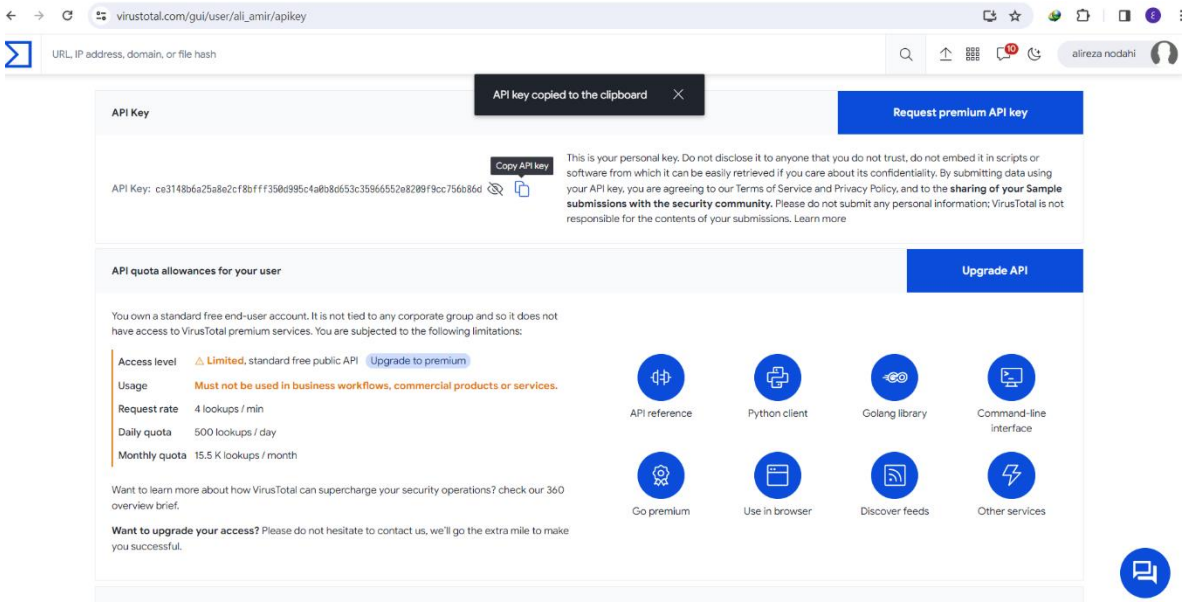
باسمه تعالی

آنتی ویروس و فایروال

پروژه درس امنیت و شبکه

اقایان : علیرضا نودهی و امیرحسین زارع کردخیلی

ابتدا به سایت ویروس تولز رفته و نام نویسی میکنیم سپس کلید تولید شده را دریافت میکنیم.



سپس کلید را دا خل کد جای گذاری میکنیم.

ما برای انتی ویروس دو برنامه داریم که یکی برای اسکن یک فایل و دیگری برای اسکن یک دایرکتوری میباشد.

file_scanner.py

```python
import hashlib
import argparse
from time import sleep
from pathlib import Path
from pprint import pprint
import requests

try:
    from key import API_KEY
except:
    API_KEY = "ce3148b6a25a8e2cf8bfff350d995c4a0b8d653c35966552e8209f9cc756b86d"


HEADERS = {"x-apikey": API_KEY}


def hash_it(file, algorithm):
    '''
    Returns hash of the file provided

    :param file: file to hash (type: str/pathlib obj) :param algorithm: algorithm to
    to use for hashing (valid algorithms: sha1 | sha256 | md5) (type: str)
    :return: file hash (type: str)
    '''
    if algorithm == "sha256":
        hasher = hashlib.sha256()
    elif algorithm == "sha1":
        hasher = hashlib.sha1()
    elif algorithm == "md5":
        hasher = hashlib.md5()
```

ابتدا باید کتابخونه های مورد نیاز خودمونو نصب کنیم.

file_scanner.py

```python
    with open(file, 'rb') as f:
        hasher.update(f.read())
    return hasher.hexdigest()


def vt_get_data(f_hash):
    '''
    The function gets the data against the file hash provided
    from the virustotal api

    :param f_hash: sha256 of the file to scan with virustotal
    :return: requests.models.Response
    '''
    print("GETTING DATA")
    url = f"https://www.virustotal.com/api/v3/files/{f_hash}"
    while True:
        response = requests.get(url, headers=HEADERS)
        if error_handle(response):
            break
    return response


def vt_post_files(file, url="https://www.virustotal.com/api/v3/files"):
    '''
    The function uploads a file to virustotal
    for analysis and returns the response from the
    virustotal api

    :param file: file to upload for analysis :param url: url to upload
    file to (use for files larger than 32MB) :return: requests.models.Response
```

e_scanner.py

```python
    file to (use for files larger than 32MB) :return: requests.models.Response
    '''
    with open(file, "rb") as f:
        file_bin = f.read()
    print("UPLOADING")
    upload_package = {"file": (file.name, file_bin)}
    while True:
        response = requests.post(url, headers=HEADERS, files=upload_package)
        if error_handle(response):
            break
    return response


def vt_get_analyses(response):
    '''
    The function returns the file hash of the uploaded file
    once the analysis of the uploaded file is available

    :param response: requests.models.Response
    :return: sha256 of the previously uploaded file (type: str)
    '''
    _id = response.json().get("data").get("id")
    url = f"https://www.virustotal.com/api/v3/analyses/{_id}"
    print(f"ID: {_id}")
    while True:
        print("WAITING FOR ANALYSIS REPORT")
        sleep(60)
        while True:
            response = requests.get(url, headers=HEADERS)
            if error_handle(response):
```

file_scanner.py

```python


def vt_get_upload_url():
    '''
    The function returns a url to upload files larger than 32MB
    to the virustotal api
    '''
    url = "https://www.virustotal.com/api/v3/files/upload_url"
    while True:
        response = requests.get(url, headers=HEADERS)
        if error_handle(response):
            break
    return response.json()["data"]


def error_handle(response):
    '''
    The function returns True if there are no errors
    and returns False otherwise

    :param response: requests.models.Response
    :return: bool
    '''
    if response.status_code == 429:
        print("WAITING")
        sleep(60)
    if response.status_code == 401:
        raise Exception("Invalid API key")
    elif response.status_code not in (200, 404, 429):
        raise Exception(response.status_code)
```

file_scanner.py

```python


def parse_response(response):
    '''
    The function extracts useful information from the respose JSON file
    and return it in JSON format.

    :param response: requests.models.Response
    :return: parsed data as json/dict
    '''
    json_obj = response.json().get("data").get("attributes")

    output = {}

    output["name"] = json_obj.get("meaningful_name")
    output["stats"] = json_obj.get("last_analysis_stats")
    output["engine_detected"] = {}

    for engine in json_obj.get("last_analysis_results").keys():
        if json_obj.get("last_analysis_results").get(engine).get("category") != "undetected":
            output.get("engine_detected")[engine] = {}
            output.get("engine_detected")[engine]["category"] = json_obj.get(
                "last_analysis_results").get(engine).get("category")
            output.get("engine_detected")[engine]["result"] = json_obj.get(
                "last_analysis_results").get(engine).get("result")

    output["votes"] = json_obj.get("total_votes")
    output["hash"] = {"sha1": json_obj.get(
        "sha1"), "sha254": json_obj.get("sha256")}
    output["size"] = json_obj.get("size")
    return output
```

file_scanner.py

```python


def bar(parsed_response):
    '''
    The function returns a bar to visually represent the engine
    detection.

    :param parsed_response: parsed dict/json from parse_response() function
    :return: bar (type: str)
    '''
    total = 72
    undetected = parsed_response.get("stats").get("undetected")
    data = f"{'@'*undetected}{' '*(total-undetected)}"
    bar = bar = f"+{'-'*total}+\n|{data}| {undetected}/{total} did not detect\n+{'-'*total}+"
    return bar

#####################################SCRIPT#####################################


parser = argparse.ArgumentParser(description="scan your files with virustotal")
parser.add_argument("file", action="store", nargs=1, help="file to scan")

parsed_arg = parser.parse_args()
#print(parsed_arg)

for f in parsed_arg.file:

    file = Path(f)

    if not file.exists():
```

```python
file_scanner.py
        raise Exception("File not found")

    # calculate file hash
    f_hash = hash_it(file, "sha256")

    # get file data against the file hash
    response = vt_get_data(f_hash)

    # if data for a specific file is not available
    # upload that file to virustotal
    if response.status_code == 404:

        # The response of vt_post_files can only be parsed by vt_get_analysis.
        # vt_post_files and vt_get_analyses should be made into a single function
        # but i left the separate in case there is a need to call vt_get_analysis
        # seperatley

        if file.stat().st_size > 32000000:
            # for files larger than 32MB
            response = vt_get_data(vt_get_analyses(
                vt_post_files(file, vt_get_upload_url())))
        else:
            # for small files
            response = vt_get_data(vt_get_analyses(vt_post_files(file)))

    if response.status_code == 200:
        # parse and print response
        parsed_response = parse_response(response)

        pprint(parsed_response, indent=2)
```

```python
file_scanner.py
        else:
            # for small files
            response = vt_get_data(vt_get_analyses(vt_post_files(file)))

    if response.status_code == 200:
        # parse and print response
        parsed_response = parse_response(response)

        pprint(parsed_response, indent=2)
        print()
        print(bar(parsed_response))
    else:
        raise Exception(response.status_code)

# 404 = upload file
# 400 = wait for resource to be available
# 401 = check user account
# 409 = resource already exists
# 429 = quota exceeded; wait 60 sec
# 200 = OK

# NOTE: The response variable is the response object of the requests module.
#       run print(type(response)) to double check the type of the variable.
# NOTE: Error handling is done with error_handle() function. The error handling
#       is build into all of the function (all function use error_handle() to
#       handle errors).

# * add better response printing
# * add support for scanning multiple file
```

سپس هر فایلی که بخواهیم با اجرای این فایل به به عنوان ارگومان پارسر ان فایل را اسکن کرده و خروجی را مشاهده میکنیم.

برای اسکن فولدرنیز به همین صورت عمل میکنیم.

folder_scanner.py

```python
import os
import time
import vt

# Replace with your actual VirusTotal API key
API_KEY = 'ce3148b6a25a8e2cf8bfff350d995c4a0b8d653c35966552e8209f9cc756b86d'

# Initialize the VirusTotal client
client = vt.Client(API_KEY)

# Replace 'path/to/folder' with the actual path of the folder you want to scan
FOLDER_PATH = input("enter yout path to folder : ")

# Set rate limiting parameters
RATE_LIMIT = 4  # Maximum number of lookups per minute
LOOKUP_INTERVAL = 60 / RATE_LIMIT  # Time between lookups in seconds

# Set daily quota parameters
DAILY_QUOTA = 5000  # Maximum number of lookups per day
LOOKUP_COUNT = 0  # Counter for number of lookups made today

#counts all files in Directory and Subdirectorys
def count_files_in_directory(path):
    file_count = 0

    for root, directories, files in os.walk(path):
        # Count files in the current directory
        file_count += len(files)

    return file_count
```

folder_scanner.py

```python
    return file_count

num_files = count_files_in_directory(FOLDER_PATH)
print(f"Number of files in directory: {num_files}")
# Define a list to store the file paths of files larger than 650MB
large_files = []
# Initialize counters for scan results
TOTAL_SCANNED = 0
TOTAL_MALICIOUS = 0
malicious_files = []
def is_malicious():
    TOTAL_MALICIOUS += 1
    malicious_files.append((filename, analysis.stats['malicious']))

# Iterate through all files in folder
for root,dirs ,filenames in os.walk(FOLDER_PATH):
    for filename in filenames:
        filepath = os.path.join(root, filename)
        # Check if file is not a directory
        if os.path.isfile(filepath):
            try:
                # Check rate limiting and daily quota limits
                if LOOKUP_COUNT >= DAILY_QUOTA:
                    print("Daily quota reached. Exiting...")
                    break
                elif LOOKUP_COUNT % RATE_LIMIT == 0 and LOOKUP_COUNT > 0:
                    time.sleep(LOOKUP_INTERVAL)

                if (os.path.getsize(filepath) / (1024 * 1024)) >= 650:
                    large_files.append(filepath)
```

```python
                if (os.path.getsize(filepath) / (1024 * 1024)) >= 650:
                    large_files.append(filepath)
                else:
                    # Scan the file and wait for completion
                    with open(filepath, 'rb') as f:
                        analysis = client.scan_file(f, wait_for_completion=True)

                # Update counters for scan results
                TOTAL_SCANNED += 1
                if 'malicious' in analysis.stats and analysis.stats['malicious'] != 0:
                    is_malicious()

                # Print scan results
                print(f"{filename}:")
                print(f"Analysis ID: {analysis.id}")
                print(f"Status: {analysis.status}")
                if 'total' in analysis.stats:
                    print(f"Total number of engines: {analysis.stats['total']}")
                if 'malicious' in analysis.stats:
                    print(f"Number of engines that detected the file as malicious: {analysis.stats['malicious']}")
                for scan in analysis.results:
                    print(f"{scan}: {analysis.results[scan]['result']}")
                print(f"{TOTAL_SCANNED} of {num_files}")
                print('')

            except vt.error.APIError as e:
                # Handle API errors
                print(f"An error occurred while scanning {filename}: {e}")

# Close the VirusTotal client connection
```

folder_scanner.py

```python
            except vt.error.APIError as e:
                # Handle API errors
                print(f"An error occurred while scanning {filename}: {e}")

# Close the VirusTotal client connection
client.close
# Print summary of scan results
print(f"Scanned {TOTAL_SCANNED} files.")
if TOTAL_MALICIOUS > 0:
    print(f"{TOTAL_MALICIOUS} files detected as malicious:")
    for filename, num_engines in malicious_files:
        print(f"{filename} ({num_engines} engines detected as malicious)")
else:
    print("No malicious files detected.")
# Print the list of large files at the end
if large_files:
    print("The following files are larger than 650MB:")
    for large_file in large_files:
        print(f"- {large_file}")
else:
    print("No files are larger than 650MB.")
print('All Scans Completed')
```

برای قسمت فایروال از قطعه کد زیر استفاده کردیم:

این کلاس مسیولیت گرفتن قواعد دارد.

```python
class FirewallRule:
    def __init__(self, src_ip, src_port, dst_ip, dst_port, dns_allowed):
        self.src_ip = src_ip
        self.src_port = src_port
        self.dst_ip = dst_ip
        self.dst_port = dst_port
        self.dns_allowed = dns_allowed
```

پکت های که رد و بدل میشوند را میخوانیم:

```python
class PacketHandler:
    def __init__(self, firewall_rules):
        self.firewall_rules = firewall_rules

    def handle_packet(self, packet_data):
        # Parse the packet data
        packet = packet_data.decode('utf-8')

        # Check if the packet is a DNS request
        if packet.startswith(''
            # Check if DNS re
            for rule in self.firewall_rules:
                if rule.dns_allowed:
                    break
            else:
                # Discard the packet if DNS requests are not allowed
                print("Discarding DNS request:", packet)
                return

        # Extract the source IP, destination IP, source port, and destination port
        src_ip, src_port, dst_ip, dst_port = packet.split(' ')[1:5]
```

(variable) firewall_rules: Any

```python
        # Check if the packet matches any of the firewall rules
        for rule in self.firewall_rules:
            if (rule.src_ip == src_ip and rule.src_port == src_port) or \
                    (rule.dst_ip == dst_ip and rule.dst_port == dst_port):
                # Allow the packet if it matches a firewall rule
                print("Allowing packet:", packet)
                return
        else:
            # Discard the packet if it doesn't match any firewall rules
            print("Discarding packet:", packet)
            return

# Define a function to read firewall rules from a file
```

```python
def read_firewall_rules(filename):
    rules = []

    with open(filename) as file:
        for line in file:
            # Parse the line into a firewall rule object
            rule_data = line.strip().split(' ')
            if len(rule_data) != 6:
                continue

            src_ip, src_port, dst_ip, dst_port, dns, allowed = rule_data
            rule = FirewallRule(src_ip, src_port, dst_ip,
                                dst_port, bool(int(allowed)))
            rules.append(rule)
    print(rules)
    return rules

# Define a function to start the firewall
```

در کنار این نرم فایلی داریم که قواعد و بهش میدیم:

کافیه فایل main.py اجرا کنیم تا رول ها اعمال بشوند.

روش دیگر برای اینکار استفاده از کتابخانه proxy.py است که میتوانیم پروکسی و ست کنیم و یک پلاگین بینویسم تا طبق خواسته ما پکت ها عبور کنند:

ابتدا باید کتابخانه را نصب کنیم و همچنیم فایل پلاگین که قرار دارد دار پکیج ها مون قرار بدهیم:

```
proxy --plugins myplg.new_plg.BlockIpAddressPlugin
```