Threat Intelligence PDF Analysis Tool

> This Python script allows you to analyze a PDF file, extract embedded URLs, and check the safety of these URLs using the VirusTotal API. It displays detailed information about the PDF file, including its general properties, the safety of embedded URLs, and the results from various antivirus engines.

Installation

1. **Python**: Ensure you have Python installed on your system. You can download Python from the official website: **Python Downloads**.
2. **VirusTotal API Key**: You need to replace the `API_KEY` variable in the script with your own VirusTotal API key. You can obtain an API key by signing up for a free account on the VirusTotal website: **VirusTotal API**.
3. **Required Modules**: This script requires several Python modules. You can install them using pip, which is a package installer for Python.

   Open your terminal or command prompt and execute the following command to install the required modules:

   ```
   shellCopy code
   pip install requests aiohttp PyMuPDF colorama tabulate
   ```

   These modules are used for making HTTP requests, handling asynchronous operations, PDF analysis, terminal output styling, and tabulating data.

Usage

Specify PDF File: In the script, replace the value of the `pdf_file_path` variable with the path to the PDF file you want to analyze.

```
pythonCopy code
pdf_file_path = r"C:\path\to\your\pdf\file.pdf"
```

Run the Script: Open your terminal or command prompt, navigate to the directory containing the script, and execute the script.

The script will upload the specified PDF file to VirusTotal for analysis, fetch and display general information, safety of embedded URLs, and detailed engine results.

View Results: The script will display detailed information about the PDF file and provide insights into the safety of embedded URLs. The script outputs a summary of the analysis, general information about the PDF, the safety of embedded URLs, and the results from antivirus engines.

Code :

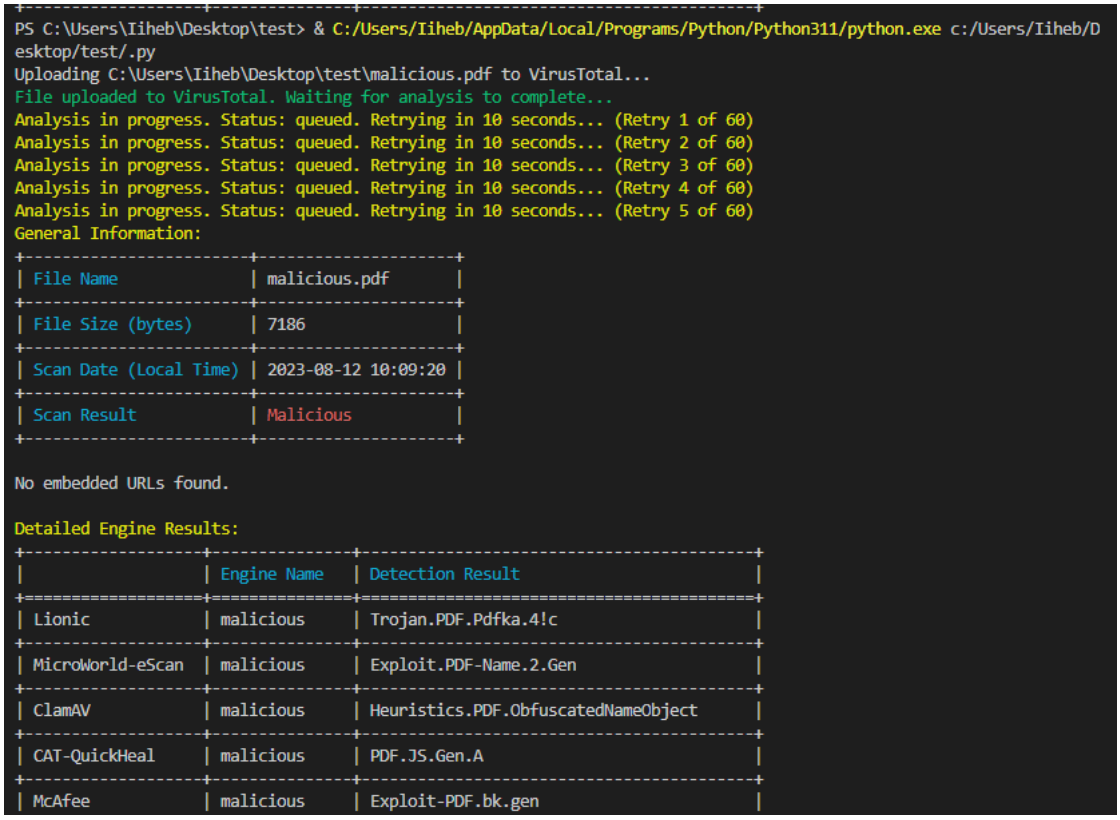
```
import requests
import os
import time
import asyncio
import fitz  # PyMuPDF
from concurrent.futures import ThreadPoolExecutor
from aiohttp import ClientSession
from colorama import Fore, Style, init
from datetime import datetime
from tabulate import tabulate

# Initialize colorama
init(autoreset=True)

# Replace with your VirusTotal API key
API_KEY = "YOUR API"

# Function to extract embedded URLs from a PDF
def extract_embedded_urls(pdf_file_path):
    try:
        urls = []
        pdf_document = fitz.open(pdf_file_path)
        for page_num in range(pdf_document.page_count):
            page = pdf_document[page_num]
            for link in page.get_links():
                if link['uri']:
                    urls.append(link['uri'])
        pdf_document.close()
        return urls
    except Exception as e:
        print(f"Failed to extract embedded URLs from the PDF: {str(e)}")
        return []

# Function to check the safety of a URL using the VirusTotal URL scan API
async def check_url_safety(session, url):
    url_scan_url = f"https://www.virustotal.com/api/v3/urls"
    headers = {
        "x-apikey": API_KEY,
    }
    params = {
        "url": url,
    }
    async with session.get(url_scan_url, headers=headers, params=params) as response:
        if response.status == 200:
            response_json = await response.json()
            if "data" in response_json:
                return response_json["data"]["attributes"]["last_analysis_stats"]["malicious"] == 0
    return False

# Function to upload a file to VirusTotal
def upload_file_to_virustotal(pdf_file_path):
    url = "https://www.virustotal.com/api/v3/files"
    headers = {
        "x-apikey": API_KEY,
    }
    with open(pdf_file_path, "rb") as file:
        files = {"file": (pdf_file_path, file)}
        response = requests.post(url, headers=headers, files=files)
        if response.status_code == 200:
            response_json = response.json()
            resource_id = response_json["data"]["id"]
            return resource_id
        else:
            print("Failed to submit the file for scanning.")
            return None

# Function to fetch general information about the file
async def fetch_general_info(session, resource_id, pdf_file_path):
    url = f"https://www.virustotal.com/api/v3/analyses/{resource_id}"
    headers = {
        "x-apikey": API_KEY,
    }
    for retry_count in range(60):
        async with session.get(url, headers=headers) as response:
            if response.status == 200:
                response_json = await response.json()
                status = response_json["data"]["attributes"]["status"]
                if status == "completed":
                    # Extract general information
                    file_name = os.path.basename(pdf_file_path)
                    file_size = os.path.getsize(pdf_file_path)
                    scan_date = datetime.fromtimestamp(os.path.getmtime(pdf_file_path)).strftime('%Y-%m-%d %H:%M:%S')
                    file_type = response_json["data"]["attributes"].get("file_info", {}).get("file_type", "N/A")
```

```
                    threat_categories = response_json["data"]["attributes"].get("tags", [])
                    popular_threat_labels = response_json["data"]["attributes"].get("popular_threat_labels", [])
                    detected_engines = [
                        (engine, result["category"], result.get("result", "N/A"))
                        for engine, result in response_json["data"]["attributes"]["results"].items()
                        if result["category"] == "malicious"
                    ]
                    embedded_urls = extract_embedded_urls(pdf_file_path)
                    url_safety_info = await check_embedded_urls_safety(session, embedded_urls)
                    scan_result = Fore.RED + "Malicious" + Style.RESET_ALL if detected_engines or any(safety == "Malicious" for _, saf

                    return file_name, file_size, scan_date, file_type, ', '.join(threat_categories) or "N/A", ', '.join(popular_threat
                elif status == "queued":
                    print(f"{Fore.YELLOW}Analysis in progress. Status: queued. Retrying in 10 seconds... (Retry {retry_count+1} of 60)
                    await asyncio.sleep(10)
                else:
                    print(f"Analysis failed with status: {status}")
                    return None, None, None, None, None, None, None, [], [], []
            else:
                print("Failed to fetch the scan result.")
                return None, None, None, None, None, None, None, [], [], []

# Function to check the safety of embedded URLs
async def check_embedded_urls_safety(session, urls):
    url_safety_info = []
    for url in urls:
        url_safe = await check_url_safety(session, url)
        url_safety_info.append((url, "Malicious" if url_safe else "Clean"))
    return url_safety_info

# Function to display detailed engine results
def fetch_engine_results(engine_results):
    if engine_results:
        headers = [Fore.CYAN + "Engine Name" + Style.RESET_ALL, Fore.CYAN + "Detection Result" + Style.RESET_ALL]
        print(f"\n{Fore.YELLOW}Detailed Engine Results:{Style.RESET_ALL}")
        print(tabulate(engine_results, headers=headers, tablefmt="grid"))
    else:
        print("\nNo malicious detections by antivirus engines.")

# Function to display embedded URLs
def fetch_embedded_urls(urls):
    if urls:
        headers = [Fore.CYAN + "Embedded URL" + Style.RESET_ALL, Fore.CYAN + "Safety" + Style.RESET_ALL]
        url_info = [[url, Fore.RED + "Malicious" + Style.RESET_ALL if safety == "Malicious" else Fore.GREEN + "Clean" + Style.RESET_AL
        print(f"\n{Fore.YELLOW}Embedded URLs:{Style.RESET_ALL}")
        print(tabulate(url_info, headers=headers, tablefmt="grid"))
    else:
        print("\nNo embedded URLs found.")

async def main():
    pdf_file_path = r"the pdf file path goes here"  # Specify the path to the PDF file
    print(f"Uploading {pdf_file_path} to VirusTotal...")
    resource_id = upload_file_to_virustotal(pdf_file_path)
    if resource_id:
        print(f"{Fore.GREEN}File uploaded to VirusTotal. Waiting for analysis to complete...{Style.RESET_ALL}")
        async with ClientSession() as session:
            file_name, file_size, scan_date, file_type, threat_categories, popular_threat_labels, scan_result, engine_results, embedde
            if file_name is not None:
                # Display the General Information table
                general_info_table = [
                    [Fore.CYAN + "File Name" + Style.RESET_ALL, file_name],
                    [Fore.CYAN + "File Size (bytes)" + Style.RESET_ALL, file_size],
                    [Fore.CYAN + "Scan Date (Local Time)" + Style.RESET_ALL, scan_date],

                    [Fore.CYAN + "Scan Result" + Style.RESET_ALL, scan_result],
                ]
                print(f"{Fore.YELLOW}General Information:{Style.RESET_ALL}")
                print(tabulate(general_info_table, tablefmt="grid"))

                # Display the Embedded URLs table
                fetch_embedded_urls(url_safety_info)

                # Display the Detailed Engine Results table
                fetch_engine_results(engine_results)
    else:
        print("Failed to upload the file to VirusTotal.")

if __name__ == "__main__":
    asyncio.run(main())
```

output :

The results provided below pertain to the analysis of two distinct PDF files.

One of the scanned PDFs has been identified as malicious, indicating the presence of suspicious or harmful content, while the other PDF has been deemed clean, suggesting the absence of such concerning elements. The ability to differentiate between these outcomes underscores the effectiveness of our analysis, enabling us to identify and flag potential security risks, thereby facilitating informed decision-making and proactive measures to safeguard digital environments.



Note

- **API Key Security**: Users must ensure the security of their VirusTotal API key. It should be kept confidential and not shared publicly.
- **Rate Limits**: The tool is designed to handle rate limits set by VirusTotal and includes retry logic when necessary.
- **File Size Limitation**: Users should be aware of any file size limitations imposed by their VirusTotal API subscription.

Enhancing Script Efficiency with Opportunities for Improvement:

The current implementation of the script presents ample room for enhancements, primarily focusing on increasing its efficiency and user-friendliness. Several key areas can be addressed to achieve these goals:

1. **Optimizing Multithreading**: A key aspect to explore is the optimization of the script's multithreading capabilities. This can significantly improve overall performance, particularly in scenarios with large-scale analysis tasks. By fine-tuning the threading model, such as exploring thread pools and asynchronous processing, we can efficiently handle concurrent tasks and reduce potential bottlenecks, leading to a smoother and more responsive experience.
2. **Hashing Function Integration**: To streamline the analysis process, integrating a hashing function can prove beneficial. By incorporating a hashing mechanism, we can create a unique signature for the file, facilitating its identification without the need for complete file uploads. This optimized approach allows us to send the hash to the VirusTotal API for scanning, potentially saving time and reducing bandwidth usage. It's crucial to ensure this hashing integration operates within a separate thread, preserving the main thread's responsiveness.
3. **Enhanced User Usability**: Usability is a paramount consideration. By refining the user interface and interaction, we can make the script more intuitive and user-friendly. Providing clear instructions, improving error handling, and presenting results in a more structured and comprehensible manner will contribute to an enhanced user experience.