PATTERN DI SICUREZZA E AUTENTICAZIONE · MODULO 4

Accesso personalizzato e sicurezza della sessione

Guida della lezione · Guida al lab / esame

LearnWisej

Guida della lezione

Gestire in prima persona la transazione di accesso, stabilire il principal applicativo e ruotare l'ID di sessione, emettere i cookie dell'host solo su HTTP, la disconnessione come revoca di un grant lato server, e chiudere i percorsi di request forgery, di origine del WebSocket e di redirect. 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

Gestire in prima persona la transazione di accesso, stabilire il principal applicativo e ruotare l'ID di sessione, emettere i cookie dell'host solo su HTTP, la disconnessione come revoca di un grant lato server, e chiudere i percorsi di request forgery, di origine del WebSocket e di redirect.

1. Il problema in produzione

I partner raggiungono AccessOps attraverso la route HTTPS pubblica, dove non c'è alcun challenge Windows su cui appoggiarsi: l'applicazione gestisce l'intero accesso, dalla casella della password fino al momento in cui la console sostituisce la vista di login. Ogni scorciatoia diventa un percorso di attacco: un blocco dell'account che rivela quali nomi esistono, un ID di sessione che sopravvive all'accesso, o una disconnessione che nasconde un pulsante mentre una seconda scheda continua a esportare. Questo modulo costruisce l'accesso dei partner e dimostra che un grant revocato viene rifiutato ovunque.

2. Concetti chiave

Concetto	Perché è importante
Transazione di accesso	La sequenza lato server limitazione → verifica → risoluzione di account e tenant → applicazione della policy → creazione del grant → definizione del principal, in cui qualsiasi passo fallito chiude il tentativo senza lasciare stato dietro di sé.
Limitazione dei login	Contatori dei fallimenti recenti per account e per indirizzo client che rifiutano ulteriori tentativi per un periodo di blocco, tenuti nel servizio di login perché un clic Wisej non può essere limitato dal middleware di ASP.NET Core.
Errore generico	Un'unica frase identica per una password sbagliata, un nome sconosciuto, un account disabilitato e un blocco, con tempi confrontabili, così che la schermata non confermi mai che un account esiste.
Principal applicativo	Il `ClaimsPrincipal` assegnato ad `Application.User` per questa sessione Wisej; è stato di sessione, non un cookie dell'host e non un token Windows.
Rotazione dell'ID di sessione	`Application.RefreshSessionId()` dà alla sessione un nuovo ID senza perderne lo stato, così che un ID esistente prima dell'accesso non valga nulla dopo.
Cookie di autenticazione dell'host	Un ticket cookie di ASP.NET Core emesso con `Secure` e `HttpOnly` da un endpoint HTTPS, necessario solo quando endpoint HTTP ordinari devono riconoscere l'utente.
Grant di login	Un record lato server con un ID casuale opaco, la chiave dell'account, il tenant, la scadenza e un flag di revoca, controllato da ogni operazione protetta; revocarlo è ciò che significa disconnettersi.

3. Il filo della lezione

Gestisci tu la transazione di accesso

La vista di login di AccessOps sulla route pubblica anonima fa esattamente una cosa in `btnSignIn_Click`: legge le due caselle, svuota la casella della password e chiama `LoginService.SignIn`. La verifica delle credenziali spetta a un sistema di identità supportato — ASP.NET Core Identity, una directory o, per gli account locali, una derivazione del framework come PBKDF2 tramite `Rfc2898DeriveBytes.Pbkdf2` con un salt per account e un confronto a tempo costante. Quando il nome utente è sconosciuto, il servizio verifica comunque contro una credenziale fittizia, così che "utente inesistente" costi lo stesso tempo di "password sbagliata".

La limitazione dei tentativi vive in quello stesso servizio. Il `LoginThrottle` di AccessOps blocca un account per 15 minuti dopo cinque fallimenti in una finestra di 15 minuti, e un indirizzo dopo venti, perché il credential stuffing prova molti nomi da un unico punto. Il rate limiting di ASP.NET Core funziona sugli endpoint accanto ad `app.UseWisej()` ma non vede un clic Wisej, che arriva attraverso il WebSocket della sessione. Leggi l'indirizzo da `Application.UserHostAddress` e ricordane i limiti: uno stesso browser può comparire sia come `::1` sia come `127.0.0.1`, e dietro un proxy il client reale arriva solo dalle

intestazioni inoltrate di cui ti fidi. I contatori nella memoria del processo sono corretti solo per un singolo server.

Ogni rifiuto — password errata, nome sconosciuto, account disabilitato, blocco — mostra una sola frase: "That user name and password combination is not valid." Il motivo reale va solo nella tabella di audit dei login. La password stessa non va da nessuna parte: non in `Application.Session`, non in un messaggio di log, non nel testo di un'eccezione.

Stabilisci il principal solo dopo che la policy è superata

Una password verificata dimostra la conoscenza di un segreto, non il diritto di usare AccessOps. Il servizio risolve l'account locale e il suo tenant, rifiuta un account disabilitato o un tenant non approvato con la stessa frase generica, crea un grant di login e solo allora costruisce il principal. L'helper `ApplicationPrincipal.Establish` del pacchetto trasforma account, tenant, grant e ruoli in claim e assegna `Application.User = new ClaimsPrincipal(new ClaimsIdentity(claims, "Portal"))`. Quel principal è stato di sessione: non emette alcun cookie, il middleware di ASP.NET Core sulle altre richieste non lo vede mai e non crea alcun token Windows.

L'ordine conta, e su Wisej.NET 4.1.7 conta anche il tipo di richiesta. Chiamare `Application.RefreshSessionId()` da un clic — un evento WebSocket — cambia l'ID sul server, ma il browser non lo adotta mai; la sua richiesta successiva avvia una sessione completamente nuova e l'accesso va perso. Chiamato durante il caricamento di una pagina, invece, il browser adotta il nuovo ID, che conserva nel sessionStorage della scheda. Per questo AccessOps verifica sul clic, parcheggia il risultato verificato sotto una chiave in sospeso, chiama `Application.Reload()` e conclude al caricamento:

```
// 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();          // prima il nuovo ID (caricamento HTTP)
    ApplicationPrincipal.Establish(p.AccountId, p.TenantId, p.GrantId, p.Roles);

    var login = Application.MainPage;
    Application.MainPage = new ConsolePage(); // sostituisce la vista di login
    login?.Dispose();
}
```

La copia di `ApplicationPrincipal` usata nel corso assegna quindi `Application.User` senza ruotare; è il caricamento a ruotare, una sola volta, prima che l'identità esista.

Cookie su HTTP, disconnessione come revoca

Un frame WebSocket non ha intestazioni di risposta, quindi un clic non può consegnare un `Set-Cookie` con `HttpOnly`. `Application.Cookies` può scrivere cookie HttpOnly solo durante `Program.Main`, `ApplicationStart`, `ApplicationRefresh` o quando `Application.IsWebSocket` è false, e `Application.Browser.CookieStorage` viene scritto da JavaScript, che non può mai impostare HttpOnly.

Quando endpoint HTTP ordinari devono riconoscere il partner, AccessOps mappa prima di `app.UseWisej()` un endpoint HTTPS che riscatta un handle monouso e di breve durata creato dalla sessione, chiama `HttpContext.SignInAsync` con lo schema `"Cookies"` e lascia che sia il gestore dei cookie a emettere `Secure`, `HttpOnly` e `SameSite=Strict`. Quel ticket non appartiene a Wisej: l'ID di sessione vive nel sessionStorage, e Wisej 4.1.7 scrive un proprio cookie HttpOnly di convalida delle richieste che va lasciato com'è.

La disconnessione non è un pulsante nascosto né un cookie cancellato. Il `GrantStore` contiene grant opachi e con scadenza; `ExportService` controlla il grant indicato dal claim `grant` del principal, il tenant e il record prima che il repository venga eseguito. La disconnessione prima revoca il grant, poi registra l'audit, fa scadere il cookie dell'host attraverso l'endpoint di disconnessione, cancella l'identità della sessione e chiama `Application.Exit()`. Impostare `Application.User` a `null` non basta — il getter ricade sul principal dell'host, e sotto l'autenticazione Windows di IIS un ricaricamento riporta subito dentro. Una seconda scheda mostra ancora Export; il suo clic successivo viene rifiutato con zero letture dal repository (fixture F05). `CookieAuthenticationEvents.OnValidatePrincipal` applica lo stesso controllo sul grant alle richieste dell'host.

Chiudi i percorsi cross-site

Il browser allega il cookie dell'host a qualsiasi richiesta, comprese quelle avviate da un altro sito, quindi ogni POST che cambia lo stato e che si fida di quel cookie — accesso, disconnessione, il passaggio del cookie — convalida un token antiforgery. L'ID di sessione di Wisej sta nel sessionStorage, che una pagina estranea non può far inviare al browser, ma l'handshake del WebSocket porta comunque i cookie e non è coperto da CORS: un piccolo middleware prima di `app.UseWisej()` rifiuta un upgrade il cui `Origin` non è un'origine di AccessOps, e l'ingress ripete la regola.

Gli URL di continuazione sono input. Un `returnUrl` proveniente da `Application.QueryString` o dalla query di un endpoint viene accettato solo quando è un percorso locale che non inizia con `//` o `/\`, oppure una voce di una allowlist esplicita sul server; qualsiasi altra cosa diventa `/`. Ricava schema e host solo dopo che `app.UseForwardedHeaders()` è stato configurato per i proxy noti. La fixture F11 — un POST o un upgrade cross-origin — deve concludersi senza alcun cambio di stato.

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:

- ✓ Implementare un accesso che verifica le credenziali con un sistema di identità supportato, limita i tentativi, restituisce errori generici e tiene le password fuori dai log e dallo stato di sessione.

- ✓ Stabilire il principal applicativo di Wisej.NET solo dopo che le policy su account e tenant sono superate, ruotare l'ID di sessione con Application.RefreshSessionId e sostituire la vista di login.
- ✓ Emettere e far scadere cookie dell'host Secure e HttpOnly da un endpoint HTTPS dedicato invece che da un evento WebSocket, e trattare la disconnessione come revoca di un grant lato server controllato da ogni operazione protetta.
- ✓ Chiudere i percorsi cross-site con l'antiforgery sui post che cambiano lo stato, i controlli sull'origine del WebSocket e una allowlist per gli URL di continuazione.

Guida al lab / esame

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

Lab pratico

Dai alla route dei partner di AccessOps il suo accesso: una vista di login sulla route pubblica anonima che verifica un'identità di fixture attraverso un verificatore supportato o un mock di addestramento che non arriva mai in produzione, rifiuta una credenziale errata e un account disabilitato con lo stesso messaggio generico, risolve l'account locale e il tenant, stabilisce il principal con ApplicationPrincipal e ruota l'ID di sessione. Appoggiala a un GrantStore di grant di login opachi, con scadenza e revocabili, che il servizio di esportazione controlla prima di ogni lettura, aggiungi un endpoint HTTPS di accesso e disconnessione per il cookie dell'host, poi apri due profili del browser, revoca il grant con la seconda scheda aperta e riprova l'esportazione in entrambe le schede e dopo un ricaricamento, e prova un post contraffatto, un upgrade WebSocket da un'origine estranea e un URL di continuazione esterno.

Punto di partenza

Apri la soluzione AccessOps del Modulo 1 con le fixture dei tenant A e B, il repository che conta le letture e il controllo di `DocumentService`, e aggiungi una `LoginPage` sulla route pubblica anonima. Eseguila su un nome host HTTPS (o su `https://localhost`) e tieni pronti due profili del browser per la prova di revoca.

Approccio suggerito

- ✓ Scrivi `LoginService.SignIn` con `LoginThrottle`, un controllo con credenziale fittizia per i nomi sconosciuti e un'unica frase di errore generica; registra gli esiti nella tabella di audit e conferma che nessun valore di password raggiunge un log o `Application.Session`.
- ✓ Aggiungi `GrantStore` (ID casuale opaco, account, tenant, scadenza, flag di revoca) e fai in modo che `ExportService` controlli il grant, il tenant e il record prima che il repository venga eseguito.
- ✓ Parcheggia l'accesso verificato sul clic, chiama `Application.Reload()` e in `ApplicationRefresh` chiama `Application.RefreshSessionId()`, poi `ApplicationPrincipal.Establish`, poi sostituisci la vista di login con la console; registra nel log il vecchio e il nuovo ID come un booleano "rotated".
- ✓ Mappa gli endpoint HTTPS di passaggio del cookie e di disconnessione prima di `app.UseWisej()`, con l'antiforgery, un controllo di `Origin` sugli upgrade WebSocket e un controllo di percorso locale per `returnUrl`; cattura gli attributi di `Set-Cookie` con il valore oscurato.
- ✓ Revoca il grant dalla scheda 1, riprova Export nella scheda 2 e dopo un ricaricamento, poi esegui le prove del post contraffatto, dell'upgrade da un'origine errata e dell'URL di continuazione esterno, e disegna il diagramma delle transizioni dello stato di autenticazione.

Deliverable richiesti

- ✓ Vista di login con errori generici, limitazione dei tentativi e gestione degli account disabilitati
- ✓ Definizione del principal con rotazione dell'ID di sessione

- ✓ Endpoint HTTPS per il cookie con gli attributi di Set-Cookie oscurati
- ✓ GrantStore con revoca applicata su tutte le schede aperte
- ✓ Diagramma delle transizioni dello stato di autenticazione e prove su post contraffatto, origine errata e URL di continuazione

Domande di autovalutazione

- ✓ La tua prima versione chiamava `Application.RefreshSessionId()` dentro `btnSignIn_Click` e la console compariva, per poi sparire al clic successivo. Che cosa ha fatto il browser con il nuovo ID, e perché il caricamento di una pagina si comporta diversamente?
- ✓ Dopo la disconnessione, la scheda 2 mostra ancora il pulsante Export e conserva ancora un principal con un claim di ruolo. Quale singolo controllo fa fallire la sua esportazione successiva, e perché cancellare soltanto il cookie dell'host non otterrebbe lo stesso risultato?
- ✓ L'ID di sessione di Wisej non sta in un cookie, eppure AccessOps ha comunque bisogno dell'antiforgery e di un controllo sull'origine. Quali richieste portano credenziali ambientali che un altro sito potrebbe far partire?

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.