**Лабораторная работа 2 «Взлом слабых паролей»**

**Задание:**

Рассмотрим, как злоумышленники пытаются взломать пароли. К несчастью для пользователя, выбор такого плохого пароля означает, что он, скорее всего, будет скомпрометирован, если файл паролей будет украден, поскольку злоумышленники в любом случае будут проверять общие слова (включая слова из других украденных баз данных) против всех хэшей. Но даже менее изощренные методы, скорее всего, тоже смогут вычислить пароль.

В этом разделе мы будем практиковаться во взломе слабых паролей с помощью наименее сложного метода: brute force (грубой силы). Это упражнение призвано продемонстрировать, почему хорошие пароли так важны.

Сценарий: у злоумышленника есть файл с паролями, именами пользователей, солями и хэшами паролей. Что они могут сделать? Ну, они могут просто перебрать все комбинации строчных букв до определенной длины, начиная, например, с "a", "b", "c" и так далее.

Листинг Функция генератора

def generate(alphabet, max\_len):  
 if max\_len <= 0: return  
 for c in alphabet:  
 yield c  
 for c in alphabet:  
 for next in generate(alphabet, max\_len-1):  
 yield c + next  
  
  
gen = generate('ab', 2)  
  
for item in gen:  
 print(item)

Напишите программу, которая выполняет следующие действия десять раз (то есть десять полных циклов с вычислением времени):

1 Случайным образом выберите одну строчную букву (max\_len = 1).

2 С помощью MD5 вычислите хэш этой начальной буквы. Это и есть "тестовый хэш".

3 В цикле пройдитесь по всем возможным строчным однобуквенным входам. 4 Хешируйте каждую букву тем же способом, что и раньше, и сравнивайте с тестовым хешем. - Когда вы найдете совпадение, остановитесь.

5 Вычислите, сколько времени ушло на поиск совпадения. Сколько времени в среднем потребовалось на поиск совпадения для случайной затравки предварительного изображения?

6 Сравните хеши для строчных, ПРОПИСНЫХ букв и цифр алфавита.

7 Используйте все печатаемые символы (string.printable). Сколько всего символов в каждом наборе входных данных? Сколько времени занимает каждый запуск? Проанализируйте использование только цифр, только букв.

8 Выполните предыдущие действия для паролей с 2, 3, 4, 5 и 6 символами.

9\* Напишите программу, которая вводит счетчик времени в SHA-256, получая на выходе хэш и преобразуя его в целое число. Пусть программа повторяет, пока не найдет хэш, который меньше заданного числа. Целевое число должно быть довольно большим, например 2255. Чтобы сделать это более похожим на блокчейн, включите в программу несколько произвольных байтов, которые будут объединены со счетчиком.

Листинг Пример использования таймера

import time  
start = time.process\_time()  
  
gen = generate('ab', 2)  
for item in gen:  
 print(item)  
  
finish = time.process\_time()  
print('Время работы в секундах: ', finish)

**Ответ:**

Код:

import hashlib  
import random  
import string  
import time  
  
def md5\_hash(text):  
 return hashlib.md5(text.encode()).hexdigest()  
  
def generate(alphabet, max\_len):  
 if max\_len <= 0: return  
 for c in alphabet:  
 yield c  
 for c in alphabet:  
 for next in generate(alphabet, max\_len-1):  
 yield c + next  
  
def test\_hash\_match(alphabet):  
 total\_time = 0  
 for \_ in range(10):  
 #случайный выбор буквы из алфавита  
 random\_letter = random.choice(alphabet)  
 #вычисление MD5 хеша выбранной буквы  
 test\_hash = md5\_hash(random\_letter)  
 #перебор всех возможных входов и сравнение хеша  
 start\_time = time.time()  
 for letter in generate(alphabet, 1):  
 if md5\_hash(letter) == test\_hash:  
 end\_time = time.time()  
 break  
 #вычисление времени поиска  
 elapsed\_time = end\_time - start\_time  
 total\_time += elapsed\_time  
 print(f'Cycle {random\_letter}: Time elapsed {elapsed\_time:.6f} seconds')  
 #среднее время поиска  
 average\_time = total\_time / 10  
 print(f'Average time for {alphabet}: {average\_time:.6f} seconds')  
  
#сравнение хешей для различных наборов символов  
print("Lowercase letters:")  
test\_hash\_match(string.ascii\_lowercase)  
  
print("\nUppercase letters:")  
test\_hash\_match(string.ascii\_uppercase)  
  
print("\nDigits:")  
test\_hash\_match(string.digits)  
  
print("\nPrintable characters:")  
test\_hash\_match(string.printable)

Функция md5\_hash(text) принимает текст и возвращает его MD5 хэш.

Функция generate(alphabet, max\_len) - генератор, который производит все возможные строки заданной длины из алфавита.

Функция test\_hash\_match(alphabet) запускает 10 циклов. Каждый цикл:

- выбирает случайную букву из алфавита и вычисляет её MD5 хэш;

- перебирает все возможные односимвольные строки из данного алфавита;

- вычисляет их хэш и сравнивается с эталонным;

- замеряет время, затраченное на поиск совпадения;

- время поиска для каждого цикла выводится на экран.

Функция вычисляет и выводит среднее время для всех 10 циклов. Программа последовательно запускает функцию test\_hash\_match для разных наборов символов: строчных букв, прописных букв, цифр и всех печатаемых символов.

для паролей с 2 символами:

import hashlib  
import random  
import string  
import time  
  
def md5\_hash(text):  
 return hashlib.md5(text.encode()).hexdigest()  
  
def generate(alphabet, max\_len):  
 if max\_len <= 0: return  
 for c in alphabet:  
 yield c  
 for c in alphabet:  
 for next in generate(alphabet, max\_len-1):  
 yield c + next  
  
def test\_hash\_match(alphabet, length):  
 total\_time = 0  
 for \_ in range(10):  
 #случайный выбор строки длиной length из алфавита  
 random\_string = ''.join(random.choice(alphabet) for \_ in range(length))  
 #вычисление MD5 хеша выбранной строки  
 test\_hash = md5\_hash(random\_string)  
 #перебор всех возможных строк заданной длины и сравнение хеша  
 start\_time = time.time()  
 for generated\_string in generate(alphabet, length):  
 if md5\_hash(generated\_string) == test\_hash:  
 end\_time = time.time()  
 break  
 #вычисление времени поиска  
 elapsed\_time = end\_time - start\_time  
 total\_time += elapsed\_time  
 print(f'Cycle {random\_string}: Time elapsed {elapsed\_time:.6f} seconds')  
 # Среднее время поиска  
 average\_time = total\_time / 10  
 print(f'Average time for {alphabet} with length {length}: {average\_time:.6f} seconds')  
  
#сравнение хешей для различных наборов символов и длины строки 2  
print("Lowercase letters (length 2):")  
test\_hash\_match(string.ascii\_lowercase, 2)  
  
print("\nUppercase letters (length 2):")  
test\_hash\_match(string.ascii\_uppercase, 2)  
  
print("\nDigits (length 2):")  
test\_hash\_match(string.digits, 2)  
  
print("\nPrintable characters (length 2):")  
test\_hash\_match(string.printable, 2)