Documento Explicativo: Fase 1 proyecto

Introducción

Este programa fue desarrollado con el objetivo de automatizar la detección de archivos maliciosos en una carpeta específica. Utiliza la API de VirusTotal para analizar archivos y organizar los resultados en carpetas de "infectados" y "limpios". Además, los resultados de los análisis se almacenan en una base de datos MySQL usando Mariadb.

Librerías Utilizadas

El programa utiliza varias librerías de Python para cumplir sus funciones:

- **os:** Manipula el sistema de archivos, permitiendo mover archivos y crear carpetas.
- **shutil**: Se utiliza para mover archivos de un lugar a otro.
- **requests**: Envía solicitudes HTTP a la API de VirusTotal.
- **time**: Controla los intervalos de espera en el análisis de los archivos.
- **mysql.connector**: Se encarga de interactuar con una base de datos MySQL para registrar los resultados.

Base de datos:

Para almacenar los resultados de los análisis lo primero fue la creación de la base de datos. Para esto utilizamos una máquina virtual Ubuntu Desktop en la cual realizamos la instalación de MariaDB como sistema de gestión de base de datos.

Con MariaDB ya instalado creamos una base de datos llamada **‘registro’** a la cual le asignamos un usuario llamado **‘admin’** con una contraseña.

Al usuario **‘admin’** se le asignan los permisos de conexiones remotas para que cualquier sistema por ejemplo nuestro windows local pueda establecer conexiones remotas con la base de datos.

También realizamos la creación de la tabla **‘resultado’** en la cual almacenamos el nombre del archivo, el resultado del análisis (infectado o limpio) y la localización donde se movió el archivo.

Estructura de la tabla ‘resultados’:

```
MariaDB [registro]> desc resultados
    -> ;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| archivo      | varchar(255) | YES  |     | NULL    |                |
| resultado    | varchar(50)  | YES  |     | NULL    |                |
| localizacion | varchar(255) | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
```

Estructura del Código

1. Conexión a la Base de Datos

La función ‘conectar_db()’ establece una conexión con nuestra base de datos llamada ‘ registro’ ubicada en una máquina virtual ubuntu desktop con IP **10.30.241.150**.

```python
def conectar_db():
    try:
        conexion = mysql.connector.connect(
            host="10.30.241.154",
            user="admin",
            password="FranPerez",
            database="registro"
        )
        return conexion
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None
```

Se define el host, el usuario, la contraseña y la base de datos donde se almacenarán los resultados. El manejo de errores está presente para asegurarse de que si ocurre algún problema durante la conexión, el programa lo capture.

2. Verificación de Archivos Procesados

Errores:

Nos apareció un shutil.error que decia que el archivo ya existía en la ubicación de destino, para solucionar este error creamos una nueva función llamada 'archivo_ya_procesado'.

Antes de analizar un archivo, el programa verifica si este ya ha sido procesado previamente. Esto se hace consultando la base de datos con la función 'archivo_ya_procesado()'. Si el archivo está registrado, el sistema lo omite.

```
Traceback (most recent call last):
  File "C:\Users\AlexPérezBarrera\Desktop\FASE1.2.py", line 135, in <module>
    procesar_archivos(directorio_a_procesar)
  File "C:\Users\AlexPérezBarrera\Desktop\FASE1.2.py", line 116, in procesar_archivos
    shutil.move(archivo, carpeta_limpios)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Program Files\Python312\Lib\shutil.py", line 845, in move
    raise Error("Destination path '%s' already exists" % real_dst)
shutil.Error: Destination path 'archivos_a_examinar\limpios\aaaaaa.exe' already exists
```

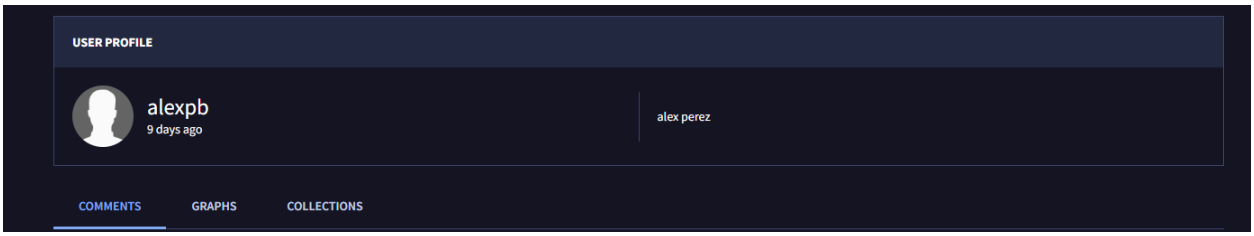
```python
def archivo_ya_procesado(nombre_archivo):
    conexion = conectar_db()
    if conexion:
        try:
            cursor = conexion.cursor()
            consulta = "SELECT COUNT(*) FROM resultados WHERE archivo = %s"
            cursor.execute(consulta, (nombre_archivo,))
            resultado = cursor.fetchone()[0]
            return resultado > 0
        except mysql.connector.Error as err:
            print(f"Error al consultar la base de datos: {err}")
            return False
        finally:
            cursor.close()
            conexion.close()
```

3. Subir Archivos a VirusTotal

La función ‘subir_archivo_a_virustotal’ toma un archivo, lo envía a la API de VirusTotal y recibe una respuesta con un identificador de análisis (‘analysis_id’), que es necesario para poder verificar el estado del análisis más tarde.

```python
def subir_archivo_a_virustotal(archivo):
    url = 'https://www.virustotal.com/api/v3/files'
    headers = {
        'x-apikey': 'f14cc7952d80c65551f70f4c9356e33d4019337f7eb76a633b99022c81be7781'
    }
    with open(archivo, 'rb') as f:
        response = requests.post(url, headers=headers, files={'file': f})
        return response.json()
```

Para esto tuvimos que registrarnos en la API de VirusTotal para poder conseguir nuestra API Key que nos ayudará a establecer conexión con el servidor y realizar peticiones de escaneado de los archivos.



4. Consultar Análisis de VirusTotal

Con el ‘analysis_id’, el programa consulta continuamente el estado del análisis hasta que esté completado o se agote el tiempo de espera.

```python
def consultar_analisis(analysis_id):
    url = f'https://www.virustotal.com/api/v3/analyses/{analysis_id}'
    headers = {
        'x-apikey': 'f14cc7952d80c65551f70f4c9356e33d4019337f7eb76a633b99022c81be7781'
    }
    response = requests.get(url, headers=headers)
    data = response.json()

    if data.get('data', {}).get('attributes', {}).get('status') == 'completed':
        return data
    else:
        return None
```

5. Procesar el Resultado

La función ‘tiene_virus’ revisa los resultados del análisis de VirusTotal. Si VirusTotal detecta algún virus marca el archivo como malicioso, devuelve **‘True’** y el archivo es movido a una carpeta de "infectados" que se encuentra ubicada en el escritorio dentro de la carpeta ‘archivos_a_examinar’.

```
def tiene_virus(resultados):
    if 'data' in resultados and 'attributes' in resultados['data']:

        resultados_motores = resultados['data']['attributes']['results']
        for motor, resultado in resultados_motores.items():
            if resultado.get('category') == 'malicious':
                print(f"Motor {motor} detectó el archivo como malicioso.")
                return True
    return False
```

6. Guardar el Resultado en la Base de Datos

La función ‘guardar_en_db’ se encarga de almacenar los resultados del análisis en una tabla llamada ‘resultados’ en la base de datos llamada registro en nuestro Ubuntu Desktop..

```
def guardar_en_db(archivo, resultado, localizacion):
    conexion = conectar_db()
    if conexion:
        try:
            cursor = conexion.cursor()
            consulta = "INSERT INTO resultados (archivo, resultado, localizacion) VALUES (%s, %
            cursor.execute(consulta, (archivo, resultado, localizacion))
            conexion.commit()
            print(f"Datos guardados en la base de datos para el archivo {archivo}")
        except mysql.connector.Error as err:
            print(f"Error al insertar en la base de datos: {err}")
        finally:
            cursor.close()
            conexion.close()
```

La tabla almacena el nombre del archivo, el resultado del análisis (infectado o limpio) y la localización donde se movió el archivo.

```
+----+-----------------+-----------+-----------------------------------------
+
|  1 | hola.exe        | Limpio    | archivos_a_examinar\limpios
|
|  2 | hola.exe        | Limpio    | archivos_a_examinar\limpios
|
|  3 | aaaaaa.exe      | Limpio    | archivos_a_examinar\limpios
|
|  4 | rvsvewaaesw.exe | Limpio    | archivos_a_examinar\limpios
|
|  5 | dfxzddrtdb.exe  | Limpio    | archivos_a_examinar\limpios\dfxzddrtdb.exe
|
|  6 | dfxzddrtdb.exe  | Limpio    | archivos_a_examinar\limpios\dfxzddrtdb.exe
|
|  7 | bille.exe       | Limpio    | archivos_a_examinar\limpios\bille.exe
|
```

7. Mover Archivos

Dependiendo del resultado del análisis, el archivo se mueve a una carpeta de "infectados" o "limpios" usando la función ‘mover_archivo’. Si el archivo ya existe en la carpeta de destino, no se mueve nuevamente por lo establecido anteriormente tras el error.

```python
def mover_archivo(archivo, destino):
    destino_completo = os.path.join(destino, os.path.basename(archivo))

    if not os.path.exists(destino_completo):
        shutil.move(archivo, destino_completo)
        return destino_completo
    else:
        print(f"El archivo {archivo} ya existe en el destino: {destino}")
        return destino_completo
```

8. Procesar archivos en un directorio

Esta es la función principal que procesa todos los archivos en el directorio especificado.

Excluye las carpetas ‘infectados’ y ‘limpios’, ya que los archivos en esas carpetas ya se han procesado.

Luego, por cada archivo, lo sube a VirusTotal, consulta los resultados y los mueve a la carpeta correspondiente según el resultado de Virustotal.