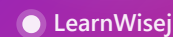


Protéger les données, les uploads, les téléchargements et les handlers

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Autoriser les uploads et les téléchargements de fichiers, écrire des handlers HTTP personnalisés sûrs, valider les entrées, encoder les sorties, et maîtriser les risques d'injection et cross-site dans une UI côté serveur. Ce que cela implique pour un développeur qui sécurise une application Wisej.NET de production — et comment aborder ce module.



Enjeux en production

Autoriser les uploads et les téléchargements de fichiers, écrire des handlers HTTP personnalisés sûrs, valider les entrées, encoder les sorties, et maîtriser les risques d'injection et cross-site dans une UI côté serveur.

1. Le problème en production

L'espace des documents confidentiels est la partie d'AccessOps qu'un attaquant convoite réellement : lettres de salaire, dossiers disciplinaires, contrats signés, dossiers du conseil d'administration. Tous les autres écrans déplacent des lignes entre des grilles ; celui-ci fait franchir la frontière à des octets, vers un navigateur que vous ne contrôlez pas, et dans l'autre sens. L'identité et les permissions ont été réglées dans les modules précédents ; celui-ci traite des endroits où un fichier, une URL ou une chaîne les contourne.

2. Concepts clés

Concept	Pourquoi c'est important
téléchargement diffusé par le serveur	Appeler <code>Application.Download</code> depuis le code serveur envoie un fichier au navigateur pendant un événement que vous avez déjà autorisé : il n'a donc aucune adresse à mettre en favori, à partager ou à deviner.
exposition des fichiers statiques	Tout ce qui se trouve sous la racine web est servi avant qu'une seule ligne de votre code ne s'exécute : les permissions du fichier se résument donc exactement à « quiconque connaît l'URL ».
événement Uploaded	Le contrôle <code>Upload</code> le déclenche sur le serveur avec les flux des fichiers reçus — le seul endroit où un fichier peut être inspecté avant d'entrer dans votre système.
vérification de la signature du contenu	Comparer les premiers octets d'un fichier avec l'en-tête connu du format, car l'extension et le type de contenu déclaré sont fournis par le client.
stockage en quarantaine	Conserver l'upload accepté hors de la racine web, sous un nom généré, jusqu'à ce que l'analyse soit saine, et seulement ensuite le rattacher au département propriétaire.
handler HTTP personnalisé	Une classe serveur enregistrée auprès de l'application qui reçoit des requêtes brutes en dehors du flux d'événements : elle n'a aucune identité de session et doit tout authentifier et tout valider elle-même.
lien signé à expiration	Une URL portant un id de document, une date d'expiration et un HMAC calculé sur les deux avec une clé détenue par le serveur, de sorte que les valeurs ne peuvent pas être modifiées et que le lien cesse de fonctionner de lui-même.

3. Déroulé de la leçon

Un téléchargement sans adresse

Ce qu'un téléchargement Wisej.NET a de plus sûr, c'est qu'il n'y a rien vers quoi pointer un lien. Le code serveur ouvre un flux, appelle `Application.Download` avec ce flux et le nom sous lequel le navigateur doit enregistrer le fichier, et les octets descendent par la connexion existante sous forme de pièce jointe. Le transfert a lieu au sein d'une session déjà authentifiée, dans un gestionnaire d'événements qui a déjà exécuté vos contrôles : aucun chemin dans le balisage, aucune URL dans l'historique, rien à transférer.

Cette propriété est facile à perdre. Placer `/files/contract-4471.pdf` dans un lien confie la fonctionnalité au serveur web, qui la sert à absolument n'importe qui, et des noms séquentiels invitent l'appelant suivant à essayer 4472. Dans AccessOps, les fichiers vivent hors du dossier publié, sous des noms générés par l'application, et la seule voie d'accès est `DocumentService.Download`.

L'ordre à l'intérieur de cette méthode de service compte. Chargez le document par son id via une requête paramétrée, en traitant l'id comme une entrée non fiable, puisqu'il provient d'une ligne de grille dans un navigateur. Demandez aux règles de permission du Module 3 — appartenance au département et périmètre du manager compris — si cet appelant peut lire cet enregistrement, et levez une exception sinon. Écrivez l'entrée d'audit. Seulement ensuite, ouvrez le fichier stocké et envoyez-le. Le gestionnaire du

bouton se contente d'appeler le service : la même règle s'applique donc depuis la grille, un écran d'approbation ou un rapport planifié.

```
public void Download(int documentId)
{
    var doc = _repository.FindById(documentId);           // parameterised
    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); // outside the web root
    Application.Download(file, doc.DisplayName);
}
```

Un upload est une entrée non fiable qui se trouve être un fichier

Le contrôle **Upload** envoie les fichiers sélectionnés au serveur et déclenche **Uploaded**, où votre gestionnaire reçoit les flux des fichiers. Tout ce que l'événement vous en dit vient du navigateur : le nom, l'extension, le type de contenu déclaré, la taille. Un client hostile peut envoyer un exécutable nommé **payslip.pdf**, ou un nom truffé de segments de répertoire parent qui sort du dossier que vous aviez prévu.

Le gestionnaire est donc une barrière à cinq contrôles, dans cet ordre. Imposez une taille maximale dans le contrôle et dans les limites de requête du serveur. Vérifiez l'extension par rapport à une liste d'autorisation de ce dont la fonctionnalité a besoin, jamais une liste de refus de ce que vous craignez. Lisez les premiers octets et confirmez qu'ils correspondent à la signature de ce format, pour qu'un exécutable renommé soit rejeté même si son extension semblait correcte. Écrivez le fichier hors de la racine web sous un nom généré par votre code, en conservant le nom d'origine de l'utilisateur comme valeur d'affichage. Enfin, mettez-le en quarantaine, lancez l'analyse antivirus, et ajoutez-le aux documents du département uniquement lorsque l'analyse est saine.

- Taille et extension rejetées avant que le flux ne soit lu jusqu'au bout.
- Signature vérifiée par rapport aux formats autorisés, pas au type déclaré.
- Nom de stockage généré par le serveur ; le nom fourni par le client n'est jamais un chemin.

Chaque rejet est un message sur lequel l'utilisateur peut agir — « PDF and Word documents up to 20 MB » — accompagné d'une entrée d'audit qui nomme l'appelant, le fichier et la raison. Le silence est pire que le refus.

Les handlers se tiennent hors de la session

Il arrive qu'un lien doive fonctionner sans le portail : un fournisseur a besoin du contrat signé, un manager transmet un document à un auditeur. C'est le rôle d'un handler HTTP personnalisé, une classe serveur enregistrée auprès de l'application qui reçoit la requête brute. Il ne fait partie d'aucune session Wisej.NET : rien n'a donc été établi sur l'appelant. Quoi que le handler réponde, il le répondra à tout Internet.

AccessOps émet un lien de partage sous la forme d'un id de document, d'une date d'expiration et d'une signature — un HMAC sur l'id et l'expiration, calculé avec une clé détenue uniquement par le serveur. Le

handler analyse l'id comme un entier, recalcule le HMAC à partir des valeurs de la query string, compare les deux à temps constant, vérifie l'expiration, et seulement ensuite charge l'enregistrement et envoie le fichier. Modifier l'id ou repousser l'expiration change la signature : la requête est donc refusée. Les liens expirés et les liens falsifiés reçoivent la même réponse courte, tous deux journalisés avec l'adresse de l'appelant, de sorte qu'un balayage à la recherche d'ids valides devient visible.

Les autres règles des handlers découlent du même fait : validez chaque entrée au lieu de faire confiance à un appelant que vous ne voyez pas, ne renvoyez aucun détail dans la réponse d'erreur, gardez une durée de vie courte — une heure dans le lab — et limitez le débit de l'endpoint pour que les documents ne puissent pas être énumérés tranquillement.

L'injection dans une UI côté serveur

Les contrôles côté serveur éliminent une grande partie des risques web ordinaires, car votre C# définit des propriétés sur des objets au lieu de concaténer du balisage. Ils n'éliminent pas l'injection, qui suit les données. Chaque requête d'AccessOps utilise des paramètres — EF Core paramètre automatiquement LINQ, le SQL brut reçoit des paramètres explicites — si bien qu'une zone de recherche contenant une apostrophe et un point-virgule est un terme de recherche, pas une instruction. La concaténation dans du SQL est le seul raccourci que ce cours n'autorise jamais.

La deuxième voie est la sortie. La plupart des contrôles Wisej.NET affichent une légende comme du texte, mais une propriété autorisée à porter du balisage l'interprétera : un label dont `AllowHtml` est activé et qui contient un nom saisi par un employé devient un hôte de scripts. Laissez ces propriétés désactivées pour le contenu utilisateur, ou encodez la valeur avant de l'affecter ; les noms de fichiers que vous réaffichez après un upload viennent eux aussi du navigateur.

La troisième voie est l'interop JavaScript. `Application.Eval` et `Application.Call` envoient du script au client, et construire ce script en y collant une chaîne utilisateur est la même erreur que construire du SQL à la main. Passez les valeurs comme arguments à une fonction qui existe déjà sur la page, et laissez le code côté client les traiter comme des données.



Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Autoriser chaque téléchargement via un service et envoyer les fichiers avec `Application.Download` plutôt que d'exposer des chemins ou des URL.
- ✓ Valider les uploads par taille, type et contenu, les stocker hors de la racine web et les analyser ou les mettre en quarantaine.
- ✓ Écrire des handlers HTTP et des endpoints personnalisés qui authentifient l'appelant et valident chaque entrée.

- ✓ Prévenir les risques d'injection et de script : requêtes paramétrées, sorties encodées, HTML sûr dans les labels et interop JavaScript sûre.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Sécurisez la fonctionnalité de documents d'AccessOps : des uploads limités en taille et en type avec inspection du contenu, stockés hors de la racine web sous un nom aléatoire et rattachés au département propriétaire, des téléchargements qui passent par l'autorisation de DocumentService et par Application.Download, un handler personnalisé pour un lien de partage signé qui expire au bout d'une heure, et une revue vérifiant que chaque requête est paramétrée et que chaque chaîne fournie par l'utilisateur est encodée avant affichage.

Point de départ

Vous ouvrez la solution AccessOps telle qu'elle était à la fin du Module 4 : une connexion fonctionnelle avec un fournisseur d'identité externe, un SessionContext portant l'identité vérifiée, et un PermissionService qui résout les rôles, les claims et les règles de périmètre par département et par manager. L'écran Documents liste déjà les documents confidentiels du département courant, et il possède déjà un contrôle Upload et un bouton Download — qui font tous deux, pour l'instant, la chose naïve : écrire les fichiers dans un dossier sous la racine web et renvoyer le fichier dont la ligne de grille se trouve contenir le nom.

Approche suggérée

- ✓ Sortez le stockage des documents du dossier publié vers un chemin configuré hors de la racine web, donnez à chaque fichier stocké un nom généré par le serveur, et conservez le nom de fichier d'origine de l'utilisateur en base uniquement comme valeur d'affichage.
- ✓ Réécrivez le gestionnaire Uploaded comme une barrière : imposez la limite de taille, vérifiez l'extension par rapport à une liste d'autorisation, lisez les premiers octets et comparez-les avec la signature du format, puis écrivez le fichier accepté dans le dossier de quarantaine et rattachez-le au département propriétaire seulement après une analyse réussie.
- ✓ Placez chaque téléchargement derrière DocumentService : chargez l'enregistrement par id avec une requête paramétrée, appelez le contrôle de permission pour l'appelant et cet enregistrement, écrivez une entrée d'audit, puis envoyez le fichier avec Application.Download, en ne laissant dans le gestionnaire du bouton que l'appel au service.
- ✓ Ajoutez le handler de lien de partage : générez une URL portant l'id du document, une expiration à une heure et un HMAC sur les deux, puis faites recalculer la signature par le handler avec une comparaison à temps constant, rejeter les liens expirés ou falsifiés avec la même réponse courte, et journaliser chaque refus.
- ✓ Passez la fonctionnalité au crible de l'injection : vérifiez que chaque requête utilise des paramètres, qu'aucune chaîne fournie par l'utilisateur n'atteint sans encodage une propriété acceptant le HTML ou un appel à Application.Eval, et terminez par la tentative de téléchargement falsifiée depuis un compte d'un autre département.

Livrables attendus

- ✓ Chaîne d'upload validée avec dossier de quarantaine
- ✓ Téléchargement autorisé via le service
- ✓ Handler de lien de partage signé et à expiration
- ✓ Revue des requêtes paramétrées et de l'encodage des sorties
- ✓ Test avec une requête de téléchargement falsifiée

Questions de révision

- ✓ Votre gestionnaire d'upload vérifie l'extension par rapport à une liste d'autorisation, et le fichier passe. Expliquez ce qu'un attaquant peut encore envoyer, et ce que la vérification de signature apporte que la vérification d'extension ne peut pas apporter.
- ✓ Un manager transmet un lien de partage à une personne extérieure à l'entreprise avant son expiration. Quelles parties de votre conception limitent les dégâts, et que faudrait-il ajouter pour révoquer ce seul lien sans changer la clé pour tout le monde ?
- ✓ Votre bouton de téléchargement est masqué pour les utilisateurs sans permission de lecture, et le test de requête falsifiée passe toujours. Expliquez précisément quelle ligne de code le fait passer, et pourquoi le bouton masqué n'y est pour rien.



Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.