Table des matières

1. Introduction

1.1. Risque sur validation mot de passe

```
boolean isEqual = userhash.length() == hash.length();
for (int i = 0; i != userhash.length(); i++) {
 isEqual &= (hash.indexOf(i) == userhash.indexOf(i));
}
return isEqual;
```

⚠ – **Warning**
- Le code `indexOf` fait une recherche il aurait fallut utiliser `charAt`

1.2. Stratégies

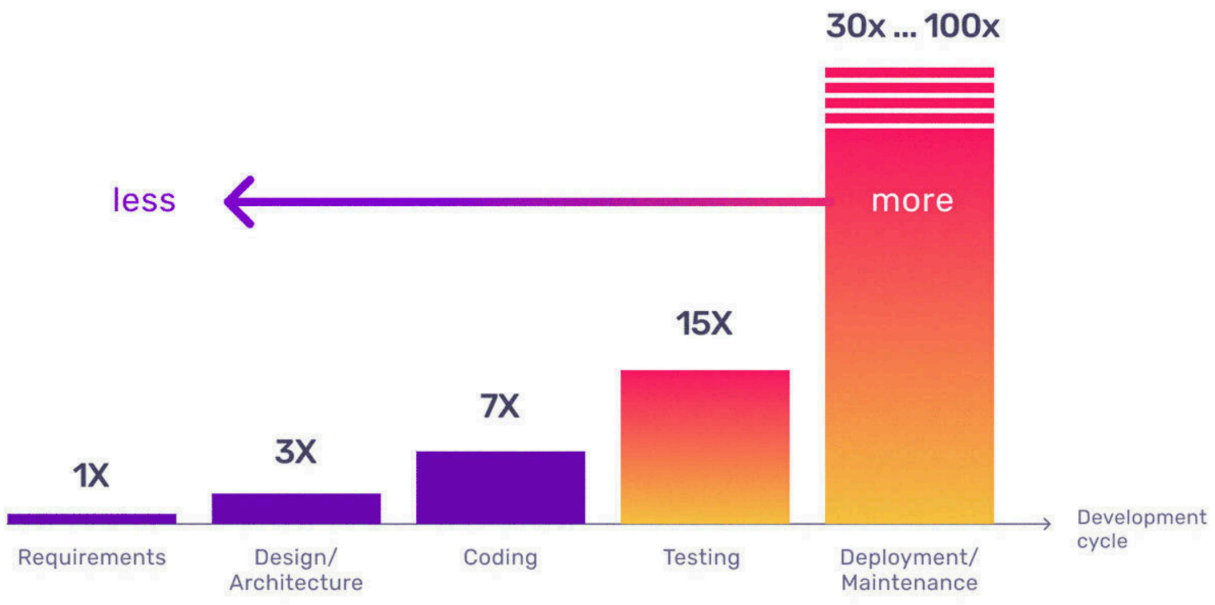
1.2.1. Penetrate & patch

On développe sans se soucier de la sécurité, puis on corrige les failles au fur et à mesure qu'elles sont découvertes. Corriger **réactivement** les problèmes lors-ce qu'ils sont signalés.

1.2.2. Défense périmétrique

Utilisation de couche de sécurité pour protéger les systèmes critiques. Par exemple, un pare-feu pour protéger un réseau interne. Firewall, WAF, SIEM, zerotrust…

1.2.3. Meilleur approche



Plus les erreurs arrivent tôt dans le cycle de développement, plus elles sont couteuses à corriger.

1.2.4. 7 royaumes

1.2.4.1. Validation des entrées

Le programme s'appuie sur des hypothèses non vérifiées sur le contenu de données utilisateurs. Utilisation d'XSS, SSRF, injections SQL, buffer overflow…

Solution: Utiliser le typage statique pour:
- Imposer la validation des entrées
- Garder trace de l'orgine des données

1.2.4.2. Utilisation incorecte des API

Une API requiert des contraintes d'utilisation mais l'utilisateur ne les respecte pas. Par exemple un double free, casts `void*` …

Solution:
- Concevoir des API de manière à rendre l'utilisation incorrecte impossible

1.2.4.3. Mécanisme de sécurité

Un mécanisme de sécurité est utilisé de manière incorrecte. Par exemple, privilèges excessifs, umask erroné, nonce constant, cookie statique, CA non vérifiée, …

Solution:
- Comprendre les mécanismes de sécurité à disposition et modéliser la menace

1.2.4.4. Concurrence

Un ordre d'exécution inhabituel dans un système distribué (multithread, cloud, …) génère un état inattendu et incorrect. Par exemples : deadlocks, TOCTTOU

Solution:
- Utiliser les techniques de prog distribuée adaptées
- Les vérifier avec du typage statique
- Concevoir des API résistantes (demander pardon, pas la permission, …)

1.2.4.5. Traitement des erreurs

Un chemin de code exceptionnel n'est pas suffisamment testé. Par exemple, code d'erreur ignoré, exception masquée, erreurs avec infos sensibles affichées à l'utilisateur…

Solution:
- Imposer le traitement des erreurs correct

1.2.4.6. Qualité du code

Les erreurs passent inaperçues quand le code est difficile à lire. Par exemples : exemple ci-dessus, goto fail , while (a = b) , …

Solution:
- Utiliser des langages modernes et de haut niveau

1.2.4.7. Isolation

Les différents agents dans le système (applications, réseau, utilisateurs) sont insuffisamment cloisonnés, Par exemples : Absence de chiffrement, CSRF, validation coté client, …

Solution :
- Modéliser la menace

1.2.4.8. Environnement

On parle de 7+1 car l'environnement dans lequel le logiciel s'exécute peut aussi être une source de vulnérabilité. Par exemples : OS compromis, dépendances non sécurisées, configuration par défaut, …

1.3. CVSS

Common Vulnerability Scoring System

1.4. PR

- **N** one: aucun accès requis
- **L** ow: accès en lecture
- **H** igh: accès en écriture

1.5. UI

- **N** one: aucune interaction requise
- **P** assive: interaction requise mais pas de l'utilisateur

- **A** ctive: interaction requise de l'utilisateur

1.6. CIA

- **N** one: aucun impact
- **L** ow: impact limité
- **H** igh: impact total

La classification récente décompose le risque en la sévérité de la vulnérabilité (Base), à la menace (patchée ou pas, exploit disponible publiquement), et à l'environnement.