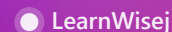


Proteggere dati, upload, download e handler

Guida della lezione · Guida al lab / esame



Guida della lezione

Autorizzare upload e download di file, handler HTTP personalizzati sicuri, validazione dell'input, codifica dell'output, rischi di injection e cross-site in una UI lato server. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

Autorizzare upload e download di file, handler HTTP personalizzati sicuri, validazione dell'input, codifica dell'output, rischi di injection e cross-site in una UI lato server.

1. Il problema in produzione

L'area dei documenti riservati è la parte di AccessOps che un attaccante vuole davvero: lettere sugli stipendi, fascicoli disciplinari, contratti firmati, documentazione del consiglio di amministrazione. Ogni altra schermata sposta righe tra griglie; questa sposta byte attraverso il confine verso un browser che non controlli, e ritorno. Identità e permessi sono stati sistemati nei moduli precedenti, e questo modulo riguarda i punti in cui un file, un URL o una stringa riescono ad aggirarli.

2. Concetti chiave

Concetto	Perché è importante
download inviato dal server	Chiamare <code>Application.Download</code> dal codice server invia un file al browser durante un evento che hai già autorizzato, quindi non ha un indirizzo da salvare, condividere o indovinare.
esposizione dei file statici	Tutto ciò che sta sotto la web root viene servito prima che giri una sola riga del tuo codice, quindi i permessi del file sono esattamente "chiunque conosca l'URL".
evento Uploaded	Il controllo <code>Upload</code> lo solleva sul server con gli stream dei file in arrivo — l'unico punto in cui un file può essere ispezionato prima di entrare nel tuo sistema.
verifica della firma del contenuto	Confrontare i primi byte di un file con l'intestazione nota del formato, perché l'estensione e il content type dichiarato sono forniti dal client.
archiviazione in quarantena	Tenere l'upload accettato fuori dalla web root con un nome generato finché la scansione non dà esito pulito, e solo allora collegarlo al reparto proprietario.
handler HTTP personalizzato	Una classe server registrata nell'applicazione che riceve richieste grezze fuori dal flusso degli eventi, quindi non ha un'identità di sessione e deve autenticare e validare tutto da sé.
link firmato con scadenza	Un URL che porta un id di documento, una scadenza e un HMAC calcolato su entrambi con una chiave tenuta sul server, così i valori non possono essere modificati e il link smette di funzionare da solo.

3. Il filo della lezione

Un download senza indirizzo

La cosa più sicura di un download Wisej.NET è che non c'è nulla da linkare. Il codice server apre uno stream, chiama `Application.Download` con quello stream e il nome con cui il browser deve salvare il file, e i byte scendono lungo la connessione esistente come allegato. Il trasferimento avviene dentro una sessione già autenticata, in un gestore di evento che ha già eseguito le tue verifiche: nessun percorso nel markup, nessun URL nella cronologia, niente da inoltrare.

È una proprietà facile da buttare via. Mettere `/files/contract-4471.pdf` in un link consegna la funzione al web server, che lo serve a chiunque, e i nomi sequenziali invitano il chiamante successivo a provare 4472. In AccessOps i file vivono fuori dalla cartella pubblicata, con nomi generati dall'applicazione, e l'unico modo per arrivarci è `DocumentService.Download`.

L'ordine dentro quel metodo di servizio conta. Carica il documento per id con una query parametrizzata, trattando l'id come input non attendibile perché arriva da una riga di griglia in un browser. Chiedi alle regole dei permessi del Modulo 3 — appartenenza al reparto e ambito del manager comprese — se questo chiamante può leggere questo record, e lancia un'eccezione se non può. Scrivi la voce di audit. Solo allora apri il file memorizzato e invialo. Il gestore del pulsante si limita a chiamare il servizio, quindi la stessa regola vale dalla griglia, da una schermata di approvazione o da un report pianificato.

```

public void Download(int documentId)
{
    var doc = _repository.FindById(documentId);           // parametrizzata
    if (doc == null || !_permissions.CanRead(_session.User, doc))
        throw new SecurityException("Documento non disponibile.");

    _audit.RecordDownload(_session.User.Id, doc.Id);

    using var file = _storage.OpenRead(doc.StoredName); // fuori dalla web root
    Application.Download(file, doc.DisplayName);
}

```

Un upload è input non attendibile che per caso è un file

Il controllo `Upload` invia al server i file selezionati e solleva `Uploaded`, dove il tuo gestore riceve gli stream dei file. Tutto ciò che l'evento ti dice su di essi arriva dal browser: il nome, l'estensione, il content type dichiarato, la dimensione. Un client ostile può inviare un eseguibile chiamato `payslip.pdf`, o un nome pieno di segmenti verso la directory superiore che esce dalla cartella che avevi previsto.

Il gestore è quindi un cancello con cinque verifiche, in ordine. Imponi una dimensione massima nel controllo e nei limiti delle richieste sul server. Verifica l'estensione rispetto a una allow-list di ciò che serve alla funzione, mai una deny-list di ciò che temi. Leggi i primi byte e conferma che corrispondono alla firma di quel formato, così un eseguibile rinominato viene scartato anche se l'estensione sembrava a posto. Scrivi il file fuori dalla web root con un nome generato dal tuo codice, conservando il nome originale dell'utente come valore da visualizzare. Infine mettilo in quarantena, esegui la scansione antimalware e aggiungilo ai documenti del reparto solo quando la scansione dà esito pulito.

- Dimensione ed estensione scartate prima di leggere lo stream fino in fondo.
- Firma verificata rispetto ai formati della allow-list, non rispetto al tipo dichiarato.
- Nome di archiviazione generato dal server; il nome del client non è mai un percorso.

Ogni scarto è un messaggio su cui l'utente può agire — "documenti PDF e Word fino a 20 MB" — più una voce di audit che indica il chiamante, il file e il motivo. Il silenzio è peggio del rifiuto.

Gli handler stanno fuori dalla sessione

A volte un link deve funzionare senza il portale: un fornitore ha bisogno del contratto firmato, un manager inoltra un documento a un auditor. Questo è un handler HTTP personalizzato, una classe server registrata nell'applicazione che riceve la richiesta grezza. Non fa parte di una sessione Wisej.NET, quindi nulla del chiamante è stato stabilito. Qualunque cosa l'handler risponda, la risponderà all'intera internet.

AccessOps emette un link di condivisione come id di documento, scadenza e firma — un HMAC su id e scadenza con una chiave tenuta solo sul server. L'handler interpreta l'id come intero, ricalcola l'HMAC dai valori della query string, confronta i due a tempo costante, verifica la scadenza, e solo allora carica il record e invia il file. Modificare l'id o spostare in avanti la scadenza cambia la firma, quindi la richiesta viene rifiutata. Link scaduti e contraffatti ricevono la stessa risposta breve, entrambi registrati nel log con l'indirizzo del chiamante, così una scansione alla ricerca di id validi diventa visibile.

Le altre regole per gli handler discendono dallo stesso fatto: valida ogni input invece di fidarti di un chiamante che non puoi vedere, non restituire dettagli nella risposta di errore, tieni breve la durata — un'ora nel lab — e applica il rate limiting all'endpoint così che i documenti non possano essere enumerati con calma.

Injection in una UI lato server

I controlli lato server eliminano buona parte dei rischi web ordinari, perché il tuo C# imposta proprietà su oggetti invece di concatenare markup. Non eliminano l'injection, che segue i dati. Ogni query di AccessOps usa parametri — EF Core parametrizza LINQ automaticamente, l'SQL grezzo riceve parametri espliciti — quindi una casella di ricerca che contiene un apice e un punto e virgola è un termine di ricerca, non un'istruzione. La concatenazione nell'SQL è l'unica scorciatoia che questo corso non ammette mai.

La seconda via è l'output. La maggior parte dei controlli Wisej.NET rende una didascalia come testo, ma una proprietà a cui è permesso contenere markup lo renderizzerà, quindi una label con `AllowHtml` attivo e il nome inserito da un dipendente diventa un host per script. Tieni disattivate queste proprietà per i contenuti degli utenti, oppure codifica il valore prima di assegnarlo; anche i nomi di file che mostri dopo un upload sono arrivati dal browser.

La terza via è l'interop JavaScript. `Application.Eval` e `Application.Call` inviano script al client, e costruire quello script incollandoci una stringa dell'utente è lo stesso errore che costruire SQL a mano. Passa i valori come argomenti a una funzione che esiste già nella pagina, e lascia che il codice lato client li tratti come dati.



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:

- ✓ Autorizzare ogni download tramite un servizio e inviare i file con `Application.Download` invece di esporre percorsi o URL.
- ✓ Validare gli upload per dimensione, tipo e contenuto, archivarli fuori dalla web root e sottoporli a scansione o quarantena.
- ✓ Scrivere handler HTTP personalizzati ed endpoint che autenticano il chiamante e validano ogni input.
- ✓ Prevenire i rischi di injection e di script: query parametrizzate, output codificato, HTML sicuro nelle label e interop JavaScript sicuro.

Guida al lab / esame

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



Lab pratico

Metti in sicurezza la funzione documenti di AccessOps: upload limitati per dimensione e tipo con content sniffing, salvati fuori dalla web root con un nome casuale e collegati al reparto proprietario; download che passano dall'autorizzazione di DocumentService e da Application.Download; un handler personalizzato per un link di condivisione firmato che scade dopo un'ora; e una revisione che verifica che ogni query sia parametrizzata e che ogni stringa fornita dall'utente sia codificata prima di essere mostrata.

Punto di partenza

Apri la soluzione AccessOps così com'era alla fine del Modulo 4: un login funzionante con un identity provider esterno, un SessionContext che porta l'identità verificata, e un PermissionService che risolve ruoli, claim e le regole di ambito per reparto e manager. La schermata Documents elenca già i documenti riservati del reparto corrente, e ha già un controllo Upload e un pulsante Download — che al momento fanno entrambi la cosa ingenua, scrivendo i file in una cartella sotto la web root e restituendo qualunque nome di file si trovi nella riga della griglia.

Approccio suggerito

- ✓ Sposta l'archivio dei documenti fuori dalla cartella pubblicata, in un percorso configurato esterno alla web root, dai a ogni file memorizzato un nome generato dal server e conserva nel database il nome di file originale dell'utente solo come valore da visualizzare.
- ✓ Riscrivi il gestore Uploaded come un cancello: imponi il limite di dimensione, verifica l'estensione rispetto a una allow-list, leggi i primi byte e confrontali con la firma del formato, poi scrivi il file accettato nella cartella di quarantena e collegalo al reparto proprietario solo dopo che la scansione è passata.
- ✓ Fai passare ogni download da DocumentService: carica il record per id con una query parametrizzata, chiama la verifica dei permessi per il chiamante e per quel record, scrivi una voce di audit, poi invia il file con Application.Download e lascia al gestore del pulsante nient'altro che la chiamata al servizio.
- ✓ Aggiungi l'handler dei link di condivisione: genera un URL che porta l'id del documento, una scadenza a un'ora e un HMAC su entrambi, poi fai ricalcolare all'handler la firma con un confronto a tempo costante, rifiutare i link scaduti o manomessi con la stessa risposta breve e registrare nel log ogni rifiuto.
- ✓ Rivedi la funzione cercando injection: verifica che ogni query usi parametri, che nessuna stringa fornita dall'utente arrivi non codificata a una proprietà che accetta HTML o a una chiamata Application.Eval, e concludi con il tentativo di download contraffatto da un account di un altro reparto.

Deliverable richiesti

- ✓ Pipeline di upload validata con cartella di quarantena

- ✓ Download autorizzato tramite il servizio
- ✓ Handler per link di condivisione firmati con scadenza
- ✓ Revisione di query parametrizzate e codifica dell'output
- ✓ Test con una richiesta di download contraffatta

Domande di revisione per lo studente

- ✓ Il tuo gestore di upload verifica l'estensione rispetto a una allow-list e il file passa. Spiega che cosa può ancora inviare un attaccante, e che cosa aggiunge la verifica della firma che la verifica dell'estensione non può dare.
- ✓ Un manager inoltra un link di condivisione a qualcuno fuori dall'azienda prima che scada. Quali parti del tuo design limitano il danno, e che cosa dovresti aggiungere per revocare quel singolo link senza cambiare la chiave per tutti?
- ✓ Il tuo pulsante di download è nascosto agli utenti senza permesso di lettura, e il test con la richiesta contraffatta passa comunque. Spiega con precisione quale riga di codice lo fa passare, e perché il pulsante nascosto non c'entra nulla.



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.