User Guide:

PYTHON-FORENSICS TOOLKIT

Written By: Michael Pedzimaz

Download Toolkit: https://github.com/mpedzi03

Preface

The following pages contain information on how to efficiently use the programs contained within what I have come to call the Python-Forensics Toolkit. The programs vary in function and utility, but all serve the purpose of being tools that can automate digital forensic activity. This toolkit is heavily founded on the work written by Chet Hosmer in his book, "Python Forensics: A Workbench for Inventing and Sharing Digital Forensic Technology". Much of the code is heavily based on the programs he has written within the chapters of his book, with the addition of my own functions and utility. If you find any of the following programs interesting and would like to dive deeper into the concepts they explore, I highly suggest your read Mr. Hosmer's book.

This toolkit contains four separate programs that each perform unique operations on a piece or pieces of data, and also a dual program GUI based ping sweep & port scanning setup as an introduction to networking investigation methods. All the programs within Python-Forensics are written in, you guessed it, the Python programming language. A huge benefit of this is that you do not need to be some advanced programmer to read and understand the code contained within the toolkit. Python syntax is very readable to the naked eye and with the addition of its strong indentation policies, proves to be very neat and organized for easy viewing. The Python language is a great choice for programming in the digital forensics environment due to many reasons, such as the plethora of modules and third-party libraries that exist, the open-source availability of source code and IDEs, consistent updates & publicly available documentation, and strong programming community that keeps it prospering.

I would like to provide a little description of the flow within which the upcoming descriptions of the programs will be displayed. This might increase the readability for some, while also setting a guideline for myself so I don't get carried away with too many details. The layout for each program will be as follows:

- A brief description as to what the program is designed to do
- A short breakdown of the program and its components
- Test case input + results interpretation
- Summary and use-case scenarios

The order in which the programs will appear will be:

PFish → PSearch → GPS_Extraction → NLTK_Query → Scanners → X-Core_PasswordGen

Considering there are many great how-to videos online that go into detail about how to setup your Python programming environment, I will not be explaining how to do that in this guide. One important thing to remember is to add your installed Python directory (e.g. C:\Python27) to your System or User PATH Environment Variable, so that you will be able to run some of these programs directly from your command prompt. Please note, however, the programs within the Python-Forensics toolkit were designed to work with Python versions 2.x, so if you are working with the newer 3.x versions, you may encounter some unforeseen errors. I should also note that I will be describing the programs from a Windows OS perspective, many of which I will be running directly from the Windows Command Prompt. Those of you working with Linux systems will have similar procedures to follow, albeit with varying input commands. Please do reference the book, “Python Forensics: A Workbench for Inventing and Sharing Digital Forensic Technology”, for a more complete tutorial of these programs on Linux OSs.

Lastly, there will be a folder /UserGuide_Images located in the repository that will contain all the images that are included in this user guide. Enjoy the presentation and please feel free to contact me with any questions, concerns, or considerations. Thank you for reading.

Pfish

The Python file system hashing program, or pfish for short, is designed for one-way file system hashing of a selected directory or file. It is separated into two files, the main driver file pfish.py and a secondary file called _pfish.py which contains all the support functions. There are many hashing algorithms available, some outdated and unsecure by modern standards, while others still in struggle of widespread adoption and acceptance in the realm of cyber security. This program incorporates the hashing algorithms MD5, SHA-224, SHA-256, SHA-384, SHA-512. You can easily add another hashing algorithm by simply adding it to the 'SelectionOfHashingAlgorithm' function.

This program can be broken down into six pieces. The 'Main' function serves to control the overall flow of the program, instantiating the Python logger while also invoking the command line parser, and finally launching the 'WalkPath' function. The 'ParseCommandLine' function parses & validates the user input to pass around the necessary parser-generated values, such as the selected hashing algorithm, directory or file rootPath, and results report output location. The 'WalkPath' function starts at the desired location and traverses every directory and file underneath, hashing with the chosen algorithm along the way. The 'HashFile' function will hash every single file in the desired path, while also pulling important metadata associated with the file and outputting these results in file-per-line fashion to our report. Next, we have our 'CSVWriter' class that allows us to create a CSV object and populate it with our results. We can modify or create different subclasses of this class to produce result documentation that suits our required program goals. Finally, we use the built-in Python Standard Library logger which will print out a general log file with important information regarding our tests, while also printing out informative, warning, or error messages.

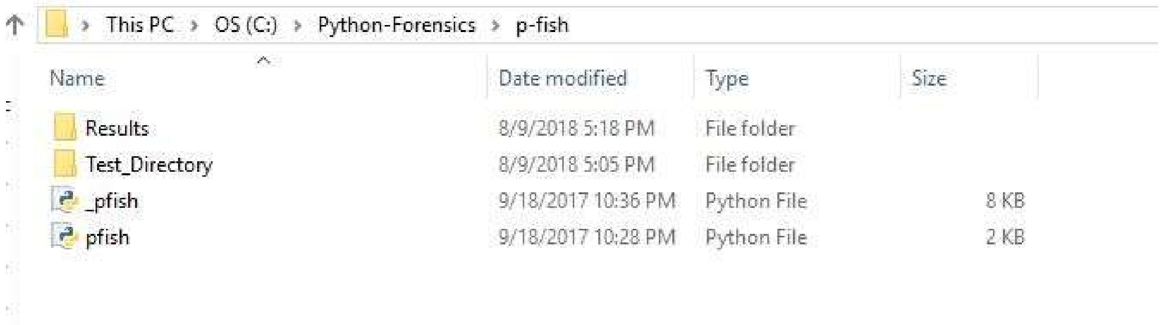
To run this program, you need to provide three mandatory arguments:

- Hashing algorithm of choice (--md5, --sha224, --sha256, --sha384, --sha512)
- Root path for hashing (-d, *or* --rootPath *followed by* "*test directory*")
- Path for report (-r, *or* --reportPath *followed by* "*results directory*")

As well as an optional argument:

- Show progress messages (-v, *or* --verbose)

Figure 1 is an example of the program setup in a standalone directory. All the proceeding programs will be stored in a similar fashion:

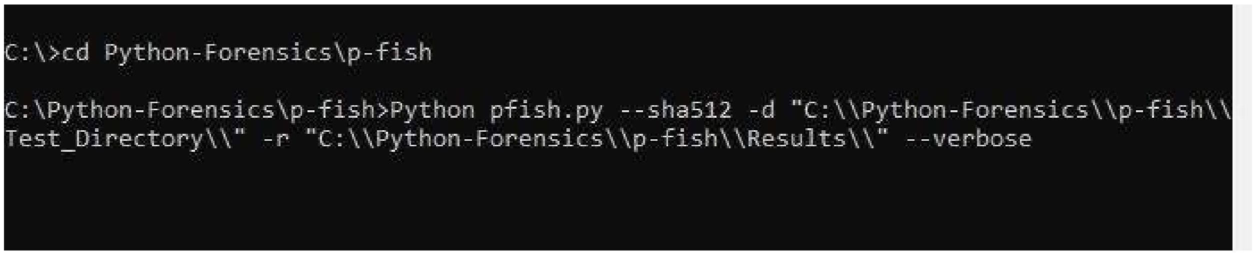


This PC > OS (C:) > Python-Forensics > p-fish

Name	Date modified	Type	Size
Results	8/9/2018 5:18 PM	File folder	
Test_Directory	8/9/2018 5:05 PM	File folder	
_pfish	9/18/2017 10:36 PM	Python File	8 KB
pfish	9/18/2017 10:28 PM	Python File	2 KB

Figure 1. Python file system hashing directory setup

Figure 2 shows how the CMD prompt input should look:

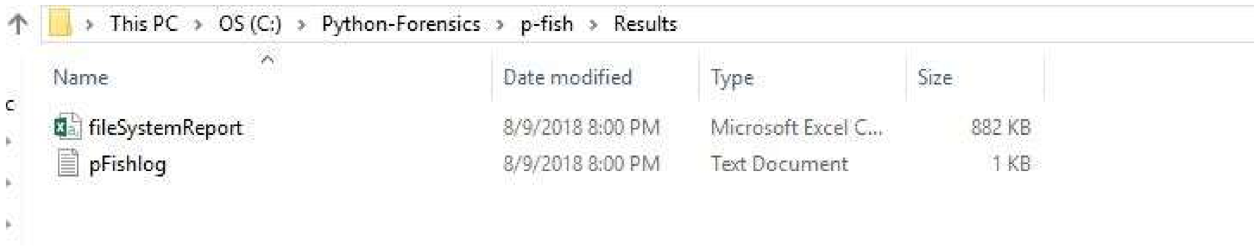


```
C:\>cd Python-Forensics\p-fish

C:\Python-Forensics\p-fish>Python pfish.py --sha512 -d "C:\\Python-Forensics\\p-fish\\
Test_Directory\\" -r "C:\\Python-Forensics\\p-fish\\Results\\" --verbose
```

Figure 2. pfish.py CMD commands

After running this line, there will be an Excel report created in our \Results directory, as well as a log file. Since the optional argument '--verbose' was inputted, we shall see each file quickly flash through the CMD prompt along with its respective hash digest.

Our \Results directory now contains the following files:

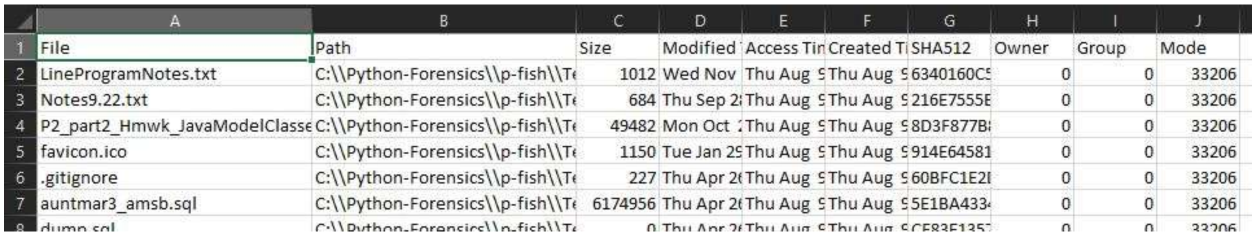


This PC > OS (C:) > Python-Forensics > p-fish > Results

Name	Date modified	Type	Size
fileSystemReport	8/9/2018 8:00 PM	Microsoft Excel C...	882 KB
pFishlog	8/9/2018 8:00 PM	Text Document	1 KB

Figure 3. pfish.py CSV and log output

The Excel report contains valuable metadata regarding our hashed files in addition to the file digest. Depending on the size of the scanned directory, the amount of lines present may be very large.

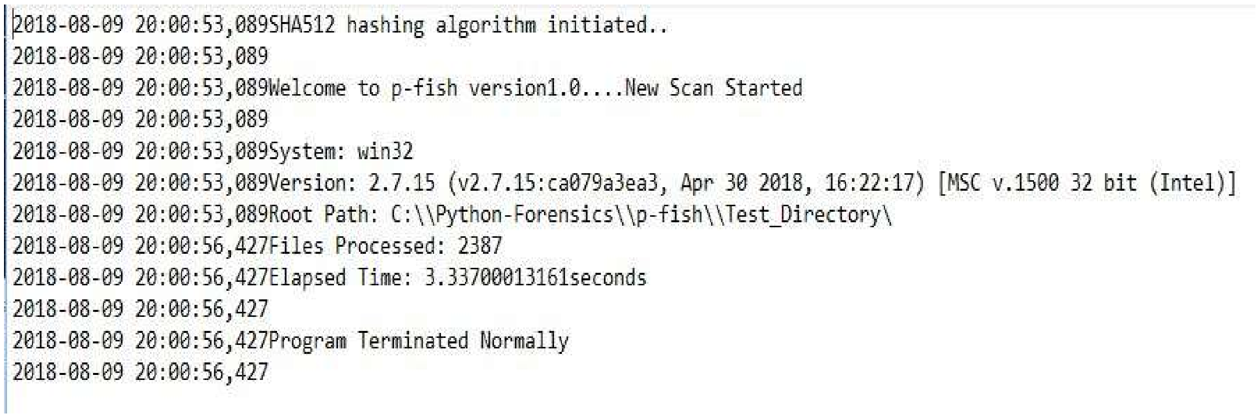


	A	B	C	D	E	F	G	H	I	J
1	File	Path	Size	Modified	Access Tin	Created Ti	SHA512	Owner	Group	Mode
2	LineProgramNotes.txt	C:\\Python-Forensics\\p-fish\\Te	1012	Wed Nov	Thu Aug 9	Thu Aug 9	6340160C5	0	0	33206
3	Notes9.22.txt	C:\\Python-Forensics\\p-fish\\Te	684	Thu Sep 2	Thu Aug 9	Thu Aug 9	216E7555E	0	0	33206
4	P2_part2_Hmwk_JavaModelClasse	C:\\Python-Forensics\\p-fish\\Te	49482	Mon Oct 2	Thu Aug 9	Thu Aug 9	8D3F877B	0	0	33206
5	favicon.ico	C:\\Python-Forensics\\p-fish\\Te	1150	Tue Jan 29	Thu Aug 9	Thu Aug 9	914E64581	0	0	33206
6	.gitignore	C:\\Python-Forensics\\p-fish\\Te	227	Thu Apr 26	Thu Aug 9	Thu Aug 9	60BFC1E2I	0	0	33206
7	auntmar3_amsb.sql	C:\\Python-Forensics\\p-fish\\Te	6174956	Thu Apr 26	Thu Aug 9	Thu Aug 9	5E1BA4334	0	0	33206
8	dump.sql	C:\\Python-Forensics\\p-fish\\Te	0	Thu Apr 26	Thu Aug 9	Thu Aug 9	CF83E1357	0	0	33206

Figure 4. fileSystemReport.csv

There will also be a log file created that serves as a neat little reference into various runtime analytics that occurred during the one-way hashing procedure. Data such as the number of files processed, the chosen hashing algorithm, the start and end time of the scan, and the basic specifications of the system performing the test are timestamped in this .txt file.

Figure 5 shows us the produced log file:



```
2018-08-09 20:00:53,089SHA512 hashing algorithm initiated..
2018-08-09 20:00:53,089
2018-08-09 20:00:53,089Welcome to p-fish version1.0....New Scan Started
2018-08-09 20:00:53,089
2018-08-09 20:00:53,089System: win32
2018-08-09 20:00:53,089Version: 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:22:17) [MSC v.1500 32 bit (Intel)]
2018-08-09 20:00:53,089Root Path: C:\\Python-Forensics\\p-fish\\Test_Directory\
2018-08-09 20:00:56,427Files Processed: 2387
2018-08-09 20:00:56,427Elapsed Time: 3.33700013161seconds
2018-08-09 20:00:56,427
2018-08-09 20:00:56,427Program Terminated Normally
2018-08-09 20:00:56,427
```

Figure 5. log file information

There are many areas in which file system hashing can prove to be useful. Evidence preservation is very important in the field of digital forensics, especially in the modern world where most communication and data distribution take place on electronic systems. It is vital for law enforcement to keep a valid and quantifiable chain of custody over digital evidence. The ability to compare hash digests of, for example an SSD belonging to a suspect, at the start of an investigation and upon presentation of evidence to a jury is crucial to the success of the respective party. Law enforcement could also search devices for a targeted hash value by hashing all the files within the root directory downwards (as our pfish program does) to see if the culprit had access to the target files. Blacklisting files as malicious code, cyber weapon files, classified or proprietary documents also serves its purpose in quickly pinpointing accountability. Whitelisting also benefits investigators by process of weeding out commonly cleared files such as OS or application executables from an investigation, making the pool of questionable files significantly smaller.

Psearch

The Python search, or Psearch program is broken down into two separate files, psearch.py & _psearch.py. As with the pfish program, _psearch.py performs the heavy lifting by containing all the support functions necessary for a successful run of the program. The purpose of this program is to provide a keyword search of a given file for a list of target keywords. We start this procedure off by reading in a file in binary mode and storing it in a binary array. Then we scan through the byte array once and replace any non-alpha characters with a '0'. The next step is the actual search of groups of alpha characters that fit the criteria of our minimum and maximum constant values, which we place into a set. The goal is to find a match between the word in the set and a word in our file of search words. A few of the additional functionalities I added to this program were the ability to search for words with disregard to suffixes preceding them, as well as an approximation search.

I will demonstrate the various ways to run the program and following that, provide some interpretation on how the results compared to each other.

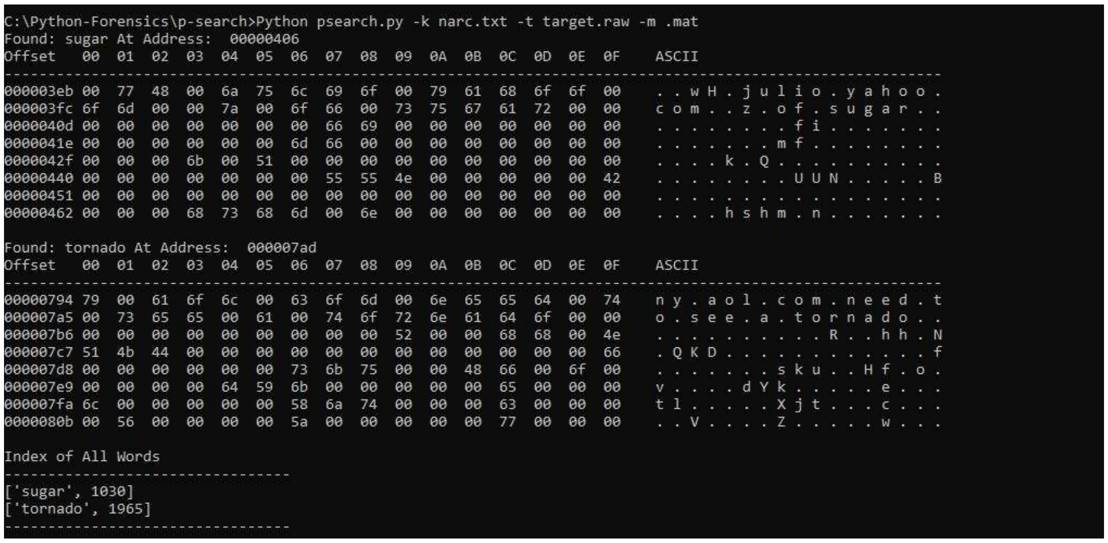
To run this program, you need to provide three mandatory arguments:

- ❖ File containing our search words (-k *or* --keyWords *followed by* "*file with keywords*")
- ❖ Target file to search (-t *or* --srchTarget *followed by* "*file to search*")
- ❖ Weighted matrix file (-m *or* --theMatrix *followed by* "*weighted matrix file*")

As well as two optional arguments:

- ❖ Show progress messages (-v, *or* --verbose)
- ❖ Activate suffix search (-s *or* --searchWordsSuffix)

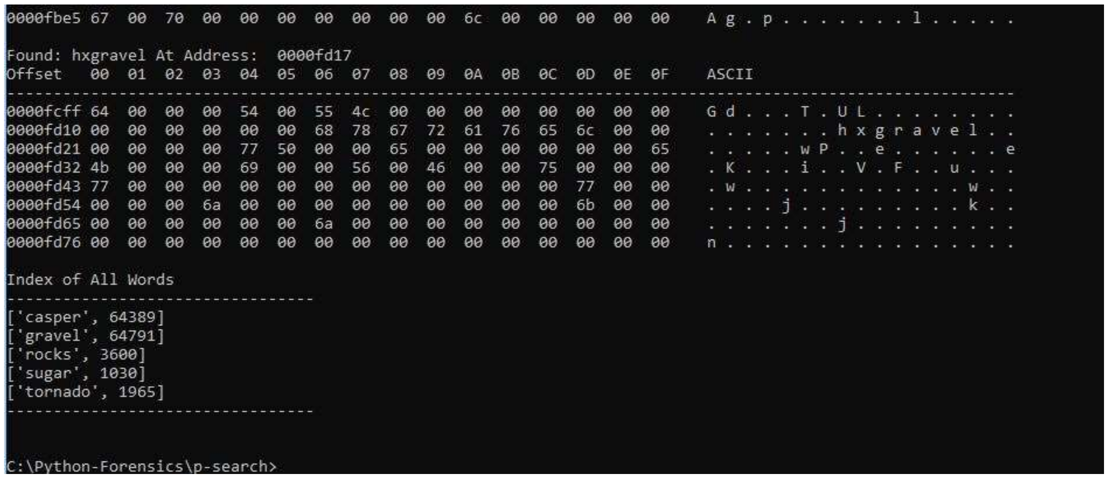
Here is a simple run of the file without the 'SearchWordsSuffix' function activation:



```
C:\Python-Forensics\p-search>Python psearch.py -k narc.txt -t target.raw -m .mat
Found: sugar At Address:  00000406
Offset   00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F    ASCII
----------------------------------------------------------------------------------------------------------------
000003eb 00  77  48  00  6a  75  6c  69  6f  00  79  61  68  6f  6f  00    . . w H . j u l i o . y a h o o .
000003fc 6f  6d  00  00  7a  00  6f  66  00  73  75  67  61  72  00  00    c o m . . z . o f . s u g a r . .
0000040d 00  00  00  00  00  00  00  66  69  00  00  00  00  00  00  00    . . . . . . . . f i . . . . . . .
0000041e 00  00  00  00  00  00  6d  66  00  00  00  00  00  00  00  00    . . . . . . . m f . . . . . . . .
0000042f 00  00  00  6b  00  51  00  00  00  00  00  00  00  00  00  00    . . . . k . Q . . . . . . . . . .
00000440 00  00  00  00  00  00  00  55  55  4e  00  00  00  00  00  42    . . . . . . . . U U N . . . . . B
00000451 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00    . . . . . . . . . . . . . . . . .
00000462 00  00  00  68  73  68  6d  00  6e  00  00  00  00  00  00  00    . . . . h s h m . n . . . . . . .

Found: tornado At Address:  000007ad
Offset   00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F    ASCII
----------------------------------------------------------------------------------------------------------------
00000794 79  00  61  6f  6c  00  63  6f  6d  00  6e  65  65  64  00  74    n y . a o l . c o m . n e e d . t
000007a5 00  73  65  65  00  61  00  74  6f  72  6e  61  64  6f  00  00    o . s e e . a . t o r n a d o . .
000007b6 00  00  00  00  00  00  00  00  00  52  00  00  68  68  00  4e    . . . . . . . . . . . R . . h h . N
000007c7 51  4b  44  00  00  00  00  00  00  00  00  00  00  00  00  66    . Q K D . . . . . . . . . . . . . f
000007d8 00  00  00  00  00  00  73  6b  75  00  00  48  66  00  6f  00    . . . . . . . . s k u . . H f . o .
000007e9 00  00  00  00  64  59  6b  00  00  00  00  00  65  00  00  00    v . . . . d Y k . . . . . . e . . .
000007fa 6c  00  00  00  00  00  58  6a  74  00  00  00  63  00  00  00    t l . . . . . X j t . . . . c . . .
0000080b 00  56  00  00  00  00  5a  00  00  00  00  00  77  00  00  00    . . V . . . . . Z . . . . . w . . .

Index of All Words
---------------------------------
['sugar', 1030]
['tornado', 1965]
---------------------------------
```

Figure 6. Test run on narc.txt using target.raw

Here are the same files tested using the suffix search algorithm. You can see that there are more words found by using this method:



```
0000fbe5 67  00  70  00  00  00  00  00  00  00  6c  00  00  00  00  00    A g . p . . . . . . . l . . . . .

Found: hxgravel At Address:  0000fd17
Offset   00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F    ASCII
------------------------------------------------------------------------------------------------------------
0000fcff 64  00  00  00  54  00  55  4c  00  00  00  00  00  00  00  00    G d . . . T . U L . . . . . . . .
0000fd10 00  00  00  00  00  00  68  78  67  72  61  76  65  6c  00  00    . . . . . . . h x g r a v e l . .
0000fd21 00  00  00  00  77  50  00  00  65  00  00  00  00  00  00  65    . . . . . w P . . e . . . . . . e
0000fd32 4b  00  00  00  69  00  00  56  00  46  00  00  75  00  00  00    . K . . . i . . V . F . . u . . .
0000fd43 77  00  00  00  00  00  00  00  00  00  00  00  00  77  00  00    . w . . . . . . . . . . . . w . .
0000fd54 00  00  00  6a  00  00  00  00  00  00  00  00  00  6b  00  00    . . . . j . . . . . . . . . k . .
0000fd65 00  00  00  00  00  00  6a  00  00  00  00  00  00  00  00  00    . . . . . . . j . . . . . . . . .
0000fd76 00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00    n . . . . . . . . . . . . . . . .

Index of All Words
---------------------------------
['casper', 64389]
['gravel', 64791]
['rocks', 3600]
['sugar', 1030]
['tornado', 1965]
---------------------------------


C:\Python-Forensics\p-search>
```

Figure 7. Same test files, but using suffix search alg.

One of the most obvious result differences between running the program's default 'SearchWords' function and the newly incorporated 'SearchWordsSuffix' function was the quantity of words that were found within the target file that was scanned. It makes complete sense that scanning the target file's alphabetic characters with the method of suffix searching yields more results than simply scanning the entire string of letters and hoping that a keyword coincidentally does not have some excess letters concatenated onto it.

Our PrintBuffer function provides us a nice visual layout that provides us with various information representing the name of a matched word within the target file and our keyword list, and location of the where the string in question was found. This can provide investigators with a concrete reference and factual representation of the match in question, which can be replicated if need be for investigative purposes. We also have the concept of approximate searching, which allows for a more open-ended approach to scanning the target file.

GPS

This Python project uses the PIL library to extract EXIF data (Exchangeable Image File Format) from image files and break apart the various pieces of the meta data into a neatly formatted CSV file that can be viewed in software such as Excel. To install the PIL library on your system, you simply need to use the pip installer as follows (example is using Win10 CMD prompt):

- pip install pillow

It is composed of multiple modules which are listed below:

GPSExtraction.py

-the main driver program that utilizes the various other modules for proper function

_modExif.py

-this module implements the PIL library and performs all PIL related tasks

_commandParser.py

-this is the location of the command line interface utility used within this Python software

_csvHandler.py

-this is our module for handling the output of our EXIF data dig placed into a neatly formatted CSV file

classLogging

-this module contains our _ForensicLog class that will be utilized for logging various program activities

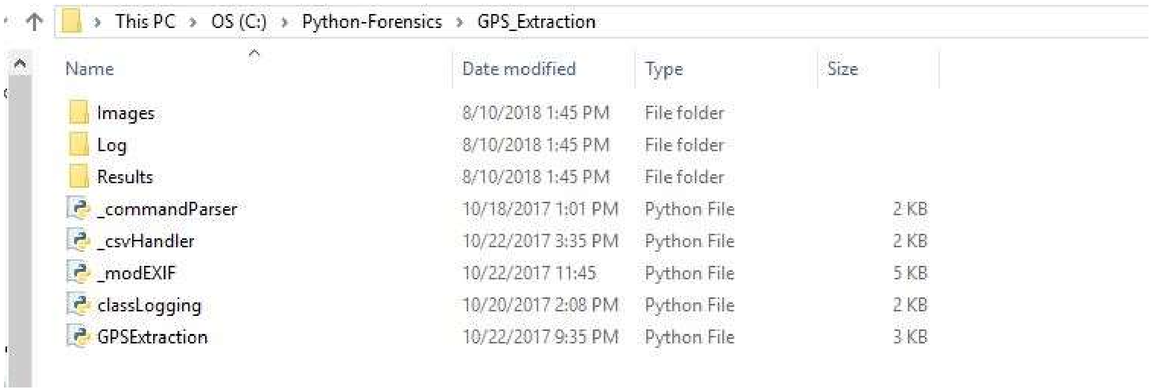
We will need to specify three required command line arguments for this program to work:

- Location of images to scan (-d)
- Destination for CSV file (-c)
- Destination for log file (-l)

To run this program using the Windows Command Prompt, the inputted string must contain all three arguments. For the sake of user readability, separating the results and logs into multiple directories is also highly recommended. The log file will provide insight into which of the images were properly processed (i.e. contained EXIF data in proper format for reading).

There is also a standalone Python file called "maliciousScript.py" which performs the task of falsifying EXIF GPS information data within an image file. This includes producing randomly generated coordinates for latitude and longitude, as well as a randomly generated time of production.

Figure 8 shows a recommended directory setup for this project:

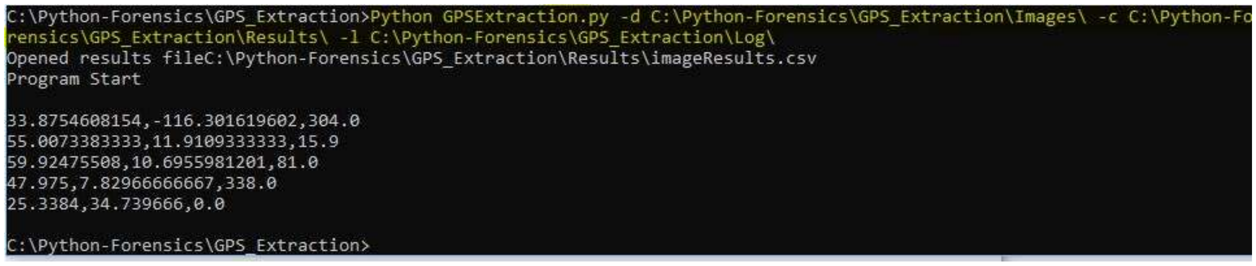


This PC › OS (C:) › Python-Forensics › GPS_Extraction

Name	Date modified	Type	Size
Images	8/10/2018 1:45 PM	File folder	
Log	8/10/2018 1:45 PM	File folder	
Results	8/10/2018 1:45 PM	File folder	
_commandParser	10/18/2017 1:01 PM	Python File	2 KB
_csvHandler	10/22/2017 3:35 PM	Python File	2 KB
_modEXIF	10/22/2017 11:45	Python File	5 KB
classLogging	10/20/2017 2:08 PM	Python File	2 KB
GPSExtraction	10/22/2017 9:35 PM	Python File	3 KB

Figure 8. Directory setup example

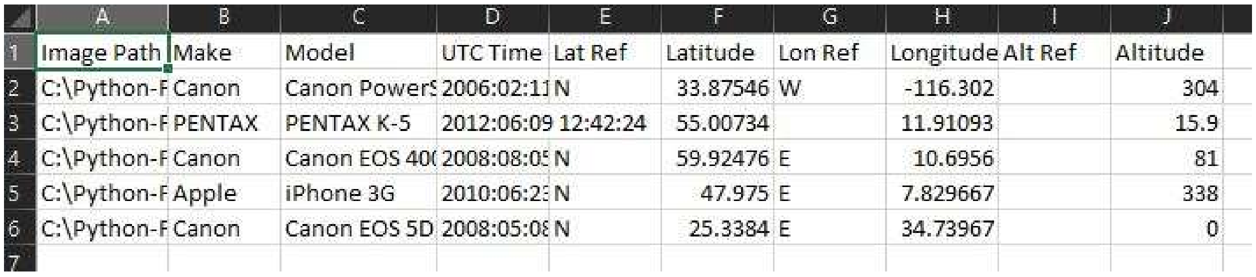
Figure 9 shows an example format of an input string using Windows CMD prompt:



```
C:\Python-Forensics\GPS_Extraction>Python GPSExtraction.py -d C:\Python-Forensics\GPS_Extraction\Images\ -c C:\Python-Fo
rensics\GPS_Extraction\Results\ -l C:\Python-Forensics\GPS_Extraction\Log\
Opened results fileC:\Python-Forensics\GPS_Extraction\Results\imageResults.csv
Program Start

33.8754608154,-116.301619602,304.0
55.0073383333,11.9109333333,15.9
59.92475508,10.6955981201,81.0
47.975,7.82966666667,338.0
25.3384,34.739666,0.0

C:\Python-Forensics\GPS_Extraction>
```

Figure 9. CMD input example

Given various sample images (included in the GitHub repository), a test run of the program gives us the results shown in Figure 10:



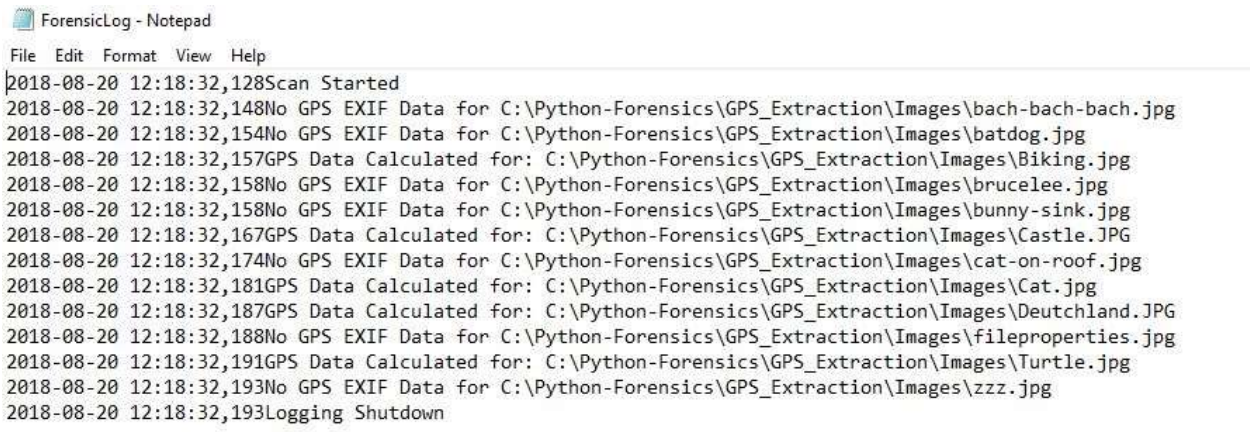
	A	B	C	D	E	F	G	H	I	J
1	Image Path	Make	Model	UTC Time	Lat Ref	Latitude	Lon Ref	Longitude	Alt Ref	Altitude
2	C:\Python-F	Canon	Canon PowerS	2006:02:11	N	33.87546	W	-116.302		304
3	C:\Python-F	PENTAX	PENTAX K-5	2012:06:09 12:42:24		55.00734		11.91093		15.9
4	C:\Python-F	Canon	Canon EOS 40(	2008:08:05	N	59.92476	E	10.6956		81
5	C:\Python-F	Apple	iPhone 3G	2010:06:23	N	47.975	E	7.829667		338
6	C:\Python-F	Canon	Canon EOS 5D	2008:05:08	N	25.3384	E	34.73967		0
7										

Figure 10. imageResults.csv

The data resulting from our directory scan tells us the location of the image in storage, the make and model of the device used to take the picture, time at which the image was taken, and the location where the image was taken.

We can see that although there are twelve images present in our \Images\ directory, not all of them contained EXIF data for extraction. Out of the twelve provided images, only five were successful inquiries.

Figure 11 is an example of the log file which provides us information regarding which images contained valid EXIF data tags, in addition to the start and send times for the scan:



```
2018-08-20 12:18:32,128Scan Started
2018-08-20 12:18:32,148No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\bach-bach-bach.jpg
2018-08-20 12:18:32,154No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\batdog.jpg
2018-08-20 12:18:32,157GPS Data Calculated for: C:\Python-Forensics\GPS_Extraction\Images\Biking.jpg
2018-08-20 12:18:32,158No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\brucelee.jpg
2018-08-20 12:18:32,158No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\bunny-sink.jpg
2018-08-20 12:18:32,167GPS Data Calculated for: C:\Python-Forensics\GPS_Extraction\Images\Castle.JPG
2018-08-20 12:18:32,174No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\cat-on-roof.jpg
2018-08-20 12:18:32,181GPS Data Calculated for: C:\Python-Forensics\GPS_Extraction\Images\Cat.jpg
2018-08-20 12:18:32,187GPS Data Calculated for: C:\Python-Forensics\GPS_Extraction\Images\Deutchland.JPG
2018-08-20 12:18:32,188No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\fileproperties.jpg
2018-08-20 12:18:32,191GPS Data Calculated for: C:\Python-Forensics\GPS_Extraction\Images\Turtle.jpg
2018-08-20 12:18:32,193No GPS EXIF Data for C:\Python-Forensics\GPS_Extraction\Images\zzz.jpg
2018-08-20 12:18:32,193Logging Shutdown
```

Figure 11. Log file showing only 5 EXIF hits

The importance of knowing where and when a picture was taken has become increasingly popular over the years. From law enforcement insisting on knowing the relative geographical position of a 911 caller to software companies wanting to gather information on pictures to better direct their applications, there has been a strong push for smart phone companies to include state of the art GPS systems and high resolution in their products. Privacy it seems is slowly taking the backseat in the hierarchy of tech innovation, for better or worse.

The ability to use image's location and time of conception can be an extremely useful for digital forensic investigators. Whether it be to validate or disprove an suspect's alibi or put together a provable timeframe in a case, this type of data could become invaluable.

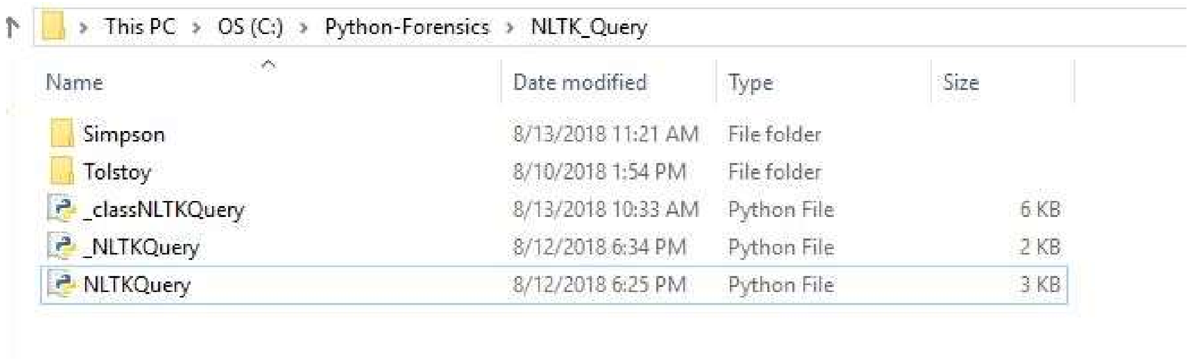
NLTK Query

This program utilizes the Natural Language Toolkit (NLTK) library to perform various diagnostic evaluations of a selected corpus, or large volume of linguistic samples. The ability to read in thousands of pages of text in a matter of seconds and then be able to generate word occurrences, concordances, similarities, parts-of-speech labels, and collocations within the text can prove to be very insightful to investigators trying to understand a piece of work. Rather than spending hours sifting through, for example, all the courtroom notes for a case, an investigator could simply press a few buttons and answer many questions he or she may have regarding the mood, style, or conversation that took place.

This natural language processing project is broken down into three separate files, NLTKQuery.py, _NLTKQuery.py, and _classNLTKQuery.py. Within our main file, NLTKQuery.py, we create a natural language processing object from our class file that performs input validation, creation of the corpus, tokenization, stop-word removal, stemming, and parts-of-speech tagging. A menu with fifteen selectable options, each offering a different analytical or functional procedure, is presented to the user during each iteration. The file _NLTKQuery.py only contains two simple functions, 'printMenu' & 'getUserSelection'. One important thing to note is that of case-sensitivity. For example, if you are tokenizing a corpus that is in all upper-case, you would need to input an upper-case word to conduct a proper search.

I have included two separate test corpuses within the repository, although you could run this analysis on any piece of work you provide. One is a composition of courtroom log files from the OJ Simpson case, and the other is an amalgam of works from the famous Russian author, Leo Tolstoy. There are many websites out there that provide free linguistic samples for you to use in your testing. I would highly recommend choosing an author that really interests you, and then running the various functions on his/her work to intricately see the style within they write, what types of words they love to use, and the overall mood that surrounds their stories.

Figure 12 is an example of a possible directory setup:



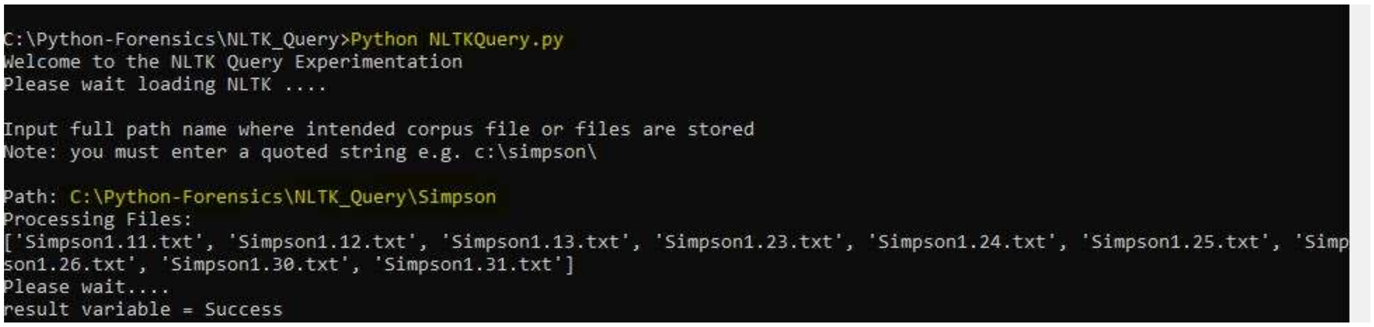
This PC > OS (C:) > Python-Forensics > NLTK_Query

Name	Date modified	Type	Size
Simpson	8/13/2018 11:21 AM	File folder	
Tolstoy	8/10/2018 1:54 PM	File folder	
_classNLTKQuery	8/13/2018 10:33 AM	Python File	6 KB
_NLTKQuery	8/12/2018 6:34 PM	Python File	2 KB
NLTKQuery	8/12/2018 6:25 PM	Python File	3 KB

Figure 12. Recommended directory setup

Depending on the size of the corpus you are attempting to work with, the processes of extracting tokens and other pre-evaluation steps may take a few minutes, depending on the computer processing power you are working with. There is a lot of information being processed behind the scenes, but once the lists and sets are built, the analyzation is remarkably fast. I will show an example of running the program from the CMD prompt in Windows 10.

Running NLTK Query with a corpus built from various OJ Simpson courtroom logs:

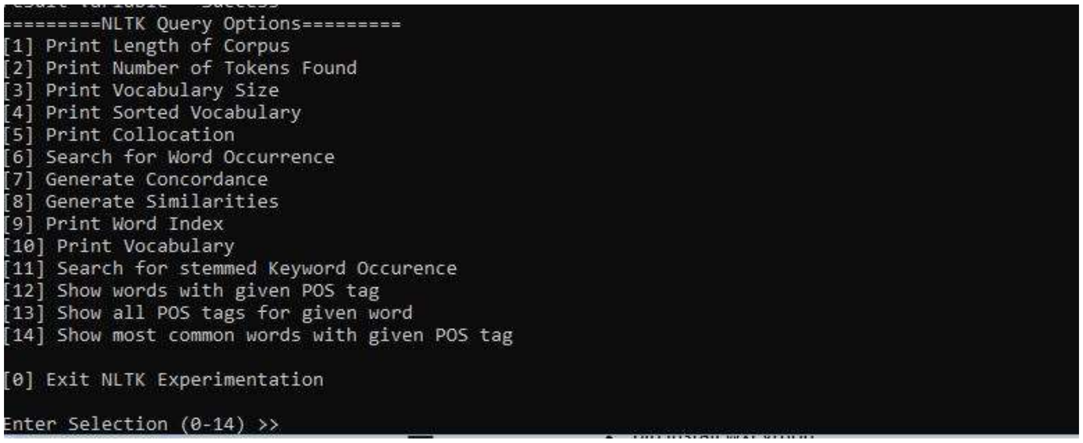


```
C:\Python-Forensics\NLTK_Query>Python NLTKQuery.py
Welcome to the NLTK Query Experimentation
Please wait loading NLTK ....

Input full path name where intended corpus file or files are stored
Note: you must enter a quoted string e.g. c:\simpson\

Path: C:\Python-Forensics\NLTK_Query\Simpson
Processing Files:
['Simpson1.11.txt', 'Simpson1.12.txt', 'Simpson1.13.txt', 'Simpson1.23.txt', 'Simpson1.24.txt', 'Simpson1.25.txt', 'Simp
son1.26.txt', 'Simpson1.30.txt', 'Simpson1.31.txt']
Please wait....
result variable = Success
```

Figure 13. CMD input example

As you can see from the prompt, the files being processed include nine separate days of courtroom discussion. I will run a variety of the menu options on the corpus to provide some examples of the analysis NLTK_Query.py could do in a fraction of the time it would take a room full of eyes to do the same.

View of the menu:

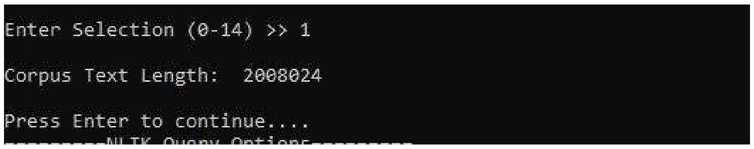


```
=========NLTK Query Options=========
[1] Print Length of Corpus
[2] Print Number of Tokens Found
[3] Print Vocabulary Size
[4] Print Sorted Vocabulary
[5] Print Collocation
[6] Search for Word Occurrence
[7] Generate Concordance
[8] Generate Similarities
[9] Print Word Index
[10] Print Vocabulary
[11] Search for stemmed Keyword Occurence
[12] Show words with given POS tag
[13] Show all POS tags for given word
[14] Show most common words with given POS tag

[0] Exit NLTK Experimentation

Enter Selection (0-14) >>
```

Figure 14. Menu printout

Many of these options provide simple statistics about the corpus in question. Others perform more extensive analysis, such as collocation, concordance, and parts of speech tagging.

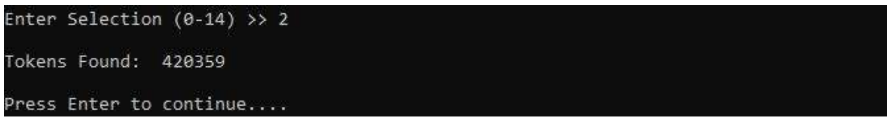
Length of Corpus:



```
Enter Selection (0-14) >> 1

Corpus Text Length:  2008024

Press Enter to continue....
```

Figure 15. Menu option 1

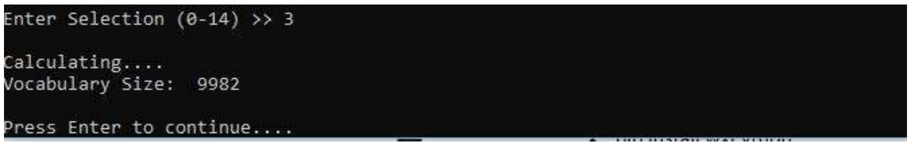
Token count:



```
Enter Selection (0-14) >> 2

Tokens Found:  420359

Press Enter to continue....
```

Figure 16. Menu option 2

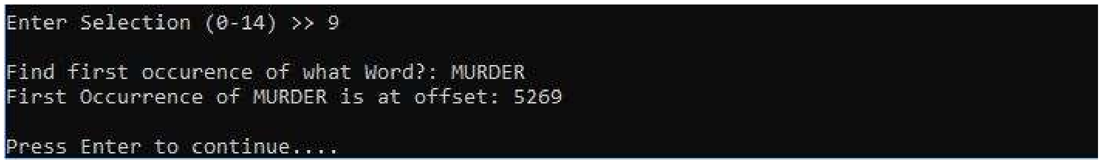
Vocabulary size:



```
Enter Selection (0-14) >> 3

Calculating....
Vocabulary Size:  9982

Press Enter to continue....
```

Figure 17. Menu option 3

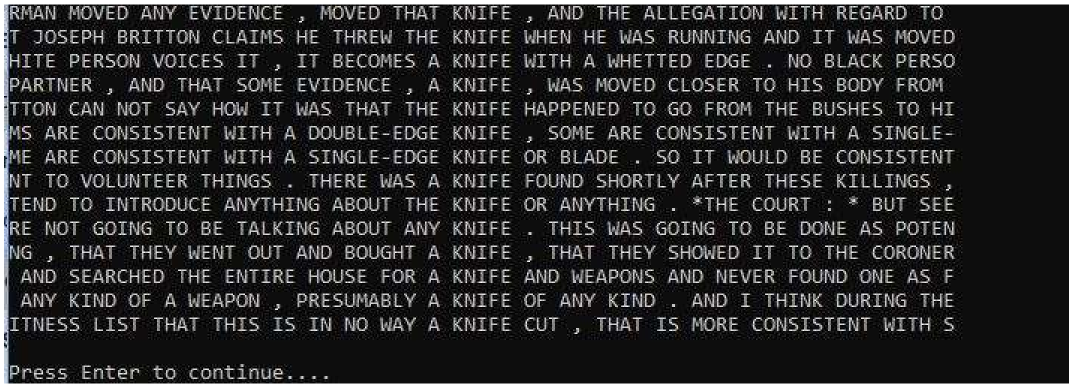
Word index (using the word “MURDER” for this example):



```
Enter Selection (0-14) >> 9

Find first occurence of what Word?: MURDER
First Occurrence of MURDER is at offset: 5269

Press Enter to continue....
```

Figure 18. Menu option 9

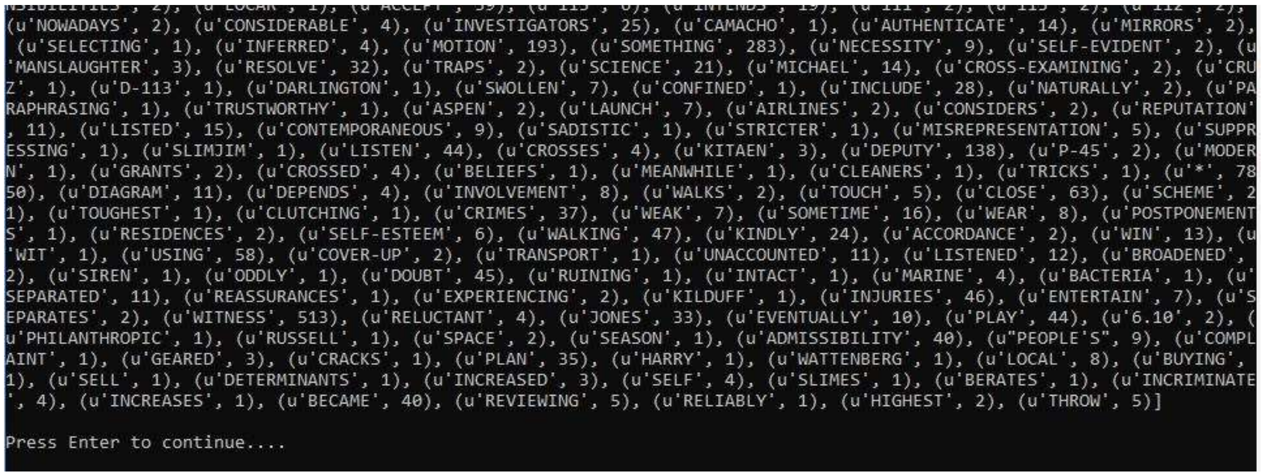
Word Concordance (word in question is “KNIFE”), which shows us the common instances in which the word was used to provide a contextual reference:



```
RMAN MOVED ANY EVIDENCE , MOVED THAT KNIFE , AND THE ALLEGATION WITH REGARD TO
T JOSEPH BRITTON CLAIMS HE THREW THE KNIFE WHEN HE WAS RUNNING AND IT WAS MOVED
HITE PERSON VOICES IT , IT BECOMES A KNIFE WITH A WHETTED EDGE . NO BLACK PERSO
PARTNER , AND THAT SOME EVIDENCE , A KNIFE , WAS MOVED CLOSER TO HIS BODY FROM
TTON CAN NOT SAY HOW IT WAS THAT THE KNIFE HAPPENED TO GO FROM THE BUSHES TO HI
MS ARE CONSISTENT WITH A DOUBLE-EDGE KNIFE , SOME ARE CONSISTENT WITH A SINGLE-
ME ARE CONSISTENT WITH A SINGLE-EDGE KNIFE OR BLADE . SO IT WOULD BE CONSISTENT
NT TO VOLUNTEER THINGS . THERE WAS A KNIFE FOUND SHORTLY AFTER THESE KILLINGS ,
TEND TO INTRODUCE ANYTHING ABOUT THE KNIFE OR ANYTHING . *THE COURT : * BUT SEE
RE NOT GOING TO BE TALKING ABOUT ANY KNIFE . THIS WAS GOING TO BE DONE AS POTEN
NG , THAT THEY WENT OUT AND BOUGHT A KNIFE , THAT THEY SHOWED IT TO THE CORONER
 AND SEARCHED THE ENTIRE HOUSE FOR A KNIFE AND WEAPONS AND NEVER FOUND ONE AS F
 ANY KIND OF A WEAPON , PRESUMABLY A KNIFE OF ANY KIND . AND I THINK DURING THE
ITNESS LIST THAT THIS IS IN NO WAY A KNIFE CUT , THAT IS MORE CONSISTENT WITH S

Press Enter to continue....
```

Figure 19. Menu option 7

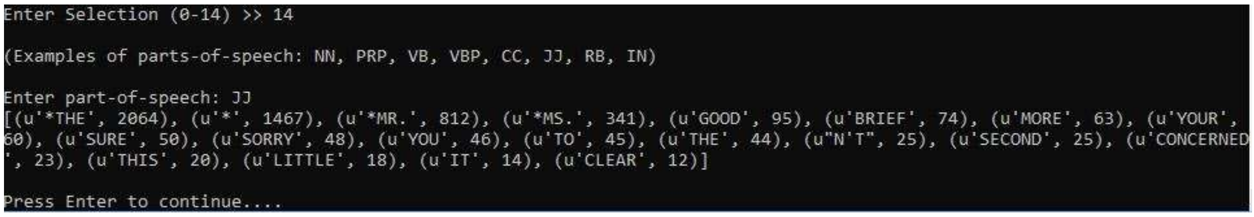
Print Vocabulary, which prints an entire list of all words used throughout the corpus with their respective counts:



```
(u'NOWADAYS', 2), (u'CONSIDERABLE', 4), (u'INVESTIGATORS', 25), (u'CAMACHO', 1), (u'AUTHENTICATE', 14), (u'MIRRORS', 2),
 (u'SELECTING', 1), (u'INFERRED', 4), (u'MOTION', 193), (u'SOMETHING', 283), (u'NECESSITY', 9), (u'SELF-EVIDENT', 2), (u
'MANSLAUGHTER', 3), (u'RESOLVE', 32), (u'TRAPS', 2), (u'SCIENCE', 21), (u'MICHAEL', 14), (u'CROSS-EXAMINING', 2), (u'CRU
Z', 1), (u'D-113', 1), (u'DARLINGTON', 1), (u'SWOLLEN', 7), (u'CONFINED', 1), (u'INCLUDE', 28), (u'NATURALLY', 2), (u'PA
RAPHRASING', 1), (u'TRUSTWORTHY', 1), (u'ASPEN', 2), (u'LAUNCH', 7), (u'AIRLINES', 2), (u'CONSIDERS', 2), (u'REPUTATION'
, 11), (u'LISTED', 15), (u'CONTEMPORANEOUS', 9), (u'SADISTIC', 1), (u'STRICTER', 1), (u'MISREPRESENTATION', 5), (u'SUPPR
ESSING', 1), (u'SLIMJIM', 1), (u'LISTEN', 44), (u'CROSSES', 4), (u'KITAEN', 3), (u'DEPUTY', 138), (u'P-45', 2), (u'MODER
N', 1), (u'GRANTS', 2), (u'CROSSED', 4), (u'BELIEFS', 1), (u'MEANWHILE', 1), (u'CLEANERS', 1), (u'TRICKS', 1), (u'*', 78
50), (u'DIAGRAM', 11), (u'DEPENDS', 4), (u'INVOLVEMENT', 8), (u'WALKS', 2), (u'TOUCH', 5), (u'CLOSE', 63), (u'SCHEME', 2
1), (u'TOUGHEST', 1), (u'CLUTCHING', 1), (u'CRIMES', 37), (u'WEAK', 7), (u'SOMETIME', 16), (u'WEAR', 8), (u'POSTPONEMENT
S', 1), (u'RESIDENCES', 2), (u'SELF-ESTEEM', 6), (u'WALKING', 47), (u'KINDLY', 24), (u'ACCORDANCE', 2), (u'WIN', 13), (u
'WIT', 1), (u'USING', 58), (u'COVER-UP', 2), (u'TRANSPORT', 1), (u'UNACCOUNTED', 11), (u'LISTENED', 12), (u'BROADENED', 
2), (u'SIREN', 1), (u'ODDLY', 1), (u'DOUBT', 45), (u'RUINING', 1), (u'INTACT', 1), (u'MARINE', 4), (u'BACTERIA', 1), (u'
SEPARATED', 11), (u'REASSURANCES', 1), (u'EXPERIENCING', 2), (u'KILDUFF', 1), (u'INJURIES', 46), (u'ENTERTAIN', 7), (u'S
EPARATES', 2), (u'WITNESS', 513), (u'RELUCTANT', 4), (u'JONES', 33), (u'EVENTUALLY', 10), (u'PLAY', 44), (u'6.10', 2), (
u'PHILANTHROPIC', 1), (u'RUSSELL', 1), (u'SPACE', 2), (u'SEASON', 1), (u'ADMISSIBILITY', 40), (u"PEOPLE'S", 9), (u'COMPL
AINT', 1), (u'GEARED', 3), (u'CRACKS', 1), (u'PLAN', 35), (u'HARRY', 1), (u'WATTENBERG', 1), (u'LOCAL', 8), (u'BUYING', 
1), (u'SELL', 1), (u'DETERMINANTS', 1), (u'INCREASED', 3), (u'SELF', 4), (u'SLIMES', 1), (u'BERATES', 1), (u'INCRIMINATE
', 4), (u'INCREASES', 1), (u'BECAME', 40), (u'REVIEWING', 5), (u'RELIABLY', 1), (u'HIGHEST', 2), (u'THROW', 5)]

Press Enter to continue....
```

Figure 10. Menu option 10

Parts of Speech analysis, in this case we are showing the most common adjectives used throughout the corpus:



```
Enter Selection (0-14) >> 14

(Examples of parts-of-speech: NN, PRP, VB, VBP, CC, JJ, RB, IN)

Enter part-of-speech: JJ
[(u'*THE', 2064), (u'*', 1467), (u'*MR.', 812), (u'*MS.', 341), (u'GOOD', 95), (u'BRIEF', 74), (u'MORE', 63), (u'YOUR', 
60), (u'SURE', 50), (u'SORRY', 48), (u'YOU', 46), (u'TO', 45), (u'THE', 44), (u"N'T", 25), (u'SECOND', 25), (u'CONCERNED
', 23), (u'THIS', 20), (u'LITTLE', 18), (u'IT', 14), (u'CLEAR', 12)]

Press Enter to continue....
```

Figure 11. Menu option 14

This type of language processing could be extended to all types of works. Whether for investigative purposes by examining a suspect’s writing style and intent in questionable documents, or for scholarly reasons by examining a past author’s entire corpus of work, this type of language processing could prove to be very useful.

Natural Language Processing is also a very interesting area of study in that it is an important pillar in the development of dialog-based systems. This idea can also be extended to the realm of artificial intelligence. For a computer program to be able to not only store information that it reads directly, but to gather enough data to produce general themes and even emotions behind a script is quite the fascinating concept.

Scanners

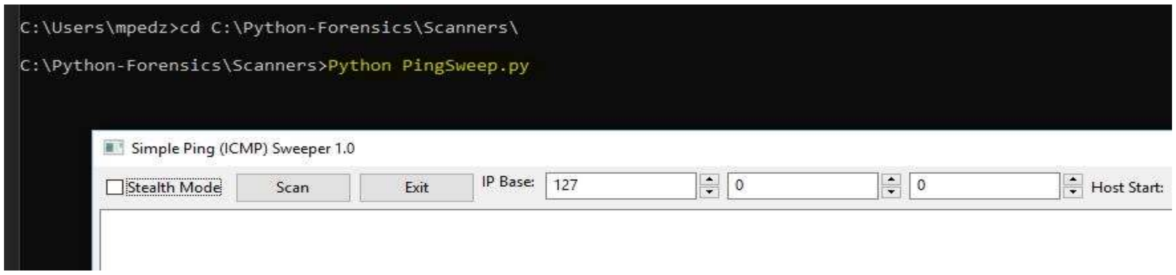
This dual set of programs pertains to investigating modern network environments. PingSweep.py & PortScanner.py perform analysis on a network, gauging available IP addresses and scanning targeted IPs for open and available ports, respectively. You will need to install the Python GUI library, 'wxPython', and the pure Python ping implementation using raw sockets library, 'pyping', for these programs to work. Both programs are in the form of a graphics user interface. Do note that you may need to maximize the application to view the entire menu bar that's stationed at the top. Also, depending on how large of a sample size you are scanning, the application may appear to be "Not Responding", during which you should give the program a moment to finish scanning and ultimately produce the results in one display.

In Win10 CMD prompt, you can install these libraries by simply inputting the commands:

- pip install pyping & pip install wxPython

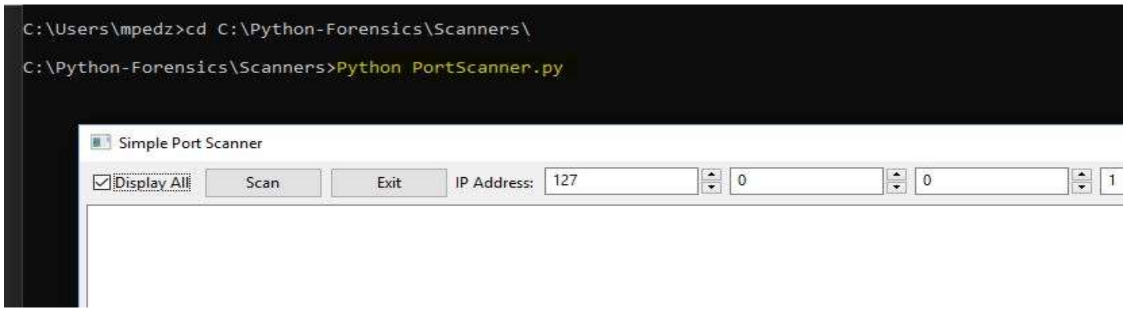
The PingSweep.py program simply pings a range of IP addresses and relays and successful hits to the user. There is also a 'Stealth Mode' checkbox towards the top right of the application, which if selected, will force a random duration timeout() delay during pings to appear less obvious during its operation.

The PortScanner.py program can be considered an extension of the PingSweep.py program, in that it provides insight into open ports on an available IP address. You simply input the desired IP address you wish to scan and proceed to enter which range of ports you would like to test for an 'open' status.



```
C:\Users\mpedz>cd C:\Python-Forensics\Scanners\

C:\Python-Forensics\Scanners>Python PingSweep.py
```

Figure 12. PingSweep.py CMD command & application interface



```
C:\Users\mpedz>cd C:\Python-Forensics\Scanners\

C:\Python-Forensics\Scanners>Python PortScanner.py
```

Figure 13. PortScanner.py CMD command & application interface

This Python-Forensics toolkit should provide insight into how a computer program could accelerate an investigative procedure, while also removing the variable of human error by providing elaborate testing that is replicable multiple times over. Each of the functionalities provided by the programs in this toolkit- file system hashing, keyword search, EXIF data extraction and manipulation, natural language processing, and network forensic analysis- are building blocks that can be enhanced or shaped to fit the purposes of most digital forensic endeavors.

Something important to note is that much of what these programs can do can also be found in various software that professionals in these types of fields already use. The ability to understand the fundamental concepts that drive these simple programs will prove to be very helpful when trying to write your own customized forensic software in the future. If not for professional reasons, I would highly recommend playing around with some of these programs to better understand how they work. One of the greatest parts about this toolkit is that basically every program within it is highly customizable. The straightforward modularization of the programs allows for easy insertion of code into various stages of the programs' core functionality.

Thank you for joining me on this journey within this small piece of a vast and growing digital forensics world. I welcome any questions or suggestions that my fellow readers may have. A final note that I would like to add would be to recommend to anyone even remotely interested in Python programming and/or digital forensics Chet Hosmer's book, "Python Forensics, A Workbench for Inventing and Sharing Digital Forensic Technology". As a senior in college, I came across this book and immediately noticed how well it assisted in my understanding of how automation and investigation could come together to break boundaries of what I thought was capable in digital forensic studies.