Propuesta de Aplicativo TuringHash

Luis PAlberto Batalla Gonzá

Dicilezembre 2024

1 Capítulo 1: Descripción del Proyecto TuringHash

Periodo: 2024.C3 Septiembre - diciembre de 2024
Grupo: 7A/B
Asignatura: Lenguajes y Autómatas
Corte: 1
Actividad: 2024.C3.07X.LA.C1.A2
Título: Propuesta de Aplicativo TuringHash
Fecha de entrega: 2024.12.03

1.1 Descripción del sistema

TuringHash es un sistema basado en la técnica de fuerza bruta para descifrar contraseñas a partir de su valor hash MD5. El sistema permite ingresar un hash MD5 y obtener el texto original mediante un proceso iterativo utilizando combinaciones de caracteres posibles. Este tipo de aplicaciones es común en la ciberseguridad para estudiar vulnerabilidades en los algoritmos de hash.

1.2 Funcionalidad del Sistema

El sistema implementará una función que tomará un hash MD5 como entrada y generará posibles combinaciones de caracteres utilizando un conjunto de caracteres (letras y números). Luego, cada combinación se comparará con el hash de entrada hasta encontrar una coincidencia.

- Entrada: Un hash MD5 a descifrar.
- Salida: El texto original que corresponde al hash, si se encuentra en el rango especificado.
- Tecnología: Flask, Python, Hashlib, y Regex para la verificación y comparación de hashes.

1.3 Cadenas válidas e inválidas

1.3.1 Cadenas válidas

El sistema acepta cadenas que puedan generar un hash MD5 correspondiente al valor de entrada. Ejemplos de cadenas válidas:

- `hello123`
- `password`
- `qwerty`

1.3.2 Cadenas inválidas

El sistema no aceptará cadenas que no correspondan a ninguna combinación válida para el hash dado. Ejemplos de cadenas no válidas:

- `nonmatchinghash`
- `invalidstring123`

1.4 Restricciones

- El sistema está limitado a una longitud máxima de 5 caracteres para las cadenas generadas.
- Solo se consideran caracteres alfanuméricos (letras y números).
- El proceso puede ser intensivo dependiendo de la longitud del hash y el número de combinaciones posibles.

2 Capítulo 2: Diseño de la Función de Fuerza Bruta

2.1 Especificación Formal

Para resolver el problema, se diseñó una función de fuerza bruta que sigue el siguiente esquema:

$$T = \{Q, \Sigma, q_0, \delta, F\}$$

donde:

- Q es el conjunto de estados, que en este caso se limita al número de caracteres procesados.
- Σ es el conjunto de caracteres posibles (alfanuméricos).
- q_0 es el estado inicial, donde comienza el proceso de generación de cadenas.

- δ es la función de transición, que se define como la creación de combinaciones de caracteres y la comparación con el hash objetivo.
- F es el conjunto de estados finales, que se alcanzan cuando se encuentra una cadena que coincide con el hash.

2.2 Diagrama del Proceso

A continuación se presenta un diagrama del proceso de fuerza bruta que implementa TuringHash:

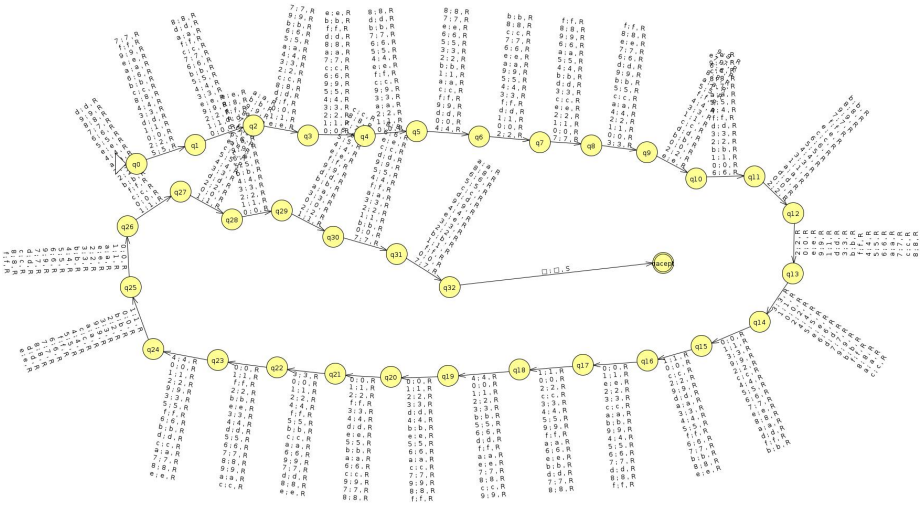

Figure 1: Diagrama de flujo del proceso de fuerza bruta.

3 Capítulo 3: Implementación en Software

3.1 Descripción de la Actividad

La implementación de TuringHash se realizó utilizando Python y Flask para la interfaz web, con un backend que gestiona la lógica de fuerza bruta. El sistema permite al usuario ingresar un hash MD5 y obtener el texto correspondiente si se encuentra dentro del rango de combinaciones posibles.

3.2 Interfaz

La interfaz gráfica del sistema permite al usuario introducir un hash MD5 y visualizar los resultados de la búsqueda en tiempo real. A continuación se presenta la interfaz de usuario del sistema:

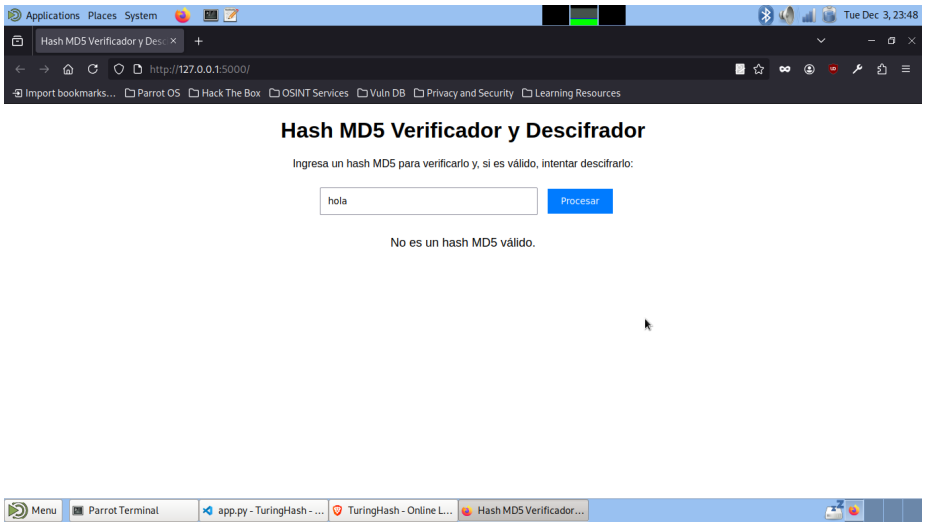

Figure 2: Interfaz gráfica del sistema TuringHash.

3.3 Código Fuente

A continuación se presenta el código fuente del componente principal, que implementa la función de fuerza bruta para descifrar el hash MD5:

```
from flask import Flask, request, jsonify
import hashlib
import itertools
import string

app = Flask(__name__)

def is_md5_hash(input_str):
    md5_pattern = r'^[0-9a-f]{32}$'
    return bool(re.match(md5_pattern, input_str))

def fuerza_bruta_md5(hash_objetivo, longitud_maxima=5):
    caracteres = string.ascii_letters + string.digits
    for longitud in range(1, longitud_maxima + 1):
        for combinacion in itertools.product(caracteres, repeat=longitud):
            texto_prueba = ''.join(combinacion)
            hash_prueba = hashlib.md5(texto_prueba.encode('utf-8')).hexdigest()
            if hash_prueba == hash_objetivo:
                return texto_prueba
    return None

@app.route('/process', methods=['POST'])
```

```
def process_hash():
    data = request.json
    input_text = data.get("hash")

    if not is_md5_hash(input_text):
        return jsonify({"message": "No es un hash MD5 válido.", "success": False})

    resultado = fuerza_bruta_md5(input_text)
    if resultado:
        return jsonify({"message": f"¡Texto encontrado! El texto es: {resultado}", "success'
    else:
        return jsonify({"message": "No se pudo descifrar el texto en el rango especificado.'

if __name__ == '__main__':
    app.run(debug=True)
```

3.4 Pruebas Realizadas

Se realizaron pruebas con diferentes valores de hash MD5 para verificar la efectividad del sistema. Las pruebas incluyeron cadenas comunes como "password" y "12345", además de otras combinaciones al azar.

3.5 Comentarios

El proyecto TuringHash ha sido una excelente oportunidad para aprender sobre las aplicaciones de fuerza bruta y cómo se pueden implementar algoritmos de descifrado en un entorno web. En proyectos futuros, se puede optimizar el rendimiento utilizando técnicas como la paralelización y el uso de GPUs.

4 Bibliografía

[1] Clase LA. Listado de propuestas de aplicativos de autómatas finitos deterministas, 2024. https://docs.google.com/spreadsheets/d/1kUtUhhcjtfIzn2-osH2RwJ9zC-gWwRV4wcR9Xu6v9k
Última consulta 2024.10.04.