Génération de mots de passe par apprentissage automatique

Pablo MOLLIER[1], Mathieu CLEMENT[2]

[1]Pablo MOLLIER, CERI
[2]Mathieu CLEMENT, CERI

Résumé

Deviner un mot de passe est l'une des problématiques les plus importantes en cybersécurité et même si la recherche a permis d'identifier diverses attaques comme la brute force ou l'attaque dictionnaire, il est également nécessaire de se protéger contre de potentielles nouvelles attaques.

Notre article va présenter un générateur de mots de passe reposant sur l'apprentissage automatique en utilisant un réseau neuronal de type LSTM.. Nous étudierons ensuite si ce type d'attaque peut efficacement deviner un mot de passe et nous la comparerons à diverses méthodes déjà existantes.

Mots clefs: password cracking, password generation, Recurrent Neural Network, Long Short-Term Memory recurrent neural networks, cybersecurity, hacking, machine learning

1. Introduction

La sécurisation des mots de passe a toujours été l'une des problématiques les plus importantes dans la sécurité informatique et les diverses méthodes pour cracker ces derniers doivent toujours être prises en compte dans le choix d'un mot de passe.

Il existe de nombreuses façons de trouver un mot de passe comme par exemple l'attaque la plus basique de type brute force ou bien l'attaque dictionnaire qui va utiliser des mots entiers. Récemment, un nouveau type d'attaque a été théorisé en utilisant l'apprentissage automatique.

Nous allons dans cet article essayer de déterminer si l'apprentissage automatique peut être utilisé dans le but de deviner un mot de passe et si oui, à quel point cette méthode est efficace en comparaison aux autres.

Pour cela, nous allons développer un générateur de mots de passes entraîné sur l'un des corpus fournit par le CERI, puis étudier sa précision ainsi que son efficacité.

Cet article sera organisé de la sorte. Nous verrons dans la première section le contexte théorique actuel dans ce domaine. Nous passerons ensuite à la seconde section qui contiendra la méthodologie que nous avons mis en œuvre. Cette partie contiendra une étude des différents corpus de mots de passe fournis afin de choisir celui le plus approprié pour l'apprentissage suivi d'une présentation du réseau neuronal et des choix que nous avons fait pour celui-ci. La troisième section présentera les résultats de notre étude ainsi qu'une évaluation de l'efficacité de la méthode présentée et la dernière section fera office de conclusion.

2. Contexte théorique

2.1. Génération de mots de passes

Il existe deux principales méthodes de génération de mot de passe [1] contre lesquelles un réseau informatique doit être protégé. Nous allons ici lister les principales ainsi que leur efficacité.

2.1.1. Brute Force

Cette méthode de génération est la plus simple et consiste simplement à générer tous les mots de passe possibles jusqu'à trouver le bon. Bien sûr, cette méthode est purement théorique car elle nécessite trop de temps et de puissance de calcul pour être utilisée de façon réaliste. En revanche, plusieurs variations existent reposant sur le même principe. Par exemple, cette méthode peut simplement tester parmi une sélection de mots de passe connus (exemple : azerty, 12345…)

. On peut également la modifier afin qu'elle ne teste qu'avec les caractères les plus utilisés sur une taille prédéfinie, etc.

2.1.2. Attaque dictionnaire

Cette attaque ressemble beaucoup à l'attaque brute force mais elle va utiliser des mots entiers plutôt que des caractères. Cela va lui permettre de trouver bien plus vite les mots de passe correspondant à des suites de mots (ex : motdepasse, salutçava). Cette méthode est à priori la plus utilisée pour trouver des mots de passe. Il est en revanche très facile de s'en protéger par exemple en rajoutant des chiffres ou bien des majuscules. La plupart des attaques dictionnaires reprennent le principe mais peuvent par exemple rajouter des chiffres en fin de mot de passe ou tester avec des majuscules en début ou milieu de mots, etc.

2.2. Génération par apprentissage automatique

Pour l'instant, les attaques reposent principalement sur les deux méthodes énumérées ci-dessus. D'autres méthodes plus complexes existent mais elles reposent également sur des informations extérieures comme les caractères les plus usités, etc. En revanche, de nouvelles techniques de génération commencent à voir le jour comme par exemple PassGAN [2] . Ce générateur est basé sur l'apprentissage automatique et utilise des réseaux de type GAN (generative adversarial network) pour construire des mots de passe.

3. Méthodologie

3.1. Corpus proposés

Les bases de données fournies par le CERI proviennent de fuites de mots passes provenant de sites divers [3]. Ces différents Corpus possèdent des caractéristiques très différentes et nous avons donc pré-sélectionnés 4 corpus de tailles variées. Notre choix va donc ce porter sur l'un de ces 4 corpus :

Id du corpus	Provenance	Nombre de mots de passes
Corpus1	elitehacker	895
Corpus2	Ashley Madison	375 853
Corpus3	hotmail	8930
Corpus4	gmail	3 132 006

3.2. Analyse des Corpus

Afin d'effectuer notre choix, nous avons développé un script python (*analyzer.py*) qui va analyser un corpus passé en paramètre et nous en donner les différentes caractéristiques dans la console. Le script va également générer deux graphiques correspondant à la fréquence des tailles de mots de passe ainsi qu'à la fréquence des caractères dans l'ensemble du corpus. Certains critères vont correspondre à la taille du mot de passe tandis que d'autres vont correspondre au contenu des mots de passe :

	Corpus1	Corpus2	Corpus3	Corpus4
Nombre de mots	895	375853	8930	3132006
Taille moyenne	6,3	7,6	8,78	8,48
Taille médiane	6	8	8	8
1er quartile taille	5	6	7	7
3e quartile taille	7	9	10	10
Ecart absolu taille	11	109	29	15305
Variance	0,02	0,00002	0,00007	0,00007
Ecart-type	0,143	0,0039	0,008	0,008
Mots ne contenant que des chars	714 (80%)	131491 (34%)	4007 (44%)	928076 (30%)
Mots ne contenant que des numéros	104 (11,6%)	46298 (12%)	1654 (19%)	468932 (15%)
Chars différents	60	92	91	69
Nombre de mots de taille < 3	9 (1%)	132 0,03 %	12 0,13 %	777 0,02 %
Nombre de mots de taille > 20	0	10 0,002 %	20 0,22 %	49883 0,1 %
Nombre moyen de caractères alphabétiques	5,52	5,4	6	5,67
Nombre moyen de caractères numériques	0,75	2,19	2,56	2,77
Nombre moyen de caractères non alphanumériques	0,18	1,67	1,68	2,1

3.3. Choix du corpus

Après analyse des corpus, nous allons pouvoir déterminer lequel est le plus adapté pour notre apprentissage.

- Corpus 1 : Ce corpus est varié au niveau des tailles mais assez peu varié au niveau du nombre de caractères différents. Il comporte également trop peu de caractères numériques et non-alpha numériques. De plus, il est très petit (895 mots).

- Corpus 2 : Ce corpus est assez peu varié au niveau des tailles de mots de passes mais il est très varié au niveau du nombre de caractères différents. De plus, il possède très peu de mots de passes anormaux (très petits ou très grands) ce qui peut améliorer l'apprentissage. Il est également d'une taille qui permet un apprentissage assez rapide sans être trop petit (375853 mots)

- Corpus 3 : Ce corpus est moyennement varié au niveau de la taille des mots de passe mais il est également très varié au niveau du nombre de caractères différents. Il comporte en revanche beaucoup de mots de passes anormaux (de taille < 3 out > 20). Il est également un peu petit avec seulement 8930 mots de passes différents ce qui va permettre un apprentissage rapide mais potentiellement moins efficace que sur un corpus plus gros. Ce corpus serait un bon candidat pour tester un apprentissage ou un concept.

- Corpus 4 : Ce corpus est aussi varié que le corpus 3 au niveau de taille des mots de passes. Il est en revanche assez peu varié au niveau de leur contenu (seulement 69 caractères différents) et il comporte beaucoup de mots de passe anormaux. De plus, il est très grand ce qui risque de causer de très longs apprentissages.

Nous avons choisi le Corpus2 pour réaliser notre étude car celui-ci va permettre des apprentissages assez rapides sans être trop petit comme le Corpus3. De plus, il est assez varié au niveau du contenu des mots de passe même si ils sont pour la plupart au alentours de 8 caractères.

3.4. Choix du réseau

Nous avons vu dans la section 2.2 que l'état de l'art au niveau de la génération de mot de passe repose sur un réseau de type GAN (generative adversarial network) [2]. Nous allons dans notre étude tester l'efficacité d'un générateur utilisant un autre type de réseau. Nous avons déterminé qu'un réseau RNN (recurrent neural network) pourrait être utilisé. Nous avons ensuite effectué une comparaison de plusieurs variantes de ces réseaux :

- RNN classique [4]: modèle simple et puissant qui retient les informations passées. En revanche, l'on peut voir apparaître des problèmes de disparition et d'explosion du gradient.

- Minimal RNN [5]: Ce réseau est plus simple et plus rapide qu'un RNN classique, mais comporte toujours les problèmes de gradient.

- RNN Encoder-Decoder [6]: Est supposé être efficace pour capturer les régularités linguistiques, mais est assez complexes

- RNN LSTM [7]: Supprime les problèmes de disparition et d'explosion du gradient en plus d'être efficace et modifiable facilement. Ce type de réseau est l'état de l'art pour une variété de problème comme la reconnaissance d'écriture manuscrite et sa génération, la reconnaissance linguistique et sa traduction, etc.

Pour notre étude, le réseau LSTM semble être un bon candidat, et c'est donc celui que nous allons utiliser.

3.5. Construction du réseau

3.5.1. Division du corpus

Nous avons tout d'abord mélangé le corpus en utilisant le script *shuffler.py,* puis nous l'avons divisé en deux parties : un corpus d'apprentissage et un corpus de test. Dans cette étude, le corpus d'évaluation n'a pas utilisé puisque nous n'allons pas calculer la précision des résultats (qui sera toujours très faible), mais plutôt la couverture des mots de passe que nous générons par rapport au corpus de test. La suite du réseau a été réalisée à l'aide de plusieurs articles. [8][9]

3.5.2. Données d'entrée / sortie

Afin de générer des mots de passe, nous allons utiliser dans notre réseau une fenêtre glissante.

En effet, nous allons passer en entrée de notre réseau une suite de caractères (correspondant à notre fenêtre), ainsi que le caractère qui devrait être prédit. Ensuite, la fenêtre glissante se déplace d'un caractère c'est à dire que le premier caractère de la séquence est supprimé et le caractère qui doit être prédit devient le dernier, tandis que le caractère suivant correspondra à la prédiction.

Au final, nous avons en entrée série de séquences de caractères ainsi que le caractère à prédire

Exemple de données d'entrée pour la série de mots de passe suivante :

- MOT DE PASSE
- AZERTY12345
- 12345678910

Séquence d'entrée	Caractère à prédire
MOT DE PAS	S
OT DE PASS	E
T DE PASSE	\n
DE PASSE\n	A
DE PASSE\nA	Z
E PASSE\nAZ	E
PASSE\nAZE	R
PASSE\nAZER	T
ASSE\nAZERT	Y
…	...
2345678910	\n

On peut remarquer que tous les mots de passes sont traités comme un texte, et qu'il n'y a pas vraiment de notion de mot de passe individuel. En revanche, nous verrons par la suite que cela n'impacte pas les résultats.

Les données de sortie vont correspondre à l'un des caractères du corpus. Par exemple dans le cas du corpus 2, il y 92 sorties possibles puisqu'il y a 92 caractères différents.

Il est également nécessaire de convertir ces données en valeurs numériques pour qu'elles puissent être traitées. Nous convertissons donc chaque lettre en un chiffre dans un dictionnaire pour les utiliser dans le réseau.

3.5.3. Le réseau

Nous avons choisi pour notre étude un réseau de type LSTM (Long Short Term Memory). Ce réseau est un réseau récurrent (RNN) qui va tenir compte des sorties précédentes pour l'apprentissage (d'où la notion de mémoire). Les sorties vont en effet être repassées en entrée lors de l'étape suivante de l'apprentissage.

Notre réseau va ici être composé de deux couches LSTM auxquelles l'on applique un Dropout de 0,2 afin d'éviter le sur-apprentissage.

Nous utilisons également une fonction de perte de type Categorical-Crossentropy car nous sommes face à une problème mutliclasse et nous utilisons également un optimiseur de type Adam.

L'apprentissage est réalisé sur 25 époques avec des batchs de taille 128. Seule l'époque minimisant la perte est sauvegardée.

L'apprentissage est réalisé sur un CPU I7-49720HQ

3.6. Génération des mots de passe

La génération des mots de passe est réalisée à partir d'un script python (*GeneratorLSTN.py*). Pour générer ces mots de passe, nous reprenons le principe de fenêtre glissante [3].

Tout d'abord, le script va charger le réseau. Il va également recharger les données afin de pouvoir convertir les sorties du réseau (qui sont sous forme de valeurs numériques) en caractères.

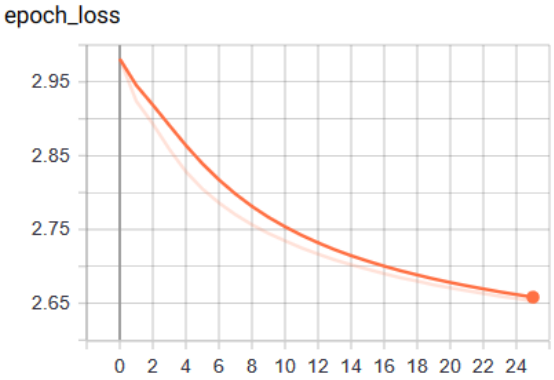
Ensuite, dans une boucle, l'on va tout d'abord définir un « seed » qui va correspondre au point de départ du réseau. Il est conseillé de finir le seed par le caractère « \n » pour que le réseau n'en tienne pas compte. Ensuite, en reprenant le principe de la fenêtre glissante, le réseau va prédire le caractère suivant le seed, puis transformer le seed en enlevant le premier caractère et en rajoutant le caractère prédit à la fin.

Pour prédire un caractère, l'on ne va pas toujours prendre celui que le réseau a prédit. On va sélectionner aléatoirement un caractère parmi la table de probabilité que le modèle a prédit. Cela va permettre d'obtenir des résultats plus variés et réalistes. Chaque caractère prédit est ensuite écrit dans un fichier texte.

4. Résultats

4.1. Fonction de perte

La fonction de perte est de type categorical cross-entropy. A la fin de l'apprentissage, l'on obtient une perte de 2,653. Voici le graphique de l'évolution de la perte en fonction des époques généré avec TensorBoard



On remarque que la perte commence à stagner à partir de la 25e époque. L'on peut donc arrêter l'apprentissage ici.

4.2. Mots de passe générés

Le générateur nous a donc généré un fichier texte contenant plusieurs mots de passe. Pour cette étude, nous avons généré 2 000 000 de mots de passe avec le réseau. Voici un échantillon de ces mots de passe.

10 premiers mots de passe générés avec le réseau LSTN
rapRrus
jafzogm96
pkrhis199
slales2
sialos33
ji7693
mksdmorafo
mowapc
mohergrein
Foydhbr32

Nous avons également, afin d'évaluer nos résultats, généré 2 000 000 de mots de passe aléatoires. Le générateur aléatoire est légèrement optimisé pour générer uniquement des mots de passe de tailles 4 à 11 et en utilisant uniquement des caractères du corpus.

Exemples de mots de passe aléatoires
Ehl7x\;&ql
a6/mFmV1U
wud=sd
f&)UkU/PjxN
qhv4yjIb;

4.3. Evaluation

4.3.1. Couverture

La première métrique que nous avons testée est une simple couverture. En effet, nous testons combien de mots de passe dans le corpus de test sont couverts par nos mots de passe générés (donc 2 000 000). Le corpus de test est de taille 75853.

Après l'évaluation avec un script python (*Evaluator.py*), nous trouvons que 2898 mots de passe ont été trouvés ce qui correspond à 3,82 % du corpus. Nous pouvons comparer ce résultat avec le générateur aléatoire, ainsi qu'avec les autres corpus :

Couverture	Générateur LSTM	Générateur Random
Corpus 2 test	2898 = 3,82 %	7 = 0,009 %
Corpus 1	89 = 9,93 %	2 = 0,2 %
Corpus 3	296 = 3,31 %	1 = 0,01 %
Corpus 4	36390 = 1,1 %	66 = 0,003 %

On peut remarquer que même si ces résultats sont assez faibles, ils restent très satisfaisant par rapport à un générateur aléatoire. On peut donc dire que le réseau peut trouver un mot de passe relativement rapidement.

4.3.2. Couverture avec Distance de Levenshtein

Nous avons également testé la couverture en la combinant avec une distance de Levenshtein.

Si la distance est inférieure à 2, alors nous considérons le mot de passe comme couvert. Le calcul de cette distance étant très long sur 2 000 000 de mots de passe et un échantillon de 75 853 (151 706 000 000 distances à calculer), nous avons réduit l'échantillon de test à 100.

Couverture	Mots de passes avec distance < 2
Corpus 2 test	51 %
Corpus 1	86 %
Corpus 3	79 %
Corpus 4	

5. Conclusion

L'objectif de cette étude était la mise au point d'un générateur de mots de passe en apprentissage automatique utilisant un réseau neuronal de type LSTM.

Au vu des résultats, nous obtenons un générateur assez satisfaisant pouvant générer des mots de passe ressemblant à des mots de passes humains. En revanche, les résultats ne sont pas meilleurs que ceux obtenus avec un réseau de type GAN [2].

L'on peut imaginer plusieurs axes d'améliorations pour notre étude. Par exemple, l'on pourrait augmenter la taille des corpus ainsi que des mots de passe générés. L'on pourrait également agrandir le réseau de neurones qui est assez basique.

L'apprentissage automatique de mot de passe représente un nouveau défi pour la sécurité informatique puisqu'elle est bien plus efficace que la méthode brute force et peut trouver des mots de passe que la méthode dictionnaire ne trouverait pas.

Bibliographie

1: Goldy Benedict, 5 Common Password-Cracking techniques used by Hackers, 2019

2: B.Hitaj, G.Ateniese, P.Gasti, F.Perez-Cruz, PassGAN: A Deep Learning Approach for Password Guessing, 2019

3: Daniel Miessler, Leaked-Databases,

4: Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, On the difficulty of training Recurrent Neural Networks, 2014

5: Chen Minmin, MinimalRNN: Toward More Interpretable and Trainable Recurrent Neural Networks, 2018

6: Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, 2014

7: Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber, LSTM: A Search Space Odyssey, 2017

8: Jason Brownlee, Text Generation With LSTM Recurrent Neural Networks in Python with Keras, 2019

9: TensorFlow, Text generation with an RNN,