Attaque par les tables ”arc-en-ciel”

7 janvier 2023

Mouad Douieb[1]et Sohaib EL Mediouni[2]

***Abstract*— Le Rainbow Attack est une technique de cracking de mots de passe qui consiste à utiliser une table de hachage précalculée appelée ”arc-en-ciel” pour essayer de retrouver le mot de passe d'un utilisateur en comparant le hachage du mot de passe soumis à la table. Cette technique est particulièrement efficace contre les mots de passe qui utilisent des caractères alphabétiques et numériques, mais qui sont relativement courts et simples. Pour se protéger contre les attaques de type Rainbow, il est recommandé d'utiliser des mots de passe longs et complexes, ainsi que de mettre en place des mesures de sécurité supplémentaires, telles que l'authentification à deux facteurs.**

I. INTRODUCTION

Ce rapport traite le sujet du Rainbow Attack qui est une technique de piratage de mots de passe qui a été mise au point dans les années 1990 et qui a depuis connu un regain d'intérêt avec l'explosion de la cybercriminalité. Cette méthode consiste à utiliser une table de hachage précalculée appelée ”arc-en-ciel” pour essayer de retrouver le mot de passe d'un utilisateur en comparant le hachage du mot de passe soumis à la table. Bien que cette technique soit particulièrement efficace contre les mots de passe qui utilisent des caractères alphabétiques et numériques, elle peut être contournée grâce à l'utilisation de mots de passe longs et complexes, ainsi qu'à la mise en place de mesures de sécurité supplémentaires. Dans cet article, nous allons examiner en détail le fonctionnement du Rainbow Attack et expliquer comment se protéger contre ce type d'attaque.

II. COMMENT FONCTIONNE L'ATTAQUE PAR LES TABLE ARC-EN-CIEL ?

L'attaque par Rainbow Table fonctionne en utilisant une base de données de mots de passe pré-calculés appelée Rainbow Table. Lorsqu'une Rainbow Table est utilisée pour cracker un mot de passe, elle effectue une recherche dans sa base de données pour trouver un mot de passe qui a été pré-calculé de manière à produire le même hash que le mot de passe à cracker. Si un mot de passe correspondant est trouvé, il est alors utilisé pour déchiffrer le mot de passe original.

Voici comment cela fonctionne en détail :

— Le hacker utilise un programme pour générer une Rainbow Table en pré-calculant les hashes de plusieurs mots de passe possibles. Cela peut prendre un certain temps, car il doit calculer des millions ou des milliards de mots de passe différents.

— Le hacker utilise alors la Rainbow Table pour essayer de cracker un mot de passe. Il commence par hasher le mot de passe qu'il essaie de cracker en utilisant la même fonction de hachage que celle utilisée pour créer la Rainbow Table.

— Le hacker recherche alors dans la Rainbow Table un mot de passe qui a été pré-calculé de manière à produire le même hash que le mot de passe qu'il essaie de cracker. Si un mot de passe correspondant est trouvé, il est utilisé pour déchiffrer le mot de passe original.

Si aucun mot de passe correspondant n'est trouvé, le hacker recommence à l'étape 2 en utilisant une nouvelle Rainbow Table ou en essayant un autre mot de passe.

Il est important de noter que plus la Rainbow Table est grande et contient de nombreux mots de passe pré-calculés, plus il est facile de cracker des mots de passe. Cependant, plus la Rainbow Table est grande, plus elle prend de temps et de ressources pour être créée et stockée.

A. Les Fonctions Réduction

1) Première expérience: On a généré des nombres premiers aléatoires (sel) pour les combiner avec nos listes de mots et générer des hash pour les comparer avec les hash challenges, pour réduire le temps de vérifications et la complexité.

2) Deuxième expérience: On a généré des nombres entre 1000 et 9 999 et on a répété le même processus, mais malheureusement, on n'a pas obtenu de résultat parce que la probabilité avec tous les nombres et mots, cela sera 6 556 626 890 946 000 cas et sans répétition de nombres, cela sera 10 000.

3) Troisième expérience: Dans la troisième expérience, on a essayé de convertir le code en Numpy pour réduire le temps de vérification au maximum et la surprise était de trouver un hashé qui était : **flakes :9608flakes :e6ef99553f09b88bd32b698a8ac9add2**

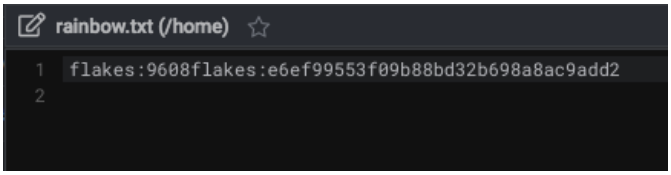

FIG. 1 – Resultat Table arc-en-ciel

[1]Etudiant en CCM Email: douieb.mouad@gmail.com
[2]Etudiant en CCM Email: sohaib.elmediouni23@gmail.com

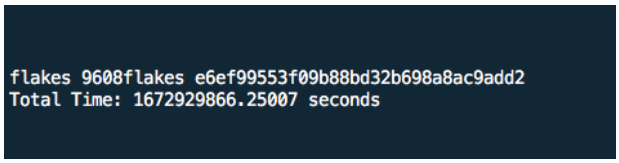

FIG. 2 – Resultat Table arc-en-ciel avec le timing

B. Explication du code

Le code ouvre un fichier de défis de hachage (hash_challenges.txt) et en lit le contenu dans une liste. Il définit également une fonction get_random_num qui génère un nombre aléatoire à quatre chiffres auxquels chaque chiffre est unique.

La fonction salt prend un mot de passe en entrée et y ajoute un ”sel” en insérant quatre caractères ’’ aléatoirement dans le mot de passe, puis en remplaçant ces caractères par quatre chiffres générés aléatoirement. Cela permet de ”saler” le mot de passe en lui ajoutant une complexité supplémentaire qui rendra le hachage du mot de passe plus difficile à craquer.

La fonction hash_md5 calcule le hachage MD5 d’une chaîne de caractères en utilisant le module hashlib. La fonction main ouvre un fichier de mots de passe (words_ccm_2023.txt) et parcourt chaque ligne du fichier. Pour chaque ligne, elle utilise la boucle for pour essayer de trouver un mot de passe ”salé” qui produira un hachage présent dans la liste de défis de hachage. Si elle en trouve un, elle l’écrit dans un fichier rainbow.txt avec le mot de passe original, le mot de passe ”salé” et le hachage associé. Le code mesure également le temps total de calcul et l’affiche à la fin.

C. Bibliothèques et Outils

1) Bibliothèques:

— random : pour générer des nombres aléatoires (utilisés pour ”saler” les mots de passe).
— hashlib : pour calculer les hachages MD5 des mots de passe ”salés”.
— time : pour mesurer le temps de calcul.
— numpy : pour travailler avec des tableaux de données.

2) Outils: AMD EPYC 7282 16-Core Processor, 4 cores

```python
import random
import hashlib


def premier_numbers():
    prime_nums_list = []
    for num in range(1000, 9999 + 1):
        if num > 1:
            for i in range(2, num):
                if (num % i) == 0:
                    break
            else:
                prime_nums_list.append(num)
    return prime_nums_list

premier_numbers_list = premier_numbers()

def salt(password):
    salt = str(random.choice(premier_numbers_list))
    string_buffer = list(password)
    for s in salt:
        pos = random.randint(0, len(string_buffer)-2)
        string_buffer.insert(pos, s)
    return string_buffer


def hash_md5(input_string):
    m = hashlib.md5()
    m.update(input_string.encode('utf-8'))
    return m.hexdigest()

def main():
    fichierMDP = open('words_ccm_2023.txt')
    for line in fichierMDP.readlines():
        for i in range(0, 1000000):
#Do the indentation just for visualisation purpose
with open('hash_challenges.txt', 'r') as f:
                my_list = f.read().splitlines()
#Do the indentation just for visualisation purpose
 salt_ = ''.join(salt(''.join(line.splitlines())))
            if hash_md5(salt_) in my_list:
#Do the indentation just for visualisation purpose
print(line + " " + salt_ + " " + hash_md5(salt_))
                return

main()
```

Code Listing 1 – With random primary numbers

```python
import random
import hashlib


def salt(password):
    salt = str(random.randint(1000, 9999))
    string_buffer = list(password)
    for s in salt:
        pos = random.randint(0, len(string_buffer))
        string_buffer.insert(pos, s)
    return string_buffer

def hash_md5(input_string):
    m = hashlib.md5()
    m.update(input_string.encode('utf-8'))
    return m.hexdigest()

def main():
    fichierMDP = open('words_ccm_2023.txt')
    for line in fichierMDP.readlines():
        for i in range(0, 90000):
            with open('hash_challenges.txt', 'r') as f:
                my_list = f.read().splitlines
                #Do the indentation
salt_ = ''.join(salt(''.join(line.splitlines())))
            if hash_md5(salt_) in my_list:
             #Do the indentation
print(line + " " + salt_ + " " + hash_md5(salt_))
                return

main()
```

Code Listing 2 – With all random numbers

```python
import random
import hashlib
import time
import numpy as np

# Ouvrir le fichier 'hash_challenges.txt' en lecture
#et le stocker dans une variable nomm e 'f',
# puis cr er une liste nomm e 'my_list'
#contenant les lignes lues    partir de 'f',
# et enfin convertir cette liste en un tableau numpy.
with open('hash_challenges.txt', 'r') as f:
    my_list = np.array(f.read().splitlines())

# generer un nombre al atoire de quatre
#chiffres sans r p t er les chiffres
def get_random_num():
    while True:
        num = random.randint(1000, 9999)
        if len(set(str(num))) == 4:
            break
    return num

def salt(password):
    salt = str(get_random_num())
    string_buffer = list(password)
    i = 0
    for s in range(4):
#Do the indentation just for visualisation purpose
pos = np.random.randint(0, len(string_buffer)+1)
        #print(pos)
        string_buffer.insert(pos, '&')

    for s in salt:
#Do the indentation just for visualisation purpose
string_buffer = ''.join(string_buffer)
.replace('&', s, 1)
    return ''.join(string_buffer)

def hash_md5(input_string):
    m = hashlib.md5()
    m.update(input_string.encode('utf-8'))
    return m.hexdigest()

def main():
    fichierMDP = open('words_ccm_2023.txt')
    lines = np.array(fichierMDP.readlines())
    total_time_start = time.time()

    for line in lines:
        time_start = time.time()
        for i in np.arange(0, 10000000):
            salt_ = ''.join(salt(''.join(line.splitlines())))

            if np.isin(hash_md5(salt_), my_list):
                f = open("rainbow.txt", "a")
                new_line = ''.join(line.splitlines()) +
                ":" + salt_ + ":" + hash_md5(salt_) + "\n"
                f.write(new_line)
                print(new_line)
  #Do the indentation just for visualisation purpose
print(f'Total Time: {time.time() - total_time_start}
seconds')
                f.close()
                break
    fichierMDP.close()
#Do the indentation just for visualisation purpose
print(f'Total Time: {time.time() - total_time_start}
seconds')

main()
```

Code Listing 3 – With Numpy where we found a result

D. Pourquoi utiliser un sel

Un sel (salt en anglais) est utilisé dans une table arc en ciel pour rendre la table plus difficile à cracker. Une table arc en ciel est une liste précalculée de valeurs de hachage qui est utilisée pour cracker des mots de passe en comparant les valeurs de hachage dans la table avec celles des mots de passe à cracker.

Un sel est ajouté au mot de passe avant de le hacher, ce qui modifie la valeur de hachage du mot de passe de manière aléatoire. Cela signifie que même si deux utilisateurs utilisent le même mot de passe, leurs valeurs de hachage seront différentes si leurs sels sont différents. Cela rend la création d'une table arc en ciel pour ce mot de passe beaucoup plus difficile, car il faudrait créer une entrée différente pour chaque combinaison possible de sel et de mot de passe.

Utiliser un sel dans une table arc en ciel est donc une mesure de sécurité importante qui empêche les attaquants de cracker facilement les mots de passe en comparant simplement les valeurs de hachage des mots de passe à ceux de la table.

III. Conclusions

L'attaque arc-en-ciel a été une méthode révolutionnaire pour craquer les mots de passe grâce à l'utilisation de tables de hachage précalculées. Cependant, elle n'est plus aussi efficace aujourd'hui en raison de l'adoption de méthodes de hachage plus robustes et de la disponibilité de dictionnaires de mots de passe plus importants. Il est important de choisir des mots de passe forts et uniques pour protéger les comptes contre les attaques de hachage, comme celle-ci.

References

[1] "A super-fast way to loop in python," *online :https://towardsdatascience.com/a-super-fast-way-to-loop-in-python-6e58ba377a00*.

[1]

```python
import random
import hashlib
import time
import numpy as np

with open('hash_challenges.txt', 'r') as f:
    my_list = np.array(f.read().splitlines())

def get_random_num():
    while True:
        num = random.randint(1000, 9999)
        if len(set(str(num))) == 4:
            break
    return num

def salt(password):
    salt = str(get_random_num())
    string_buffer = list(password)
    i = 0
    for s in range(4):
        pos = np.random.randint(0, len(string_buffer)+1)
        string_buffer.insert(pos, '&')

    for s in salt:
        string_buffer = ''.join(string_buffer).replace('&', s, 1)
    return ''.join(string_buffer)

def hash_md5(input_string):
    m = hashlib.md5()
    m.update(input_string.encode('utf-8'))
    return m.hexdigest()

def main():
    fichierMDP = open('words_ccm_2023.txt')
    lines = np.array(fichierMDP.readlines())
    total_time_start = time.time()

    for line in lines:
        time_start = time.time()
        for i in np.arange(0, 10000000):
            salt_ = ''.join(salt(''.join(line.splitlines())))

            if np.isin(hash_md5(salt_), my_list):
                f = open("rainbow.txt", "a")
                new_line = ''.join(line.splitlines()) + ":" + salt_ + ":" + hash_md5(salt_) + "\n"
                f.write(new_line)
                print(new_line)
                print(f'Total Time: {time.time() - total_time_start} seconds')
                f.close()
                break
    fichierMDP.close()
    print(f'Total Time: {time.time() - total_time_start} seconds')

main()
```