debe fallar y la reservada al grupo debe funcionar; dentro del ámbito, los resultados se invierten. Si ambas identidades leen ambos archivos, revisa las ACL heredadas y la pertenencia al grupo de administradores locales. Registra `WindowsIdentity.GetCurrent()` con su `Name`, su SID `User` y su `ImpersonationLevel`, y elimina la identidad que devuelve.

.NET Framework, la opción impersonate y los enganches de los hilos

En .NET Framework hay cuatro conmutadores que suenan parecido y hacen cosas distintas: el servicio de rol de autenticación de Windows de IIS, `<authentication mode="Windows" />` de System.Web, `<identity impersonate>` de System.Web y la clave `impersonate` de Wisej.NET en Default.json. El laboratorio mantiene desactivada la suplantación de System.Web y usa `using (identity.Impersonate()) { … }` alrededor de una única lectura síncrona de archivo, con la identidad tomada de `Application.UserIdentity` (la identidad de inicio de sesión de la petición, que solo existe en la compilación para .NET Framework).

Con `"impersonate": true`, que se lee una vez y se guarda en caché para todo el proceso, Wisej.NET produce `ImpersonationBegin` al inicio de cada hilo de petición y de cada tarea iniciada con `Application.StartTask`, e `ImpersonationEnd` cuando termina. En .NET Framework, salvo que un controlador establezca `e.Handled = true`, Begin llama a `UserIdentity.Impersonate()` y End elimina ese contexto. Los eventos se producen alrededor del hilo de petición: `ImpersonationBegin` se dispara antes de `ThreadBegin`, e `ImpersonationEnd` antes de `ThreadEnd`. (La documentación antigua dice lo contrario). Son eventos estáticos de todo el proceso: adjúntalos una sola vez al iniciar la aplicación y, si solo los observas, deja `Handled` en false para que el framework limpie lo que empezó. Un `Task.Run` directo no recibe nada de esto.

El laboratorio compara la opción desactivada y activada para un clic y para el cuerpo de un `Application.StartTask`, con WebSocket habilitado y en modo solo HTTP, cada uno en una sesión nueva.

Una lectura RunImpersonated delimitada en ASP.NET Core

En .NET 8 y .NET 10 los dos eventos se siguen disparando cuando `impersonate` es true, pero Wisej.NET no hace nada de forma predeterminada: el .NET moderno no tiene `WindowsIdentity.Impersonate()` ni un contexto de suplantación que mantener abierto entre un Begin y un End. El patrón admitido es un ámbito que gestiona su propia reversión. AccessOps lo envuelve en un `ScopedDocumentReader` que recibe un ID de documento, nunca una ruta:

```
[SupportedOSPlatform("windows")]
public byte[] Read(HttpContext context, int documentId)
{
    if (context.User.Identity is not WindowsIdentity identity || !identity.IsAuthenticated)
        throw new UnauthorizedAccessException();
    // policy first: grant, tenant and record, then a canonical path under the root
    string path = _catalog.ResolveApprovedPath(documentId);

    return WindowsIdentity.RunImpersonated(identity.AccessToken, () ⇒
    {
        using (var current = WindowsIdentity.GetCurrent()) _trace.Record("inside", current);
        return File.ReadAllBytes(path);
    });
}
```

El navegador nunca elige una ruta; el catálogo asigna el ID a un archivo bajo una única raíz aprobada solo después de que la decisión de la aplicación permita la lectura. `RunImpersonated` restaura el token anterior cuando el delegado retorna o lanza una excepción, que es lo que demuestra el fixture F03: solicita un

documento aprobado cuyo archivo falta, deja que se escape `FileNotFoundException` y la fila "después" debe volver a mostrar la identidad del grupo. El caso de ACL denegada usa un archivo que existe, para que "no encontrado" nunca se haga pasar por "no permitido". Mantén el ámbito síncrono; usa `RunImpersonatedAsync` solo cuando el trabajo esperado termine dentro de la petición, y no guardes nunca el token en el estado de la sesión ni se lo entregues a trabajo de larga duración.

Ejecuta el lector donde el token de la petición está vivo: un endpoint autorizado asignado antes de `app.UseWisej()`, o código que llega al `HttpContext` actual a través de `IHttpContextAccessor`. Un evento de Wisej por WebSocket ve el principal capturado cuando se abrió el socket, y el host gestiona la vida de ese token, así que el laboratorio registra ese caso en lugar de depender de él.

Clasificar el segundo salto

Una lectura que funciona en una carpeta NTFS local y falla contra `\\files01\confidential` o con un inicio de sesión integrado de SQL no suele ser un fallo de suplantación. Haz tres preguntas. Primer salto: ¿llegó el navegador a IIS con Kerberos? Un encabezado Negotiate por sí solo no lo demuestra; `klist` en el cliente debería mostrar un vale para el SPN HTTP del sitio, y NTLM no se puede delegar. Delegación: ¿puede la cuenta del grupo de aplicaciones delegar, mediante delegación restringida, a ese SPN de destino exacto, y la cuenta del usuario no está marcada como confidencial? Autorización en el backend: ¿concede realmente el permiso del recurso compartido, de NTFS o de SQL acceso a este usuario? El fixture F04 es ese caso; cambia el código solo cuando tengas las tres respuestas.

Después, decide. El acceso delegado del usuario mantiene la ACL del backend como autoridad, pero necesita delegación restringida configurada para cada destino, nunca sin restricciones. Una identidad de servicio con privilegios estrechos traslada la decisión a AccessOps: el servicio comprueba la concesión, el tenant y el registro, y una entrada de auditoría nombra a la persona en cuyo nombre leyó. El registro de la decisión indica cuál usa AccessOps y por qué.

Sé honesto sobre el origen de cada prueba. Las lecturas suplantadas, los resultados de las ACL, la comparación en .NET Framework y el segundo salto necesitan IIS con autenticación de Windows y cuentas del dominio: solo se verifican en el entorno de laboratorio. En una máquina de desarrollo, el PolicyHarness demuestra lo que va primero: la decisión de la aplicación que debe permitir la lectura antes de que se abra ningún ámbito de suplantación.



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:

- ✓ Explicar la diferencia entre la identidad de Windows adjunta a una petición y el token efectivo usado en una operación de archivo, y demostrar el acceso con una ACL en lugar de con un nombre de usuario.
- ✓ Configurar la suplantación acotada en .NET Framework y comparar la opción impersonate de Wisej.NET, sus eventos ImpersonationBegin e ImpersonationEnd y Application.StartTask en eventos HTTP y WebSocket.
- ✓ Implementar una lectura WindowsIdentity.RunImpersonated delimitada en ASP.NET Core con una ruta canónica elegida por el servidor, y verificar que el token se revierte tras una excepción.
- ✓ Clasificar un fallo de segundo salto —SPN, destino de la delegación restringida o ACL del backend— y decidir entre el acceso delegado del usuario y una identidad de servicio restringida con auditoría explícita.

Guía del laboratorio / examen

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

Laboratorio práctico

Añade a AccessOps un lector de documentos confidenciales con dos carpetas, una que solo puede leer el grupo de aplicaciones y otra que solo puede leer un usuario de laboratorio concreto del dominio, y un ScopedDocumentReader construido a partir del ayudante WindowsScopedRead (ASP.NET Core) o Net48ScopedRead (.NET Framework) que abre un archivo elegido por ID en el servidor. Anota el sujeto de la aplicación, la identidad de la petición y la identidad efectiva antes, dentro y después de la lectura acotada, fuerza una excepción con un fixture aprobado que falta y muestra que se recupera la identidad habitual, compara la opción impersonate de Wisej.NET desactivada y activada para un clic y un Application.StartTask en los modos HTTP y WebSocket en .NET Framework y, donde exista un segundo servidor, clasifica un fallo remoto de SMB o de SQL integrado por primer salto, delegación y autorización en el backend. Las lecturas suplantadas, los resultados de las ACL y el segundo salto solo se verifican en el entorno de laboratorio con la autenticación de Windows de IIS y cuentas del dominio; en una máquina de desarrollo, el PolicyHarness demuestra la decisión de la aplicación que debe permitir la lectura antes de que se abra ningún ámbito de suplantación.

Punto de partida

Abre AccessOps tal como lo dejó el Módulo 2: el sitio de intranet en IIS con la autenticación de Windows, la política de recursos del Módulo 1 detrás de los servicios de documentos y el PolicyHarness en verde. En el entorno de laboratorio, crea las dos carpetas de fixtures bajo una única raíz aprobada y concede a la cuenta del grupo y al usuario de laboratorio del dominio sus derechos de lectura por separado; en una máquina de desarrollo, trabaja en el PolicyHarness y en el código del lector hasta que llegues a ese entorno.

Enfoque sugerido

- ✓ Añade un catálogo de documentos que asigne IDs de documento del servidor a rutas canónicas bajo la raíz aprobada, y haz que `ScopedDocumentReader` llame a la política de recursos antes de resolver ninguna ruta.
- ✓ Envuelve la lectura en `WindowsIdentity.RunImpersonated` (o en `using (identity.Impersonate())` en .NET Framework) y registra `WindowsIdentity.GetCurrent()` antes, dentro y después, junto al sujeto de la aplicación y la identidad de la petición.
- ✓ Lee el fixture reservado al grupo y el reservado al usuario fuera y dentro del ámbito, solicita después un documento aprobado cuyo archivo falta y confirma que la fila "después" vuelve a mostrar la identidad del grupo (F03).
- ✓ En el entorno aislado de .NET Framework, establece `"impersonate": true`, adjunta una sola vez al arrancar controladores de registro a `ImpersonationBegin` e `ImpersonationEnd`, y repite las observaciones para un clic y un `Application.StartTask` con WebSocket activado y desactivado.

- ✓ Donde haya disponible un servidor de archivos o un SQL Server, reproduce F04 y anota por separado el primer salto, el destino de la delegación y la ACL del backend antes de escribir la decisión de migración.

Entregables requeridos

- ✓ ScopedDocumentReader con una ruta canónica resuelta en el servidor
- ✓ Matriz de identidad y resultado antes, dentro y después
- ✓ Traza de reversión con fallo forzado
- ✓ Comparación de la opción impersonate y de StartTask en HTTP y WebSocket
- ✓ Clasificación del segundo salto y decisión de migración para el recurso remoto

Preguntas de repaso

- ✓ Tu traza muestra el nombre del usuario de laboratorio dentro del ámbito y, sin embargo, ambos archivos de fixture se abren en todas las filas. ¿Qué te dice eso sobre la evidencia y qué compruebas primero?
- ✓ En .NET 10, el controlador de `ImpersonationBegin` registra en cada clic. ¿Por qué la lectura del archivo se sigue ejecutando como el grupo, y qué tendría que cambiar para que no fuera así?
- ✓ El recurso compartido remoto solo deniega a través de AccessOps, mientras que el mismo usuario lo abre desde su escritorio. ¿Cuál de las tres comprobaciones del segundo salto descarta eso, y cuáles dos quedan?

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.