# Cybersécurité et générateur de nombres aléatoires

How not to generate Passwords
A quick history of CVE-2020-27020

Date : 21 avril 2022
Speaker : Mathis Hammel (CodinGame)
Format : conférence (45mn)

Au cours de cette conférence, Mathis va nous apprendre à ne pas reproduire les erreurs faites par de grands acteurs de l'IT.

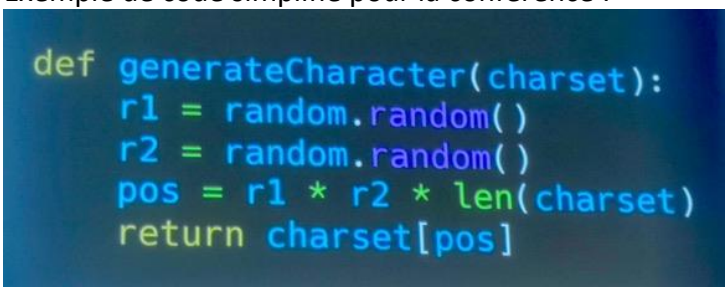
Question : « Pourquoi utiliser un gestionnaire de mot de passe ? »
Réponse : « Tout simplement pour contenir les attaques. »
Bonnes pratiques dans le choix des mots de passe :

- Utiliser des mots de passe sécurisés (longs et avec caractères spéciaux, chiffres et des changements de casse)
- Ne surtout pas utiliser le même mot de passe sur différents comptes car un hacker pourrait compromettre un des comptes et pourrait alors réutiliser le mot de passe piraté sur un autre compte qu'a cet utilisateur sur une autre plateforme.

Sur le darkweb il existe des plateformes pour vendre des mots de passe.
Un dump de 10 millions d'email/mot de passe coute 5 à 10 dollars les premiers jours sur le marché noir.
Il existe des sites permettant de savoir si un mot de passe a été compromis (Exemple : https://haveibeenpwned.com )
Des comptes Netflix / Spotify sont également revendus sur le blackmarket (compter 20 euros par an).

Avec des mots de passe forts et variés, l'attaque est contenue : seul un compte est compromis.
Selon Mathis, sans un **bon gestionnaire de mot de passe**, un mot de passe n'est pas sécurisé car pas facilement mémorisable.
Exemple de gestionnaire de mots de passe : le gestionnaire OSS Keypass codé en C#.
Mais que faire lorsque le gestionnaire de mot de passe contient des failles ?

Dans la suite de son talk, Mathis prend pour cible **Kaspersky Password Manager** (KPM).
Exemple de code simplifié pour la conférence :



Le code de KPM montre un index obtenu en multipliant 2 nombres entiers entre eux.
KPM utilise un charset dynamique : dès qu'un caractère est tiré au sort, il est replacé à la fin de la liste. La distribution de probabilité change donc au cours du tirage. KPM biaise vers des caractères rares jamais choisis par des humains (ex : x, w, y). Cela permet de prémunir un un peu des attaques par chaînes de markov, mais cela diminue l'entropie des mots de passe.

**Chaînes de markov** ou markov-based bruteforce : pour craquer plus vite des mots de passe, on apprend des règles à un logiciel sur ce à quoi ressemble un mot de passe créé par un humain. Exemple : ***th*** suivit de ***e*** (the) en anglais, ***2*** suivi souvent de ***022*** (année 2022)

Un mot de passe de 16 caractères avec une bonne entropie ne se bruteforce pas.

PRNG security : **Pseudo**-Random Number Generators.
Courant dans les langages de programmation. Exemple : java.util.Random en Java.
Certaines implémentations sont plus mauvaises que d'autres, surtout celle de Java.
Java : algorithme Linear Congruential Generators (LCG)
Xo = racine (seed)
Xn+1 = (a.Xn + c) % m

La seed est souvent générée à partir de l'heure courante.
Les constantes a et c sont définies en dur dans le langage.
La variable m permet de ne garder qu'un nombre de bits limité.

Si à un moment on arrive à récupérer 2 valeurs du LCG on peut préduire la suite.
Démo live de Mathis pour casser le Random de Java : nécessité de quand même bruteforcer la seed.
Cette démo n'aurait pas fonctionné avec la classe java.security.SecureRandom

Pour générer des nombres imprédictibles, il faut utiliser un Cryptographically Secure Pseudorandom Number Generator (CSPRNG). Des algos sont généralement dispos dans les packages/modules de crypto ou même nativement dans le langage.
La classe SecuredRandom est un CSPRNG.

Kaspersky utilise le générateur **Mersenne Twister** qui n'est pas cryptographiquement sûr.
Extrait de wikipedia : *« Mersenne Twister, contrairement à l'algorithme Blum Blum Shub, est insuffisant pour une utilisation en cryptographie car des algorithmes tels que Berlekamp-Massey ou Reed-Sloane permettent d'en prédire le comportement. Il reste cependant très utilisé dans tous les domaines hors de la cryptographie en raison de son efficacité. »*

Tous les PRNGs demandent en entrée un peu d'aléatoire (la seed).
2 instances avec la même seed vont générer la même séquence de nombres. Exemple live avec *random.seed(1234)* de Python. Pratique pour debugger mais pas génial en termes de sécu.
La génération de la seed doit être sécurisée.

Kaspersky utilise time(0) comme seed.
A la même seconde donné, 2 utilisateurs utilisant le gestionnaire Kaspersky à la même seconde ont donc le même mot de passe.
Seulement 31 millions de mot de passes différents par an, ce qui rend l'attaque par bruteforce très facile (moins d'1 mn).

Le chercheur français ayant trouvé la faille CVE-2020-27020 a gagné 10 k$. Les utilisateurs ont été invités à changer tous leurs mots de passe. Autant dire que beaucoup vont chercher un autre gestionnaire de mot de passe.

## Conclusion

Essayer d'intégrer la sécurité à toutes les étapes du projet.
Pratiques à appliquer : **Security Shift Left**, **DevSecOps**, **Security By Design**

Questions :

- Personnellement, Mathis utilise Dashlane car il a un copain qui travaille là-bas
- Mathis pense que le mot de passe va continuer à être utilisé et ne fait pas confiance à la biométrie pour certains usages
- Les générateurs hardwares :
    - Random.org : antennes pointées vers l'atmosphère captant du bruit atmosphérique.
- Quid du master password des générateurs de mot de passe et de la base de données sous-jacente ? Préconisation : utiliser un mot de passe particulièrement robuste pour pas être craqué par force brute