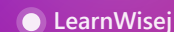


Audit, segreti e configurazione

Guida della lezione · Guida al lab / esame



Guida della lezione

Audit trail per i comandi sensibili, correlation ID, gestione dei segreti, configurazione d'ambiente, logging senza far trapelare dati. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

Audit trail per i comandi sensibili, correlation ID, gestione dei segreti, configurazione d'ambiente, logging senza far trapelare dati.

1. Il problema in produzione

Ora AccessOps autentica le persone, ne risolve i permessi e protegge i propri documenti — e nulla di tutto questo aiuta la mattina in cui qualcuno chiede chi ha approvato la richiesta 4471, da quale indirizzo, e se qualcuno era stato rifiutato prima. I cinque moduli precedenti hanno deciso che cosa è consentito; questo decide che cosa è ricostruibile dopo. Chiude anche la falla che trasforma un piccolo incidente in uno grande: una password del database o un client secret OIDC in un file committato due anni fa.

2. Concetti chiave

Concetto	Perché è importante
audit trail	Un registro di business di ogni comando sensibile — chi ha agito, su che cosa, quando e da dove — conservato per un periodo dichiarato perché qualcuno dovrà risponderne.
archiviazione append-only	Una tabella di audit su cui l'account dell'applicazione può fare solo <code>INSERT</code> . Se il codice che scrive la cronologia può anche riscriverla, la cronologia non dimostra nulla.
evidenza di manomissione	Una sequenza senza buchi più una catena di hash, in cui l'hash di ogni riga copre i propri campi e l'hash precedente, così una riga modificata spezza la catena.
correlation ID	Un identificativo generato all'inizio di un comando e ripetuto su ogni riga di log e di audit che esso produce, così un singolo clic può essere ricostruito in seguito.
scope di logging	Un blocco di proprietà implicito — <code>ILogger.BeginScope</code> o un enricher di Serilog — che collega il correlation ID e l'autore a tutto ciò che viene registrato al suo interno.
archivio dei segreti	Il posto in cui una credenziale vive fuori dalla build: user secrets in locale, variabili d'ambiente in un container, un vault gestito in produzione — oppure una managed identity, senza nulla di memorizzato.
oscuramento	Una regola su che cosa non deve mai arrivare a un sink di log — password, token, cookie, stringhe di connessione e dati personali — fatta rispettare dal logging strutturato, non dalla memoria.

3. Il filo della lezione

Il record che sopravvive alla discussione

Due cose diverse vengono chiamate "logging". Una riga di log è diagnostica e può essere campionata o eliminata a rotazione dopo una settimana. Una riga di audit è una prova: ha un periodo di conservazione, un responsabile e un lettore che non è uno sviluppatore. AccessOps ha bisogno di un `AuditService` come unico punto di ingresso per il secondo tipo, richiamato da ogni comando sensibile — approvare una richiesta, concedere o revocare un permesso, scaricare un documento riservato, e ogni tentativo di login del Modulo 2. Nient'altro scrive nella tabella di audit.

Una riga risponde a sei domande: l'identificativo stabile dell'autore, il nome dell'azione (`Approval.Approve` , `Document.Download`), il tipo e l'ID del destinatario, il timestamp UTC dall'orologio del server, l'indirizzo e lo user agent del client, e il correlation ID. Registra anche l'esito, perché un diniego è spesso più interessante di un successo. In Wisej.NET l'indirizzo di rete e lo user agent del browser del client arrivano dal contesto applicativo per sessione e non da qualcosa che la pagina invia, quindi catturali una sola volta quando il `SessionContext` viene creato dopo il login. Prendi anche l'autore da quel contesto verificato — mai da un controllo o da un parametro fornito dal chiamante.

Poi rendi la riga difficile da cancellare. Il login al database dell'applicazione riceve `INSERT` e niente altro, così un bug o un'istruzione iniettata non possono cancellare in silenzio un martedì intero. Dai a ogni riga una sequenza senza buchi e un hash sui propri campi più l'hash della riga precedente; un passaggio

pianificato ricalcola la catena e indica la prima riga che non corrisponde più. Questa è *evidenza* di manomissione, non prevenzione — rende visibile una modifica invece che impossibile. Infine, scrivi la riga di audit nella stessa transazione della modifica di business, così un'approvazione senza il suo record è irraggiungibile.

```
using (_log.BeginScope(new Dictionary<string, object>
    { ["CorrelationId"] = ctx.CorrelationId, ["UserId"] = ctx.UserId }))
using (var tx = _db.BeginTransaction())
{
    _approvals.Approve(requestId, ctx);           // la modifica di business
    _audit.Record(AuditEntry.For(ctx,             // autore, indirizzo, correlazione
        "Approval.Approve", "ApprovalRequest", requestId));
    tx.Commit();                                 // entrambe, o nessuna
}
```

Un ID, una storia

Il correlation ID viene creato dove inizia un comando dell'utente — in cima a un gestore di evento snello, prima che venga chiamato qualsiasi servizio. Basta un `Guid`; se usi già `System.Diagnostics.Activity`, usa il trace identifier dell'activity corrente così che l'ID corrisponda a tutto ciò che parla il formato di trace W3C. Mettilo nel contesto del comando che porta anche l'autore e l'indirizzo del client, e passa quel contesto a ogni servizio. Non tornare a pescare nella sessione dall'interno di un servizio: l'elaborazione in background avviata con `Application.StartTask` può sopravvivere alla sessione che lo ha avviato.

Collegare l'ID alle righe di log non è compito del chiamante. Apri uno scope di logging attorno al comando — `ILogger.BeginScope` con un dizionario di proprietà, oppure il `LogContext` di Serilog con un enricher — e ogni riga scritta al suo interno porterà automaticamente il correlation ID e l'autore, comprese le righe delle librerie sottostanti. Usa il logging strutturato così che l'ID sia una proprietà con nome nel sink, il che fa di "mostrami tutto per la correlazione 7f3c" una query invece di un grep. Poi rendilo visibile: quando un comando fallisce, chiudi la frase cortese con l'ID come riferimento.

I segreti appartengono all'ambiente, non al repository

A questo punto AccessOps ha tre credenziali: la stringa di connessione al database, il client secret OIDC del Modulo 4 e la chiave di firma dei link di condivisione con scadenza del Modulo 5. Nessuna di esse va in un file tracciato. In sviluppo usa gli user secrets di .NET — il progetto riceve uno `UserSecretsId` e i valori vivono fuori dalla cartella della soluzione, quindi un clone del repository non contiene credenziali. In deployment forniscili come variabili d'ambiente o leggili da un vault gestito tramite un provider di configurazione. Meglio ancora, usa una managed identity: l'app si autentica come se stessa e riceve un token di breve durata, quindi non c'è alcun segreto memorizzato che possa trapelare.

Funziona perché la configurazione di ASP.NET Core è stratificata e le sorgenti successive prevalgono: `appsettings.json`, poi la sovrapposizione per ambiente, poi gli user secrets in sviluppo, poi le variabili d'ambiente, poi un provider di vault. I file committati contengono la struttura e valori predefiniti innocui; i valori che hanno la forma di un segreto arrivano ambiente per ambiente. Il `Default.json` di Wisej.NET configura lo shell — tema, timeout di sessione, finestra di avvio — e non è un archivio di segreti più di quanto lo sia `Web.config`.

Un segreto che non puoi ruotare con calma è un segreto che ruoterai male alle 3 di notte, quindi qui la procedura di rotazione è un deliverable. Annota, per ogni segreto: il responsabile, l'archivio, la scadenza e i passi di sostituzione. Progetta per avere due credenziali valide contemporaneamente — un secondo client secret, un secondo login al database — così la rotazione è un riavvio progressivo invece di un fermo. E una volta che un valore è stato committato, cancellare la riga non serve: ruotalo e revoca quello vecchio.

Registra l'incidente, non la persona

I log vengono copiati negli indici di ricerca, spediti a servizi di terze parti, aperti sui portatili del supporto e conservati molto più a lungo di quanto chiunque intenda. Devono contenere abbastanza da ricostruire un incidente e niente che sia dannoso di per sé. Non registrare mai una password (nemmeno una fallita), un token, un header di autorizzazione, un cookie, una stringa di connessione o il contenuto di un documento riservato.

I dati personali richiedono la stessa disciplina per un motivo diverso: una riga di audit è un registro legittimo con una politica di conservazione, mentre una riga di debug che contiene l'indirizzo di casa di un dipendente ne è una copia fuori controllo. Registra l'ID utente stabile invece di nome ed email, e l'ID del documento invece di un nome di file che può essere a sua volta sensibile. Il logging strutturato aiuta, perché registri proprietà con nome prese da una allowlist invece di serializzare un intero oggetto richiesta e sperare.

- Mai nel log: password, token, cookie, stringhe di connessione, contenuti dei documenti.
- Registra per riferimento: ID utente, ID documento, ID richiesta — non nomi o dati personali in testo libero.
- Fallo rispettare nel sink: una policy di mascheramento, più un test che verifica che un marcatore di segreto noto non arrivi mai nell'output.

Chiudi il cerchio con una revisione più che con una regola: percorri da capo a fondo l'output di log di un'approvazione reale e di un download reale, e marca ogni campo come necessario, oscurato o rimosso.



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:

- ✓ Registrare chi ha fatto che cosa, quando e da dove per ogni comando sensibile con un audit trail a prova di manomissione.
- ✓ Portare un correlation ID attraverso i log così che un'azione dell'utente possa essere tracciata tra i servizi.
- ✓ Tenere i segreti fuori dal controllo di versione usando configurazione d'ambiente, archivi dei segreti e managed identity.

- ✓ Registrare abbastanza per indagare su un incidente senza registrare password, token o dati personali.

Guida al lab / esame

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



Lab pratico

Aggiungi audit e gestione dei segreti ad AccessOps: un `AuditService` che registra approvazioni, modifiche ai permessi e download di documenti con utente, orario, indirizzo e correlation ID; una tabella di audit append-only; un correlation ID creato per ogni comando e scritto in ogni riga di log; stringhe di connessione e client secret OIDC spostati nella configurazione d'ambiente con una procedura di rotazione documentata; e una revisione dei log che oscura i campi sensibili.

Punto di partenza

Apri la soluzione AccessOps così com'è alla fine del Modulo 5: un `SessionContext` verificato creato al login, un `PermissionService` e una guard di autorizzazione dentro `DocumentService` e `ApprovalService`, l'accesso OIDC, e la funzione documenti con la sua pipeline di upload e i link di condivisione firmati. La schermata di approvazione, l'editor dei permessi per gli amministratori e l'elenco dei documenti funzionano tutti — semplicemente non lasciano traccia di ciò che hanno fatto, e la stringa di connessione al database e il client secret OIDC stanno ancora in un file di configurazione tracciato dal controllo di versione.

Approccio suggerito

- ✓ Crea la tabella di audit con una sequenza senza buchi, autore, azione, tipo e ID del destinatario, timestamp UTC, indirizzo del client, user agent, esito, correlation ID e una colonna di hash, poi concedi al login applicativo sul database solo `INSERT` — niente `UPDATE`, niente `DELETE`.
- ✓ Scrivi `AuditService` come unico scrittore di quella tabella, prendendo l'autore e i dati del client dal `SessionContext` verificato, calcolando l'hash di ogni riga sui propri campi più l'hash della riga precedente, e facendo partecipare l'insert alla stessa transazione della modifica di business.
- ✓ Aggiungi un contesto per comando che genera un correlation ID in cima a ogni gestore di evento, passalo a `ApprovalService`, `DocumentService` e all'editor dei permessi, e apri uno scope di logging così che ogni riga scritta durante il comando porti l'ID e l'autore.
- ✓ Sposta la stringa di connessione, il client secret OIDC e la chiave di firma dei link di condivisione fuori dai file di configurazione tracciati, negli user secrets per lo sviluppo e nelle variabili d'ambiente per il deployment, poi scrivi la procedura di rotazione con responsabili, date di scadenza e la sovrapposizione con due credenziali valide.
- ✓ Chiama `AuditService` da approvazione, concessione e revoca di permessi e download di documenti — registrando i dinieghi oltre ai successi — poi esegui un'approvazione e un download e rivedi ogni riga di log risultante cercando password, token, header, stringhe di connessione e dati personali.

Deliverable richiesti

- ✓ `AuditService` e tabella di audit append-only

- ✓ Correlation ID in ogni comando e in ogni riga di log
- ✓ Segreti spostati nella configurazione d'ambiente
- ✓ Procedura di rotazione dei segreti
- ✓ Revisione dell'oscuramento nei log

Domande di revisione per lo studente

- ✓ La tua tabella di audit è in sola scrittura per l'applicazione, e ogni riga è concatenata con gli hash. Indica una cosa che un attaccante con lo stesso login al database dell'applicazione può ancora fare al trail, e spiega esattamente come la mostrerebbe il passaggio di verifica.
- ✓ Un'approvazione parte dalla schermata di approvazione e termina in un task in background avviato con `Application.StartTask` dopo che l'utente si è disconnesso. Da dove prende quel task il correlation ID e l'autore, e che cosa si rompe se li legge invece dalla sessione?
- ✓ Uno sviluppatore rimuove il client secret OIDC dal file di configurazione tracciato e fa il push della modifica. Perché il segreto è ancora compromesso, e che cosa deve succedere prima che l'incidente sia davvero chiuso?



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.