PATTERN DI SICUREZZA E AUTENTICAZIONE · MODULO 3

Impersonificazione e delega di Windows

Guida della lezione · Guida al lab / esame

LearnWisej

Guida della lezione

Identità della richiesta contro token effettivo, impersonificazione circoscritta su .NET Framework e con WindowsIdentity.RunImpersonated su ASP.NET Core, l'impostazione impersonate di Wisej.NET con i suoi hook ImpersonationBegin e ImpersonationEnd, il ripristino del token e il secondo hop di Kerberos. Che cosa significa per uno sviluppatore che mette in sicurezza i percorsi di accesso, identità e autorizzazione di un'app Wisej.NET di produzione — e come affrontare questo modulo.

Focus di produzione

Identità della richiesta contro token effettivo, impersonificazione circoscritta su .NET Framework e con WindowsIdentity.RunImpersonated su ASP.NET Core, l'impostazione impersonate di Wisej.NET con i suoi hook ImpersonationBegin e ImpersonationEnd, il ripristino del token e il secondo hop di Kerberos.

1. Il problema in produzione

AccessOps conserva i documenti riservati su uno storage di file Windows le cui ACL sono state scritte per le persone, non per il server web. La route intranet conosce già il membro del personale, quindi viene la tentazione di fare impersonificazione ovunque e lasciar decidere Windows. Così una singola lettura di file diventa un cambio di identità a livello di processo, e fallisce in modo confuso non appena il file si trova su un secondo server.

2. Concetti chiave

Concetto	Perché è importante
identità della richiesta	La `WindowsIdentity` che l'autenticazione Windows di IIS ha associato alla richiesta HTTP; dice chi ha chiamato, ma da sola non apre nulla.
token effettivo	Il token che il thread presenta quando Windows apre un oggetto, letto con `WindowsIdentity.GetCurrent()`; normalmente l'identità del pool; l'utente solo dentro un ambito di impersonificazione.
impersonificazione circoscritta	Eseguire una sola operazione delimitata con il token dell'utente e ripristinare alla fine, anche quando lancia un'eccezione.
impostazione `impersonate`	L'opzione di Default.json dietro `Configuration.Impersonate` (predefinita `false`) che fa sollevare a Wisej.NET i suoi eventi di impersonificazione attorno ai thread delle richieste e ai task di `Application.StartTask`.
`ImpersonationBegin` / `ImpersonationEnd`	Eventi `HandledEventHandler` a livello di processo; su .NET Framework la gestione predefinita fa l'impersonificazione e poi la annulla, e impostare `Handled` sopprime quel comportamento predefinito.
prova tramite ACL	La prova che l'accesso è venuto dai gruppi del token confrontati con un'ACL reale: un file che solo il pool può leggere, uno che solo un utente di lab nominato può leggere.
secondo hop	Il server web che riusa l'identità dell'utente verso un'altra macchina, cosa che richiede la delega Kerberos verso una destinazione consentita e un'ACL sul backend che conceda l'accesso all'utente.

3. Il filo della lezione

Chi chiama non è chi apre il file

Una richiesta autenticata da Windows porta due fatti facili da fondere. L'identità della richiesta è ciò che l'host ha autenticato: su ASP.NET Core raggiunge la sessione Wisej.NET come `Application.User`, la cui `Identity` è una `WindowsIdentity` sotto l'autenticazione Windows di IIS. Il token effettivo è ciò che il thread in esecuzione presenta al sistema operativo, e in una richiesta normale è ancora l'identità dell'application pool. Windows non chiede mai un nome quando apre un file; confronta il SID utente e i SID di gruppo del token con l'ACL del file. Il soggetto applicativo di AccessOps del Modulo 1 è un terzo fatto, separato.

Per questo il lab dimostra l'accesso con le ACL, non con i nomi: una cartella che solo l'application pool può leggere, una che solo un utente di lab del dominio nominato può leggere. Prima dell'impersonificazione la lettura riservata all'utente deve fallire e quella riservata al pool riuscire; dentro l'ambito gli esiti si invertono. Se entrambe le identità leggono entrambi i file, controlla le ACL ereditate e l'appartenenza agli amministratori locali. Registra `WindowsIdentity.GetCurrent()` con il suo `Name`, il SID `User` e l'`ImpersonationLevel`, ed elimina con dispose l'identità che restituisce.

.NET Framework, l'impostazione impersonate e gli hook dei thread

Su .NET Framework quattro opzioni hanno nomi simili e fanno cose diverse: il servizio ruolo Windows Authentication di IIS, `<authentication mode="Windows" />` di System.Web, `<identity impersonate>` di System.Web e la chiave `impersonate` di Wisej.NET in Default.json. Il lab tiene disattivata l'impersonificazione di System.Web e usa `using (identity.Impersonate()) { … }` attorno a una sola lettura sincrona di file, con l'identità presa da `Application.UserIdentity` (l'identità di logon della richiesta, che esiste solo nella build per .NET Framework).

Con `"impersonate": true`, che viene letto una volta sola e messo in cache per tutto il processo, Wisej.NET solleva `ImpersonationBegin` all'inizio di ogni thread di richiesta e di ogni task avviato con `Application.StartTask`, e `ImpersonationEnd` quando termina. Su .NET Framework, a meno che un gestore non imposti `e.Handled = true`, Begin chiama `UserIdentity.Impersonate()` ed End elimina quel contesto. Gli eventi vengono sollevati attorno al thread della richiesta: `ImpersonationBegin` scatta prima di `ThreadBegin`, e `ImpersonationEnd` prima di `ThreadEnd`. (La documentazione più vecchia dice il contrario.) Sono eventi statici a livello di processo: collegali una volta all'avvio dell'applicazione e, se ti limiti a osservarli, lascia `Handled` a false così che il framework ripulisca ciò che ha avviato. Un semplice `Task.Run` non riceve nulla di tutto questo.

Il lab confronta l'impostazione disattivata e attivata per un clic e per il corpo di un `Application.StartTask`, con il WebSocket abilitato e in modalità solo HTTP, ciascuno in una sessione nuova.

Una lettura RunImpersonated delimitata su ASP.NET Core

Su .NET 8 e .NET 10 i due eventi scattano ancora quando `impersonate` è true, ma per impostazione predefinita Wisej.NET non fa nulla: il .NET moderno non ha `WindowsIdentity.Impersonate()` né un contesto di impersonificazione da tenere aperto tra un Begin e un End. Il pattern supportato è un ambito che gestisce da sé il proprio ripristino. AccessOps lo incapsula in uno `ScopedDocumentReader` che riceve l'ID di un documento, mai un percorso:

```
[SupportedOSPlatform("windows")]
public byte[] Read(HttpContext context, int documentId)
{
    if (context.User.Identity is not WindowsIdentity identity || !identity.IsAuthenticated)
        throw new UnauthorizedAccessException();
    // prima la policy: grant, tenant e record, poi un percorso canonico sotto la radice
    string path = _catalog.ResolveApprovedPath(documentId);

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

Il browser non sceglie mai un percorso; il catalogo associa l'ID a un file sotto un'unica radice approvata solo dopo che la decisione dell'applicazione ha consentito la lettura. `RunImpersonated` ripristina il token precedente quando il delegato ritorna o lancia un'eccezione, ed è proprio ciò che dimostra la fixture F03:

richiedi un documento approvato il cui file manca, lascia uscire la `FileNotFoundException`, e la riga "after" deve mostrare di nuovo l'identità del pool. Il caso dell'ACL negata usa un file che esiste, così che "non trovato" non si spacci mai per "non consentito". Mantieni l'ambito sincrono; usa `RunImpersonatedAsync` solo quando il lavoro atteso si completa dentro la richiesta, e non memorizzare mai il token nello stato di sessione né passarlo a lavoro di lunga durata.

Esegui il lettore dove il token della richiesta è vivo: un endpoint autorizzato mappato prima di `app.UseWisej()`, oppure codice che raggiunge l'`HttpContext` corrente attraverso `IHttpContextAccessor`. Un evento Wisej sul WebSocket vede il principal catturato all'apertura del socket, e la durata di quel token è gestita dall'host, quindi il lab registra quel caso invece di farci affidamento.

Classificare il secondo hop

Una lettura che funziona su una cartella NTFS locale e fallisce verso `\\files01\confidential` o verso un login integrato di SQL di solito non è un bug di impersonificazione. Fai tre domande. Primo hop: il browser ha raggiunto IIS con Kerberos? Un'intestazione Negotiate da sola non lo dimostra; `klist` sul client dovrebbe mostrare un ticket per lo SPN HTTP del sito, e NTLM non può essere delegato. Delega: l'account dell'application pool è autorizzato a delegare, tramite delega vincolata, esattamente a quello SPN di destinazione, e l'account dell'utente non è contrassegnato come sensibile? Autorizzazione sul backend: il permesso sulla condivisione, su NTFS o su SQL concede davvero l'accesso a questo utente? La fixture F04 è proprio questo caso; modifica il codice solo dopo aver ottenuto tutte e tre le risposte.

Poi decidi. L'accesso delegato dell'utente mantiene l'ACL del backend come fonte autorevole, ma richiede una delega vincolata configurata per ogni destinazione, mai una delega non vincolata. Un'identità di servizio con privilegi ristretti sposta la decisione dentro AccessOps: il servizio controlla il grant, il tenant e il record, e una voce di audit nomina la persona per conto della quale ha letto. Il registro delle decisioni dice quale delle due usa AccessOps e perché.

Sii onesto su da dove viene ogni prova. Le letture con impersonificazione, gli esiti delle ACL, il confronto su .NET Framework e il secondo hop richiedono IIS con l'autenticazione Windows e account di dominio: si verificano solo nell'ambiente di lab. Su una macchina di sviluppo il PolicyHarness dimostra ciò che viene prima: la decisione dell'applicazione che deve consentire la lettura prima che si apra qualsiasi ambito di impersonificazione.

Regola di sicurezza

Tieni separate le identità e decidi sul server. Il principal dell'host, il principal applicativo di Wisej.NET e il token Windows effettivo sono tre fatti diversi, e nessuno di essi è un permesso. Ogni lettura protetta controlla il grant corrente, il tenant e il record al confine del servizio prima che il repository venga eseguito, e un pulsante nascosto, un claim di ruolo o una scheda disconnessa non dimostrano nulla finché quel controllo non rifiuta.

Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Spiegare la differenza tra l'identità Windows associata a una richiesta e il token effettivo usato in un'operazione su file, e dimostrare l'accesso con un'ACL invece che con un nome utente.
- ✓ Configurare l'impersonificazione circoscritta su .NET Framework e confrontare l'impostazione impersonate di Wisej.NET, i suoi eventi ImpersonationBegin e ImpersonationEnd e Application.StartTask tra eventi HTTP e WebSocket.
- ✓ Implementare su ASP.NET Core una lettura WindowsIdentity.RunImpersonated delimitata con un percorso canonico scelto dal server, e verificare che il token venga ripristinato dopo un'eccezione.
- ✓ Classificare un errore del secondo hop — SPN, destinazione della delega vincolata o ACL del backend — e scegliere tra l'accesso delegato dell'utente e un'identità di servizio ristretta con audit esplicito.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.

Lab pratico

Aggiungi ad AccessOps un lettore di documenti riservati con due cartelle, una leggibile solo dall'application pool e una solo da un utente di lab del dominio nominato, e uno ScopedDocumentReader costruito a partire dall'helper WindowsScopedRead (ASP.NET Core) o Net48ScopedRead (.NET Framework) che apre un file scelto tramite un ID lato server. Registra il soggetto dell'app, l'identità della richiesta e l'identità effettiva prima, durante e dopo la lettura circoscritta, forza un'eccezione su una fixture approvata mancante e mostra che l'identità ordinaria riprende, confronta su .NET Framework l'impostazione impersonate di Wisej.NET disattivata e attivata per un clic e per un Application.StartTask in modalità HTTP e WebSocket e, dove esiste un secondo server, classifica un errore remoto SMB o di login integrato SQL per primo hop, delega e autorizzazione sul backend. Le letture con impersonificazione, gli esiti delle ACL e il secondo hop si verificano solo nell'ambiente di lab con l'autenticazione Windows di IIS e account di dominio; su una macchina di sviluppo il PolicyHarness dimostra la decisione dell'applicazione che deve consentire la lettura prima che si apra qualsiasi ambito di impersonificazione.

Punto di partenza

Apri AccessOps come l'ha lasciato il Modulo 2: il sito intranet su IIS con l'autenticazione Windows, la policy sulle risorse del Modulo 1 dietro i servizi dei documenti e il PolicyHarness tutto verde. Nell'ambiente di lab, crea le due cartelle delle fixture sotto un'unica radice approvata e concedi all'account del pool e all'utente di lab del dominio i rispettivi diritti di lettura separati; su una macchina di sviluppo, lavora nel PolicyHarness e sul codice del lettore finché non arrivi a quell'ambiente.

Approccio suggerito

- ✓ Aggiungi un catalogo dei documenti che associa gli ID lato server a percorsi canonici sotto la radice approvata, e fai in modo che `ScopedDocumentReader` chiami la policy sulle risorse prima di risolvere qualsiasi percorso.
- ✓ Racchiudi la lettura in `WindowsIdentity.RunImpersonated` (o in `using (identity.Impersonate())` su .NET Framework) e registra `WindowsIdentity.GetCurrent()` prima, durante e dopo, accanto al soggetto dell'app e all'identità della richiesta.
- ✓ Leggi la fixture riservata al pool e quella riservata all'utente fuori e dentro l'ambito, poi richiedi un documento approvato il cui file manca e conferma che la riga "after" mostra di nuovo l'identità del pool (F03).
- ✓ Nella sandbox .NET Framework imposta `"impersonate": true`, collega una volta all'avvio dei gestori di logging a `ImpersonationBegin` e `ImpersonationEnd`, e ripeti le osservazioni per un clic e per un `Application.StartTask` con il WebSocket attivato e disattivato.
- ✓ Dove sono disponibili un file server o SQL Server, riproduci F04 e registra separatamente il primo hop, la destinazione della delega e l'ACL del backend prima di scrivere la decisione di migrazione.

Deliverable richiesti

- ✓ ScopedDocumentReader con un percorso canonico risolto dal server
- ✓ Matrice di identità ed esiti prima, durante e dopo
- ✓ Traccia del ripristino dopo un errore forzato
- ✓ Confronto dell'impostazione impersonate e di StartTask tra HTTP e WebSocket
- ✓ Classificazione del secondo hop e decisione di migrazione per la risorsa remota

Domande di autovalutazione

- ✓ La tua traccia mostra il nome dell'utente di lab dentro l'ambito, eppure entrambi i file delle fixture si aprono in ogni riga. Che cosa ti dice questo sulle prove, e che cosa controlli per primo?
- ✓ Su .NET 10 il gestore di `ImpersonationBegin` scrive nel log a ogni clic. Perché la lettura del file viene comunque eseguita come il pool, e che cosa dovrebbe cambiare perché non succeda?
- ✓ La condivisione remota rifiuta l'accesso solo passando da AccessOps, mentre lo stesso utente la apre dal proprio desktop. Quale dei tre controlli del secondo hop esclude questo fatto, e quali due restano?

Regola di sicurezza

Tieni separate le identità e decidi sul server. Il principal dell'host, il principal applicativo di Wisej.NET e il token Windows effettivo sono tre fatti diversi, e nessuno di essi è un permesso. Ogni lettura protetta controlla il grant corrente, il tenant e il record al confine del servizio prima che il repository venga eseguito, e un pulsante nascosto, un claim di ruolo o una scheda disconnessa non dimostrano nulla finché quel controllo non rifiuta.