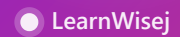


Schutz von Daten, Uploads, Downloads und Handlern

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Autorisierung von Datei-Uploads und -Downloads, sichere eigene HTTP-Handler, Eingabevalidierung, Ausgabe-Encoding, Injection und Cross-Site-Risiken in einer serverseitigen UI. Was das für Entwickler bedeutet, die eine Wisej.NET-App im Produktivbetrieb absichern – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Autorisierung von Datei-Uploads und -Downloads, sichere eigene HTTP-Handler, Eingabevalidierung, Ausgabe-Encoding, Injection und Cross-Site-Risiken in einer serverseitigen UI.

1. Das Problem im Produktivbetrieb

Der Bereich für vertrauliche Dokumente ist der Teil von AccessOps, auf den es ein Angreifer wirklich abgesehen hat: Gehaltsschreiben, Disziplinarakten, unterschriebene Verträge, Unterlagen für die Geschäftsleitung. Jeder andere Bildschirm verschiebt Zeilen zwischen Grids; dieser hier bewegt Bytes über die Grenze zu einem Browser, den Sie nicht kontrollieren, und wieder zurück. Identität und Berechtigungen wurden in den vorigen Modulen geklärt, und in diesem Modul geht es um die Stellen, an denen eine Datei, eine URL oder ein String daran vorbeiläuft.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
vom Server gestreamter Download	Der Aufruf von <code>Application.Download</code> im Servercode streamt eine Datei während eines bereits autorisierten Events an den Browser, sodass es keine Adresse gibt, die man als Lesezeichen speichern, weitergeben oder erraten könnte.
Preisgabe statischer Dateien	Alles unterhalb des Web-Roots wird ausgeliefert, bevor eine einzige Zeile Ihres Codes läuft; die Berechtigung für die Datei lautet also genau „wer die URL kennt“.
Uploaded-Event	Das <code>Upload</code> -Control löst es auf dem Server mit den eingehenden Datei-Streams aus – der einzige Ort, an dem eine Datei geprüft werden kann, bevor sie in Ihr System gelangt.
Prüfung der Inhaltssignatur	Die ersten Bytes einer Datei werden mit dem bekannten Header des Formats verglichen, denn Dateieindung und angegebener Content-Type stammen vom Client.
Quarantäne-Speicher	Der angenommene Upload wird außerhalb des Web-Roots unter einem generierten Namen gehalten, bis der Scan unauffällig ist, und erst dann mit der zuständigen Abteilung verknüpft.
eigener HTTP-Handler	Eine bei der Anwendung registrierte Serverklasse, die rohe Requests außerhalb des Event-Flusses erhält; sie hat keine Session-Identität und muss daher alles selbst authentifizieren und validieren.
signierter, ablaufender Link	Eine URL mit Dokument-ID, Ablaufzeit und einem HMAC, der mit einem serverseitig gehaltenen Schlüssel über beide Werte berechnet wird, sodass sich die Werte nicht bearbeiten lassen und der Link von selbst ungültig wird.

3. Der rote Faden der Lektion

Ein Download ohne Adresse

Das Sicherste an einem Wisej.NET-Download ist, dass es nichts gibt, worauf man verlinken könnte. Der Servercode öffnet einen Stream, ruft `Application.Download` mit diesem Stream und dem Dateinamen auf, unter dem der Browser speichern soll, und die Bytes wandern als Anhang über die bestehende Verbindung. Die Übertragung findet innerhalb einer bereits authentifizierten Session statt, in einem Event-Handler, der Ihre Prüfungen schon ausgeführt hat: kein Pfad im Markup, keine URL im Verlauf, nichts zum Weiterleiten.

Diese Eigenschaft ist schnell verspielt. `/files/contract-4471.pdf` in einen Link zu schreiben, übergibt die Funktion an den Webserver, der die Datei an jeden ausliefert, und fortlaufende Namen laden den nächsten Aufrufer ein, es mit 4472 zu versuchen. In AccessOps liegen die Dateien außerhalb des veröffentlichten Ordners unter Namen, die die Anwendung erzeugt, und der einzige Weg hinein ist `DocumentService.Download` .

Die Reihenfolge innerhalb dieser Service-Methode ist entscheidend. Laden Sie das Dokument per ID über eine parametrisierte Abfrage und behandeln Sie die ID als nicht vertrauenswürdige Eingabe, denn sie stammt aus einer Grid-Zeile in einem Browser. Fragen Sie die Berechtigungsregeln aus Modul 3 –

einschließlich Abteilungszugehörigkeit und Managerzuständigkeit –, ob dieser Aufrufer diesen Datensatz lesen darf, und werfen Sie andernfalls eine Exception. Schreiben Sie den Audit-Eintrag. Erst dann öffnen Sie die gespeicherte Datei und streamen sie. Der Button-Handler ruft nur den Service auf, sodass dieselbe Regel gilt, ob der Aufruf aus dem Grid, einem Genehmigungsbildschirm oder einem geplanten Bericht kommt.

```
public void Download(int documentId)
{
    var doc = _repository.FindById(documentId);           // parametrisiert
    if (doc == null || !_permissions.CanRead(_session.User, doc))
        throw new SecurityException("Document not available.");

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

    using var file = _storage.OpenRead(doc.StoredName);    // außerhalb des Web-Roots
    Application.Download(file, doc.DisplayName);
}
```

Ein Upload ist nicht vertrauenswürdige Eingabe, die zufällig eine Datei ist

Das **Upload** -Control sendet die ausgewählten Dateien an den Server und löst **Uploaded** aus, wo Ihr Handler die Datei-Streams erhält. Alles, was Ihnen das Event über sie mitteilt, kam aus dem Browser: der Name, die Endung, der angegebene Content-Type, die Größe. Ein feindseliger Client kann eine ausführbare Datei namens **payslip.pdf** schicken oder einen Namen voller Parent-Directory-Segmente, der aus dem vorgesehenen Ordner herausführt.

Der Handler ist also ein Tor mit fünf Prüfungen in fester Reihenfolge. Erzwingen Sie eine Maximalgröße im Control und in den Request-Limits des Servers. Prüfen Sie die Endung gegen eine Allow-List dessen, was die Funktion braucht, nie gegen eine Deny-List dessen, was Sie befürchten. Lesen Sie die ersten Bytes und bestätigen Sie, dass sie zur Signatur dieses Formats passen, sodass eine umbenannte ausführbare Datei abgewiesen wird, auch wenn ihre Endung harmlos aussah. Schreiben Sie die Datei außerhalb des Web-Roots unter einem Namen, den Ihr Code erzeugt, und behalten Sie den ursprünglichen Namen des Benutzers nur als Anzeigewert. Stellen Sie sie schließlich unter Quarantäne, führen Sie den Malware-Scan aus und fügen Sie sie erst dann den Dokumenten der Abteilung hinzu, wenn der Scan unauffällig ist.

- Größe und Endung werden abgewiesen, bevor der Stream zu Ende gelesen ist.
- Die Signatur wird gegen die erlaubten Formate geprüft, nicht gegen den angegebenen Typ.
- Der Speichername wird vom Server erzeugt; der Client-Name ist nie ein Pfad.

Jede Ablehnung ist eine Meldung, mit der der Benutzer etwas anfangen kann – „PDF- und Word-Dokumente bis 20 MB“ – plus ein Audit-Eintrag mit Aufrufer, Datei und Grund. Schweigen ist schlimmer als Ablehnung.

Handler stehen außerhalb der Session

Manchmal muss ein Link ohne das Portal funktionieren: Ein Lieferant braucht den unterschriebenen Vertrag, eine Managerin leitet ein einzelnes Dokument an einen Prüfer weiter. Das ist ein eigener HTTP-Handler, eine bei der Anwendung registrierte Serverklasse, die den rohen Request erhält. Er ist nicht Teil

einer Wisej.NET-Session, es ist also nichts über den Aufrufer festgestellt. Was immer der Handler beantwortet, beantwortet er dem gesamten Internet.

AccessOps stellt einen Freigabelink als Dokument-ID, Ablaufzeit und Signatur aus – ein HMAC über ID und Ablaufzeit mit einem Schlüssel, der nur auf dem Server liegt. Der Handler parst die ID als Integer, berechnet den HMAC aus den Werten des Query-Strings neu, vergleicht beide in konstanter Zeit, prüft die Ablaufzeit und lädt erst dann den Datensatz und streamt die Datei. Wer die ID ändert oder die Ablaufzeit hinausschiebt, ändert die Signatur, und der Request wird abgewiesen. Abgelaufene und gefälschte Links erhalten dieselbe kurze Antwort, beide werden mit der Adresse des Aufrufers protokolliert, sodass ein Scan nach gültigen IDs sichtbar wird.

Die übrigen Regeln für Handler folgen aus derselben Tatsache: Validieren Sie jede Eingabe, statt einem Aufrufer zu vertrauen, den Sie nicht sehen, geben Sie in der Fehlerantwort keine Details preis, halten Sie die Lebensdauer kurz – im Lab eine Stunde – und begrenzen Sie die Anfragerate des Endpunkts, damit sich Dokumente nicht in aller Ruhe durchprobieren lassen.

Injection in einer serverseitigen UI

Serverseitige Controls beseitigen einen großen Teil der üblichen Web-Risiken, weil Ihr C#-Code Eigenschaften an Objekten setzt, statt Markup zusammenzufügen. Injection beseitigen sie nicht, denn Injection folgt den Daten. Jede Abfrage in AccessOps verwendet Parameter – EF Core parametrisiert LINQ automatisch, rohes SQL erhält explizite Parameter –, sodass ein Suchfeld mit einem Anführungszeichen und einem Semikolon ein Suchbegriff ist und keine Anweisung. Verkettung in SQL ist die eine Abkürzung, die dieser Kurs nie erlaubt.

Der zweite Weg ist die Ausgabe. Die meisten Wisej.NET-Controls rendern eine Beschriftung als Text, aber eine Eigenschaft, die Markup enthalten darf, rendert es auch – ein Label mit aktiviertem `AllowHtml` und einem von Mitarbeitenden eingegebenen Namen darin wird so zum Skript-Host. Lassen Sie diese Eigenschaften für Benutzerinhalte ausgeschaltet oder encodieren Sie den Wert vor der Zuweisung; auch die Dateinamen, die Sie nach einem Upload zurückgeben, kamen aus dem Browser.

Der dritte Weg ist JavaScript-Interop. `Application.Eval` und `Application.Call` senden Skript an den Client, und dieses Skript durch Einfügen eines Benutzer-Strings zu bauen, ist derselbe Fehler, wie SQL von Hand zusammenzusetzen. Übergeben Sie Werte als Argumente an eine Funktion, die bereits auf der Seite existiert, und lassen Sie den clientseitigen Code sie als Daten behandeln.



Sicherheitsgewohnheit

Dem Browser wird nie vertraut. Jede relevante Entscheidung – wer der Benutzer ist, was er tun darf, welchen Datensatz er sehen darf – wird auf dem Server in einem Service getroffen, und die UI spiegelt sie nur wider. Existiert eine Regel nur als versteckter Button, dann existiert sie nicht.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ jeden Download über einen Service autorisieren und Dateien mit `Application.Download` streamen, statt Pfade oder URLs preiszugeben.

- ✓ Uploads nach Größe, Typ und Inhalt validieren, sie außerhalb des Web-Roots speichern und scannen oder unter Quarantäne stellen.
- ✓ eigene HTTP-Handler und Endpunkte schreiben, die den Aufrufer authentifizieren und jede Eingabe validieren.
- ✓ Injection- und Skript-Risiken verhindern: parametrisierte Abfragen, encodierte Ausgabe, sicheres HTML in Labels und sicheres JavaScript-Interop.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Sichern Sie die Dokumentenfunktion von AccessOps ab: Uploads, begrenzt nach Größe und Typ und mit Inhaltsprüfung, außerhalb des Web-Roots unter einem zufälligen Namen gespeichert und mit der zuständigen Abteilung verknüpft; Downloads, die über die Autorisierung von DocumentService und Application.Download laufen; ein eigener Handler für einen signierten Freigabelink, der nach einer Stunde abläuft; und ein Review, das bestätigt, dass jede Abfrage parametrisiert ist und jeder vom Benutzer gelieferte String vor der Anzeige encodiert wird.

Ausgangspunkt

Sie öffnen die AccessOps-Solution in dem Stand, den sie am Ende von Modul 4 hatte: ein funktionierendes Login mit einem externen Identity Provider, ein SessionContext mit der verifizierten Identität und ein PermissionService, der Rollen, Claims und die Zuständigkeitsregeln für Abteilung und Manager auflöst. Der Dokumente-Bildschirm listet bereits die vertraulichen Dokumente der aktuellen Abteilung auf und hat bereits ein Upload-Control und einen Download-Button – beide tun derzeit das Naive: Sie schreiben Dateien in einen Ordner unterhalb des Web-Roots und streamen zurück, welcher Dateiname auch immer gerade in der Grid-Zeile steht.

Vorgeschlagenes Vorgehen

- ✓ Verlegen Sie den Dokumentenspeicher aus dem veröffentlichten Ordner an einen konfigurierten Pfad außerhalb des Web-Roots, geben Sie jeder gespeicherten Datei einen vom Server erzeugten Namen und behalten Sie den ursprünglichen Dateinamen des Benutzers in der Datenbank nur als Anzeigewert.
- ✓ Schreiben Sie den Uploaded-Handler als Tor neu: Größenlimit erzwingen, die Endung gegen eine Allow-List prüfen, die ersten Bytes lesen und mit der Formatsignatur vergleichen, dann die angenommene Datei in den Quarantäne-Ordner schreiben und erst nach bestandenem Scan mit der zuständigen Abteilung verknüpfen.
- ✓ Stellen Sie jeden Download hinter DocumentService: den Datensatz per ID mit einer parametrisierten Abfrage laden, die Berechtigungsprüfung für den Aufrufer und diesen Datensatz aufrufen, einen Audit-Eintrag schreiben, dann die Datei mit Application.Download streamen und im Button-Handler nichts als den Service-Aufruf belassen.
- ✓ Ergänzen Sie den Handler für Freigabelinks: eine URL mit Dokument-ID, einer Ablaufzeit eine Stunde in der Zukunft und einem HMAC über beide erzeugen; der Handler berechnet die Signatur mit einem Vergleich in konstanter Zeit neu, weist abgelaufene oder manipulierte Links mit derselben kurzen Antwort ab und protokolliert jede Ablehnung.
- ✓ Prüfen Sie die Funktion auf Injection: Bestätigen Sie, dass jede Abfrage Parameter verwendet und kein vom Benutzer gelieferter String unencodiert eine HTML-fähige Eigenschaft oder einen Application.Eval-Aufruf erreicht, und schließen Sie mit dem gefälschten Download-Versuch von einem Konto aus einer anderen Abteilung ab.

Geforderte Ergebnisse

- ✓ Validierte Upload-Pipeline mit Quarantäne-Ordner
- ✓ Autorisierter Download über den Service
- ✓ Handler für signierte, ablaufende Freigabelinks
- ✓ Review von parametrisierten Abfragen und Ausgabe-Encoding
- ✓ Test mit einer gefälschten Download-Anfrage

Wiederholungsfragen

- ✓ Ihr Upload-Handler prüft die Endung gegen eine Allow-List, und die Datei besteht die Prüfung. Erklären Sie, was ein Angreifer trotzdem noch senden kann und was die Signaturprüfung hinzufügt, das die Prüfung der Endung nicht leisten kann.
- ✓ Eine Managerin leitet einen Freigabelink vor dessen Ablauf an jemanden außerhalb des Unternehmens weiter. Welche Teile Ihres Designs begrenzen den Schaden, und was müssten Sie ergänzen, um genau diesen einen Link zu widerrufen, ohne den Schlüssel für alle zu ändern?
- ✓ Ihr Download-Button ist für Benutzer ohne Leseberechtigung ausgeblendet, und der Test mit der gefälschten Anfrage besteht trotzdem. Erklären Sie genau, welche Codezeile dafür gesorgt hat, dass er besteht, und warum der ausgeblendete Button nichts damit zu tun hatte.



Sicherheitsgewohnheit

Dem Browser wird nie vertraut. Jede relevante Entscheidung – wer der Benutzer ist, was er tun darf, welchen Datensatz er sehen darf – wird auf dem Server in einem Service getroffen, und die UI spiegelt sie nur wider. Existiert eine Regel nur als versteckter Button, dann existiert sie nicht.