Laboratoire SLH 24 - Questions

Auteur

- Harun Ouweis

Question 1

Quelle est la différence en termes de risque entre les passkeys et l'authentification unique (SSO)?

Réponse

Les passkeys et l'authentification unique (SSO) partagent l'objectif de simplifier l'authentification tout en réduisant les risques de "credentials stuffing". Cependant, ils diffèrent par leurs mécanismes et risques spécifiques.

Explication

- **Passkeys**:
    - Utilisent des clés cryptographiques basées sur le standard FIDO2, sans dépendre de mots de passe.
    - Elles éliminent le risque de phishing et d'attaques par force brute, car les données d'authentification ne sont jamais transmises ni stockées sur un serveur centralisé.
    - La gestion sur l'appareil de l'utilisateur renforce la sécurité mais dépend d'une bonne protection de l'appareil.
- **SSO**:
    - Centralise l'accès à plusieurs services via un fournisseur tiers (par exemple, Google, Facebook).
    - Bien qu'il simplifie l'accès, une faille du compte SSO expose tous les services liés à une attaque.
    - Vulnérable aux attaques de phishing et aux compromissions du fournisseur d'identité.

En résumé, les passkeys offrent un modèle décentralisé et intrinsèquement plus sécurisé, tandis que le SSO reste vulnérable en cas de compromission centralisée.

Question 2

Concernant la validation d'entrées pour les images, quelles sont les étapes effectuées? Et comment stockez-vous les images?

Réponse

Les étapes de validation et le stockage des images visent à garantir la sécurité et l'intégrité des données uploadées.

Explication

- **Étapes de validation**:

  1. **Vérification de l'extension** :

     - L'extension des fichiers uploadés est vérifiée pour s'assurer qu'ils se terminent par `.jpg`.
     - Cela est réalisé par la condition :

     ```
     if !filename.ends_with(".jpg") {
         return Err((StatusCode::BAD_REQUEST, "Only .jpg files are
     allowed").into());
     }
     ```

  2. **Limitation de la taille des fichiers** :

     - La taille maximale des fichiers est fixée à 5 Mo pour éviter les attaques de déni de service.
     - Cela est contrôlé avec :

     ```
     if file_bytes.len() > MAX_UPLOAD_SIZE {
         return Err((StatusCode::BAD_REQUEST, "Uploaded file is too
     large").into());
     }
     ```

  3. **Validation des dimensions des images** :

     - La largeur et la hauteur des images sont limitées à 1920x1080 pixels pour garantir la cohérence.
     - Cela est validé via :

     ```
     if image.width() > MAX_IMG_WIDTH || image.height() >
     MAX_IMG_HEIGHT {
         return Err((StatusCode::BAD_REQUEST, "Image dimensions are too
     large").into());
     }
     ```

  4. **Analyse du contenu de l'image** :

     - La crate `image` est utilisée pour analyser les métadonnées des images.
     - Par exemple, le type de couleur est vérifié :

     ```
     if image.color() != image::ColorType::Rgb8 && image.color() !=
     image::ColorType::Rgba8 {
         return Err((StatusCode::BAD_REQUEST, "Unsupported image
     ```

```
format").into());
}
```

- **Stockage**:
  - **Répertoire des uploads** :
    - Les fichiers sont sauvegardés dans un répertoire défini par `consts::UPLOADS_DIR`. Si le répertoire n'existe pas, il est créé dynamiquement :

```
if !Path::new(uploads_dir).exists() {
    create_dir_all(uploads_dir).map_err(|_| {
        (StatusCode::INTERNAL_SERVER_ERROR, "Failed to create
uploads directory")
    })?;
}
```

  - **Noms uniques pour les fichiers** :
    - Un UUID est utilisé pour générer un nom unique pour chaque fichier :

```
let unique_filename = format!("{}.jpg", Uuid::new_v4());
```

  - **Sauvegarde sur disque** :
    - Les fichiers validés sont écrits sur le disque dans le répertoire des uploads :

```
let mut file = File::create(&file_path).map_err(|_| {
    (StatusCode::INTERNAL_SERVER_ERROR, "Failed to save uploaded
file")
})?;
file.write_all(&file_bytes).map_err(|_| {
    (StatusCode::INTERNAL_SERVER_ERROR, "Failed to write file
content")
})?;
```

Les informations supplémentaires, telles que le chemin du fichier et le texte associé, sont sauvegardées dans un fichier `.yaml` via `serde_yaml`. Cette approche garantit une gestion sécurisée et organisée des posts.

Question 3

Que pensez-vous de la politique de choix des noms de fichiers uploadés? Y voyez-vous une vulnérabilité? Si oui, suggérez une correction.

Réponse

La politique de choix des noms de fichiers uploadés actuelle utilise des noms uniques générés dynamiquement avec un UUID. Cela réduit considérablement les risques de collision et d'exécution de fichiers malveillants, mais certaines vulnérabilités potentielles peuvent persister.

Explication

- **Points positifs** :
  - L'utilisation d'un **UUID** garantit que chaque fichier a un nom unique, évitant ainsi les collisions accidentelles ou intentionnelles.
  - Les noms de fichiers ne sont pas directement dérivés des entrées utilisateur, réduisant le risque de path traversal ou de script injection.
- **Vulnérabilités potentielles** :
  1. **Path Traversal** :
     - Si le chemin d'accès ou le nom de fichier est manipulé avant d'être stocké, cela pourrait exposer des vulnérabilités de type path traversal, permettant à un utilisateur malveillant d'accéder à des fichiers sensibles.
  2. **Absence d'un contrôle renforcé du chemin** :
     - Si des entrées non validées ou mal nettoyées sont utilisées pour construire des chemins de fichiers, cela pourrait compromettre la sécurité.
  3. **Exposition des noms de fichiers dans l'interface utilisateur** :
     - Si les noms générés (même aléatoires) sont exposés directement à l'utilisateur, cela pourrait révéler la structure interne ou permettre la reconnaissance de schémas prévisibles.
- **Corrections suggérées** :
  1. **Utilisation de répertoires isolés** :
     - Stocker les fichiers dans des répertoires spécifiques (par exemple, séparés par utilisateur ou par type de fichier) pour minimiser les risques en cas de fuite de noms.
  2. **Renforcement de la validation des chemins** :
     - Toujours utiliser des chemins absolus et empêcher l'utilisation de séquences dangereuses comme `../` :

```
if filename.contains("..") {
    return Err((StatusCode::BAD_REQUEST, "Invalid file
name").into());
}
```

  3. **Encapsulation des noms dans une logique de sécurité** :
     - Les noms de fichiers ne devraient jamais être exposés directement. Utiliser des alias ou des références dans la base de données pour pointer vers les fichiers stockés.
  4. **Protéger l'accès aux fichiers** :

- Limiter l'accès aux fichiers uploadés par un mécanisme d'authentification ou de permissions.