SLH - Sécurité logicielle haut niveau

Validation des entrées

13 octobre 2025

Table des matières

1 Introduction

Les catégories d'erreurs le plus souvent associés aux entrées utilisateur sont:

- **Buffer overflow**: validation incorrecte de la taille des entrées.
- **XSS**: validation incorrecte de contenu utilisateur (avec codage incorrect de la sortie).
- **File upload of dangerous type**: validation incorrecte du type de fichier.
- **Path traversal**: validation incorrecte des chemins de fichiers.
- **SQL Injection**: validation incorrecte de l'absence de métacaractères.

1.1 Exemples

```yaml
countries:
- GB
- IE
- FR
- DE
- NO
```

Ici `NO` est un code de pays valide, mais si l'application le reconnaît comme `Norway` (Norvège) mais la spec `YAML` l'interprête comme un `No` (Non), cela peut causer des erreurs inattendues.

2 Fondamentaux

2.1 Cohérence de la validation

Un problème souvent rencontré est que la validation des entrées peut se faire à plusieurs endroits, par exemple côté client (JavaScript) et côté serveur (Python, Java, etc.). Si les règles de validation ne sont pas parfaitement alignées entre ces deux couches, des incohérences peuvent survenir, permettant à des entrées invalides de passer à travers.

> **i Info**
>
> Il est important de factoriser la logique de validation pour éviter les duplications et les divergences. Cela peut être fait en utilisant des bibliothèques de validation partagées ou en centralisant la logique de validation dans une couche intermédiaire.

2.1.1 Quand valider?

Le risque principal de la validation tardive est que des données malveillantes peuvent déjà avoir été traitées par le système avant d'être validées, ce qui peut entraîner des vulnérabilités de sécurité. Des risques seraient:

- Contenu dangereux dans des logs ou dans un stockage persistant
- Side channels sur le temps, ou les messages d'erreurs

C'est pourquoi il est recommandé de valider les entrées dès que possible.

2.2 Nettoyer les entrées

Si la validation ne passe pas, nous pourrions être tentés de nettoyer les données d'entrées, mais tenter de nettoyer une entrée est dangereux:

- Nettoyer correctement est difficile, voire impossible suivant les cas
- Les cas à nettoyer seront des chemins de code peu testés

2.2.1 Exemple

```
<ScRiPt>
```

```
<script defer>
```

```
<scr<script>ipt>
```

```
<img src='a' onerror='...'>
```

et plein[1] d'autres!

Fig. 1. – Capture des slides du cours – Exemple de nettoyage dangereux

> **⚠ Warning**
>
> Il est crucial d'avoir des tests permettant de valider que les entrées soient gérées correctement. De plus, il faut absolument tester des cas négatifs, c'est-à-dire des entrées invalides, pour s'assurer que le système réagit de manière appropriée (par exemple, en rejetant l'entrée avec un message d'erreur clair) plutôt que de tenter de les nettoyer ou de les accepter.

2.3 Niveaux de validation

Il existe 3 niveaux de validation des entrées utilisateur, chacun ayant son propre rôle et ses propres techniques:

1. **Validation syntaxique**: vérifier qu'une chaîne de caractères appartient à un langage.
2. **Validation sémantique**: vérifier qu'une information a un sens cohérent dans un contexte particulier.
3. **Validation pragmatique**: vérifier la véracité d'une proposition logique

2.3.1 Exemple adresse mail

- **Syntaxique**: un nom de boîte contenant des caractères valides, suivi par un `@`, suivi d'un nom DNS valide -> privilégier des librairies éprouvées.
- **Sémantique**: le nom de domaine a une structure et un nombre de composants réalistes, le TLD est connu; le domaine existe, l'enregistrement MX existe.
- **Pragmatique**: L'utilisateur est capable de communiquer une valeur secrète envoyée sur cette adresse.

3 Validation Sémantique & Pragmatique

3.1 Validation pragmatique

La validation pragmatique est souvent réalisée par des mécanismes externes, tels que l'envoi d'e-mails de confirmation, la vérification par SMS, ou l'utilisation de services tiers pour valider des informations spécifiques. Ces méthodes permettent de s'assurer que l'utilisateur possède réellement les informations qu'il fournit.

> ⚠ **Warning**
>
> La validation pragmatique démontre l'accès mais pas la propriété. Par exemple, envoyer un e-mail de confirmation à une adresse fournie par l'utilisateur prouve qu'il a accès à cette adresse, mais ne garantit pas qu'il en est le propriétaire légitime.

3.1.1 Générer et stocker des secrets

Les secrets générés doivent respecter certaines contraintes:

- Suffisamment longs pour résister aux attaques par force brute.
- Suffisamment aléatoires pour éviter les prédictions.
- Doit être à usage unique pour éviter les attaques par rejeu.
- Doit être limité dans le temps pour réduire la fenêtre d'attaque.

4 Validation Syntaxique

4.1 Allow vs Deny Lists

Lors de la validation syntaxique, il est crucial de choisir entre l'utilisation de listes d'autorisation (allow lists) et de listes de refus (deny lists). Les listes d'autorisation spécifient explicitement les entrées acceptables, tandis que les listes de refus énumèrent les entrées interdites.

4.2 Mesures générales

Plusieurs mesures sont recommandées pour une validation syntaxique efficace:

- S'appuyer sur les types
- Canonicaliser les valeurs avant de filtrer
- Toujours vérifier les tailles des valeurs et leurs intervalles
- Utiliser des libs:
  - Java: Commons Validator
  - Python: Pydantic, Marshmallow
  - JS: validator.js

4.3 Expressions régulières

Les expressions régulières sont un outil puissant pour la validation syntaxique, mais elles doivent être utilisées avec précaution.

4.3.1 Rust

```rust
use regex::Regex;
let re = Regex::new(r"^\d{2}-\d{2}-\d{4}$")
  .expect("re compile error");
assert!(re.is_match("01-01-2024"));
```

Précompilation

```rust
use std::sync::LazyLock;
static MY_REGEX: LazyLock<Regex> = LazyLock::new(|| {
  Regex::new(r"^\d{2}-\d{2}-\d{4}$")
    .expect("re compile error")
});
```

5 Encodage des Sorties

Lors de la validation des entrées, il est également essentiel de considérer l'encodage des sorties. Même si une entrée est validée correctement, elle doit être encodée de manière appropriée avant d'être utilisée dans différents contextes (HTML, SQL, etc.) pour prévenir les attaques telles que les injections SQL ou les attaques XSS.

5.1 Adressage indirect

- Utiliser des identifiants opaques pour communiquer avec l'extérieur, et les réassocier aux valeurs correspondantes côté serveur
- Le principe du REST dans sa définition originale: les URL comme des identifiants opaques.

5.2 Identifiants uniques

Identifiants uniques globaux, sécurisés, conviviaux: prenez 2 sur les 3 (pourquoi ?)

- 1+2: **clés publiques**
- 1+3: **noms de domaine**
- 2+3: **keyring**