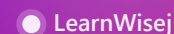


Autenticazione e identità di sessione

Guida della lezione · Guida al lab / esame



Guida della lezione

Flussi di login, gestione delle password, un SessionContext con un'identità verificata, timeout, logout, "ricordami" e session fixation. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

Flussi di login, gestione delle password, un SessionContext con un'identità verificata, timeout, logout, "ricordami" e session fixation.

1. Il problema in produzione

AccessOps contiene anagrafiche dei dipendenti, documenti riservati e richieste di approvazione, quindi la prima schermata che chiunque raggiunge è anche il primo controllo che conta. Tutto ciò che il resto di questo corso fa rispettare — permessi, regole a livello di record, audit trail — poggia su un unico presupposto: che il server sappia chi c'è dall'altra parte di questa sessione e sia in grado di dimostrarlo. Il Modulo 1 ha disegnato i confini di fiducia; questo modulo mette un'identità verificata dentro il confine di sessione e impedisce che venga indovinata, presa in prestito o fatta sopravvivere oltre il dovuto.

2. Concetti chiave

Concetto	Perché è importante
hash lento con salt	Una derivazione a senso unico della password con un salt casuale per utente e un numero di iterazioni volutamente costoso, così che una tabella rubata non possa essere invertita e ogni tentativo costi lavoro vero.
confronto a tempo costante	Confrontare due hash con un numero fisso di operazioni usando <code>CryptographicOperations.FixedTimeEquals</code> , così il tempo di risposta del server non rivela mai quanti byte di un tentativo erano giusti.
SessionContext	L'oggetto creato solo dopo la verifica di una password, che contiene id del dipendente, reparto, ruoli e dati di accesso, ed è memorizzato in <code>Application.Session</code> come unico punto a cui qualsiasi codice chiede chi è l'utente.
timeout di sessione	Il periodo di inattività dopo il quale Wisej.NET elimina la sessione lato server e tutto ciò che contiene, configurato per l'applicazione in <code>Default.json</code> invece che stabilito a occhio schermata per schermata.
session fixation	Un attacco in cui l'identificatore che ha portato la pagina di login anonima porta anche, dopo, la sessione autenticata, così che un identificatore piazzato dall'attaccante diventa un identificatore con accesso effettuato.
throttling del login	Contare i fallimenti per account e per indirizzo del client e rifiutare ulteriori tentativi per un periodo di blocco: è questo che trasforma una macchina per tirare a indovinare senza limiti in pochi tentativi all'ora.
record di audit del login	Una riga per tentativo con il nome utente provato, l'ora, l'indirizzo del client e l'esito reale — il posto in cui viene scritto il motivo specifico del fallimento, dato che lo schermo ne mostra sempre solo uno generico.

3. Il filo della lezione

Verificare una password che non conservi mai

AccessOps non memorizza password. Per ogni dipendente memorizza tre valori: l'algoritmo e il numero di iterazioni usati, un salt casuale prodotto da un generatore crittografico, e i byte derivati. L'accesso esegue la stessa derivazione sulla password appena digitata, usando il salt e il numero di iterazioni di quel dipendente, e confronta il risultato con i byte memorizzati. Non viene decifrato nulla, perché nulla è stato cifrato — un hash va in una sola direzione, e il salt fa sì che due dipendenti che hanno scelto la stessa password producano valori memorizzati completamente diversi.

Qui sono adatte due implementazioni. PBKDF2 tramite `Rfc2898DeriveBytes` dà un controllo esplicito: SHA-256, un salt di 16 byte, un output di 32 byte e un numero di iterazioni abbastanza alto da far durare una verifica una frazione di secondo percepibile. Il `PasswordHasher` di ASP.NET Core Identity, invece, impacchetta in un'unica stringa l'indicatore dell'algoritmo, il numero di iterazioni, il salt e l'hash, e restituisce un `PasswordVerificationResult` il cui valore `SuccessRehashNeeded` ti dice di ripetere la

derivazione con i parametri attuali. In entrambi i casi il costo è il punto: ogni tentativo dell'attaccante deve pagare una derivazione completa.

La password in sé vive per la durata di una chiamata di metodo. Viene letta dalla `TextBox` di login, passata a `PasswordService.Verify` e dimenticata: mai un campo della finestra, mai un valore nella riga di audit, mai parte di un messaggio di eccezione e mai una riga di log — nemmeno troncata.

```
public bool Verify(string password, StoredCredential stored)
{
    var candidate = Rfc2898DeriveBytes.Pbkdf2(
        Encoding.UTF8.GetBytes(password),
        stored.Salt,
        stored.Iterations,
        HashAlgorithmName.SHA256,
        stored.Hash.Length);

    return CryptographicOperations.FixedTimeEquals(candidate, stored.Hash);
}
```

Un'identità verificata, e una sola

Quando `Verify` restituisce true, il servizio di login costruisce un `SessionContext`: l'id del dipendente, il nome visualizzato, l'id del reparto, se questa persona gestisce qualcuno, i ruoli risolti dal database in quel momento, l'ora di accesso e l'indirizzo del client. Viene memorizzato una sola volta in `Application.Session`, che è per singola sessione del browser, quindi nulla passa tra il responsabile operativo e l'addetto alle paghe. Un campo statico sarebbe condiviso da tutte le sessioni del processo.

Da quel momento è l'unica fonte di verità. Le schermate non tengono una copia dell'id utente e i servizi non ricevono come parametro l'id del chiamante, perché un parametro è qualcosa che sceglie il codice chiamante — `DocumentService` legge il contesto da sé. Quando la griglia dei documenti mostra un reparto, quel reparto arriva dal contesto e non da una combo box che si sarebbe potuto chiedere al browser di cambiare. È l'abitudine del corso tradotta in codice: una regola che vive solo in ciò che la UI ha proposto non è una regola.

Il contesto viene creato dopo la verifica e mai prima. Non esiste un contesto anonimo con un id vuoto in attesa di essere riempito, perché un oggetto che esiste in entrambi gli stati prima o poi verrà considerato affidabile in quello sbagliato. Ogni schermata di AccessOps tranne il login richiede il contesto e rimanda indietro il browser quando manca, e un accessor tipizzato che lancia un'eccezione quando la sessione non contiene nulla trasforma tutto questo in un'abitudine invece che in una nota di revisione.

Timeout, logout per inattività e un Logout che fa sul serio

Una sessione Wisej.NET è stato sul server, e la sua durata è configurazione, non codice. Il timeout di sessione si imposta nel `Default.json` dell'applicazione e, quando scade, il server elimina la sessione insieme alle finestre, ai controlli e al `SessionContext` che contiene. Scegli un valore che l'azienda possa difendere — venti minuti vanno bene per un portale che mostra fasce retributive e approvazioni firmate — e tienilo più breve di qualsiasi timeout dei proxy davanti all'app, così il comportamento è quello che hai configurato.

Il logout per inattività è la metà visibile della stessa regola. Un `Wisej.Web.Timer` riavviato a ogni attività dell'utente avvisa un minuto prima della scadenza, poi esegue la stessa pulizia del pulsante Logout: ripulisce il contesto, scarta tutto ciò che era in cache sulla persona, scrive il record di audit e chiama `Application.Exit()`. Entrambi i percorsi chiamano un unico metodo, perché due routine di pulizia divergono col tempo, e quella che nessuno clicca è quella che si dimentica qualcosa.

- La sessione termina sul server — chiudere la scheda del browser non è un logout.
- Ogni stato di schermata che nominava una persona sparisce prima che la sessione venga eliminata.
- Il browser arriva alla pagina di login, e tornare indietro nella cronologia non mostra alcun dato.

Brute force, credential stuffing e fixation

Una pagina di login senza throttling è una macchina per tirare a indovinare senza limiti. `LoginThrottle` tiene contatori per account e per indirizzo del client; dopo cinque fallimenti in una breve finestra l'account viene rifiutato per un periodo di blocco, e i contatori si azzerano in caso di successo. Dai a ogni risposta di fallimento la stessa forma oltre che le stesse parole: quando il nome utente non esiste, esegui comunque la verifica su una credenziale fittizia fissa, così un account inesistente e una password errata costano lo stesso tempo.

Il messaggio a schermo è un'unica frase generica per ogni fallimento — utente sconosciuto, password errata, account bloccato, disabilitato. Il motivo reale va nella riga di audit, dove un amministratore può vedere trenta tentativi con utente sconosciuto da un solo indirizzo in dieci minuti. Quello schema è credential stuffing: coppie raccolte dalla violazione di qualcun altro e provate in massa, fermate dal contatore per indirizzo più che da quello per account, dato che ogni account può essere provato una sola volta.

La session fixation è la più sottile. L'identificatore che ha portato la pagina di login anonima non deve portare l'identità verificata, quindi dopo la verifica la sessione precedente al login termina e il browser prosegue su una nuova, senza nulla copiato. Il "ricordami" segue la stessa disciplina: mai una password memorizzata e mai un valore indovinabile, ma un token casuale lungo per dispositivo, salvato come hash, legato a un solo dipendente, con una scadenza, revocabile dalla console di amministrazione e ricontrollato sull'account attivo prima di ricostruire il `SessionContext`.



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:

- ✓ Implementare una pagina di login che verifica le credenziali rispetto a un hash con salt e non memorizza mai la password.

- ✓ Creare un `SessionContext` che tiene un'identità verificata in `Application.Session` ed è l'unica fonte di verità su chi è l'utente.
- ✓ Configurare timeout di sessione, logout per inattività e logout esplicito così che lo stato venga eliminato e non possa essere riutilizzato.
- ✓ Proteggere il flusso di login da brute force, credential stuffing e session fixation.

Guida al lab / esame

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



Lab pratico

Implementa l'autenticazione di AccessOps: una pagina di login con blocco dell'account dopo cinque tentativi falliti, la verifica della password con un hash moderno con salt, un SessionContext creato solo dopo la verifica e memorizzato nella sessione, un timeout di inattività che riporta l'utente al login e ne ripulisce lo stato, un Logout esplicito che chiude la sessione e un record di audit per ogni tentativo di accesso.

Punto di partenza

Apri la soluzione AccessOps creata nel Modulo 1, che è ancora uno shell: una finestra di avvio, una pagina di login segnaposto il cui pulsante apre semplicemente la console principale, e il documento del modello delle minacce che ha indicato l'autenticazione come primo controllo da costruire. La tabella dei dipendenti esiste con nomi e reparti ma senza credenziali, e nulla nella soluzione sa ancora chi ha effettuato l'accesso.

Approccio suggerito

- ✓ Scrivi per primo `PasswordService`, con un metodo Hash che genera un salt casuale nuovo e un metodo Verify che ripete la derivazione con il salt e il numero di iterazioni memorizzati e confronta con un controllo di uguaglianza a tempo costante; aggiungi alla tabella dei dipendenti le colonne salt, numero di iterazioni e hash e crea due account di test tramite il servizio.
- ✓ Definisci `SessionContext` con id del dipendente, nome visualizzato, id del reparto, flag di manager, ruoli e ora di accesso, più un accessor tipizzato su `Application.Session` che restituisce il contesto corrente e lancia un'eccezione quando nessuno ha effettuato l'accesso.
- ✓ Costruisci `LoginService.SignIn` come unico percorso che verifica una credenziale: chiedi a `LoginThrottle` se l'account è bloccato, verifica la password, scrivi il record di audit per ogni esito e, in caso di successo, azzera i contatori, chiudi la sessione precedente al login e crea il SessionContext per quella nuova.
- ✓ Ricostruisci la pagina di login in modo che il gestore del pulsante si limiti a chiamare `LoginService` e a reagire al risultato — un'unica frase generica per ogni fallimento, la console in caso di successo — e fai sì che ogni altra schermata richieda il contesto e reindirizzi al login quando manca.
- ✓ Imposta il timeout di sessione in `Default.json`, aggiungi il timer di inattività con il suo avviso un minuto prima, e scrivi un'unica routine di Logout che ripulisce il contesto, registra la disconnessione e chiama `Application.Exit()`, richiamata sia dal pulsante sia dal percorso di inattività.

Deliverable richiesti

- ✓ Pagina di login con blocco dell'account e messaggi di errore generici
- ✓ Servizio di hashing e verifica delle password

- ✓ SessionContext con identità verificata
- ✓ Timeout di inattività e logout esplicito
- ✓ Record di audit dei login

Domande di revisione per lo studente

- ✓ La tua pagina di login risponde a una password errata in 480 ms e a un nome utente sconosciuto in 6 ms. Spiega che cosa impara un attaccante da questa sola differenza, e descrivi le due modifiche che eliminano il segnale senza cambiare il messaggio.
- ✓ Un collega propone di tenere il dipendente autenticato in una proprietà statica perché "è più semplice che andare a pescare nella sessione ogni volta". Descrivi esattamente che cosa vede il secondo utente contemporaneo, e in quale momento le due sessioni interferiscono per la prima volta.
- ✓ Il tuo pulsante Logout ripulisce il SessionContext ma non chiude la sessione. Indica due cose che restano recuperabili dopo, e di' quale consideri più grave in un portale che mostra documenti riservati.



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.