

Malicious File Detection Tool

**Pauline Finlay**

**B00107945**

***Department of Informatics***

***School of Informatics & Engineering***

***Technological University Dublin, Blanchardstown***

***Dublin 15.***

**Word Limit for Assignment: 3000 Actual Word Count: 2463**

**BSc (Hons) in Computing in Digital Forensics & Cyber Security**

**DFCS H4013 Application Security**

**10th January 2021**

**Plagiarism Declaration**

I hereby certify that this material, which I am submitting for assessment for the course title of BSc. (Hons) In Computing in Digital Forensics and Cyber Security is my own work and that I have:

* Clearly referenced/listed all sources as appropriate
* Given the sources of all pictures, data etc. that is not our own
* Any work of any other student(s) either past or present is clearly referenced
* Not submitted for assessment work previously submitted for any other course, degree or qualification
* Not incorporated any text acquired from external agencies other than extracts from attributed sources (including online facilities)
* Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
* My work may be electronically checked for plagiarism, including, but not exclusively, by the use of plagiarism detection software and stored for future comparison
* Any false claim for this work will be penalised in accordance with the College’s regulations

**Signed:** Pauline Finlay **Date:** 10/01/2021

**Executive Summary**

This program takes a selected file path from the user and walks through the files generating an md5 hash for each file in path. These hash values are then compared against a benign hash list and any hashes not eliminated in this comparison check are sent to Virus Total, <https://www.virustotal.com/> via an API call. A report is then generated and outputted to a new text file via a JSON string.

Contents

[1. Introduction 1](#_Toc61178235)

[2. Application Features 2](#_Toc61178236)

[2.1. main.py 2](#_Toc61178237)

[2.2. funct.py 2](#_Toc61178238)

[2.2.1. hashfiles(mypath) 2](#_Toc61178239)

[2.2.2. querydb() 2](#_Toc61178240)

[2.2.3. malhash() 2](#_Toc61178241)

[2.2.4. queryvt() 2](#_Toc61178242)

[3. Feature Implementation 3](#_Toc61178243)

[3.1. mainmenu.py 3](#_Toc61178244)

[3.2. funct.py 3](#_Toc61178245)

[3.2.1. hashfiles(mypath) 4](#_Toc61178246)

[3.2.2. querydb() 5](#_Toc61178247)

[3.2.3. malhash() 6](#_Toc61178248)

[3.2.4. queryvt() 7](#_Toc61178249)

[4. Testing 8](#_Toc61178250)

[4.1. File Path 8](#_Toc61178251)

[4.1.1. Test 1 8](#_Toc61178252)

[4.1.2. Test 2 8](#_Toc61178259)

[4.1.3. Test 3 8](#_Toc61178260)

[4.1.4. Test 4 9](#_Toc61178269)

[4.2. Apikey 9](#_Toc61178270)

[4.2.1. Test 1 9](#_Toc61178272)

[4.2.2. Test 2 9](#_Toc61178280)

[5. Code 10](#_Toc61178281)

[5.1. main.py 10](#_Toc61178282)

[5.2. funct.py 12](#_Toc61178289)

[6. Conclusion 14](#_Toc61178290)

[7. References 15](#_Toc61178291)

[8. Appendix 16](#_Toc61178292)

[8.1. External Libraries 16](#_Toc61178293)

[8.1.1. hashlib 16](#_Toc61178294)

[8.1.2. json 16](#_Toc61178305)

[8.1.3. os 16](#_Toc61178306)

[8.1.4. requests 16](#_Toc61178307)

[8.1.5. time 16](#_Toc61178308)

[8.1.6. Traceback 16](#_Toc61178309)

[8.2. Zipped Folder 17](#_Toc61178310)

**Table of Figures**

[Figure 1: import statements in main.py 3](#_Toc61178311)

[Figure 2: import statements in funct.py 4](#_Toc61178312)

[Figure 3: hashfiles(mypath) function 4](#_Toc61178313)

[Figure 4: querydb() function 5](#_Toc61178314)

[Figure 5: malhash() function 6](#_Toc61178315)

[Figure 6: queryvt() function 7](#_Toc61178316)

[Figure 7: Invalid file path entered – Testing folder does not exist 8](#_Toc61178317)

[Figure 8: Invalid file path 8](#_Toc61178318)

[Figure 9: Valid file path entered program progresses 8](#_Toc61178319)

[Figure 10: Permission error accessing C:\Program Files 9](#_Toc61178320)

[Figure 11: User prompt to enter apikey 9](#_Toc61178321)

[Figure 12: Program terminates in error when invalid API key entered 9](#_Toc61178322)

# Introduction

This program walks through a given file path selected by the user, checking path validity and calculating the md5 hash value for each file in the path.

The calculated hash values are then compared to a benign hash value list called ‘benign.txt’ disregarding benign values and outputting any hash value which cannot be identified to a new text file called ‘suspicious.txt’.

The hash values in this file are then parsed to Virus Total via an API call to check if they have been identified as malicious. The API key used was the free version and is limited to 4 searches per minute and 500 per day.

This program was not tested on a Virtual Machine and so did not contain any malware sample files, to replicate the results received when a malware file hash is uploaded to Virus Total a known malware file hash has been hardcoded into the program to illustrate the results.

The virus total report generated is then written to text file called ‘report.txt’ for the user.

The file path used for testing was C:\Program Files from a Window10 Home machine, the calculated hash values from the first run make up the ‘benign.txt’ file and so in essence the program is checking for any modification to the original files and new files.

# Application Features

The application consists of two python files ‘***main.py’*** from which the application is run and a separate file ‘***funct.py’*** which contains the functions used in the application.

## ***main.py***

This is the main file which runs the application calling functions from the ***‘funct.py’*** file through an import statement.

## ***funct.py***

The file contains the functions used in the application.

## **hashfiles(mypath)**

This function takes a file path from a user prompt and loops through all the files in the path generating the md5 hash value for each and outputting each result on a new line in a text file ***‘ hashlist.txt’***. In order to deal with files which cannot be accessed they are bypassed and the file root and name outputted to a separate file **‘adminaccessfiles.txt’** for further investigation.

## querydb()

This function takes each md5 hash generated in ***‘hashlist.txt’*** and checks it against the ‘***benign.txt’*** file to match and eliminate known benign file hashes.

## malhash()

This function reads the unidentified hashes from the ***‘malicious.txt’*** file and returns them as a list to the user.

## queryvt()

This function sends each positively identified malicious file hash to Virus Total and generates a report for each writing the report to a text file ***‘report.txt’****.* The free API key version was used so checks are limited to 4 per minute or 500 per day in order to comply with this calls are throttled and limited.

# Feature Implementation

As it was deemed necessary for the successful running of the application to follow a specific running order user input was kept to a minimum, namely:

* Entering the file path
* Entering the user Virus Total API key

## ***mainmenu.py***

This file runs the application taking user inputs for file **path** and **apikey**. The file path option is tested for validity prompting the user to re-enter path if an invalid path is entered. An invalid API key is not tested against, but it would be more feasible for the user to hard code their individual API key to prevent user error. The functions from ‘***funct.py’*** are accessed via an import statement. The **os** library is also used to check the validity of the file path chosen by the user.

Hash checks sent to Virus Total are throttled and limited in the execution of the program also. As no actual malware was included in the test environment, there is a hard coded known malware hash hardcoded into the file to illustrate findings in the Virus Total report.

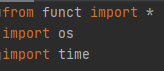


Figure 1: import statements in main.py

## ***funct.py***

This file contains all the functions used in the application and are called into the ‘***main.py’*** file through an import statement – from funct import \*. The external libraries used in this file are:

* **import hashlib** – to generate file hashes in selected file path.
* **import json** – to output the Virus Total report to a text file as json string.
* **import os** – this is used to walkthrough the path creating a hash digest of the files in path by interacting with the underlying Operating System and also to check file path validity.
* **import requests** - this is used to send the HTTP request to Virus Total to check for malicious file hashes.
* **Import traceback** – used to handle the general errors reporting to stack trace at point of failure

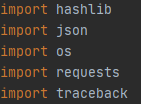


Figure 2: import statements in funct.py

## **hashfiles(mypath)**

This function walks through the files in the user specified path generating an md5 hash values for each file in path. Large file sizes are handled by reading the file in chunks and the Error 13 Permission Denied error is handled by a try: except block.

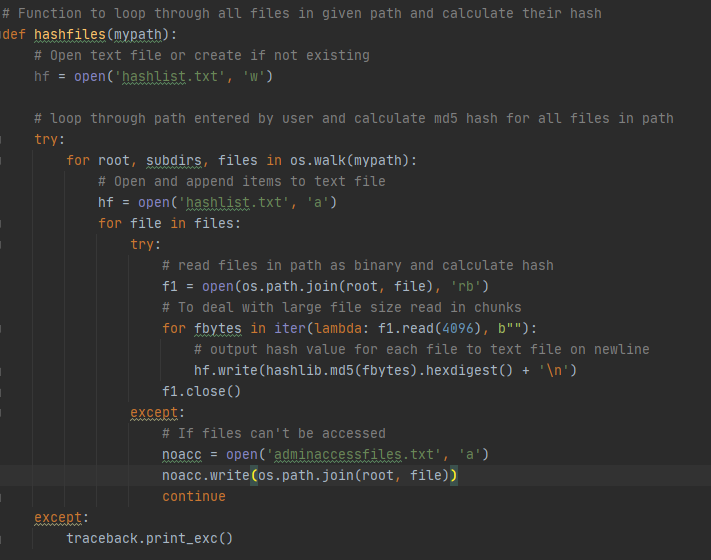


Figure 3: hashfiles(mypath) function

## querydb()

This function compares the file hashes generated by the **hashfiles(mypath)** function by comparing the results in two text files, ***‘hashlist.txt’*** and a sample list of benign hash values ***‘benign.txt’***. Hash values which are not identified are written to a new text file ***‘suspicious.txt’*** to be later analysed by sending to the Virus Total site, [https://www.virustotal.com](https://www.virustotal.com/gui/), to be checked for a malicious file score.

Note: NIST provides a list of benign hash values which can be downloaded and referenced against. This was downloaded but I encountered privilege issues accessing it so created my own base file of hashes to compare against. Testing was on the C:\Program Files the first run of hashes is the **‘benign.txt’** file and further checks on the same path check for additions or modifications to this path.

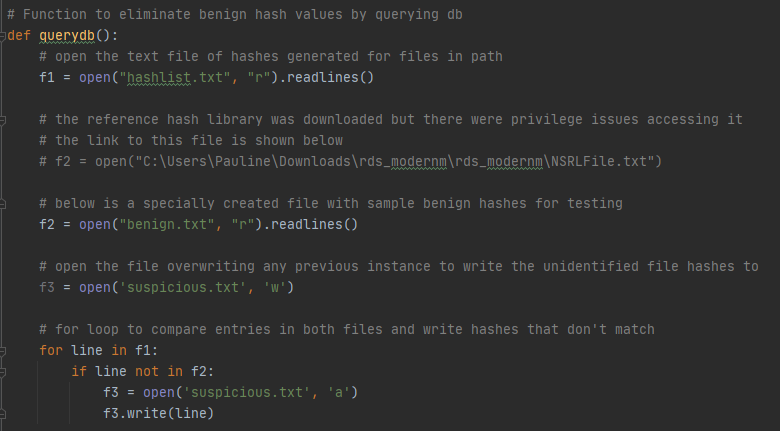


Figure 4: querydb() function

## malhash()

This function takes each line in the ***‘suspicious.txt’*** file and return the hash value in the line to the user as a list using a for loop.

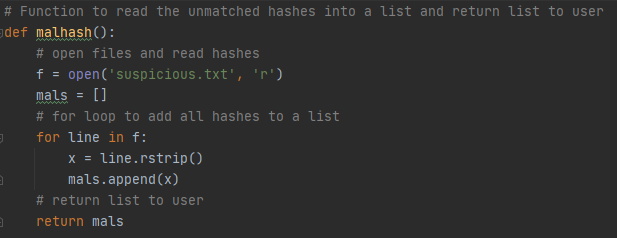


Figure 5: malhash() function

## queryvt()

This function sends the suspicious file hashes identified to Virus Total via a HTTP request taking 2 parameters the **apikey** of the user and the file hash as a **resource.** The Virus Total report is then outputted to a new text file ***‘report.txt’*** as a json string. It uses the **requests** library to send the request to Virus Total.

The Virus Total API key used is the free version and can only be used to query 4 hash values per minute to throttle the speed a sleep timer is used for 15 seconds after each request. There is also a maximum limit of 500 hash values per day. To deal with this requested are throttled using sleep timer and limited using range.

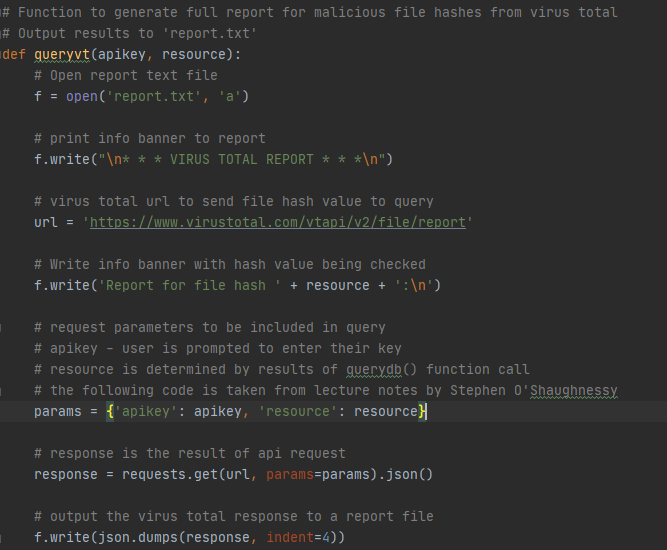


Figure 6: queryvt() function

# Testing

## File Path

### Test 1

This test is for a seemingly genuine file path entered in the correct format but the folder ***Testing*** at the end of the path does not exist, so it returns an ‘***Invalid Path’*** message and prompts the user to re-enter a valid file path.

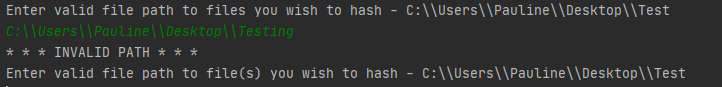


Figure 7: Invalid file path entered – Testing folder does not exist



### Test 2

This test takes in gibberish from the user and does not resemble a valid file path, it too returns an ‘***Invalid Path’*** message and prompts the user to re-enter a valid file path.

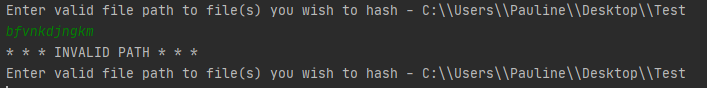


Figure 8: Invalid file path

### Test 3

This test shows a valid file path entered by the user and thus the program continues outputting information messages to the user.

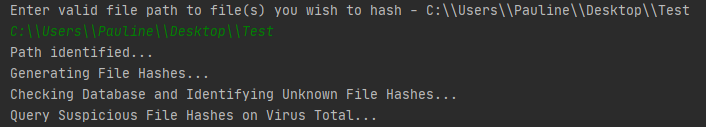


Figure 9: Valid file path entered program progresses



### Test 4

When trying to access the C:\Program Files path normal user encounters a Permission error when trying to access the files.

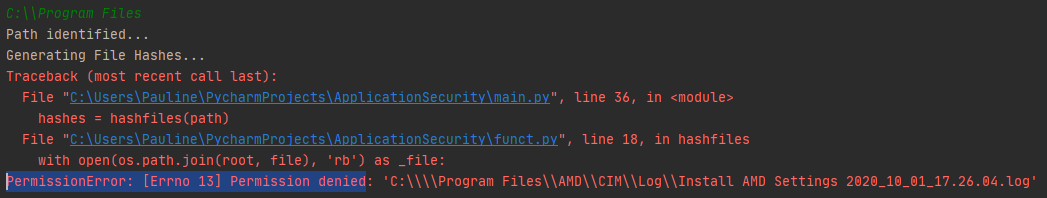


Figure 10: Permission error accessing C:\Program Files

In order to overcome this a try except block was inserted bypassing in file(s) where permission is denied and outputting their file root and name to separate text file **‘adminaccessfiles.txt’**.

## Apikey



### Test 1

The user is prompted to enter their Virus Total API key in order to send the request to the Virus Total website to check the suspicious file hashes. When a valid API key is entered the program continues and outputs information messages to the user. Once the program terminates a report is generated from the Virus Total results and is outputted to a text file ***‘report.txt’***.

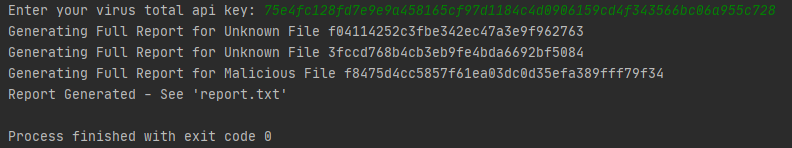


Figure 11: User prompt to enter apikey



### Test 2

An invalid API key will cause the following program error, this has not been tested against, but it would be more beneficial to he user to hard code their API key eliminating this error.



Figure 12: Program terminates in error when invalid API key entered

# Code

## main.py

1. # Programmer: Pauline Finlay  
   # Date: 10/01/2021  
   # Main file which imports functions from funct.py  
     
     
   from funct import \*  
   import os  
   import time  
     
   # print statements to display program info to user  
   print("\* \* \* Malicious File Detection Tool \* \* \*\n".upper())  
   print("\* \* \* PROGRAM FUNCTIONS \* \* \*")  
   print("This program allows the user to select a file path, it then loops")  
   print("through all files in this path generating their md5 hash value.")  
   print("Next known benign file hashes are excluded by checking reference database.")  
   print("Suspicious file hashes are then forwarded to VirusTotal for analysis.")  
   print("A full report is then generated for all identified Malicious Samples")  
   print("\* \* \* \* \* \* \* \* \* \* \*\n")  
     
     
   # choose path to folder to generate files hashes  
   # prompt user to enter valid file path for walk-through  
   path = input("Enter valid file path to files you wish to hash - C:\\\\Users\\\\Pauline\\\\Desktop\\\\Test\n")  
     
   # Check whether the specified path exists or not  
   isExist = os.path.exists(path)  
     
   # while loop to handle invalid file path entry  
   while not isExist:  
    print("\* \* \* Invalid Path \* \* \*".upper())  
    # prompt to reenter file path  
    path = input("Enter valid file path to file(s) you wish to hash - C:\\\\Users\\\\Pauline\\\\Desktop\\\\Test\n")  
    # Check whether the specified path exists or not  
    isExist = os.path.exists(path)  
     
   else:  
    print("Path identified...")  
     
    # 1 Walk-through File Path & Generate File Hashes  
    print("Generating File Hashes...")  
    # Function call to generate the file hashes for files in path  
    hashes = hashfiles(path)  
     
    # 2 Check File Hashes against benign hash list  
    print("Checking Database and Identifying Unknown File Hashes...")  
    # Function call to check hashes against list of benign hashes  
    querydb()  
     
    # 3 Query Virus Total for Suspicious File Hashes  
    print("Query Suspicious File Hashes on Virus Total...")  
    # Virus total api key for testing if necessary  
    # apikey = '75e4fc128fd7e9e9a458165cf97d1184c4d0906159cd4f343566bc06a955c728'  
    apikey = input("Enter your virus total api key:\t")  
     
    # Check to see if api key is valid  
    # isExist = os.path.exists(path)  
     
    f = open('report.txt', 'w')  
    # Function call to extract identified hashes from text file and add to queryvt() function call  
    mals = malhash()  
    for x in mals:  
    i = 1  
    if i < 500:  
    queryvt(apikey, x)  
    print("Generating Full Report for Unknown File " + x)  
    # throttle speed to limit check to 4 values per minute  
    time.sleep(15)  
    i += 1  
    else:  
    print("Daily Search Limit Reached")  
     
    # this program was not run on a VM with malware samples  
    # this is the hash value for a known malicious file for testing  
    x = 'f8475d4cc5857f61ea03dc0d35efa389fff79f34'  
    print("Generating Full Report for Malicious File " + x)  
    # Query vt to generate a report on a malicious file  
    queryvt(apikey, x)  
     
    # 4 Virus Total Report generated written to 'report.txt'  
    print("Report Generated - See \'report.txt\'")

## funct.py

# Programmer: Pauline Finlay  
# Date: 10/01/2021  
# Functions file to be used in main file through import statement  
  
  
import hashlib  
import json  
import os  
import requests  
import traceback  
  
  
# Function to loop through all files in given path and calculate their hash  
def hashfiles(mypath):  
 # Open text file or create if not existing  
 hf = open('hashlist.txt', 'w')  
  
 # loop through path entered by user and calculate md5 hash for all files in path  
 try:  
 for root, subdirs, files in os.walk(mypath):  
 # Open and append items to text file  
 hf = open('hashlist.txt', 'a')  
 for file in files:  
 try:  
 # read files in path as binary and calculate hash  
 f1 = open(os.path.join(root, file), 'rb')  
 # To deal with large file size read in chunks  
 for fbytes in iter(lambda: f1.read(4096), b""):  
 # output hash value for each file to text file on newline  
 hf.write(hashlib.md5(fbytes).hexdigest() + '\n')  
 f1.close()  
 except:  
 # If files can't be accessed  
 noacc = open('adminaccessfiles.txt', 'a')  
 noacc.write(os.path.join(root, file))  
 continue  
 except:  
 traceback.print\_exc()  
  
  
# Function to eliminate benign hash values by querying db  
def querydb():  
 # open the text file of hashes generated for files in path  
 f1 = open("hashlist.txt", "r").readlines()  
  
 # the reference hash library was downloaded but there were privilege issues accessing it  
 # the link to this file is shown below  
 # f2 = open("C:\Users\Pauline\Downloads\rds\_modernm\rds\_modernm\NSRLFile.txt")  
  
 # below is a specially created file with sample benign hashes for testing  
 f2 = open("benign.txt", "r").readlines()  
  
 # open the file overwriting any previous instance to write the unidentified file hashes to  
 f3 = open('suspicious.txt', 'w')  
  
 # for loop to compare entries in both files and write hashes that don't match  
 for line in f1:  
 if line not in f2:  
 f3 = open('suspicious.txt', 'a')  
 f3.write(line)  
  
  
# Function to read the unmatched hashes into a list and return list to user  
def malhash():  
 # open files and read hashes  
 f = open('suspicious.txt', 'r')  
 mals = []  
 # for loop to add all hashes to a list  
 for line in f:  
 x = line.rstrip()  
 mals.append(x)  
 # return list to user  
 return mals  
  
  
# Function to generate full report for malicious file hashes from virus total  
# Output results to 'report.txt'  
def queryvt(apikey, resource):  
 # Open report text file  
 f = open('report.txt', 'a')  
  
 # print info banner to report  
 f.write("\n\* \* \* VIRUS TOTAL REPORT \* \* \*\n")  
  
 # virus total url to send file hash value to query  
 url = 'https://www.virustotal.com/vtapi/v2/file/report'  
  
 # Write info banner with hash value being checked  
 f.write('Report for file hash ' + resource + ':\n')  
  
 # request parameters to be included in query  
 # apikey - user is prompted to enter their key  
 # resource is determined by results of querydb() function call  
 # the following code is taken from lecture notes by Stephen O'Shaughnessy  
 params = {'apikey': apikey, 'resource': resource}  
  
 # response is the result of api request  
 response = requests.get(url, params=params).json()  
  
 # output the virus total response to a report file  
 f.write(json.dumps(response, indent=4))

# Conclusion

This program is designed to check for malicious files by uploading their md5 hash value to Virus Total via an API call. Due to access errors checking against the NIST NSRL file it now uses it’s own benign hash file to run checks against for changes in the Program Files path outputting file hashes which do not match to an suspicious file and read from here to the function which sends the request to Virus Total.

# References

Requests: HTTP for Humans™ — Requests 2.25.1 documentation. 2020. Requests: HTTP for Humans™ — Requests 2.25.1 documentation. [ONLINE] Available at: https://requests.readthedocs.io/en/master/. [Accessed 31 December 2020].

os — Miscellaneous operating system interfaces — Python 3.9.1 documentation. 2020. os — Miscellaneous operating system interfaces — Python 3.9.1 documentation. [ONLINE] Available at: https://docs.python.org/3/library/os.html. [Accessed 31 December 2020].

Python JSON. 2020. Python JSON. [ONLINE] Available at: https://www.w3schools.com/python/python\_json.asp. [Accessed 31 December 2020].

hashlib — Secure hashes and message digests — Python 3.9.1 documentation. 2020. hashlib — Secure hashes and message digests — Python 3.9.1 documentation. [ONLINE] Available at: https://docs.python.org/3/library/hashlib.html. [Accessed 31 December 2020].

O’Shaughnessy, S. 2020. Moodle - TU Dublin Blanchardstown Campus: Log in to the site. [ONLINE] Available at: https://vle-bn.tudublin.ie/pluginfile.php/354792/mod\_resource/content/0/Cyber\_Python\_2020\_part\_2.pdf. [Accessed 31 December 2020].

"Getting the SHA-1 (or MD5) hash of a directory « Python recipes « ActiveState Code", *Code.activestate.com*, 2021. [Online]. Available: https://code.activestate.com/recipes/576973/. [Accessed: 10- Jan- 2021].

# Appendix

## External Libraries

### hashlib

This module implements a chosen hashing algorithm. For the purposes of this program md5 was the selected hashing algorithm.



### json

JSON is text written with JavaScript object notation and in this program is used to write the output from the Virus Total report converting it into a json string using the ***json.dumps()*** method.

### os

This allows the user to interact with the underlying operating system. In this instance it was used to walkthrough the selected file path and verify the validity of the file path.

### requests

This allows the user to make HTTP requests in python. In this instance it is used to send the request to Virus Total to query the file hashes.

### time

This allows the user to throttle the hash values being sent to Virus Total for checking to 4 per minute as allowed by the free version API key used.

### Traceback

This is used to handle general access errors reporting to stack trace at time of failure.

## Zipped Folder

Included with this report are:

* The two python files
  + main.py
  + funct.py
* The benign list used for comparison
  + benign.txt
* The file containing Program Files which could not be accessed
  + adminaccesfiles.txt
* Sample Testing Files (based on small test sample of 13 files not appearing in benign.txt)
  + hashlist.txt
  + suspicious.txt
* A sample Virus Total Report File
  + report.txt