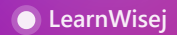


Härtungs-Checkliste und Abschlussprojekt

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

HTTPS und Security-Header, Cookie-Flags, Session-Einstellungen, Aktualisierung von Abhängigkeiten, Rate Limiting, Härtung des Deployments und das Sicherheits-Review von AccessOps. Was das für Entwickler bedeutet, die eine Wisej.NET-App im Produktivbetrieb absichern – und wie Sie dieses Modul lesen.

**Fokus: Produktivbetrieb**

HTTPS und Security-Header, Cookie-Flags, Session-Einstellungen, Aktualisierung von Abhängigkeiten, Rate Limiting, Härtung des Deployments und das Sicherheits-Review von AccessOps.

1. Das Problem im Produktivbetrieb

AccessOps prüft Passwörter inzwischen gegen einen gesalzenen Hash, löst Berechtigungen in einem Service auf, streamt Dokumente nur über `DocumentService` und protokolliert jede Genehmigung im Audit-Trail. Nichts davon hilft, wenn der erste Request im Klartext unterwegs ist, wenn ein Skript das Session-Cookie lesen kann oder wenn der Build ein Paket mit einer veröffentlichten Sicherheitsmeldung ausliefert. Die letzte Meile gehört dem Host: das Zertifikat, die Response-Header, die Cookie-Flags, die Request-Limits und die Version von allem im Abhängigkeitsgraphen.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
HSTS	Ein <code>Strict-Transport-Security</code> -Header, der den Browser anweist, für diesen Host eine festgelegte Zeit lang kein einfaches HTTP zu verwenden, sodass ein späterer Besuch nie beim Redirect abgefangen wird.
Content-Security-Policy	Ein Header, der die Origins auflistet, von denen eine Seite Skripte, Styles und Frames laden darf; er macht aus einem eingeschleusten Skript einen blockierten Request statt Code, der in einer aktiven Session läuft.
frame-ancestors	Die CSP-Direktive (<code>X-Frame-Options</code> ist das ältere Gegenstück), die verhindert, dass eine andere Website AccessOps in einen Frame einbettet und einen Manager dazu bringt, auf Approve zu klicken.
Cookie-Flags	<code>Secure</code> hält das Session-Cookie von einfachem HTTP fern, <code>HttpOnly</code> hält es von JavaScript fern, und <code>SameSite</code> verhindert, dass es bei einem Request mitgeschickt wird, den eine andere Website gestartet hat.
Request-Limits	Obergrenzen für Body- und Header-Größe auf Host-Ebene, die einen übergroßen Upload abweisen, bevor auch nur eine Zeile AccessOps-Code läuft.
Token Bucket	Ein Zähler, der einen Versuch nur zulässt, solange er ein Token enthält, und sich mit fester Rate wieder auffüllt, sodass der Datenverkehr kurz stoßweise ansteigen darf und dann auf Schritttempo gebremst wird.
Audit der Abhängigkeiten	Eine wiederholbare Prüfung jedes direkten und transitiven Pakets sowie der Runtime gegen veröffentlichte Sicherheitsmeldungen – dokumentiert mit Datum und Entscheidung.

3. Der rote Faden der Lektion

HTTPS ist Sache des Hosts, HSTS Sache des Browsers

TLS wird außerhalb Ihres C#-Codes terminiert: von IIS mit einem gebundenen Zertifikat, von Kestrel mit einem konfigurierten Zertifikat oder vom Reverse Proxy vor beiden. Ihr Teil ist es, einfaches HTTP nutzlos zu machen. Leiten Sie jeden HTTP-Request auf Host-Ebene um und senden Sie dann `Strict-Transport-Security` mit einem langen `max-age`, damit der Browser das unsichere Schema gar nicht mehr versucht. Ergänzen Sie `includeSubDomains` erst, wenn jede AccessOps-Subdomain wirklich TLS ausliefert, und betrachten Sie `preload` als eine Entscheidung, die Sie nicht schnell rückgängig machen können. Ein Deployment-Detail erwischt viele: Hinter einem Proxy erreicht der Request die Anwendung unverschlüsselt, und solange die Verarbeitung der Forwarded-Header nicht aktiviert ist, hält sich die App für unsicher und markiert nichts als `Secure`.

Header sind die günstigsten Kontrollen in diesem Kurs, und sie werden einmal für die Pipeline gesetzt statt pro Bildschirm. `X-Content-Type-Options: nosniff` verhindert, dass ein Browser rät, ein hochgeladenes Dokument sei in Wahrheit HTML. `Referrer-Policy: strict-origin-when-cross-origin` verhindert, dass ein Dokumentbezeichner in die Logs einer anderen Website gelangt. `frame-ancestors 'none'` verweigert das Einbetten in Frames komplett. Setzen Sie sie dort, wo jede Response vorbeikommt, einschließlich des

Handlers für signierte Freigabelinks aus Modul 5 – eine separate URL und genau die, die gern vergessen wird.

Die Content-Security-Policy erfordert hier Sorgfalt, weil im Browser der Wisej.NET-Client läuft. Dieser Client, seine Theme-Ressourcen und Ihre Widget-Skripte kommen vom selben Origin, also deckt `default-src 'self'` sie ab; die Layout-Engine schreibt Inline-Styles, daher braucht `style-src` meist `'unsafe-inline'`; und Interop, das zur Laufzeit Client-Code erzeugt, zwingt Sie, `script-src` zu erweitern. Deployen Sie zuerst als `Content-Security-Policy-Report-Only`, klicken Sie sich durch jeden Bildschirm, lesen Sie die Verstöße und setzen Sie die Richtlinie erst dann durch.

```
app.UseForwardedHeaders();           // TLS hat der Proxy terminiert, nicht wir
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();
});
// der Wisej.NET-Handler läuft danach, also trägt jede Response die Header
```

Cookies, Session-Limits und ein Button, der kein Endpunkt ist

Sobald der Login erfolgreich ist, ist das Session-Cookie das Credential, also bekommt es alle drei Flags. `Secure` bedeutet, dass es nie ohne TLS übertragen wird. `HttpOnly` bedeutet, dass selbst eine erfolgreiche Skript-Injection in einem eigenen Widget es nicht lesen kann. `SameSite` auf `Lax` oder `Strict` bedeutet, dass ein Request, den eine andere Website gestartet hat, es nicht mitführt – die günstigste Abwehr gegen eine gefälschte Genehmigung. In ASP.NET Framework sind das Einstellungen an den Cookie- und Session-State-Elementen der `Web.config`; in ASP.NET Core sind es Optionen der Cookie-Policy. Prüfen Sie sie in den Entwicklertools des Browsers, nicht durch erneutes Lesen der Konfiguration.

Die Session-Lebensdauer hat zwei Hälften. Wisej.NET liest seine eigenen Einstellungen aus der JSON-Konfigurationsdatei der Anwendung neben der Startseite, und dort gehört das Session-Timeout des Frameworks hin: die Rückfallebene, die eine verlassene Session und ihren serverseitigen Zustand zurückholt. Es ist nicht der Leerlauf-Logout, den Sie in Modul 2 gebaut haben, denn der Client hält seine Verbindung am Leben, und ein verlassener Browser kann wie Datenverkehr aussehen – Leerlauf muss ab dem letzten echten Benutzer-Event gemessen werden, und der explizite Logout muss den `SessionContext` in `Application.Session` entsorgen, damit der Zurück-Button auf einer Login-Seite landet. Request-Limits sind die dritte Einstellung: Begrenzen Sie Body- und Header-Größe am Host, damit ein übergroßer Upload abgewiesen wird, bevor `DocumentService` überhaupt betreten wird.

Rate Limiting teilt sich entlang einer wichtigen Linie. Der Handler für Freigabelinks ist eine echte URL, also kann Middleware ihn direkt begrenzen – ein nach Client-Adresse partitionierter Token-Bucket-Limiter, der mit 429 antwortet, ohne zu verraten, ob der Link existierte. Der Login-Button ist keine URL: Jedes Wisej.NET-Event, vom Scrollen im Grid bis zum Klick auf Sign in, erreicht innerhalb einer bestehenden

Session denselben Anwendungsendpunkt, sodass Middleware sie nicht unterscheiden kann. Das Rate Limiting für den Login lebt daher im Service, als Bucket, der nach dem übermittelten Benutzernamen und der Client-Adresse geschlüsselt ist und geprüft wird, bevor der Hash verifiziert wird. Eine Ablehnung liefert dieselbe generische Meldung wie ein falsches Passwort, wird mit ihrer Korrelations-ID im Audit-Trail festgehalten und zählt gegen einen gemeinsamen Speicher statt gegen ein statisches Feld, wenn AccessOps auf mehreren Servern läuft.

Gepatcht – und das nachweislich

Führen Sie `dotnet list package --vulnerable --include-transitive` aus und lesen Sie die gesamte Ausgabe: Das Paket, das Sie nie ausgewählt haben und das drei Ebenen tiefer hereingezogen wurde, ist der übliche Befund. Ergänzen Sie `--outdated` und `--deprecated` für das, was lediglich veraltet oder aufgegeben ist. Aktuelle .NET-SDKs prüfen Sicherheitsmeldungen außerdem schon beim Restore, wodurch das Audit zur Build-Ausgabe wird statt zu einer Aufgabe, an die sich jemand erinnern muss. Nehmen Sie es in die Pipeline auf, damit eine neue Sicherheitsmeldung einen Build scheitern lässt, statt erst in einem Penetrationstest aufzutauchen.

Pakete sind nur ein Teil des Graphen. Die .NET-Runtime, der Webhost und das Container-Basis-Image brauchen jeweils einen Patch-Rhythmus und eine dokumentierte Version. Wisej.NET selbst ist eine versionierte Abhängigkeit: Pinnen Sie sie, lesen Sie vor einem Wechsel die Release Notes und testen Sie Theme und eigene Widgets nach einem Major-Upgrade erneut. JavaScript von Drittanbietern im Web-Root ist für den Paketgraphen unsichtbar, listen Sie also jede Datei von Hand mit Version und Herkunft auf. Und dann halten Sie das Audit schriftlich fest – Datum, Werkzeug, Befunde, eine Entscheidung pro Befund (aktualisieren, abmildern oder mit Ablaufdatum akzeptieren) und den Termin für das nächste Review.

Das Review vor dem Release, Modul für Modul

Das Ergebnis des Abschlussprojekts ist ein Review, keine Funktion. Nehmen Sie die Platzierungsübersicht der Kontrollen aus Modul 1 und gehen Sie den Kurs vorwärts durch; prüfen Sie dabei die App und nicht die Absicht, und testen Sie als Angreifer statt als Benutzer: Fordern Sie als normaler Mitarbeiter das Dokument einer anderen Abteilung an, spielen Sie einen Freigabelink nach Ablauf seiner Stunde erneut ab, rufen Sie eine Genehmigungsmethode als jemand auf, der nicht der Manager ist, und senden Sie zwanzig falsche Passwörter in zehn Sekunden.

- Modul 2 – Sperre, Hashing, verifizierte Identität, Leerlauf-Timeout, Logout, der den Zustand entsorgt.
- Modul 3 – jede Service-Methode geschützt, Abteilungs- und Managerregeln unterhalb der UI, die gefälschte Anfrage abgewiesen.
- Modul 4 – PKCE, Validierung von ID-Token und Nonce, Claim-Mapping, Abmeldung, die beide Sessions beendet.
- Modul 5 – Upload-Validierung, Downloads über den Service, signierte ablaufende Links, parametrisierte Abfragen, encodierte Ausgabe.
- Modul 6 – Audit-Zeilen für sensible Befehle, Korrelations-IDs in den Logs, weder Secrets noch personenbezogene Daten in einem von beiden.
- Modul 7 – HTTPS und HSTS, der Header-Satz, Cookie-Flags, Session- und Request-Limits, Rate Limiting, ein datiertes Audit.

Jeder Befund erhält einen Schweregrad, einen Verantwortlichen und entweder eine Korrektur oder ein akzeptiertes Risiko mit Ablaufdatum; eine Checkliste mit unerklärten Lücken ist nicht vollständig. Die Freigabe bedeutet genau eines: Jede in Modul 1 gezogene Grenze hat jetzt eine Kontrolle, die ein Service durchsetzt, nicht ein Bildschirm.



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:

- ✓ HTTPS, HSTS, sichere Cookie-Flags und Security-Header für ein Wisej.NET-Deployment konfigurieren.
- ✓ Session-Einstellungen, Request-Limits und Rate Limiting so abstimmen, dass sie Missbrauch standhalten.
- ✓ Abhängigkeiten und Runtime mit einem wiederholbaren Prozess auf aktuellem Patch-Stand halten.
- ✓ ein Sicherheits-Review vor dem Release anhand einer Checkliste durchführen und das AccessOps-Abschlussprojekt abliefern.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Schließen Sie das AccessOps-Abschlussprojekt ab: HTTPS mit HSTS erzwingen und Security-Header setzen, Cookies als Secure und HttpOnly mit einer strikten SameSite-Richtlinie markieren, Session-Timeout und Request-Größenlimits festlegen, Rate Limiting für Login und Freigabelinks ergänzen, ein Audit der Abhängigkeiten durchführen und die Ergebnisse festhalten, dann die vollständige Checkliste dieses Kurses gegen die App durchgehen und jeden Befund beheben.

Ausgangspunkt

Öffnen Sie die AccessOps-Solution in dem Stand nach Modul 6: eine Login-Seite mit Sperre, ein `SessionContext` mit der verifizierten Identität, ein `PermissionService`, der `DocumentService` und `ApprovalService` schützt, externe Anmeldung über OpenID Connect, eine Upload- und Download-Pipeline mit einem Handler für signierte Freigabelinks und ein `AuditService`, der mit Korrelations-IDs versehene Zeilen schreibt. An der Host-Konfiguration wurde noch nichts angefasst: Die App antwortet in der Entwicklung über einfaches HTTP, ihre Cookies tragen keine Flags, und kein Paket wurde gegen eine Sicherheitsmeldung geprüft.

Vorgeschlagenes Vorgehen

- ✓ Binden Sie ein Zertifikat, leiten Sie HTTP auf Host-Ebene auf HTTPS um, ergänzen Sie `Strict-Transport-Security` und prüfen Sie die Verarbeitung der Forwarded-Header, falls ein Reverse Proxy vor AccessOps sitzt.
- ✓ Schaffen Sie eine Stelle in der Pipeline, die `X-Content-Type-Options`, `Referrer-Policy`, `frame-ancestors` und eine Content-Security-Policy setzt – zunächst im Report-only-Modus, bis jeder AccessOps-Bildschirm und der Handler für Freigabelinks ohne Verstöße laden.
- ✓ Markieren Sie das Session-Cookie als `Secure` und `HttpOnly` mit einer strikten `SameSite`-Richtlinie, legen Sie das Wisej.NET-Session-Timeout in der Konfigurationsdatei der Anwendung fest und begrenzen Sie die Größe des Request-Bodys am Host.
- ✓ Setzen Sie einen nach Benutzername und Client-Adresse geschlüsselten Token Bucket in den Login-Service und einen Middleware-Limiter auf die Freigabelink-URL, und halten Sie jede Ablehnung mit ihrer Korrelations-ID im Audit-Trail fest.
- ✓ Führen Sie `dotnet list package --vulnerable --include-transitive` aus, dokumentieren Sie die Befunde mit Entscheidung und Verantwortlichem, gehen Sie dann die Checkliste der sieben Module gegen die laufende App durch und beheben Sie, was sie findet.

Geforderte Ergebnisse

- ✓ Konfiguration von HTTPS, HSTS und Security-Headern
- ✓ Einstellungen zur Härtung von Cookies und Session
- ✓ Rate Limiting für Login und Freigabelinks

- ✓ Bericht zum Audit der Abhängigkeiten
- ✓ Ausgefüllte Checkliste des Sicherheits-Reviews mit Korrekturen

Wiederholungsfragen

- ✓ Middleware kann das Rate Limiting für den Handler der Freigabelinks übernehmen, aber nicht für den Sign-in-Button. Was unterscheidet die beiden Requests in einer Wisej.NET-App, und wo muss das Login-Limit stattdessen leben?
- ✓ Ihre Content-Security-Policy wird durchgesetzt, und das Grid wird ohne Styles dargestellt. Welche Direktive ist die wahrscheinliche Ursache, und wie hätte Ihnen der Report-only-Modus das mitgeteilt, bevor es das Deployment tat?
- ✓ Das Audit der Abhängigkeiten listet ein verwundbares Paket auf, das in AccessOps niemand direkt referenziert. Wie ist es in den Build gelangt, und welche zwei ehrlichen Wege gibt es, den Befund zu schließen?



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.