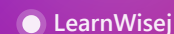


Checklist di irrobustimento e capstone

Guida della lezione · Guida al lab / esame



Guida della lezione

HTTPS e header di sicurezza, flag dei cookie, impostazioni di sessione, aggiornamento delle dipendenze, rate limiting, irrobustimento del deployment e la revisione di sicurezza di AccessOps. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

HTTPS e header di sicurezza, flag dei cookie, impostazioni di sessione, aggiornamento delle dipendenze, rate limiting, irrobustimento del deployment e la revisione di sicurezza di AccessOps.

1. Il problema in produzione

Ora AccessOps verifica le password rispetto a un hash con salt, risolve i permessi in un servizio, invia i documenti solo tramite `DocumentService` e sottopone ad audit ogni approvazione. Nulla di tutto questo aiuta se la prima richiesta viaggia in chiaro, se uno script può leggere il cookie di sessione, o se la build distribuisce un package con un advisory pubblicato. L'ultimo miglio spetta all'host: il certificato, gli header di risposta, i flag dei cookie, i limiti delle richieste e la versione di ogni elemento del grafo delle dipendenze.

2. Concetti chiave

Concetto	Perché è importante
HSTS	Un header <code>Strict-Transport-Security</code> che dice al browser di rifiutare HTTP in chiaro per questo host per un tempo dichiarato, così una visita successiva non viene mai intercettata sul redirect.
Content-Security-Policy	Un header che elenca le origini da cui una pagina può caricare script, stili e frame; trasforma uno script iniettato in una richiesta bloccata invece che in codice eseguito in una sessione attiva.
frame-ancestors	La direttiva CSP (<code>X-Frame-Options</code> è l'equivalente più vecchio) che impedisce a un altro sito di inserire AccessOps in un frame e indurre un manager a cliccare Approve.
flag dei cookie	<code>Secure</code> tiene il cookie di sessione lontano da HTTP in chiaro, <code>HttpOnly</code> lo tiene lontano da JavaScript, e <code>SameSite</code> impedisce che venga allegato a una richiesta avviata da un altro sito.
limiti delle richieste	Tetti a livello di host sulla dimensione del corpo e degli header, che rifiutano un upload troppo grande prima che giri una sola riga di codice di AccessOps.
token bucket	Un contatore che concede un tentativo solo finché contiene un token e si ricarica a velocità fissa, così il traffico può avere una piccola raffica e poi rallenta fino quasi a fermarsi.
audit delle dipendenze	Una verifica ripetibile di ogni package diretto e transitivo, e del runtime, rispetto agli advisory pubblicati — registrata con una data e una decisione.

3. Il filo della lezione

HTTPS è compito dell'host, HSTS è compito del browser

TLS viene terminato fuori dal tuo C#: da IIS con un certificato associato, da Kestrel configurato con un certificato, o dal reverse proxy davanti a entrambi. La tua parte è rendere inutile HTTP in chiaro. Reindirizza ogni richiesta HTTP a livello di host, poi invia `Strict-Transport-Security` con un `max-age` lungo così che il browser smetta del tutto di tentare lo schema non sicuro. Aggiungi `includeSubDomains` solo quando ogni sottodominio di AccessOps serve davvero TLS, e considera `preload` una decisione che non puoi annullare in fretta. Un dettaglio di deployment coglie spesso di sorpresa: dietro un proxy la richiesta arriva all'applicazione non cifrata, quindi se non è abilitata la gestione dei forwarded headers l'app crede di essere in un contesto non sicuro e non marca nulla come `Secure`.

Gli header sono i controlli più economici di questo corso, e si impostano una volta per la pipeline invece che schermata per schermata. `X-Content-Type-Options: nosniff` impedisce a un browser di indovinare che un documento caricato sia in realtà HTML. `Referrer-Policy: strict-origin-when-cross-origin` impedisce che un identificativo di documento finisca nei log di un altro sito. `frame-ancestors 'none'` rifiuta del tutto l'inserimento in un frame. Impostali dove passa ogni risposta, compreso l'handler dei link di condivisione firmati del Modulo 5 — un URL separato, e quello che ci si dimentica.

La Content-Security-Policy richiede attenzione, perché il browser sta eseguendo il client Wisej.NET. Quel client, le risorse del tema e gli script dei tuoi widget arrivano dalla stessa origine, quindi `default-src`

'self' li copre; il motore di layout scrive stili inline, quindi `style-src` di solito ha bisogno di `'unsafe-inline'`; e un interop che costruisce codice client a runtime ti costringe ad allargare `script-src`. Distribuiscila prima come `Content-Security-Policy-Report-Only`, passa per ogni schermata, leggi le violazioni, poi rendila vincolante.

```
app.UseForwardedHeaders();           // TLS l'ha terminato il proxy, non noi
app.UseHttpsRedirection();
app.UseHsts();                       // Strict-Transport-Security
app.Use(async (ctx, next) =>
{
    var h = ctx.Response.Headers;
    h["X-Content-Type-Options"] = "nosniff";
    h["Referrer-Policy"] = "strict-origin-when-cross-origin";
    h["Content-Security-Policy-Report-Only"] = AccessOpsCsp;
    await next();
});
// L'handler di Wisej.NET gira dopo, quindi ogni risposta porta gli header
```

Cookie, limiti di sessione e un pulsante che non è un endpoint

Una volta riuscito il login, il cookie di sessione è la credenziale, quindi riceve tutti e tre i flag. `Secure` significa che non viaggia mai senza TLS. `HttpOnly` significa che anche un'injection di script riuscirà in un widget personalizzato non può leggerlo. `SameSite` impostato a `Lax` o `Strict` significa che una richiesta avviata da un altro sito non lo porta con sé — la difesa più economica contro un'approvazione contraffatta. Su ASP.NET Framework sono impostazioni degli elementi cookie e session-state di `Web.config`; su ASP.NET Core sono opzioni della cookie policy. Verificali negli strumenti per sviluppatori del browser, non rileggendo la configurazione.

La durata della sessione ha due metà. Wisej.NET legge le proprie impostazioni dal file di configurazione JSON dell'applicazione accanto alla pagina iniziale, ed è lì che va il timeout di sessione del framework: la rete di sicurezza che recupera una sessione abbandonata e il suo stato lato server. Non è il logout per inattività che hai costruito nel Modulo 2, perché il client mantiene viva la connessione e un browser abbandonato può sembrare traffico — l'inattività va misurata dall'ultimo vero evento dell'utente, e il logout esplicito deve eliminare il `SessionContext` in `Application.Session` così che il pulsante Indietro arrivi a una pagina di login. I limiti delle richieste sono la terza impostazione: limita la dimensione del corpo e degli header sull'host così che un upload troppo grande venga rifiutato prima di entrare in `DocumentService`.

Il rate limiting si divide lungo una linea importante. L'handler dei link di condivisione è un URL vero, quindi un middleware può limitarlo direttamente — un limiter a token bucket partizionato per indirizzo del client, che risponde 429 senza rivelare se il link esisteva. Il pulsante di login non è un URL: ogni evento Wisej.NET, dallo scorrimento di una griglia a un clic su Sign in, arriva allo stesso endpoint dell'applicazione dentro una sessione esistente, quindi il middleware non li distingue. Il rate limiting del login vive quindi nel servizio, come bucket indicizzato per nome utente inviato e indirizzo del client e verificato prima dell'hash. Un rifiuto restituisce lo stesso messaggio generico di una password errata, viene registrato nell'audit con il suo correlation ID, e conta su un archivio condiviso invece che su un campo statico quando AccessOps gira su più server.

Aggiornato, e dimostrabilmente

Esegui `dotnet list package --vulnerable --include-transitive` e leggi tutto: il package che non hai mai scelto, tirato dentro tre livelli più sotto, è il problema tipico. Aggiungi `--outdated` e `--deprecated` per ciò che è semplicemente vecchio o abbandonato. Gli SDK .NET recenti verificano gli advisory anche durante il restore, il che trasforma l'audit in output della build invece che in un compito che qualcuno si deve ricordare. Mettilo nella pipeline così che un nuovo advisory faccia fallire una build invece di emergere in un penetration test.

I package sono solo una parte del grafo. Il runtime .NET, il web host e l'immagine base del container hanno ciascuno bisogno di una cadenza di aggiornamento e di una versione registrata. Wisej.NET stesso è una dipendenza con versione: fissala, leggi le release notes prima di aggiornare, e ritesta il tema e i widget personalizzati dopo un aggiornamento maggiore. Il JavaScript di terze parti nella web root è invisibile al grafo dei package, quindi elenca ogni file a mano con la sua versione e la sua provenienza. Poi metti l'audit per iscritto — data, strumento, problemi trovati, una decisione per ciascuno (aggiornare, mitigare, o accettare con una scadenza) e la data della prossima revisione.

La revisione pre-rilascio, modulo per modulo

Il deliverable del capstone è una revisione, non una funzionalità. Prendi la mappa di posizionamento dei controlli del Modulo 1 e ripercorri il corso in avanti, verificando l'app e non le intenzioni, e testa come un attaccante invece che come un utente: richiedi il documento di un altro reparto come dipendente qualsiasi, riusa un link di condivisione dopo la sua ora, chiama un metodo di approvazione come qualcuno che non è il manager, invia venti password sbagliate in dieci secondi.

- Modulo 2 — blocco dell'account, hashing, identità verificata, timeout di inattività, logout che elimina lo stato.
- Modulo 3 — ogni metodo di servizio protetto da guard, regole per reparto e manager sotto la UI, la richiesta contraffatta rifiutata.
- Modulo 4 — PKCE, validazione di ID token e nonce, mappatura dei claim, disconnessione che chiude entrambe le sessioni.
- Modulo 5 — validazione degli upload, download tramite il servizio, link firmati con scadenza, query parametrizzate, output codificato.
- Modulo 6 — righe di audit per i comandi sensibili, correlation ID nei log, nessun segreto né dato personale in nessuno dei due.
- Modulo 7 — HTTPS e HSTS, l'insieme degli header, flag dei cookie, limiti di sessione e delle richieste, rate limiting, un audit datato.

Ogni problema riceve una gravità, un responsabile e una correzione oppure un rischio accettato con una scadenza; una checklist con vuoti inspiegati non è completa. L'approvazione finale significa una cosa sola: ogni confine disegnato nel Modulo 1 ha ora un controllo che viene fatto rispettare da un servizio, non da una schermata.



La regola sulla sicurezza

Il browser non è mai considerato affidabile. Ogni decisione che conta — chi è l'utente, che cosa può fare, quale record può vedere — viene presa sul server, in un servizio, e la UI si limita a rifletterla. Se una regola esiste solo come pulsante nascosto, non esiste.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Configurare HTTPS, HSTS, flag sicuri dei cookie e header di sicurezza per un deployment Wisej.NET.
- ✓ Regolare impostazioni di sessione, limiti delle richieste e rate limiting per resistere agli abusi.
- ✓ Mantenere aggiornati dipendenze e runtime con un processo ripetibile.
- ✓ Condurre una revisione di sicurezza pre-rilascio basata su una checklist e consegnare il capstone di AccessOps.

Guida al lab / esame

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



Lab pratico

Completa il capstone di AccessOps: imponi HTTPS con HSTS e imposta gli header di sicurezza, marca i cookie come Secure e HttpOnly con una policy SameSite restrittiva, imposta il timeout di sessione e i limiti di dimensione delle richieste, aggiungi il rate limiting su login e link di condivisione, esegui un audit delle dipendenze e registrane i risultati, poi applica all'app l'intera checklist di questo corso e correggi ogni problema rilevato.

Punto di partenza

Apri la soluzione AccessOps così com'è dopo il Modulo 6: una pagina di login con blocco dell'account, un `SessionContext` che tiene l'identità verificata, un `PermissionService` che protegge `DocumentService` e `ApprovalService`, l'accesso esterno tramite OpenID Connect, una pipeline di upload e download con un handler per i link di condivisione firmati, e un `AuditService` che scrive righe marcate con il correlation ID. Nella configurazione dell'host non è ancora stato toccato nulla: in sviluppo l'app risponde su HTTP in chiaro, i suoi cookie non hanno flag, e nessun package è stato verificato rispetto a un advisory.

Approccio suggerito

- ✓ Associa un certificato e reindirizza HTTP su HTTPS a livello di host, aggiungi `Strict-Transport-Security`, e verifica la gestione dei forwarded headers se davanti ad AccessOps c'è un reverse proxy.
- ✓ Aggiungi un unico punto nella pipeline che imposta `X-Content-Type-Options`, `Referrer-Policy`, `frame-ancestors` e una Content-Security-Policy, partendo in modalità report-only finché ogni schermata di AccessOps e l'handler dei link di condivisione non si caricano senza violazioni.
- ✓ Marca il cookie di sessione come `Secure` e `HttpOnly` con una policy `SameSite` restrittiva, imposta il timeout di sessione di Wisej.NET nel file di configurazione dell'applicazione, e limita la dimensione del corpo delle richieste sull'host.
- ✓ Metti dentro il servizio di login un token bucket indicizzato per nome utente e indirizzo del client, e un limiter nel middleware sull'URL dei link di condivisione, registrando nell'audit ogni rifiuto con il suo correlation ID.
- ✓ Esegui `dotnet list package --vulnerable --include-transitive`, registra i problemi trovati con una decisione e un responsabile, poi applica all'app in esecuzione la checklist dei sette moduli e correggi ciò che emerge.

Deliverable richiesti

- ✓ Configurazione di HTTPS, HSTS e header di sicurezza
- ✓ Impostazioni di irrobustimento di cookie e sessione
- ✓ Rate limiting su login e link di condivisione

- ✓ Report dell'audit delle dipendenze
- ✓ Checklist della revisione di sicurezza completata, con le correzioni

Domande di revisione per lo studente

- ✓ Un middleware può applicare il rate limiting all'handler dei link di condivisione ma non al pulsante Sign in. Che cosa hanno di diverso le due richieste in un'app Wisej.NET, e dove deve vivere invece il limite sul login?
- ✓ La tua Content-Security-Policy è vincolante e la griglia viene renderizzata senza stili. Quale direttiva è la causa probabile, e come te lo avrebbe detto la modalità report-only prima che lo facesse il deployment?
- ✓ L'audit delle dipendenze elenca un package vulnerabile che nessuno in AccessOps referencia direttamente. Come è entrato nella build, e quali sono i due modi onesti per chiudere il problema?



La regola sulla sicurezza

Il browser non è mai considerato affidabile. Ogni decisione che conta — chi è l'utente, che cosa può fare, quale record può vedere — viene presa sul server, in un servizio, e la UI si limita a rifletterla. Se una regola esiste solo come pulsante nascosto, non esiste.