PATTERN DI SICUREZZA E AUTENTICAZIONE · MODULO 1

Confini di identità in Wisej.NET

Guida della lezione · Guida al lab / esame

LearnWisej

Guida della lezione

Le tre identità di una richiesta Wisej.NET — principal dell'host, principal applicativo e token Windows effettivo — Application.User e il ciclo di vita della sessione, e un rifiuto a livello di risorsa al confine del servizio che sopravvive alla revoca del ruolo. 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

Le tre identità di una richiesta Wisej.NET — principal dell'host, principal applicativo e token Windows effettivo — Application.User e il ciclo di vita della sessione, e un rifiuto a livello di risorsa al confine del servizio che sopravvive alla revoca del ruolo.

1. Il problema in produzione

AccessOps serve il personale interno attraverso un punto di ingresso Windows sulla intranet e i partner attraverso un punto di ingresso HTTPS pubblico, ed entrambi arrivano agli stessi documenti riservati. Ogni richiesta porta con sé più identità contemporaneamente, e un bug che le confonde si manifesta come un tenant che legge il record di un altro tenant, o come un lettore revocato che riesce ancora a esportare. Questo modulo separa le identità e mette la decisione dove la si può dimostrare: al confine del servizio, prima che il repository venga eseguito.

2. Concetti chiave

Concetto	Perché è importante
principal della richiesta dell'host	Ciò che l'host ha autenticato per la richiesta o per l'upgrade WebSocket — `HttpContext.User` su ASP.NET Core; anonimo in una semplice esecuzione su Kestrel.
principal applicativo	Il soggetto che il tuo accesso assegna alla sessione Wisej.NET dopo che un verificatore attendibile ha restituito un account: ID dell'account, tenant, ID del grant.
identità Windows effettiva	Il token che il sistema operativo controlla quando il processo apre un file — l'account dell'application pool o del processo, a meno che il codice non faccia impersonificazione.
override di sessione	Un valore assegnato ad `Application.User` nella sessione corrente; prevale sul principal della richiesta finché non viene riportato a `null`.
grant	Un record lato server, revocabile e con scadenza, secondo cui un account può leggere in un tenant; ogni operazione protetta ne legge lo stato corrente.
confine del servizio	L'unico metodo che ogni percorso — clic, download, job in coda — attraversa prima del repository, così che un solo rifiuto li copra tutti.
repository che conta le letture	Un repository di test che conta le proprie letture, così che un rifiuto sia dimostrato a zero chiamate al repository invece che dedotto da una schermata vuota.

3. Il filo della lezione

Tre identità, e nessuna di esse è un permesso

Segui `CONTOSO\reader1` sulla route intranet. IIS autentica la richiesta, quindi il principal dell'host è un'identità Windows; eppure, quando `DocumentService` apre un file, il sistema operativo controlla il token dell'application pool, non quello di reader1, a meno che il codice non faccia impersonificazione (Modulo 3). Un partner federato (Modulo 5) ha un principal applicativo valido e nessun token Windows. L'esportazione notturna non ha alcun utente connesso, eppure l'account del processo può leggere l'intera cartella dei documenti.

Registrale come tre osservazioni separate e non lasciare mai che una prenda il posto di un'altra. Nessuna è una decisione di autorizzazione: il principal dell'host dice chi si è connesso, un claim di ruolo dice che cosa era vero al momento dell'accesso, il token del processo dice che cosa consente il sistema operativo. AccessOps deve rispondere a una domanda più ristretta a ogni lettura protetta: il grant corrente consente a questo account di leggere questo record in questo tenant, adesso?

`Visible` ed `Enabled` descrivono la schermata. Wisej.NET verifica sì sul server che un clic provenga da un controllo abilitato e visibile, ma i dati legati a una colonna nascosta della griglia arrivano comunque al browser, e lo stesso documento è raggiungibile da un gestore di evento, da un percorso di download e da un job. La regola appartiene al servizio che tutti questi chiamano.

Come si risolvono Application.User e IsAuthenticated

`Application.User` è un `IPrincipal` statico con un getter in due passi. Se in questa sessione è stato assegnato un valore, restituisce quell'override di sessione; altrimenti restituisce il principal della richiesta — su ASP.NET Core l'`HttpContext.User` catturato all'arrivo della richiesta o dell'upgrade WebSocket: una `WindowsIdentity` sotto l'autenticazione Windows di IIS, un principal non autenticato in un'esecuzione Kestrel di sviluppo. `Application.IsAuthenticated` è semplicemente `User?.Identity?.IsAuthenticated == true`: non sa nulla di tenant, ruolo o proprietà. Impostare `Application.User = null` rimuove l'override e ricade sul principal dell'host, quindi "disconnettere" in quel modo un utente Windows lo lascia autenticato.

Assegna l'override solo dopo che un verificatore attendibile ha restituito un account; una `ClaimsIdentity` costruita da un nome utente digitato lascia al browser la scelta della propria identità. L'helper `ApplicationPrincipal` del lab costruisce i claim dall'account verificato — ID dell'account immutabile, tenant, ID del grant, ruoli — con il tipo di autenticazione `"Portal"`. Usa come chiave quell'ID, mai un nome visualizzato o un'email.

```
// Eseguito al caricamento di una pagina, dopo che il verificatore di test ha restituito un
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();   // prima il nuovo ID, al caricamento
    Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"));
}
```

`Application.UserIdentity` esiste solo nella build per .NET Framework (net48), dove è l'identità di logon di System.Web; la build net8.0 a cui fa riferimento un AccessOps su .NET 10 non la dichiara. Su .NET 10 leggi il principal Windows attraverso `Application.User` e verifica che la sua `Identity` sia una `WindowsIdentity` prima di farci affidamento.

Il ciclo di vita della sessione e il lavoro che sta fuori

Il primo caricamento HTTP costruisce l'interfaccia, gli eventi successivi arrivano attraverso il WebSocket, e ogni transizione della sessione è un evento a sé. `ApplicationStart` scatta per una nuova sessione dopo `Program.Main`; `ApplicationRefresh` scatta a un ricaricamento del browser o a un cambio di URL nella stessa sessione; `SessionTimeout` annuncia la scadenza per inattività (imposta `Handled` per sopprimere la finestra di proroga predefinita); `ApplicationExit` scatta quando la sessione viene terminata o eliminata, ed è l'unico evento sollevato alla scadenza. Usalo per la pulizia, mai come revoca: quella sta nell'archivio dei grant.

`Application.RefreshSessionId` emette un nuovo ID di sessione senza perdere la sessione, chiudendo la session fixation dopo l'accesso — ma solo al caricamento di una pagina. Da un clic sul WebSocket il server

cambia ID, il browser non adotta mai quello nuovo e la sua richiesta successiva avvia una sessione nuova. Per questo AccessOps verifica sul clic, parcheggia l'account sotto una chiave in sospeso, chiama `Application.Reload()` e in `ApplicationRefresh` ruota prima l'ID, poi stabilisce il principal. Oscura `Application.SessionId` nelle tracce.

Fuori dalla richiesta, l'identità ambientale viene meno. `Application.StartTask` ripristina la sessione sul suo thread del pool; un semplice `Task.Run` o il callback di un timer non ne hanno alcuna, e `Application.Update` ne ripristina una solo da un `Application.Current` memorizzato. Per questo l'esportazione in background riceve un soggetto esplicito — ID dell'account, tenant, ID del grant, ID del record — e `DocumentService` ricontrolla il grant quando il job parte. Se due profili del browser mostrano lo stesso account, cerca un campo statico che contiene il principal.

Il rifiuto al confine del servizio

`DocumentService.Read(subject, tenant, recordId)` carica il grant corrente e rifiuta quando manca, è revocato, è scaduto o non concede la lettura, oppure quando il suo tenant è diverso da quello richiesto; solo allora chiama il repository. Un controllo sul ruolo non basta: il Reader del tenant A è davvero un Reader, il record 17 esiste in entrambi i tenant e `IsInRole("Reader")` lascia passare la lettura orizzontale.

La prova è un numero. Nella fixture F02 il Reader del tenant A richiede il record 17 del tenant B: rifiuto, e il conteggio delle letture non si muove. Poi revoca il grant Reader dalla console di amministrazione e fai clic su Export nella schermata che mostra ancora il pulsante: il servizio legge il grant corrente e rifiuta prima che si apra qualsiasi file. Ogni decisione scrive un evento di audit con un soggetto opaco, l'azione, la classificazione, la decisione e un ID di correlazione — mai cookie, ID di sessione o token.

La console autonoma `PolicyHarness` (net8.0 nel pacchetto; net10.0 funziona allo stesso modo) applica la regola senza Wisej.NET, IIS o un identity provider: una lettura consentita, poi i rifiuti per confine di tenant, ruolo mancante, account sconosciuto, grant scaduto e grant revocato a zero chiamate al repository. L'harness dimostra la policy ovunque; l'app dimostra che ogni percorso la usa.

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:

- ✓ Distinguere il soggetto applicativo, il principal della richiesta dell'host e l'identità Windows effettiva di una richiesta AccessOps, e spiegare perché nessuno di essi è da solo una decisione di autorizzazione.
- ✓ Spiegare come si risolvono Application.User e IsAuthenticated, compresi l'override di sessione e il ripiego sul principal dell'host, e perché Application.UserIdentity esiste solo su .NET Framework.

- ✓ Tracciare il ciclo di vita della sessione — ApplicationStart, ApplicationRefresh, SessionTimeout, ApplicationExit e Application.RefreshSessionId — e passare un soggetto esplicito al lavoro che viene eseguito fuori dalla sessione.
- ✓ Applicare un controllo su tenant e record al confine del servizio e dimostrare che il rifiuto avviene prima che il repository legga, anche dopo che il ruolo è stato revocato su una schermata aperta.

Guida al lab / esame

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

Lab pratico

Crea la soluzione AccessOps con i tenant A e B, ciascuno con un record 17 dal contenuto diverso, un account Reader nel tenant A e un account senza assegnazioni. Costruisci un DocumentService che controlla il grant corrente, il tenant e il record prima di chiamare un repository che conta le proprie letture, stabilisci il principal di sessione da un verificatore di test attendibile con ApplicationPrincipal e aggiungi una vista diagnostica (tenuta fuori dalle route pubbliche e rimossa prima del packaging) che mostra il tipo di autenticazione, il flag di autenticazione, un soggetto oscurato e l'identità effettiva del sistema operativo. Traccia l'avvio, un clic, un ricaricamento del browser e un job in background in due profili del browser, richiedi il record 17 del tenant B come Reader del tenant A, revoca il grant Reader tra il caricamento della pagina e l'Export su una schermata già aperta, ed esegui il PolicyHarness autonomo per mostrare le stesse decisioni di consenso e rifiuto senza Wisej, IIS o un identity provider.

Punto di partenza

Una nuova applicazione web Wisej.NET chiamata AccessOps su `net10.0` con il pacchetto `Wisej-4` 4.1.7, eseguita su Kestrel sulla tua macchina, più il progetto console `PolicyHarness` del pacchetto aggiunto alla stessa soluzione. Questo modulo non richiede IIS, un dominio Windows o un identity provider: ogni passo viene eseguito in locale, e l'"identità effettiva del sistema operativo" che registri è semplicemente l'account con cui gira il processo `dotnet`.

Approccio suggerito

- ✓ Prepara prima le fixture: il record 17 nel tenant A e nel tenant B con contenuti diversi, un grant Reader per il tenant A e un account senza grant, e un repository il cui `ReadCount` puoi mostrare a schermo.
- ✓ Scrivi `DocumentService.Read(subject, tenant, recordId)` in modo che carichi il grant corrente e rifiuti un grant mancante, revocato o scaduto, un diritto di lettura mancante e un tenant non corrispondente prima di toccare il repository.
- ✓ Collega l'accesso a un verificatore di addestramento che non arriva mai in produzione: verifica sul clic, parcheggia l'account, chiama `Application.Reload()` e al caricamento chiama `ApplicationPrincipal.Establish`, che ruota l'ID di sessione prima di impostare `Application.User`.
- ✓ Aggiungi la vista diagnostica dietro un'opzione di compilazione solo per il debug, poi compila la matrice a quattro colonne per l'avvio, un clic, un ricaricamento e un job in background in due profili del browser, compresi un job avviato con `Application.StartTask` e uno avviato con `Task.Run` che riceve un soggetto esplicito.
- ✓ Esegui i rifiuti — il record 17 del tenant B come Reader del tenant A, poi Export dopo aver revocato il grant su una schermata aperta — verifica che `ReadCount` non si sia mosso, esegui `PolicyHarness` e conferma che la build pubblicata non contiene alcuna vista diagnostica.

Deliverable richiesti

- ✓ Soluzione AccessOps con le fixture dei tenant A e B e un repository che conta le letture
- ✓ Controllo di accesso alle risorse in DocumentService con verifiche su grant, tenant e record
- ✓ Vista diagnostica delle identità, rimossa dalla build pacchettizzata
- ✓ Matrice delle identità a quattro colonne per richiesta, sessione, operazione ed esito
- ✓ Esecuzione di PolicyHarness con una lettura consentita e due rifiuti a zero chiamate al repository

Domande di autovalutazione

- ✓ La tua vista diagnostica mostra la stessa identità effettiva del sistema operativo per entrambi i profili del browser, mentre i soggetti applicativi sono diversi. Che cosa ti dice su quale identità controllerà il file system, e che cosa cambierebbe su IIS con l'autenticazione Windows?
- ✓ Dopo la revoca del grant Reader, il pulsante Export è ancora sulla schermata aperta e `Application.User` porta ancora il claim del ruolo Reader. Quale singola lettura nel tuo codice fa scattare il rifiuto, e perché il claim di ruolo non è quella lettura?
- ✓ La tua esportazione in background gira su `Task.Run` un'ora dopo il clic. Che cosa riceve esattamente, che cosa ricontrolla quando parte, e che cosa andrebbe storto se leggesse invece `Application.User`?

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.