Rapport Cryptologie – Générateur de mots de passe

Sarah Neau Monier – sarah.neau-monier@epita.fr
Gael Gnonlonfoun – gael.gnonlonfoun@epita.fr

Novembre 2020

Toute personne souhaitant assurer la sécurité de ses différents comptes se doit d'avoir différents mots de passe pour chacun... Générer un mot de passe *fort* peut être automatisé. C'est donc ici qu'intervient un générateur de mots de passe. Pratique, il permet de générer des mots de passe de façon rapide et sûre.

Table des matières

1 Fonctionnement

Un générateur de mots de passe se compose d'une entrée, d'un slider pour déterminer la longueur du mot de passe ainsi que de différents choix de sous-alphabets.

Afin de garantir l'utilisation de tous les sous-alphabets, un minimum d'un caractère par sous-alphabet retenu est obligatoirement contenu dans le mot de passe généré.

2 Interface

Pour garantir un choix astucieux du mot de passe, un générateur de mot de passe a été créé. Le choix de la longueur du mot de passe, tout comme ses caractères, est laissé au choix de l'utilisateur. Par défaut, sa longueur est fixée à 16 et l'utilisation des nombres et lettres est cochée afin que le mot de passe par défaut soit fort.

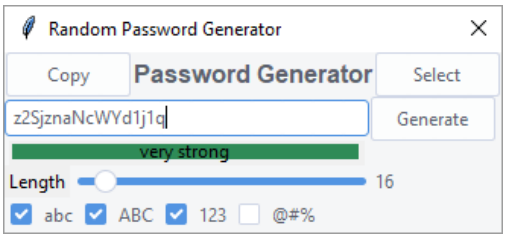

Figure 1: Générateur de mots de passe

3 Indicateur de force d'un mot de passe

Afin de garantir au mieux l'adoption de mots de passe forts, un indicateur dynamique permet de visualiser la force de son mot de passe (*password_strength.py*).

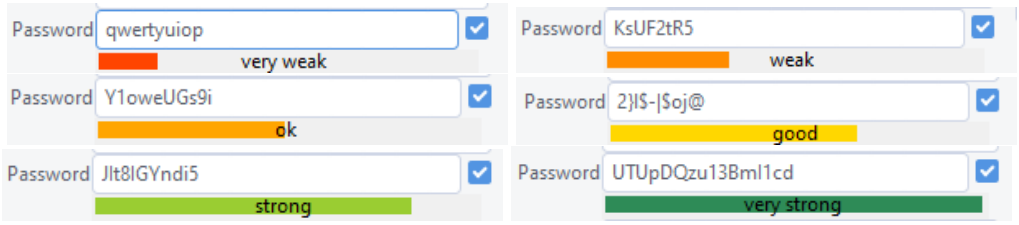

Figure 2: Indicateur de force du mot de passe

On calcule le nombre de combinaisons en fonction des types de caractère présents dans le mot de passe et de sa longueur. Le nombre de combinaisons pour une longueur donnée étant la multicombinaison de k parmi n, avec k la longueur du mot de passe et n celle de l'alphabet, multipliée par le nombre de permutations uniques de chacune de ces multicombinaisons. Nous avons approximé en utilisant le nombre de permutations car, pour calculer les permutations uniques, il nous aurait fallu itérer sur chaque multicombinaison, ce qui reviendrait à faire le brute force et prendrait des années pour les mots de passe forts. Nous ne sommes intéressés que par un ordre de grandeur du résultat (minutes, années, décennies, ...) et cette approximation n'a donc pas de conséquences notables.

```python
if number_combinations < TIME_IN_NS.HOUR:
    return PWD_STRENGTH.VERY_WEAK.value
elif number_combinations < TIME_IN_NS.DAY:
    return PWD_STRENGTH.WEAK.value
elif number_combinations < TIME_IN_NS.YEAR:
    return PWD_STRENGTH.OK.value
elif number_combinations < 10 * TIME_IN_NS.YEAR:
    return PWD_STRENGTH.GOOD.value
elif number_combinations < 500 * TIME_IN_NS.YEAR:
    return PWD_STRENGTH.STRONG.value
else:
    return PWD_STRENGTH.VERY_STRONG.value
```

Figure 3: Définition des forces de mots de passe

L'alphabet est construit comme étant la somme des sous-alphabets (minuscules, majuscules, ponctuations, chiffres) utilisés dans le mot de passe.

Le nombre de combinaisons total est la somme du nombre de combinaisons pour chaque longueur de mot de passe en-dessous de la longueur réelle (selon l'hypothèse qu'un attaquant essaierait les combinaisons moins longues en premier).

En nous basant sur l'hypothèse qu'une combinaison pourrait être testée par un attaquant en une nanoseconde, nous avons fixé des seuils arbitraires de correspondance entre temps requis et force de mot de passe (faible, fort, ...).

Aussi, l'indicateur a été construit après l'étude de la durée théorique de brute-forçage du mot de passe (*time_cracker.py*) sur la base de cette hypothèse.

category	length	duration	time_ref	password	strength
lower	4	20385	nanoseco	xenc	very weak
lower	5	590409	nanoseco	iglki	very weak
lower	6	17691129	nanoseco	owbmpi	very weak
lower	7	5.48E+08	nanoseco	hravkdr	very weak
lower	8	18	second	iybkcehn	very weak
lower	9	10	minute	mfsvmwg	very weak
lower	10	5	hour	yfmoteve	weak
lower	11	8	month	jvwwthvu	ok
lower	12	286	month	lsnxmmxt	ok
lower	13	29	year	vbzcxgenc	strong
lower	14	1098	year	yoyqqpze	strong
lower	15	42798	year	bljfmjenu	strong
lower	16	1710785	year	zafmgnvf	strong
lower	17	70098271	year	ecxkpulkx	strong
lower	18	2.94E+09	year	dpnvrwnc	strong
lower	19	1.26E+11	year	vvamcsgy	strong
lower	20	5.56E+12	year	vbmhuwv	strong
lower	21	2.50E+14	year	sqrbzviad	strong
lower	22	1.15E+16	year	nttfpjwqh	strong
lower	23	5.40E+17	year	xgkfmggg	strong
lower	24	2.59E+19	year	utdgensfr	strong

Figure 4: Force mot de passe - extrait

Ainsi, nous avons pu constater qu'un mot de passe de moins de 8 caractères sera toujours très faible, peu importe le jeu de caractères utilisé.

À partir de ce constat, nous avons généré un fichier contenant la liste des mots de passe dont la longueur est supérieure à 8 caractères et étant considérés comme fragiles car souvent utilisés (*word_set.py*). Pour cela, nous avons téléchargé la liste de ces derniers sur le site suivant : *github*

rockyou.

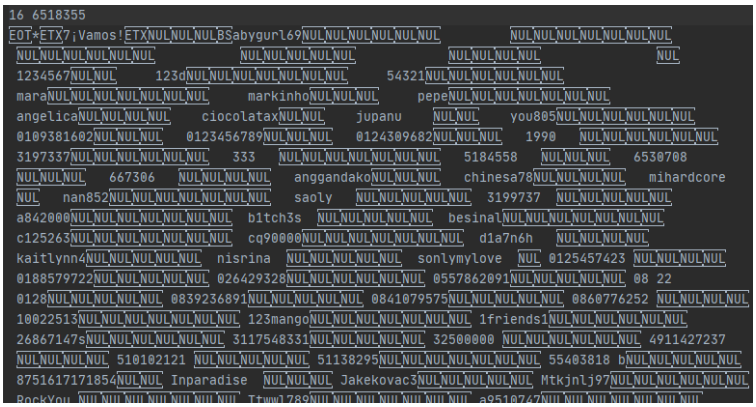

Figure 5: *rockyou.txt*

Pour permettre l'utilisation de grands fichiers en gardant le programme léger et non coûteux en mémoire, nous traitons le fichier pour permettre une recherche dichotomique directement dans le fichier, sans avoir à charger l'ensemble des mots en mémoire.

Le fichier traité actuel ne pèse que 100Mo, mais on pourrait utiliser un fichier beaucoup plus gros sans compromettre ni les performances ni l'utilisation en ressources.

Pour ce faire, nous devons une première fois charger tous les mots en mémoire pour les trier. Puis, nous devons aligner tous les mots sur une même longueur pour pouvoir aveuglément naviguer d'un mot à l'autre dans le fichier. Pour chaque mot, il y a un certain nombre de caractères nuls servant à l'alignement, ce qui résulte en un ratio dégradé d'espace disque pour information stockée. Afin de ne pas trop dégrader ce ratio, nous avons limité la taille maximum d'un mot de passe à 16 (inclus).

Les mots de 8 caractères étant déjà considérés comme très faibles par notre indicateur de temps, nous avons gardé uniquement les mots de passe de 9 caractères et plus dans cette base de mots communs. On remarque qu'elle fait bien son travail puisqu'elle permet de détecter des mots de passe tels que "qwertyuiop" comme très faible au lieu de faible, ou encore "mybaby yana" comme très faible au lieu de correct (ok).

4 Améliorations

4.1 Presse-papier

Il n'y a pas de fonctionnalité permettant de vider le presse-papier automatiquement après usage ou après un certain temps. Ce pourrait être un bon axe d'amélioration car garder un mot de passe en clair dans le presse-papier est très risqué.

4.2 Construction fichier mots de passe faibles

Idéalement, il faudrait construire des blocs séparés pour chaque longueur de mot de passe et chercher dans le bloc correspondant à la taille souhaitée. Cela nous permettrait de ne pas jeter les mots de passe communs de 17 caractères ou plus, et de corriger le problème de ratio de stockage dégradé tout en conservant les mêmes performances.

5 Conclusion

L'utilisation d'un générateur de mots de passe semble indispensable dans l'adoption de mots de passe forts. Par ailleurs, indiquer la force du mot de passe choisi permet l'adoption de bonnes pratiques quant au choix de mots de passe, incitant l'utilisateur a en construire un fort.

En posant l'hypothèse sur le nombre de combinaisons pouvant être testées en un temps donné, nous avons pu établir un indice de force satisfaisant.

6 Nombre de caractères

Ce rapport contient 6013 caractères dont 936 espaces.

Liste des images