TALLER PRÁCTICO FUERZA BRUTA Y RABIN-KARP

Complejidad Algorítmica – Corte 3 – 2024-2

Federico Vargas Rozo

Requisitos

1. Se hizo uso del Motor de Bases de Datos ‘MySQL’ versión 8.0 en la máquina local.
   1. HOST: localhost
   2. USUARIO: root
   3. CLAVE: *Cuasimodo8*
   4. DB: *ca\_tallerFB*
2. Se hizo uso del lenguaje de programación Python 3.12, con las siguientes librerías/módulos:
   1. **pymysql**Permite la funcionalidad de interactuar con bases de datos MySQL o MariaDB.
   2. **re**  
      Permite la funcionalidad de Expresiones Regulares (*Regular Expressions*) para búsqueda de patrones en cadenas de texto. Se utiliza en el ejercicio al momento de realizar las verificaciones de ‘limpieza’ del archivo de contraseñas.
   3. hashlib  
      Proporciona métodos para implementar algoritmos de hash criptográficos como MD5, SHA-1, SHA-256, y demás. De estos, se utilizará MD5.
   4. time  
      Proporciona métodos para trabajar con tiempo, permitiendo contabilizar los tiempos de ejecución del algoritmo de ataque.
   5. configparser

Proporciona funcionalidad de lectura y escritura de archivos de propiedades y configuraciones del programa. Utilizado para cargar los valores de múltiples variables.

Explicación del Algoritmo

1. Método clean\_passwords() “Limpiar Contraseñas”

Carga cada una de las contraseñas del archivo PasswordsUEB.txt, las separa por líneas, limpia los posibles caracteres vacíos antes y después de cada contraseña, y procede a realizar ciertas verificaciones, con el objetivo de descartar contraseñas “inútiles”. Aquellas contraseñas que pasen las verificaciones satisfactoriamente serán añadidas a una lista definitiva, que será la lista por utilizar en el transcurso de la ejecución del programa. Estas verificaciones son:

* Que la contraseña no esté previamente en la lista.
* Que la contraseña no tenga una longitud de menos de 4 caracteres.
* Que la contraseña no contenga únicamente 1 tipo de carácter.
* Que la contraseña no sea únicamente numérica.

1. Método hash\_cleaned\_passwords() “Hashear Contraseñas Limpias”

Itera por cada una de las contraseñas limpias de la función anterior y añade a un diccionario la pareja clave-valor:

(clave en texto plano: hash MD5 de la clave)

Esto se hace de esta manera para tener un acceso más rápido tanto al hash MD5 como a la clave en texto plano original.

1. Método execute\_attack() “Ejecutar Ataque”

Ejecuta el Ataque de Diccionario como tal.

* 1. Inicia realizando la conexión a la Base de Datos MySQL llamada ‘ca\_tallerFB’.
  2. Ejecuta la consulta: SELECT login, clave FROM Usuario;
  3. Almacena el resultado de la consulta en una tupla (registro) de tuplas (columnas).
  4. Itera por cada registro:
     1. Almacena el usuario en la variable ‘login’
     2. Almacena el hash MD5 de la clave del usuario en la variable ‘clave’
     3. Inicializa contadores de las comparaciones a 0
     4. Inicia el temporizador
     5. Itera por cada ítem del diccionario del método hash\_cleaned\_passwords()
        1. Aumenta las comparaciones totales
        2. Si ‘clave’ es igual al valor del hash MD5 generado del diccionario
           1. Aumentar las comparaciones correctas
           2. Terminar el ciclo
        3. Si no
           1. Aumentar las comparaciones incorrectas
     6. Termina el temporizador
     7. Calcula el tiempo de ejecución
     8. Imprime resultados
  5. Cierra la conexión con la Base de Datos

Resultados

Texto

Descripción generada automáticamente

Fig. 1: Ejecución del programa y sus resultados en consola.

**Explicación de la Solución**

La solución utiliza el algoritmo de *Rabin-Karp* cuyo funcionamiento se caracteriza por el uso de funciones Hash para comparar la cadena de texto (hash MD5 del cada usuario en la BD) y el patrón (hash MD5 generado de las contraseñas del archivo). Generalmente, ocurre una última comparación de verificación *‘carácter a carácter’* del texto a comparar, sin embargo, en nuestro caso, como el texto son los Hashes MD5 no es posible realizar esta comparación pues, no se conoce la cadena de texto original, es decir, la clave del usuario NO HASHEADA.

Al tratar de hacer la comparación *‘carácter a carácter’* nos topamos con que hace lo mismo que la operación *‘texto1’ = ‘texto2’*.

Por lo tanto, esta comparación no se observa en la solución.

De la misma manera, no se realiza una solución pura de Fuerza Bruta Clásica, pues no es posible bajo las condiciones dadas y la naturaleza del problema. Esto es debido a que, como el problema trata puramente con hashes, no es posible realizar comparaciones *‘carácter a carácter’* de cada contraseña, pues, nuevamente, no se conoce la clave NO HASHEADA, y estaríamos realizando el mismo proceso mencionado previamente. Únicamente se conocen las contraseñas del archivo *.txt* y sus hashes MD5 respectivos. Se podría llegar a pensar que uniendo todas las contraseñas del archivo y realizando el proceso de comparación y desplazamiento a manera de Fuerza Bruta podría llegar a servir, pero esto no es el caso, pues, nuevamente, no se conoce la longitud de la clave correcta y, por lo tanto, no es posible encontrar una solución acertada comparando y desplazando los textos.

Es decir, y, en conclusión, gran parte del problema radica en que se está tratando con Hashes. Esto limita los algoritmos posibles para llegar a una solución. Se pueden utilizar algoritmos que utilicen la metodología de *Rabin-Karp* pues son estos los que permiten comparaciones entre Hashes, detectar colisiones, solucionarlas, y finalmente, dar una contraseña correcta.

**¿Cómo ejecutar el programa?**

Tener el servidor MySQL prendido, con las credenciales establecidas, y el script SQL adjunto al taller ejecutado con los dos registros definidos. Tener en el mismo directorio que el archivo ***‘main.py’*** el archivo ***‘PasswordsUEB.txt’***.Tener las librerías y módulos definidos en los **Requisitos** instalados en la máquina**:**

* Por CMD: *pip install pymysql*

Ejecutar el programa ***‘main.py’***:

* Por CMD: *python main.py*

**Bibliografía y Referencias**

1. PyMySQL. (2023). *Documentación de PyMySQL*. [En Línea]. Disponible en: <https://pymysql.readthedocs.io/en/latest/user/examples.html>
2. Re. (2009). *Re – Regular expressions operations.* [En Línea]. Disponible en: <https://docs.python.org/3/library/re.html>
3. Ing. Wilson Rojas Reales. (2024). *“Complejidad Algorítmica – Fuerza Bruta”.* Complejidad Algorítmica, Pregrado en Ingeniería de Sistemas, Universidad El Bosque. [Presentación de Diapositivas].