Manchester Metropolitan University

# BLOCK BASED SIGNATURE DETECTION FOR FORENSIC TRIAGE

**ETHICS APPROVAL NUMBER: 13311**

**Project (6G6Z1101_1920_9Z6)**

Karan Nihalani - 17023122

## Acknowledgements

I would like to thank Dr. Robert Hegarty and Dr. Syed Aziz Shah for their aid during this project.

## Abstract

This project portrays the intention to explore further into Block Based Signature detection within the Forensic Triage and dives further into the explanation and analysis of different types of signature detection needed in a digital forensics investigation. An examination is carried out on a large dataset consisting of over 3,800 image files, with the goal of comparing the efficiency between scanning entire file hashes and scanning only the hashes of the 2nd block of all these image files. When analysing the hashes, for the purpose of visualising the results output, the duration for the completion of all file hashes and all 2nd block hashes are timed and files are also read for matching hashes, to prevent the lack of integrity in the dataset. The results obtained from the analysis process show that the comparisons carried out are confirmed to be just as predicted, accomplishing its objective. These will be useful when carrying out digital investigations in the professional world, when scanning file blocks in large datasets for corrupted or malicious data, therefore, fulfilling its purpose in saving time while studying criminal evidence.

## Contents

# 1. Introduction

## 1.1 Background

The use of technology keeps on growing exponentially and as this happens, an increase in cybercrimes involving digital devices are performed and the employment of digital forensics in solving such cases are demonstrated to be crucial, especially when it features within the digital forensic triage process. The digital forensic triage consists of all the elements necessary in carrying out an effective and complete investigation, from collecting and analysing digital evidence to prioritizing the evidence itself and within the assembly and analysis of metadata there are a multitude of tools and techniques carried out or applied, depending on the type of data the investigator can work with.

One of the methodologies digital forensic analysts and investigators make use of is known as signature detection or signature-based detection. Looking at its definition, signature detection is referred to as “an approach which considers attack patterns as signatures and further compares signatures of known attacks to incoming attacks for detection” (IGI Global, 2015), suggesting the disclosure and identification of malicious data or activity via the comparison of various signatures found in digital evidence metadata. The benefit of this technique is mainly the accuracy for decreasing the rate of false positives, where false positives suggest the “error of accepting an alternative hypothesis (the real hypothesis of interest) when the results can be attributed to chance”, in other words, the “error of rejecting a null hypothesis when it is actually true” (UC Berkeley, 2017).

Within the signature-based methods used in industries’ forensic triage process, there are two types which stand out the most, file-based signature detection and block-based signature detection. A simple analogy of both would be that file signature detection is utilised to perform the comparison of signatures of the entire file itself, as the name makes it clear, whereas block-based signature detection performs the analysis on the specific blocks within the file system, where these are “stored as separate pieces, each with a unique identifier” (IBM, 2019).

Although signature detection in general is known to be flawed in only detecting known attacks and eventually resulting in obligatory and necessary updates, it also can consume too much time if a live forensic investigation is taking place. This is very important as volatile data is at risk of being lost and the running services in a computer “run at a much higher priority than processes, and many users are unaware that these services actually exist” and because of “their high priority and lack of attention by the typical end user, they are a common target for hackers” (ELSEVIER SciTech Connect, 2013). This program will be attempting to challenge the idea that block-based signature detection compared to file-based signature detection is determined to be less time-consuming and equally as efficient if it were to be applied within the current digital forensic triage. The final product will be a program that proves this concept. The user will execute a python script at terminal-level in a Linux based O.S, within a directory with a large dataset of image files as their recovered digital evidence and once run, the user will be able to visualise the entire image dataset undergoing file-based signature detection first and proceeding this, the same dataset will undergo block-based signature detection, but only for the second block of the file. At the terminal window, the time taken for both processes will be demonstrated, along with the number of matches found within the dataset, to detect anomalies. This is shown to the user so that it will be easier for them to observe the difference in duration between the two methods, allowing them to draw a conclusion from their observation.

## 1.2 Aims

A1 - I will research how signature detection is employed during forensic triage.

A2 - Identify the key requirements of the triage process.

A3 - Evaluate different approaches to signature detection (block and file based)

A4 - Recommend the most appropriate approach based on the results of my evaluation.

## 1.3 Objectives

O1 - Carry out research and read about signature detection and its uses in today's world.

O2 - Explain key concepts and different types of signatures that can be detected in the early stages of an investigation.

O3 - Design an experiment to compare block and file based.

O4 - Identify appropriate software libraries to develop the experiment/software

O5 - Identify the requirements for signature detection in digital forensics

O6 - Design an experiment to evaluate and compare file and block-based approaches to signature detection for digital forensic triage.

O7 - To able to implement the appropriate and correct software to be able to conduct my experiment.

O8 - Test the software to ensure it functions correctly

O9 - Evaluate your software using a standardised data set, perform comparison between block and file-based approaches to demonstrate the benefits of block-based triage

O10 - To be able to evaluate my project against the aims and objectives presented above.

O11 - To document any future work I will have identified from working during this project.

## 1.4 Potential Issues

A potential issue I might face during this project would be an issue with the software libraries imported into my program's code. This could be since some libraries wouldn't be installed properly because of a network or system error, during the installation process. I could also face the issue where certain libraries might not work, as they would only be implemented on a certain version of Python.

Another potential issue I can face during the timeframe of the project, is an issue with hardware failure, where a problem would involve the constant requirement for a high amount of memory, which can cause the device to slow down and cause unwanted errors, e.g. lags, freezing pages and programs crashing. In addition, a device can be physically damaged anytime throughout the time the project is being completed, where broken screens or broken keyboards or internal circuit damage can result in a delay concerning the completion of project tasks.

Invasive attacks on the computing device can also be considered as a potential challenge within this project. The laptop can suffer some invasive attacks, like viruses, malware or hacking attacks. This can occur due to the unwanted downloads of unwanted files from the Internet.

## 1.5 Report Outline

The outline for this report is as seen below:

**Section 2 – Literature Review**

An insight on heavily related work on the subject, including examples of professional alternatives provided by analysts working in industry.

**Section 3 – Product Design**

The description of the components of the source code, including goals and explanation for libraries and functions used, utilising screenshots for better visualisation.

**Section 4 – Product Implementation**

A walkthrough on the development and presentation of the product making use of screenshots for better examination.

**Section 5 – Evaluation**

Using results from carrying out testing on the product, a reflection is made on the product, clarifying the significance of the results

**Section 6 – Conclusion**

An overview on the entire project, where strengths, weaknesses and potential additions are discussed. Any further work is also discussed.

# 2. Literature Review

## 2.1 Study Abstract

The purpose of this project is to create a product that is able to provide a comparison between two different types of signature-based detection techniques that would be involved during the forensic triage. This will be created in a Python-compatible editor, such as Bluefish on a Linux operating system and will be displayed through a terminal. Further research into other potential signature detection techniques for the forensic triage process will be looked into, as well as, the use of signature-based detection in the professional cybercriminal investigation scene.

## 2.2 Introduction

The use of signature-based detection techniques is currently widely made use of in the professional workplace and have been used since the past few years as a common technique to locate malicious data in a large dataset of files. Even though large corporations and security organisations such as, *McAffee*, *Symantec* and *TrendMicro* and others are able to provide signature detection in many of their security countermeasures and mitigation methodologies, if their signature database is not updated, the "IDS systems will fail to detect some of the intrusion attacks" (Ebrary.net), enhancing the importance of constant monitoring during an investigation for possible unknown threats.

The purpose of this literature review is to explore and understand a variety of different examples of signature-based detection techniques used in innovative products proposed by highly qualified forensic experts in the field and how signature detection is used to overcome various common cybersecurity problems to prevent more evidence and data loss during the forensic triage. The benefits and drawbacks of each of the products researched below are delved into to provide a critical analysis of the papers.

## 2.3 Related Work

### SlackStick

A live investigation is defined as the method to investigate or find data within devices that are powered on. It is made clear that the purpose for this is to "acquire volatile data that would otherwise get lost if the computer system is turned off" (Council of Europe, 2013) or overwritten. Said volatile data consists of data that is stored in memory (RAM), cache and registries and can be lost once the device is turned off, whereas we observe the opposite with non-volatile data, as the data would remain in the computer, once the device is turned back on.

The authors of this study highlight the events that take place during the search and seizure in a live forensic investigation, from the fact that digital devices need to be turned off before being taken into labs for further research and study, after carrying out research at the scene, to its purpose: to be able to retrieve relevant volatile data that can be lost, once device is turned off, for example, open ports and running applications or software. Due to these factors, there are various disadvantages present in live investigations, which include the fact that the forensic techniques, such as file hashing used in labs, which would not be able to be employed at the scene of the crime because of the amount of time needed by the forensics software for the preparation of the evidence to be analysed and hashed can be excessive.

In this study, we are presented with a solution to issues like these, called, '*SlackStick.*' *SlackStick* is designed to identify relevant files to the investigation or "non-volatile evidence triage during a live forensics investigation." (R. Hegarty, J. Haggerty, 2015). The functionality of the *SlackStick* software is expressed to be executed via an external device, like flash memory and instead of carrying out the entire search signature on the entire file and blocks of data, it can detect characteristics from the underlying file and carry out the search effectively. Nonetheless, five objectives and/or requirements will have to be considered. The first one being, a fast-speed analysis of a large amount of data, as during a forensic investigation, the size and amount of files can mount up to a couple of terabytes of data found on storage devices. Secondly, the "operating system and resource agnostics" would be necessary, as we would be dealing with a variety of different types of devices and software, where there wouldn't be much access to professional forensic applications. Further on, "minimal false positives and false negatives" are taken into consideration, due to the negative impact false positives bring during the triage stage of an investigation. Apart from this, false negatives can result in the missing of evidence during the ongoing inspection.  The fourth objective would be to prevent much user interaction, as not many "first responders" have a base knowledge on the forensic triage, so the goal would be to reduce human error. Finally, the last requirement would be "partial file detection," to highlight the importance of identifying file that have been deleted or even fragmented.

The respective software presents various advantages and somewhat minor disadvantages to the outcomes of this methodology. The various advantages include the idea that, as the software is inserted into the device via an external device, it "accesses persistent data stores in read-only mode" (R.Hegarty, J.Haggerty, 2015), therefore, the evidence in the computer is left untampered. This leads to the circumstance that investigators who lack experience, would be able to carry out file searches for certain data needed, reducing their time needed to learn complex forensics programs. Another advantage is that reliability and the software's perceptivity for harmful attacks are decreased, as multiple signatures for a single file of interest are permitted. One of the few disadvantages *SlackStick* presents is its live USB implementation, as simply put, it can or will weaken the software's performance.

## **Extrusion Detection of Illegal Files in Cloud-Based Systems**

When files, such as, media, documents and more are uploaded to and downloaded from a cloud-based system, perimeter-based defences like, firewalls and IDS (Intrusion Detection Systems) are utilized to prevent denial of service attacks, alongside protecting the virtual machines. Although these defences provide a mostly efficient protective barrier, it isn't able to protect cloud system users from other violations, such as "Intellectual property theft" and "cloud providers being held accountable for illegal file storage" (R. Hegarty, J. Haggerty, 2015).

*XDet* is presented as a solution to issues like these, which can be used beside the other security methodologies mentioned above for extrusion detection and to ensure a more robust and secure system. Extrusion detection is defined as "the opposite of an 'Intrusion Detection System' (IDS)" (PC Magazine Encyclopedia, 2019), where in an extrusion detection system, the software inspects whether an attack is performed by the computer.  *XDet* is going to be proven to be crucial to combat various security challenges brought forward by cloud-based software, mostly in handling the user's personal information. These include, making sure that all data that might be confidential is very well protected and isn't leaked, due to data being stored on the cloud, the user has more potential to be the victim of an attack.

The process *XDet* utilizes is conditional and is broken down into steps for our understanding. Firstly, the user or cloud provider selects what files are deemed as private. From the user's perspective, these files can include, sensitive photos and confidential data and from the cloud provider's point of view, these can include, stolen files, indecent images and other data that shouldn't be uploaded to the storage servers. If a file does not meet the criteria, a signature is created from the file, but instead of making use of the Transport Layer, as is done in IDS' and firewalls, this software involves the Application Layer. Files are then sent over the network a ""packet reassembly is carried out and the file is searched for the signature" (R. Hegarty, J. Haggerty, 2015). If the signature finds a match, a report would be generated and/or the upload or download process would be halted, until further action takes place.

Furthermore, there are advantages and disadvantages this software brings to the table. The generation of such reports mentioned before can turn out to be advantageous, as they can be made use of to track attackers, since data such as, which users would be uploading or downloading inappropriate data and their IP addresses can be displayed, making their whereabouts known. Also, when an upload/download process is blocked, the system prevents further file transfers, until the user validates themselves, applying the use of "two-factor authentication" and passwords. Generally, *XDet* has the benefit that it would be protected by other countermeasures already within the provider's perimeter, strengthening the overall software. Another positive aspect that *XDet* provides is its encrypted connections (SSL) within the transfers to and from the cloud. Moreover, when files are downloaded, the data is encrypted, and the data would be decrypted before writing to storage. An important limitation presented is the fact that because a tremendous amount of data is stored in cloud environments, it is probable that false positives and false negatives may occur. Finally, as a multitude of devices are available, a "legitimate" file download may be from any number of computers a cookie would be residing in.

## **Private Searching for Sensitive File Signatures**

While performing research on this paper, it was highlighted that issues can occur when search for sensitive file signatures within or on an untrusted server and while, during an investigation, a computer which results in containing crucial classified information is confiscated, its databases will end up being considered a target for attacks or "local exploits" (J. Solis, 2011). Examples of these "local exploits" may be considered interior databases that have content which has potential to be leaked via the influence of malware.

The fairly obvious solution provided is a scanner which searches for sensitive classified signatures on unreliable digital devices, all without revealing any information. This approach mentions its objective as not allowing the user or host to have an awareness about the signatures that are being found or searched, preventing the unwanted exposure of investigation that is proven useful during an ongoing case. John Solis and his team at Sandia National Labs, California, make sure to present us with a new method, in which he mentions is "inspired by existing private stream searching systems." This is revealed to be Paillier encryption. Paillier encryption is explained to be an encryption technique developed by computer scientist, Pascal Paillier in 1999, in which its end goal is to "provide public key encryption" (SpringerLink, 2011), where the user would be provided with a pair of keys, a public and private key. Messages can therefore be encrypted with their public key and decrypted with their private key. This is defined as a "type of keypair-based cryptography" (Daylighting Society, 2017). Solis proposes the idea of using this encryption type to build a "simple bit-mask" that pinpoints all classified signatures already located on a certain host. This begins to introduce benefits and drawbacks relevant to the methodology recommended as an answer.

A benefit that relates back to the Paillier encryption methodology is the fact that information will not be easily leaked or leaked at all, since all of the processes that take place are over "encrypted ciphertexts," even in conditions when the computer we are working on is considered to be unreliable. Another benefit highlighted after carrying out tests and inspecting the results, was that the time taken to perform all file hashes, along with other calculation was proven to be done efficiently. We can say that overall, it provides the first steps to an adaptable and competent solution. Moving on, the drawbacks seen with this method used, concern topics such as the variability of different file sizes and the effect that this has on actually determining the efficiency of the approach after looking at the results, as the execution time is suggested to be more accurately measured when the "function of files" is measured, instead of the amount of files there are. It is further on mentioned that, "reducing the communication overhead in large networked environments" (J. Solis, 2011) and having multiple scanners run on this large network would cause over-exertion and "overwhelm the administrator". This will be assigned as an area for improvement.

### Context-Based File Block Classification

Typically, in the hard disk or the storage media of a device, files or data are stored in what we call 'sequences' of data blocks, which eventually have to be pieced together again when recovering this information, through techniques, such as, the file carving process. Within the file carving process, we observe that we do this by locating block sequences or patterns, through the use of signatures, like the headers and footers of files, but unfortunately, said techniques used at present day are proven to have a low accuracy. To go deeper into what this paper will present, we will explore the purpose of what file block classification is and the approach that is presented.

The purpose of file block classification is defined to be "to assign a file type to a file block based only on its content" (L. Sportiello, S. Zanero, 2012) and the paper researched contains an approach presented for a "context-based classification" technique. This approach reports that it takes compound files into consideration, a compound file being a file or document that contains more than one data type within its data structure, for example, the file could contain text, audio files, video files etc. The target projected for this solution is to be able to improve "block-by-block classification" methods used in today's world of forensic investigations. It attempts to do this by exploiting the section where the file blocks that correspond to the same file are found on the storage media of the device. This study is also written to demonstrate to the reader the negative effect that compound files have on a step-by-step block classification method used in today's day, which allows us to highlight the idea that it is important to emphasize on focusing on compound file when planning and carrying out a statistical file block classification procedure.

During the investigative process of discovering an answer for these issues, a few problems involving compound files are made clear. These types of files have different types of structures, where data can be encoded as a "primitive type" within and embedded in the structure. When this occurs, the files are unable to present "uniform" properties, as "the blocks related to embedded data are statistically different from the other file blocks" (L. Sportiello, S. Zanero, 2012). This reason, along with other minor drawbacks, why it is necessary to make sure to ensure to have compound files that do not contain embedded data.

A solution this report proposes is a method that improves the overall performance of "single block file block classifiers." This results in being advantageous, as it is proved that false-positives and false-negatives are decreased, utilizing training sets that are only assigned to file or data types that are "primitive", therefore, improving performance, as a whole. Another solution provided that ends up being a strength is to insert a certain number of pre-assigned non-classification blocks, to prevent files from being misclassified. Apart from the fact that this solution improves block-classification performance, it's also fairly simple and general and can also be applied alongside other block classification algorithms.

## 2.4 Summary

In this literature review, we were able to view how crucial it is to maintain raw data found in digital devices found at crime scenes, especially when the data is considered to be volatile and can be lost at a moment when the device's power is turned off. Therefore, the implementation and development of the *SlackStick* software is critical via the adoption of signature searching for underlying characteristics in a file, at a live investigation in order to prevent scenarios like the ones mentioned above from occurring often with both, trained professionals and untrained investigators.

Moreover, we are already aware of the complexity of cloud-based systems when trying to transfer files to and from a server in a safe and efficient manner. *XDet*, the software provided as the solution to issues like these, demonstrates its key point by utilizing signature detection in halting illegal or malicious uploads and downloads to the cloud system and producing reports containing information like, IP Addresses, for example, to have the ability to track the location of the person or computer carrying out the illegal activity. This proves to be useful in reducing the attack risk on devices in said cloud networks.

Signature detection and signature searching are further highlighted to be the main aspect for tackling issues within forensic investigations, especially, when devices are confiscated for further research. This is the time when said computers are most vulnerable for leaks of confidential information and the approach provided is logical employment of Paillier encryption so that signature search would be carried out over "encrypted ciphertexts", as mentioned previously, hence preventing the leak of information.

Further on, the last paper studied in this literature review concerns file block classification and adds to the forensic investigative process but dives deeper into the recovery and file carving procedure, explaining how time consuming statistical block-by-block classification techniques can be. To tackle this, a file block classification methodology is proposed, but focusing on the contents of the data and the exploitation of sections where the file block belongs to the same file in the computer's storage, although, always taking into consideration the presence of embedded data in compound files.

To conclude, we are able to observe a direct correlation between the necessary need of the different types of signature detection techniques and signature search methodologies, emphasizing the capability for improving approaches used in current investigations carried out today. Using the information gathered from this literature review, I hope to create an efficient signature detection program, which is able to demonstrate the effectiveness of block-based signature detection, compared to file-based signature detection.

# 3. Product Design

## 3.1 Initial Concept

The early plan for the product of this project was pretty straightforward, as we would have to demonstrate a method in which we can observe a comparison between two types of signature-based detection methodologies that could be used in the professional world. The early ideas ended up turning out to be very similar to the final product generated. With this in mind, we can say that a presentation of the comparison of both the types of signature detection techniques turned out to be successful.

The two types of signature-based detection techniques to be compared were file-based signature detection and block-based signature detection, where these would be carried out for a large dataset of files, in this case, image files. For the file-based detection on the image files, the methodology would be carried out on the entire images file itself, whereas for the block-based technique, the methodology would be carried out on only the 2nd block of that certain image file. To understand this better, within a file, we can find blocks located and everything the file system does is made up of calculations or operations done on these blocks. **Figure 1** below very simply demonstrates a file system with its blocks labelled by numbers.

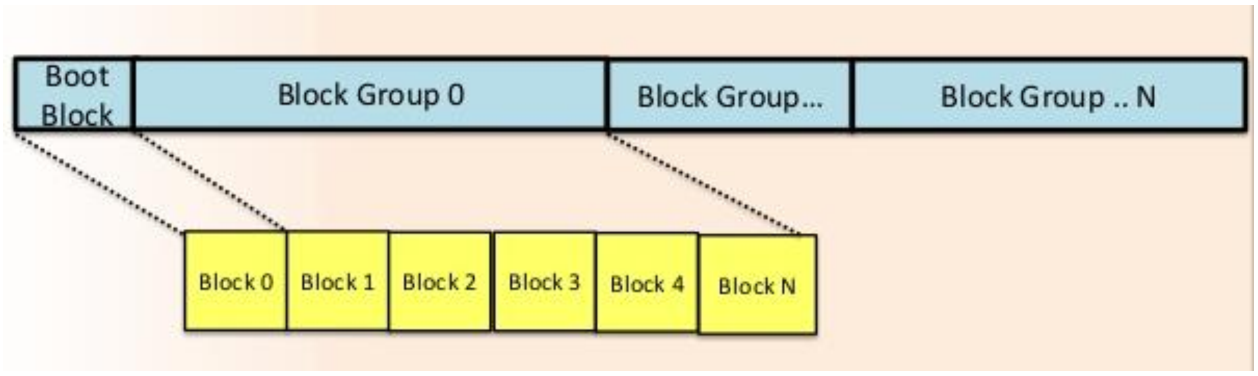


**Figure 1**

Delving deeper into the signature-based detection being executed for the files, hashes would be needed to be generated. Hashes or hash functions are used for data integrity and are essentially defined as the generation of a "value or values from a string or text using a mathematical function" (Technopedia, 2017). A simple analogy of a hash would be that it is described as a unique 'fingerprint' of the file. When attempting to attack or compromise a system, it is almost impossible to do so without performing changes within the file system itself, therefore altering the hash within the file, as well. This is the reason hashing was used when carrying out the different types of signature detection, enabling us to view if there is any corrupted files within the dataset. There are different types of hashing algorithms such as, MD5 algorithms, SHA-256 and SHA-1. In the product, it was decided that the MD5 algorithm would be utilised. This is due to the fact that MD5 algorithms themselves are faster to be generated, therefore making it a good option for use to minimalise margin of error in our comparison. In addition, the program would be developed making use of a Virtual Manchine with Linux as the operating system.

## 3.2 Program Flow Design

The general idea for this project was to accurately demonstrate the comparison between the file-based signature detection and the signature detection technique on the 2nd block of the files, with the end-goal of the block-based signature detection method being faster in duration. The concept is portrayed as to be a very simple one. The user would initially be able to download the dataset with the 3,000+ image files and would then proceed to open up the Linux terminal and navigate to the directory of the image dataset and the code script itself and then the user would be able to execute the program. Once executed, the terminal would display that the first signature detection technique used, the file-based method and then would continue to mention if there are any hash matches found within the file, along with the time taken to carry out the entire process. The same function would be carried out for the second technique tested, the block-based signature detection technique. Finally, the program would come to a halt as it has been able to go through the entire dataset twice, once for each method, leaving the user with the visualisation of the duration between both of the signature detection techniques. The systems flowchart below (**Figure 2**) demonstrates the flow of data within the program described.

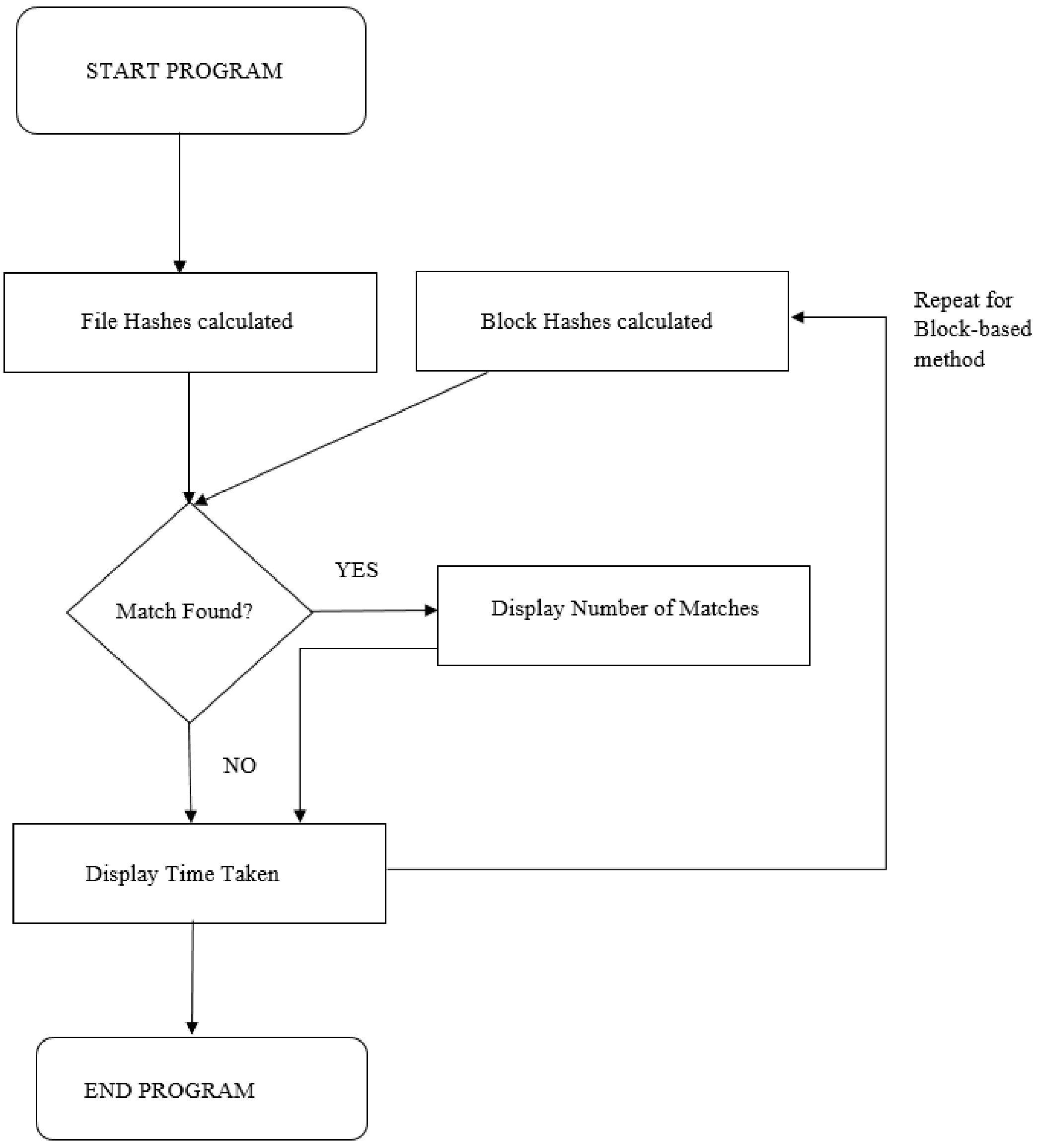


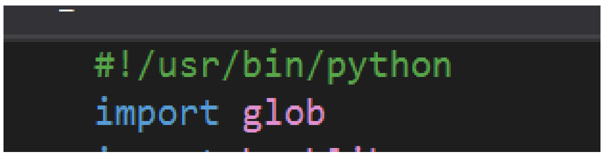
**Figure 2**

## 3.3 Use of Python Libraries

To be able to generate the desired outcome of this product, a variety of Python libraries had to be used when developing the source code. The Python Standard Library is very broad and contains many different facilities that are ready to be used. The Python libraries are able to provide "standardized solutions for many problems that occur in everyday programming" (The Python Software Foundation, 2020) and therefore, enhancing the ability to create smooth scripts to be executed professionally.

### 3.3.1 `glob` – Pathname Expression

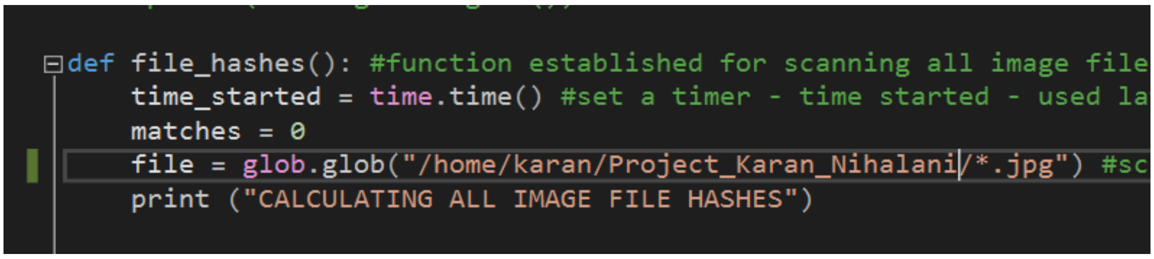
The first library we come across in our Python script for the program is `glob`. This library is defined to be used as a module that allows the user to locate a file in a directory, thus providing a pathname for the files, "matching a specified pattern according to the rules used by the Unix shell, although [the] results are returned in arbitrary order" (The Python Software Foundation, 2020). The screenshot of the script below (**Figure 3**) demonstrates the library imported at the beginning of the source code.



```
#!/usr/bin/python
import glob
```

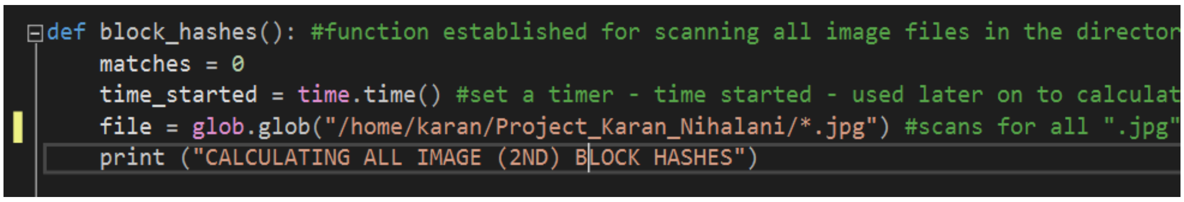
**Figure 3 – `glob` library (import)**

This library imported allows us to implement `glob` in the body of the code. To be more specific, `glob.glob` is input in the code to give the user the ability to return a possibly empty list of path names that match the specific pathname, `home/karan/Documents/Project_Karan_Nihalani/`, for example. It is also determined to be relative, containing "shell-style wildcards." The "shell-style wildcards" here, represented as `*.jpg`, as shown in the screenshots below (**Figure 4.1 and Figure 4.2**) as well, allows the user to navigate to that certain directory and locate all files with the extension '.jpg.'



```
def file_hashes(): #function established for scanning all image file
    time_started = time.time() #set a timer - time started - used la
    matches = 0
    file = glob.glob("/home/karan/Project_Karan_Nihalani/*.jpg") #sc
    print ("CALCULATING ALL IMAGE FILE HASHES")
```

**Figure 4.1 – `glob` (file hash function)**

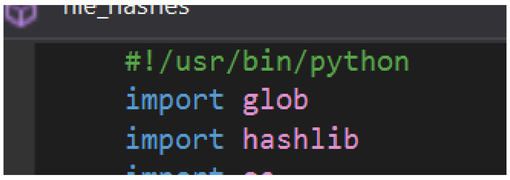


```
def block_hashes(): #function established for scanning all image files in the director
    matches = 0
    time_started = time.time() #set a timer - time started - used later on to calculat
    file = glob.glob("/home/karan/Project_Karan_Nihalani/*.jpg") #scans for all ".jpg"
    print ("CALCULATING ALL IMAGE (2ND) BLOCK HASHES")
```

**Figure 4.2 – `glob` (file hash function)**

### 3.3.2 `hashlib` – Producing secure hashes (MD5)

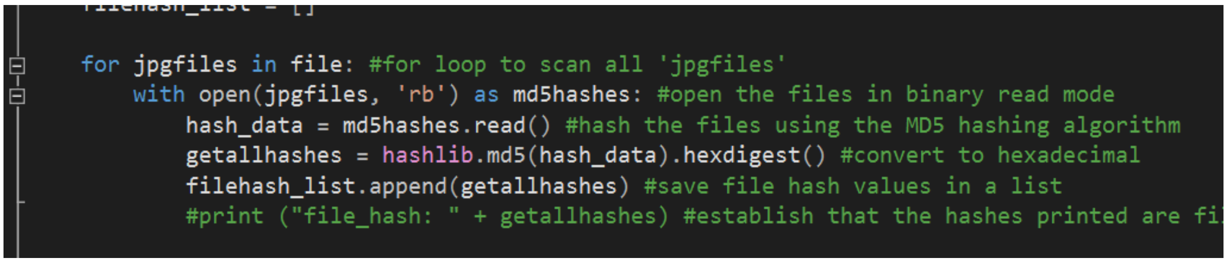
Right after the `glob` library, we are introduced to the `hashlib` library. This is one of the most important or the most important module that is included in the source code for this product. The `hashlib` library is basically what will enable the user to generate all the hashes for each file in the image dataset. In our case, the algorithm used in an MD5 algorithm. The library is described to have only "one constructor method named for each type of hash", therefore returning a "hash object with the same simple interface" (The Python Software Foundation, 2020). **Figure 5** below, demonstrates the import added to the beginning of the source code.



```
#!/usr/bin/python
import glob
import hashlib
```

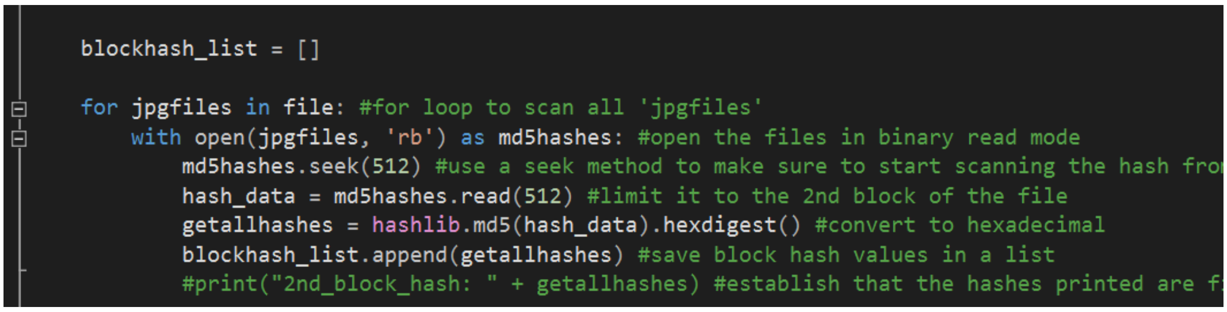
**Figure 5 – `hashlib` (import)**

Importing `hashlib` into our script now allows the user to generate fully functional MD5 hashes. Within both the functions for both the file-based signature detection and block-based signature detection, similar syntax has been employed. First, we make sure to loop through all the image files, which ensures that no file is skipped when performing the hash calculation. We then give the data to be hashed, using the MD5 algorithm a name and in this case, it is `hash_data`. After this, the `hash_data` is converted into hexadecimal, making use of '`. hexdigest`'. The `hash.hexdigest ()` attribute consists of a string returned of double length, "containing only hexadecimal digits" (The Python Software Foundation, 2020). This concludes the employment of the MD5 hashing algorithm for both signature detection techniques, as shown in **Figure 6.1** and **Figure 6.2**.



```
for jpgfiles in file: #for loop to scan all 'jpgfiles'
    with open(jpgfiles, 'rb') as md5hashes: #open the files in binary read mode
        hash_data = md5hashes.read() #hash the files using the MD5 hashing algorithm
        getallhashes = hashlib.md5(hash_data).hexdigest() #convert to hexadecimal
        filehash_list.append(getallhashes) #save file hash values in a list
        #print ("file_hash: " + getallhashes) #establish that the hashes printed are fi
```

**Figure 6.1** – `hashlib` & `hexdigest` in function (File Hash)

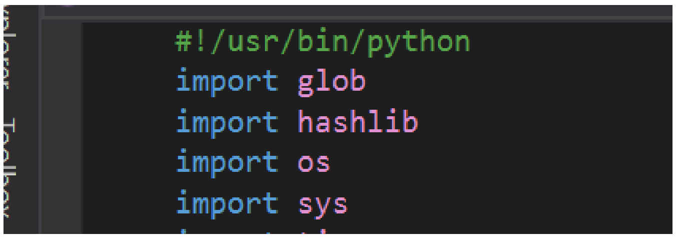


```
blockhash_list = []

for jpgfiles in file: #for loop to scan all 'jpgfiles'
    with open(jpgfiles, 'rb') as md5hashes: #open the files in binary read mode
        md5hashes.seek(512) #use a seek method to make sure to start scanning the hash fro
        hash_data = md5hashes.read(512) #limit it to the 2nd block of the file
        getallhashes = hashlib.md5(hash_data).hexdigest() #convert to hexadecimal
        blockhash_list.append(getallhashes) #save block hash values in a list
        #print("2nd_block_hash: " + getallhashes) #establish that the hashes printed are f
```

**Figure 6.2** – `hashlib` & `hexdigest` in function (Block Hash)

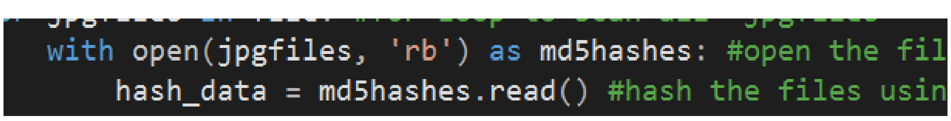
### 3.3.3 General Imports – `os` & `sys`

To make sure the python script is executed with no errors and no complications, there are a couple of more general libraries that are imported right at the beginning of the code (**Figure 7**). As defined, the `os` library is used to "provide a portable way of using operating system dependant functionality" (The Python Software Foundation, 2020). This is important for the product, as we need to access the directories with images to achieve the desired output, therefore interacting with the Linux operating system. In the body of the script, the interaction between the source code and the Linux OS is demonstrated, with the syntax `with open (jpgfiles, 'rb') as md5hashes` and `md5hashes.read()` (**Figure 8.1** and **Figure 8.2**).



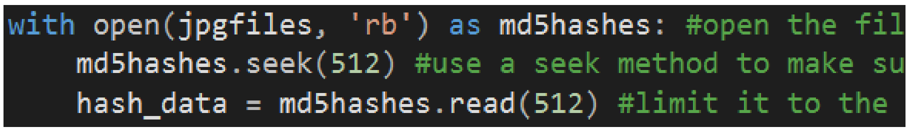
```
#!/usr/bin/python
import glob
import hashlib
import os
import sys
```

**Figure 7** – `os` & `sys` (import)



```
with open(jpgfiles, 'rb') as md5hashes: #open the fil
    hash_data = md5hashes.read() #hash the files usin
```

**Figure 8.1** – `os` functionality (File Hash function)



```
with open(jpgfiles, 'rb') as md5hashes: #open the fil
    md5hashes.seek(512) #use a seek method to make su
    hash_data = md5hashes.read(512) #limit it to the
```

**Figure 8.2** – `os` functionality (Block Hash function)

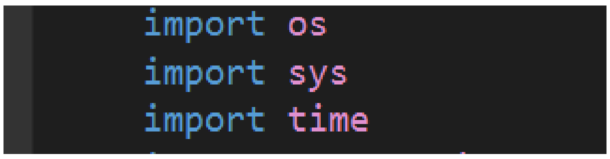
To add to this, the `sys` library is imported, where it "provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter" (The Python Software Foundation, 2020).

### 3.3.35 The `seek` method

To add to this, as viewed solely in Figure 8.2 above, there is an extra line input above `hash_data = md5hashes.read (512)`, which is `md5hashes.seek (512)`, which is key in attempting to carry out the block-based detection for the 2nd blocks of the image files. The seek method or function in Python is described to "change the position of the File Handle to a given specific position" (GeeksforGeeks, 2019). The syntax `md5hashes.seek (512)` states to make use of a seek method to make sure to start scanning the hash from the 512th byte or the second block of the file and `hash_data = md5hashes.read (512)` limits it to the 2nd block of the file, rather than the entire file itself.

### 3.3.4 The `time` library

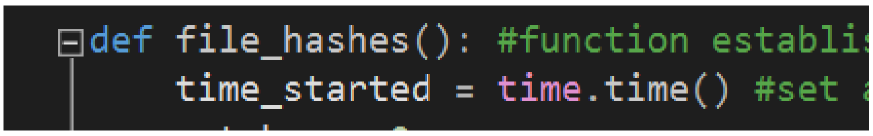
The `time` library used in the script is imported below the previous two libraries, as seen in **Figure 9**. This library is proved to be of use in the final program produced, as the signature detection process is timed to evaluate the efficiency and speed for the same amount of images in a dataset. Apart from this, the library allows for the user to make use of "various time-related functions" (The Python Software Foundation, 2020).



```
import os
import sys
import time
```

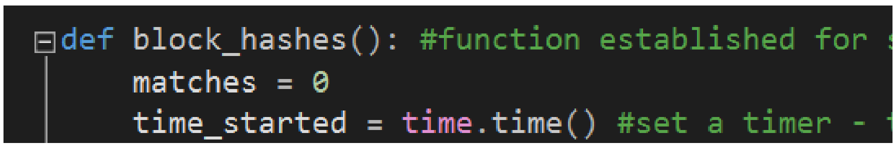
**Figure 9** – `time` library (import)

The time function used for the product is to essentially find the difference between the end time and the time the process started. This is demonstrated in **Figure 10.1 and Figure 10.2**, where near the start of the functions, the time started is defined by `time.time()`, allowing the product to commence timing the process. Further on, we are able to see in **Figure 10.3** that a simple equation is carried out to be called '`time_elapsed`', where we subtract the time the process started from the actual time taken.



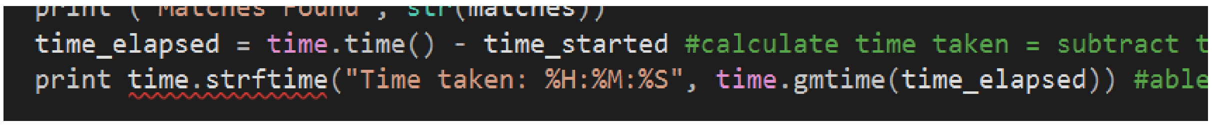
```
def file_hashes(): #function establi
    time_started = time.time() #set
```

**Figure 10.1** – `time.time()` (File Hash function)



```
def block_hashes(): #function established for
    matches = 0
    time_started = time.time() #set a timer -
```

**Figure 10.2** – `time.time()` (Block Hash function)



```
print ("Matches found", str(matches))
time_elapsed = time.time() - time_started #calculate time taken = subtract t
print time.strftime("Time taken: %H:%M:%S", time.gmtime(time_elapsed)) #able
```

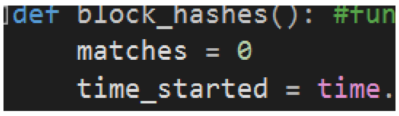
**Figure 10.3** – `time_elapsed` equation (both functions)

Above we can also observe that right underneath the equation, the time, represented in hours, minutes and seconds is printed onto the terminal screen. The attribute `time.strftime` is used to "convert a tuple or `struct _time` representing a time as returned by `gmtime()` or `localtime()` to a string as specified by the *format* argument" (The Python Software Foundation), which in this case the format argument is represented in hours, minutes and seconds, '`%H:%M:%S`', as mentioned before.

## 3.4 Hash Matches

Before exploring the source code to discover the methodology used to find and store matched hashes that have been generated, it is important to know what matched hashes mean in situations involving the forensic triage and the criminal investigative process. As mentioned before in this report, when a signature-based method is deployed, its objective is to locate any matching signatures and if any match is found, an "alert is generated, if a match is not found then the traffic flows without any problem" (U. H. Rao, U. Nayak, 2014). Finding a match is crucial in attempting to preserve the data's integrity, where that match could signify a file with malicious intent.

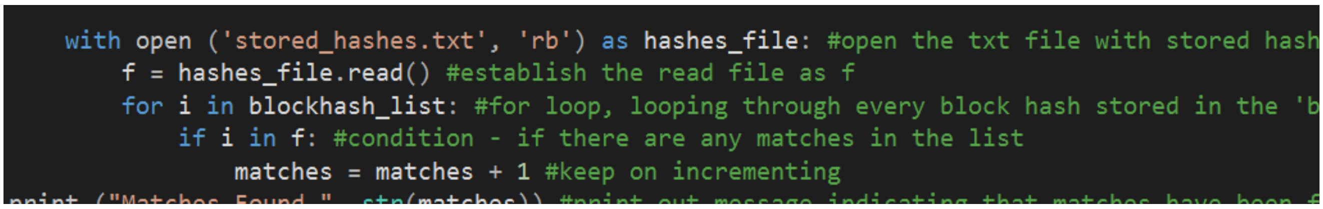
Within the script, the matches are set at a value of '0', before the signature detection process begins and the function is executed (**Figure 11**).



```
def block_hashes(): #fun
    matches = 0
    time_started = time.
```

**Figure 11** – `matches` counter (beginning)

Once the signature detection process is being performed and the hashes are being created, the file with stored hashes, '`stored_hashes.txt`' within the directory is opened in binary mode. A loop is then implemented, looping through every block and file hash within the list of hashes being generated and if there are any matches in the list, it would be printed out on the terminal screen. **Figure 12** demonstrated the code used for both the functions for the file-based signature detection technique and the block-based technique.



```
    with open ('stored_hashes.txt', 'rb') as hashes_file: #open the txt file with stored hash
        f = hashes_file.read() #establish the read file as f
        for i in blockhash_list: #for loop, looping through every block hash stored in the 'b
            if i in f: #condition - if there are any matches in the list
                matches = matches + 1 #keep on incrementing
print ("Matches Found ", str(matches)) #print out message indicating that matches have been f
```

**Figure 12** – signature match (code body)

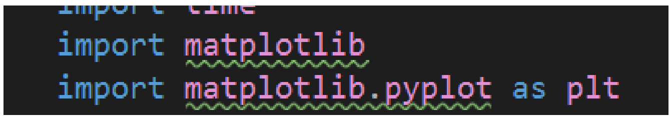
## 3.5 Additional Comments

There are a few additional comments added to the design of this product that have to be pointed out that are represented as potential improvements, if this product would be enhanced in a certain way.

### 3.5.1 `matplotlib` in Python

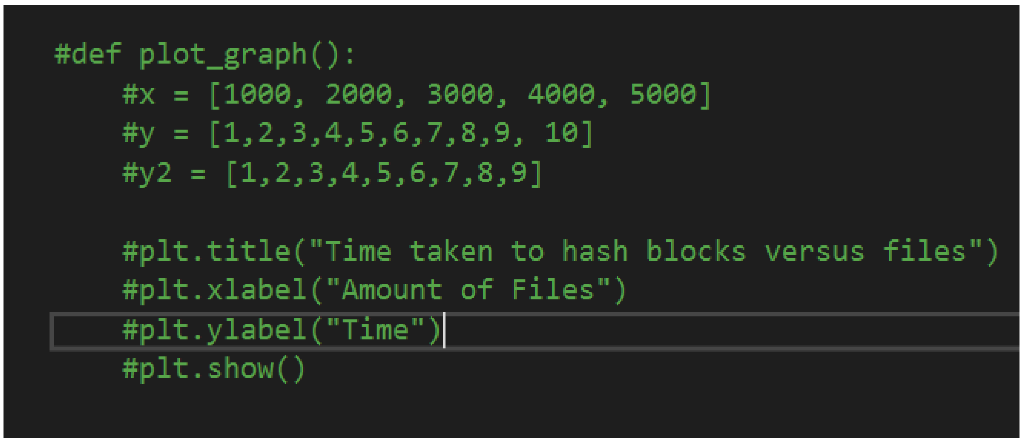
In Python, the module `matplotlib` allows for "graphing and data visualization" (PythonProgramming, 2014), allowing the user to better understand results in 2D plotted arrays like line graphs, for example.

In **Figure 13**, we are able to see the imports for this module added right at the top of the code script itself and commented out right below both of the functions, we can see a function that would plot the amount of matches found for both the file and block detection methods, taking into account a vast amount of images (**Figure 14**).



```
import time
import matplotlib
import matplotlib.pyplot as plt
```

**Figure 13** – `matplotlib` (import)



```
#def plot_graph():
    #x = [1000, 2000, 3000, 4000, 5000]
    #y = [1,2,3,4,5,6,7,8,9, 10]
    #y2 = [1,2,3,4,5,6,7,8,9]

    #plt.title("Time taken to hash blocks versus files")
    #plt.xlabel("Amount of Files")
    #plt.ylabel("Time")
    #plt.show()
```

**Figure 14** – `plot_graph` function

### 3.6 Summary

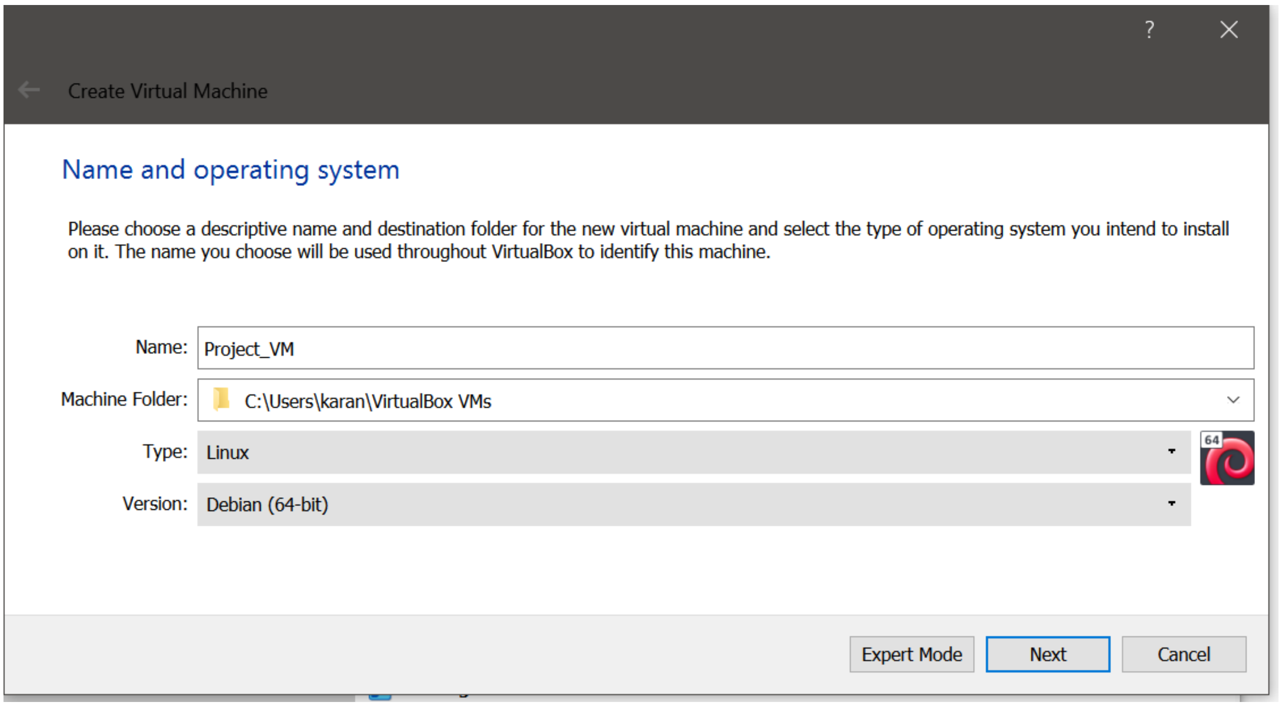
The design for this project has been given a great amount of consideration, demonstrating above the amount of time and research undergone to apply the correct libraries, modules and process to create a successful product that could be employed in a real world scenario.

# 4. Product Implementation

The section we will be focusing on after the code for the product has been developed, it the implementation of the product itself. This chapter will focus mainly on the operating system used and the program's functionalities within it.

## 4.1 Linux Operating System and 'Virtual Box'

The operating system used to execute the product is created on the virtual machine, 'Virtual Box', which can be downloaded. Once the application is downloaded and running, the steps to set up the operating system are followed. **Figure 15** demonstrates the step in this process, where the Linux OS,



Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Project_VM

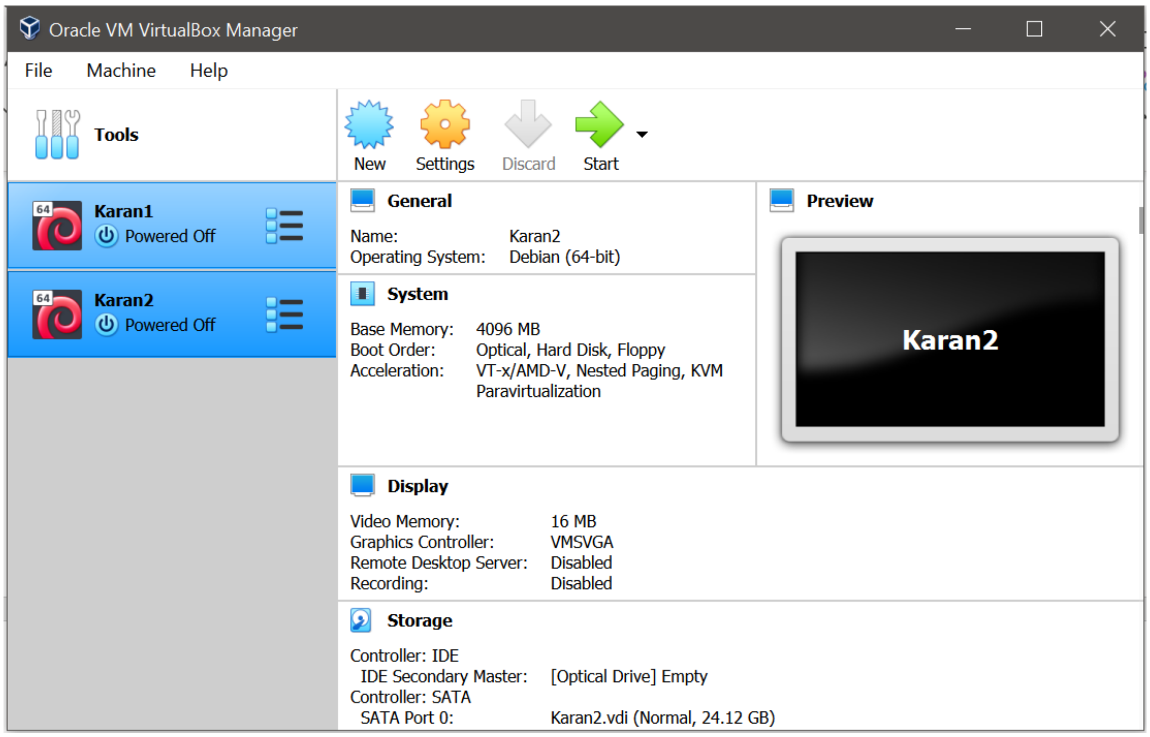
Machine Folder: C:\Users\karan\VirtualBox VMs

Type: Linux

Version: Debian (64-bit)

more precisely, the Debian software is chosen to be set up. Debian is defined as a "volunteer project that has developed and maintained a GNU/Linux operating system for well over a decade" (Debian, 2020), "[using] the Linux kernel and other components obtained from the GNU project (M. Rouse, 2006).

It is of high importance to make use of Virtual Machines these types of projects, as they can result in being beneficial for the program's developer. There are various benefits to them, but perhaps a benefit that stands out among the rest is the fact that there can be "multiple OS environments [that] can exist simultaneously on the same machine, isolated from each other" (O. Weis, 2019), therefore, if there is any issue or problem with one VM, the user can always revert to the main screen (**Figure 16**) and create a new Virtual Machine.



**General**

Name: Karan2
Operating System: Debian (64-bit)

**System**

Base Memory: 4096 MB
Boot Order: Optical, Hard Disk, Floppy
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

**Display**

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

**Storage**

Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Karan2.vdi (Normal, 24.12 GB)

**Figure 16** – 'Virtual Box' home screen

## 4.2 Navigation of directories

The Virtual Machine starts up and the user is presented with the desktop for this Debian Linux operating system (**Figure 17**).

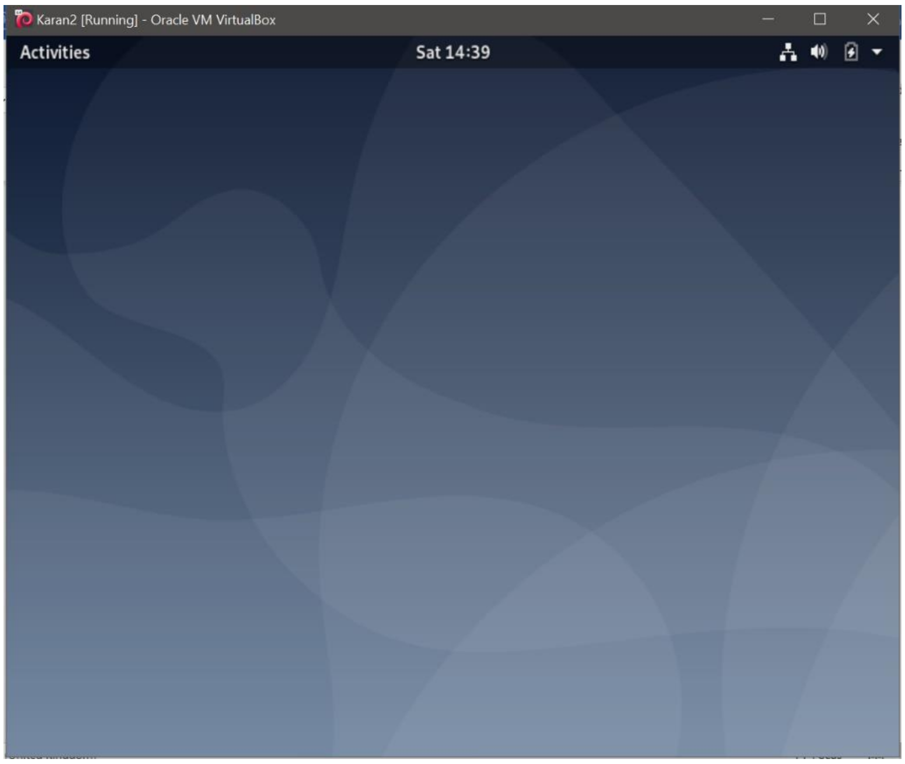
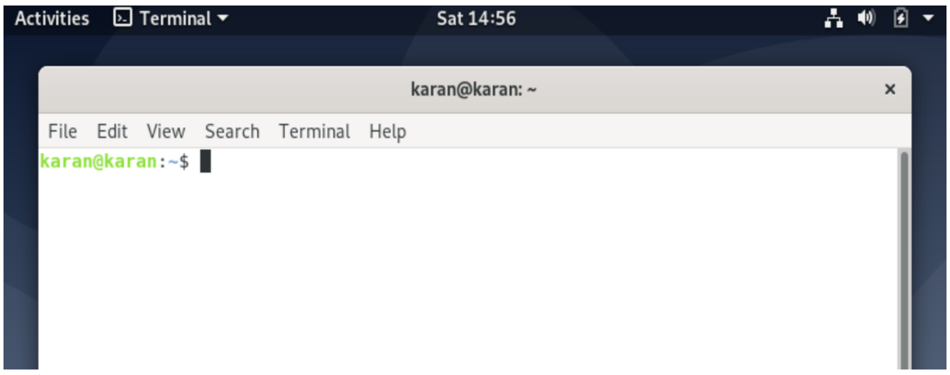


**Figure 17** – VM Desktop Screen

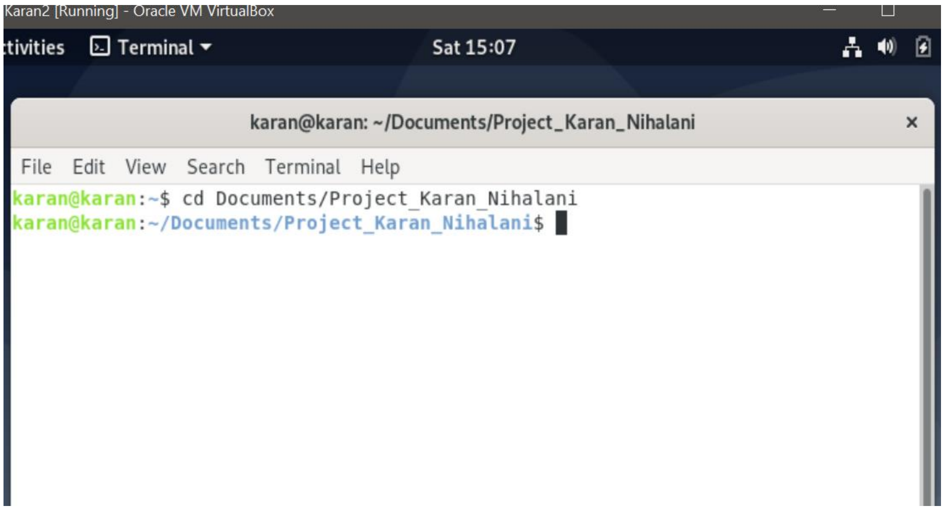
Once on the screen, the user will have to navigate to the system's terminal (**Figure 18**), which is used to execute the program itself, but before executing the program, the navigation to the directory where the folder with the Python code script and the image files are located have to navigated to via the terminal itself. The command `cd` is used to advance directories and revert directories. The official syntax used with `cd` is `cd [option] [directory]`. When attempting to navigate directories, in this case, the 'relative pathname' would be used, meaning that it would be "much less tedious to use" (The Linux Information Project, 2007), providing the user with easier accessibility.



```
karan@karan:~$ 
```

**Figure 18** – The Terminal

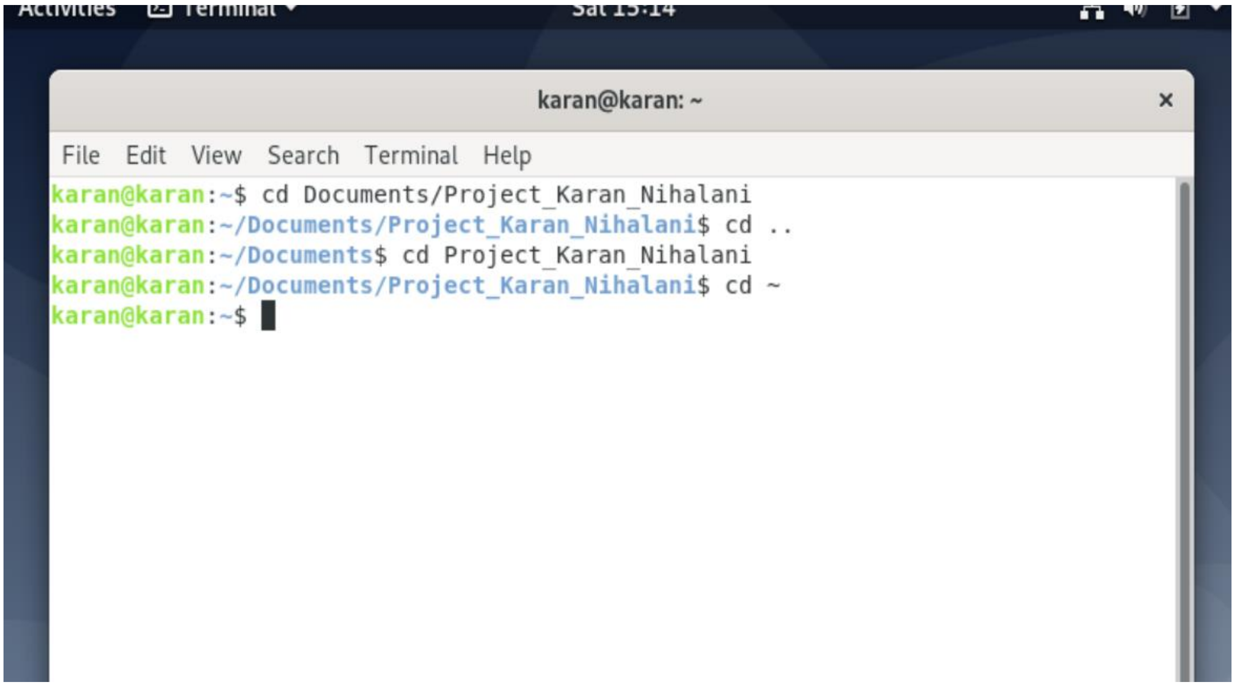
Further on, to navigate to the desired directory needed to implement the product, the syntax `cd Documents/Project_Karan_Nihalani` is used, as we find ourselves in the 'home' or 'root' directory to begin with. This leads us to finally being able to run the program (**Figure 19**).



```
karan@karan:~$ cd Documents/Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ 
```

**Figure 19** – Navigating to the product's directory

A final note to be made in navigating directories is the fact that the user can go back a directory, making us of the syntax `cd ..` and the user can also directly go back to the 'home' or 'root' directory, typing `cd ~` as shown below (**Figure 20**).



```
karan@karan:~$ cd Documents/Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ cd ..
karan@karan:~/Documents$ cd Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ cd ~
karan@karan:~$ 
```

**Figure 20** – Reverting directories

## <u>4.3 Data storage and location</u>

Before approaching the final step to implement the working product, the data location and content will be looked further into. This section explains the location where the data is stored, along with demonstrating the content within the folder. As mentioned above, all the data is physically found within the sub-folder 'Project_Karan_Nihalani', which can be found manually within the 'Documents' folder at the 'home' directory (**Figure 21**).

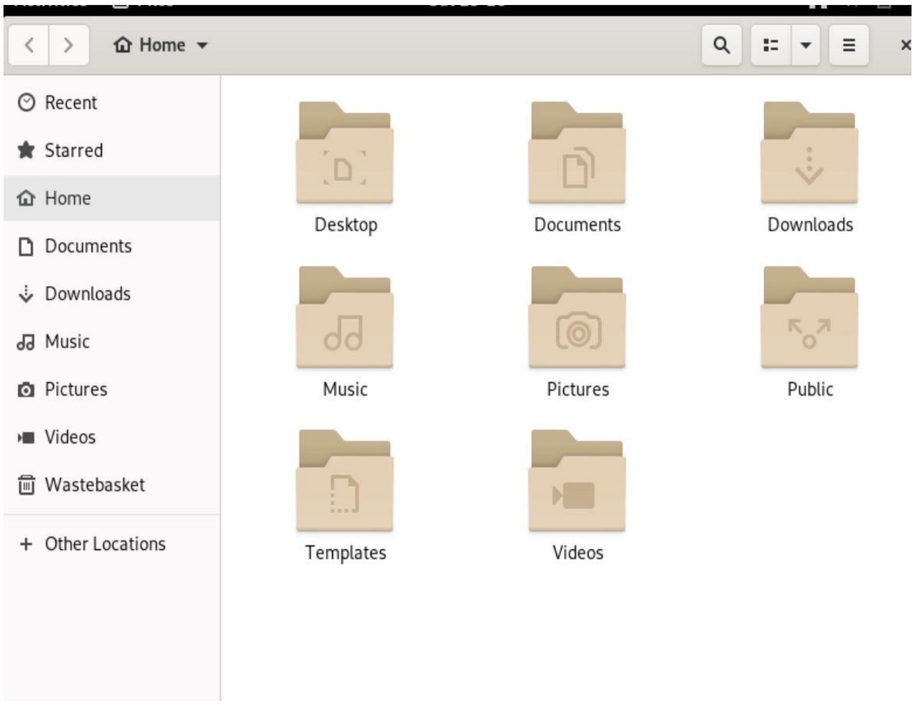
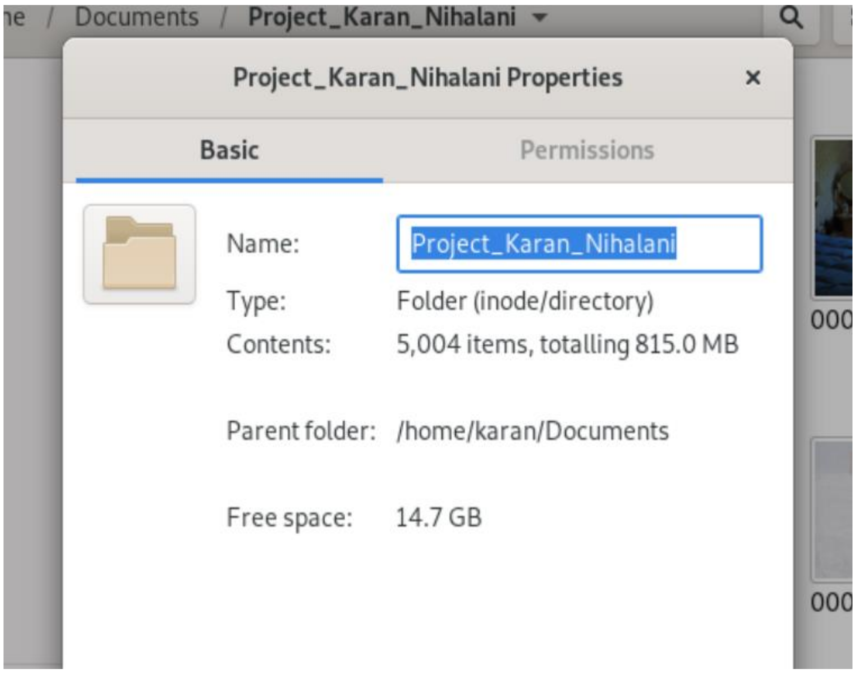


**Figure 21** – 'home' directory folder with 'Documents' folder to be accessed

Furthermore, **Figure 22** clearly demonstrates the content found within the 'Project_Karan_Nihalani' folder within the 'Documents' folder itself, where there are more than 3000 items found, which include the product's code script, written in Python, the images themselves and the text file with any stored hashes that are found or not (Figure 23).



Project_Karan_Nihalani Properties

Name: Project_Karan_Nihalani
Type: Folder (inode/directory)
Contents: 5,004 items, totalling 815.0 MB
Parent folder: /home/karan/Documents
Free space: 14.7 GB

**Figure 22** – project folder properties

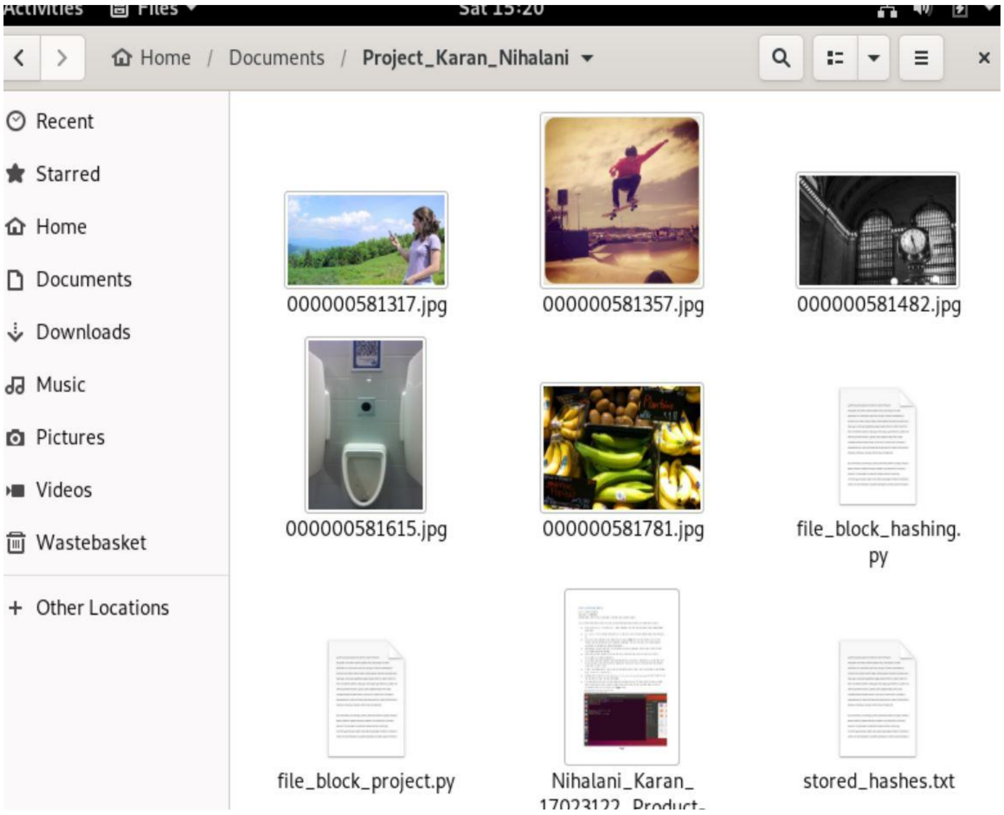
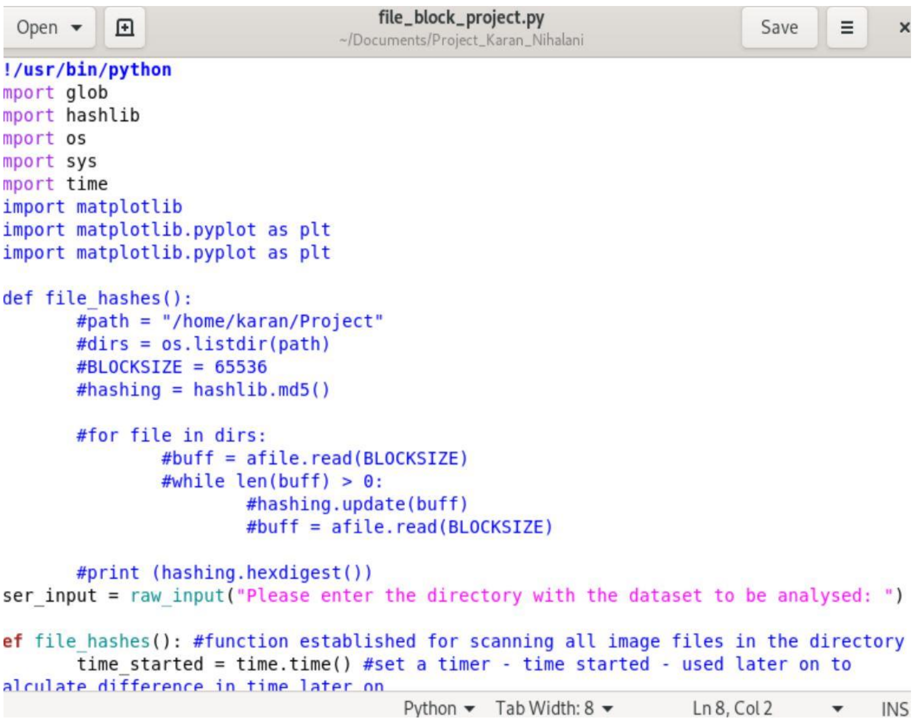


**Figure 23** – project folder content

The final script that is used to implement the product is the file named, ‘file_block_project.py’, for which we can see its contents opened in a text editor in **Figure 24** below.

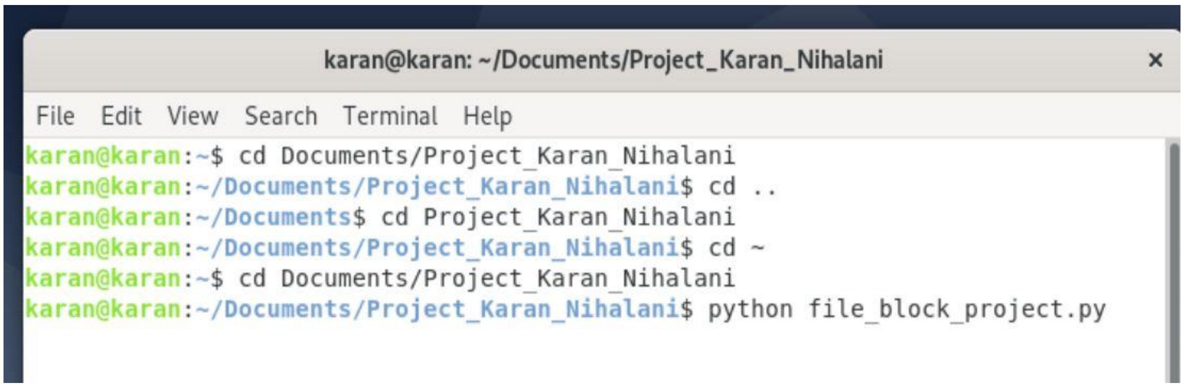


```
!/usr/bin/python
mport glob
mport hashlib
mport os
mport sys
mport time
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

def file_hashes():
        #path = "/home/karan/Project"
        #dirs = os.listdir(path)
        #BLOCKSIZE = 65536
        #hashing = hashlib.md5()

        #for file in dirs:
                #buff = afile.read(BLOCKSIZE)
                #while len(buff) > 0:
                        #hashing.update(buff)
                        #buff = afile.read(BLOCKSIZE)

        #print (hashing.hexdigest())
ser_input = raw_input("Please enter the directory with the dataset to be analysed: ")

ef file_hashes(): #function established for scanning all image files in the directory
        time_started = time.time() #set a timer - time started - used later on to
alculate difference in time later on
```

**Figure 24** – section source code screenshot

## <u>4.4 Program execution and Terminal display</u>

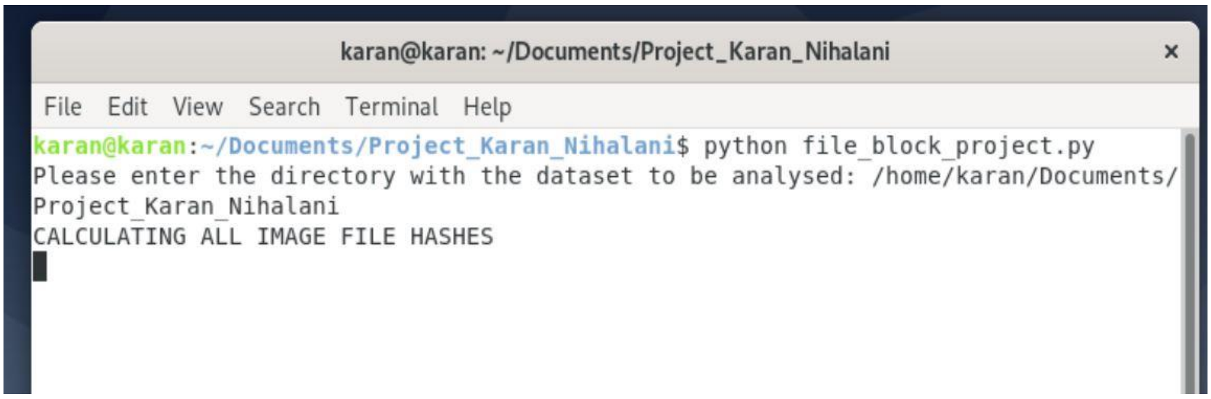
Finally, when it comes to the intended display after the program is implemented, the layout on the terminal seems to be a fairly simple one, as the objective in hand is to display a clear comparison between both the signature detection techniques, via the time taken to calculate hashes for the 3000+ images in the folder. **Figure 25** demonstrates the navigation to the target folder in the terminal to execute the Python script, with the syntax `python file_block_project.py`.


karan@karan: ~/Documents/Project_Karan_Nihalani
File Edit View Search Terminal Help

```
karan@karan:~$ cd Documents/Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ cd ..
karan@karan:~/Documents$ cd Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ cd ~
karan@karan:~$ cd Documents/Project_Karan_Nihalani
karan@karan:~/Documents/Project_Karan_Nihalani$ python file_block_project.py
```

**Figure 25** – Python script execution via terminal

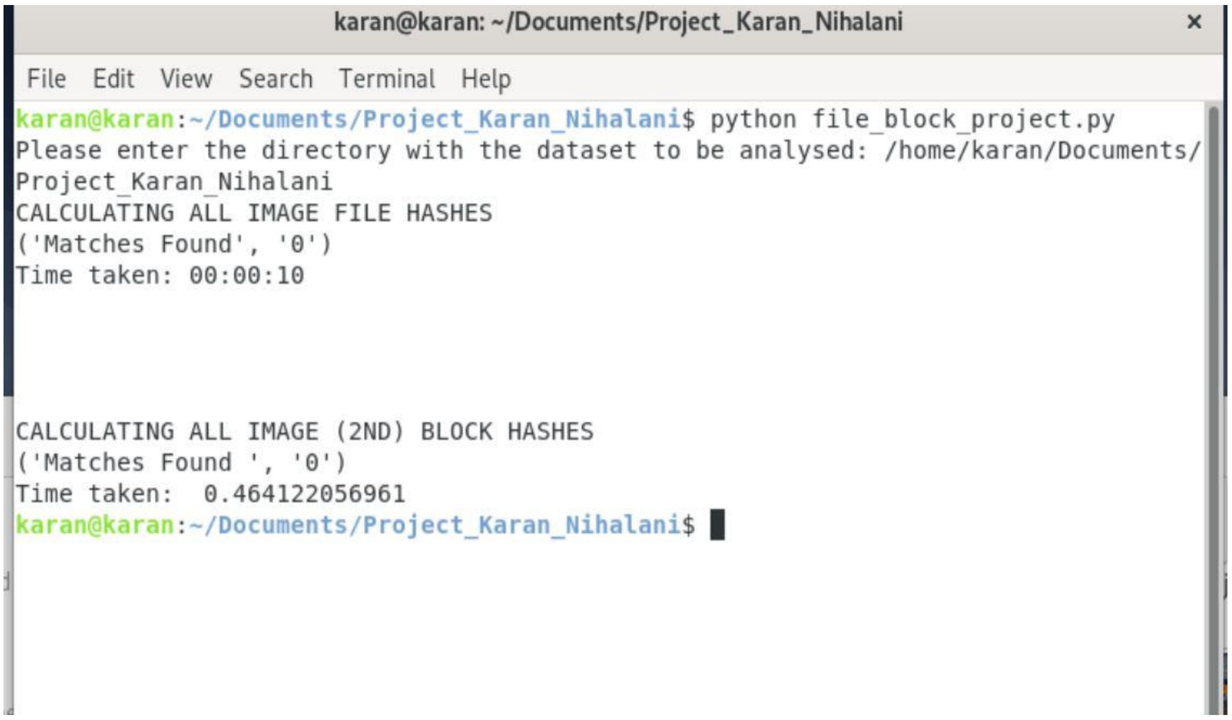
Immediately after this, the user is presented with the text ‘Calculating Image File Hashes’, after entering the desired directory, demonstrating that the process of calculating hashes for the entire image files has begun and that the first signature detection technique is taking place. As seen in the image below (**Figure 26**), the user must wait a while to see some results displayed.


karan@karan: ~/Documents/Project_Karan_Nihalani
File Edit View Search Terminal Help

```
karan@karan:~/Documents/Project_Karan_Nihalani$ python file_block_project.py
Please enter the directory with the dataset to be analysed: /home/karan/Documents/
Project_Karan_Nihalani
CALCULATING ALL IMAGE FILE HASHES
```

**Figure 26** – ‘Calculating All Image File Hashes’ screen

Eventually, the user will be able to view the final results (**Figure 27**) on the terminal screen where the calculation for all image file hashes are done, as well as the calculations for all 2nd block file hashes and whether any matches have been found when performing the signature detection for each. In addition to this, we can view that the duration of the methodology has been timed to demonstrate the clear comparison between both the file-based and block-based signature detection techniques. The time is displayed in hours, minutes and seconds for the file-based signature detection method, whereas the time is represented in seconds only for the block-based method, as it was predicted that the block-based method would be quicker in performing signature detection.


karan@karan: ~/Documents/Project_Karan_Nihalani
File Edit View Search Terminal Help

```
karan@karan:~/Documents/Project_Karan_Nihalani$ python file_block_project.py
Please enter the directory with the dataset to be analysed: /home/karan/Documents/
Project_Karan_Nihalani
CALCULATING ALL IMAGE FILE HASHES
('Matches Found', '0')
Time taken: 00:00:10




CALCULATING ALL IMAGE (2ND) BLOCK HASHES
('Matches Found ', '0')
Time taken:  0.464122056961
karan@karan:~/Documents/Project_Karan_Nihalani$
```

**Figure 27** – Final Result screen, displayed on terminal

## 5. Evaluation

The Evaluation section will examine the outcome of the project's design. This will include the testing undertaken to demonstrate the success rate of the product and a further analysis on the results of the product itself.

### 5.1 Evaluation Design & Testing

This section helps explain the process undertaken for the designing and testing of the final product.

#### 5.1.1 An Evaluation's purpose

The reason to carry out an evaluation process for a project or product, as it can help with the current outcome produced. One of the ways it can provide the product with aid is in improving the current version of the product for future purposes, where an example for improvement could be the program's performance in either speed, functionality or accessibility. Furthermore, the program's accountability can be enhanced, as "a program that can account for its public investments by demonstrating progress and results is more likely to be viewed as successful" (Office of Energy Efficiency & Renewable Energy, 2020), leaving exterior users and investors with a better impression. This evaluation will emphasise the steps taken to fully evaluate the product in depth.

### 5.1.2 Tests carried out by the developer

While coming up with performing tests by the developer on the product, the emphasis has been focused more on testing for bugs within the code, especially when hashing and carrying out the duration for both methods, and assessing the final functionality of the product, where the developer in this scenario has chosen to test the product via the criteria of test data. Test data signifies data that is "used as input to perform the tests for identifying and locating the defects", where data input can be compared to the actual outcome. The so called 'Alpha Testing' is carried out to ensure that the code itself does not have syntax errors.

## 5.2 Developer Testing

Below, **Figure 28** demonstrates the evidence of the tests being carried out and recorded in a table, establishing the expected and actual outcomes of the data input.

| Input & Program processes | Expected Outcome | Actual Outcome | Comments (if needed) |
|---|---|---|---|
| `cd <directory with program and dataset>` | Directory with code and dataset exists & is accessed. | Directory with code and dataset exists & is accessed. | Works successfully. |
| `python file_block_project.py` | Terminal finds python script and runs it. | Terminal finds python script and runs it. | Works successfully. |
| **Directory prompt in terminal, e.g:** `/home/karan/Documents/Project_Karan_Nihalani` | Program accepts as valid directory. | Program accepts input as valid directory. | Works successfully. |
| **File hashes being calculated** | MD5 hashes carried out on entire file. | MD5 hashes carried out on entire file, presenting text for confirmation. | Works successfully. |
| **Block hashes being calculated** | MD5 hashes carried out on 2nd Block of file. | MD5 hashes carried out on 2nd Block of file, presenting text for confirmation. | Works successfully. |
| **Matches found** | If any matches, number of matches displayed, if '0' matches, zero matches displayed. | Zero matches are displayed. | Works successfully. |
| **Timing both techniques** | Calculating File Hashes should take more time than 2nd Block Hashes. | Calculating File Hashes takes 10 secs more than Block Hashes | Works successfully. |

**Figure 28** – Testing Table

## 5.3 Analysing Results
Looking further into the analysis of the results, we can identify the fact that a clear comparison between both file-based and block-based signature detection is made evident, deeming the outcome as successful. If the program was to be developed further, another method to demonstrate the comparison between both signature detection techniques would be created to help the user visualise the outcome. An example is commented out within the code itself, where `matplotlib` could allow the developer to create a line graph demonstrating the amount of time taken to perform the hashes, relative to an increase of images.

## 5.4 Personal Evaluation
I am personally satisfied with the outcome of this project. The product itself resulted in having a positive conclusion, as the results showed this as well. The upcoming section will discuss my thoughts on each area of developing this project.

### 5.4.1 Evaluation of Research
During the past 3 years doing the Computer Forensics and Security course, I was already relatively familiar with the basic concept of the Forensic Triage process and the processes and analysis of results in a cybercriminal investigation, but this is the first time I was challenged to go further into this subject via researching a multitude of technical papers and attempting to locate the most appropriate and relevant information for my project. At first, I thought that it was extremely difficult finding any reliable papers that would relate to something as specific as signature-based detection, but as I managed to alter my research technique, I started to encounter articles that related more to this topic. Even though cloud-based detection is not specified in the project brief, this was an interesting review to add, as the use of cloud-based systems is growing exponentially and will continue growing as a primary source of storage in the future.

### 5.4.2 Evaluation of Product Development and Design
Commencing the design and code development for this project was the most challenging aspect of all, as there were many technical obstacles that would come up occasionally, impeding further progress at the time. The idea of using a virtual machine with a Linux-based operating system and creating a Python-based script was quickly established, as this was a comfortable base to begin working on and I was already familiar with both. Furthermore, making use of Python made it easier when researching syntax for the program and input libraries that could be used. Unfortunately, some time was lost, as the project was halted while attempting to debug my code every time I attempted to run it, therefore not allowing me to show a further improvement in the code itself, e.g proceeding with the `matplotlib` graph.

### 5.4.3 Evaluation of Product Implementation
The implementation of the product was straightforward as the program was to be executed at terminal level in the virtual machine. Fortunately, this did not take up much time for the project, which allowed me to solve any issues I had at the design stage of the project, with a comfortable amount of time. The only issue I was to face was the fact that at times there were issues with files being considered to large for the virtual machine's storage capacity, which was quickly dealt with.

## 5.5 Summary
As previously mentioned, I am glad with the outcome of this project, as it fulfils the objectives that were lay out in the beginning, highlighting the comparison between both signature-based detection methodologies, although more could be added in the future to enhance the efficiency of the product.

## 6. Conclusion

The work performed during this entire report involved designing a visual experiment comparing file-based and block-based signature detection techniques. A great amount of time was invested in researching key aspects of signature detection and the different methods of signature detection that can be used during the forensic triage in today’s world. The product itself had one final objective or goal to be achieved and this was kept in mind when designing the product and the goal was met, but there were certain aspects that could have been added for the product’s improvement, if time allowed it. If this project were to be done again, I would have prioritised the design much more, decreasing the amount of time given for the product’s implementation and research. This would have allowed me to better represent the results. In conclusion, despite of the lack of time for certain aspects and the troubles encountered during the design stage, the overall impressions are successful.

## References

IGI Global, 2015. "What Is Signature Based Detection" [Online] [Accessed on 3rd March 2020]. https://www.igi-global.com/dictionary/an-auto-reclosing-based-intrusion-detection-technique-for-enterprise-networks/46076

UC Berkeley, 2017. "Type I and Type II errors" [Online] [Accessed on 3rd March 2020]. https://www.stat.berkeley.edu/~hhuang/STAT141/Lecture-FDR.pdf

IBM, 2019. "Block Storage" [Online] [Accessed on 3rd March 2020]. https://www.ibm.com/cloud/learn/block-storage#toc-what-is-bl-FugUvNFG).

ELSEVIER SciTech Connect, 2013. "Incident Response: Live Forensics and Investigations, *Chapter 5, Case Study: Live versus Postmortem*" [Online] [Accessed on 3rd March 2020]. http://scitechconnect.elsevier.com/wp-content/uploads/2013/09/Incident-Response-Live-Forensics-and-Investigations.pdf

Ebrary.net, 2020. "How Does Detection Work?" [Online] [Accessed on 1st April 2020]. https://ebrary.net/26722/computer_science/detection_work

Council of Europe, 2013. "Live Data Forensics - or - Why volatile data can be crucial for your cases" [Online] [Accessed on 1st April 2020]. https://www.coe.int/en/web/octopus/blog/-/blogs/live-data-forensics-or-why-volatile-data-can-be-crucial-for-your-cases

Robert Hegarty, John Haggerty, 2015. "SlackStick: Signature-Based File Identification for Live Digital Forensics Examinations" [Online] [Accessed on 1st April 2020]. http://e-space.mmu.ac.uk/608682/2/SlackStick%20Final%20Version.pdf

Robert Hegarty, John Haggerty, 2015. "Extrusion detection of illegal files in cloud-based systems" [Online] [Accessed on 1st April 2020]. http://e-space.mmu.ac.uk/617083/1/authorFinalVersion.pdf

PC Magazine Encyclopedia, 2019. "Definition of: Extrusion Detection" [Online] [Accessed on 1st April 2020]. https://www.pcmag.com/encyclopedia/term/58880/extrusion-detection

John Solis, 2011. "Private Searching for Sensitive File Signatures" [Online] [Accessed on 1st April 2020]. https://www.scitepress.org/Papers/2011/34667/34667.pdf

SpringerLink, 2011. "Paillier Encryption and Signature Schemes" [Online] [Accessed on 1st April 2020]. https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_488

Daylighting Society, 2017. "Pallier Cryptosystem" [Online] [Accessed on 1st April 2020]. https://paillier.daylightingsociety.org/about

Luigi Sportiello, Stefano Zanero, 2012. "Context-Based File Block Classification" [Online] [Accessed on 1st April 2020]. http://dl.ifip.org/db/conf/ifip11-9/df2012/SportielloZ12.pdf

Technopedia, 2017. “Hashing” [Online] [Accessed on 3rd April 2020].
https://www.techopedia.com/definition/14316/hashing

The Python Software Foundation, 2020. “The Python Standard Library” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/3/library/

The Python Software Foundation, 2020. “`glob` – Unix style pathname pattern expression” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/3/library/glob.html

The Python Software Foundation, 2020. “`hashlib` – Secure hashes and message digests” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/2/library/hashlib.html

The Python Software Foundation, 2020. “`os` – Miscalleneous operating system interfaces” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/3/library/os.html

The Python Software Foundation, 2020. “`sys` – System-specific parameters and functions” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/3/library/sys.html

GeeksforGeeks, 2019. “Python seek() function” [Online] [Accessed on 3rd April 2020].
https://www.geeksforgeeks.org/python-seek-function/

The Python Software Foundation, 2020. “`time` – Time access and conversions” [Online] [Accessed on 3rd April 2020].
https://docs.python.org/3/library/time.html

Umesh Hodeghatta Rao, Umesha Nayak, 2014. “The InfoSec Handbook: An Introduction to Information Security, *How Does Detection Work?*” [Online] [Accessed on 4th April 2020].
https://link.springer.com/content/pdf/10.1007%2F978-1-4302-6383-8.pdf

PythonProgramming, 2014. “Matplotlib Crash Course Python Tutorial” [Online] [Accessed on 4th April 2020].
https://pythonprogramming.net/matplotlib-python-3-basics-tutorial/

Debian, 2020. “Debian is the rock on which Ubuntu is built” [Online] [Accessed on 6th April 2020].
https://ubuntu.com/community/debian

M. Rouse, 2006. “Debian” [Online] [Accessed on 6th April 2020].
https://searchdatacenter.techtarget.com/definition/Debian

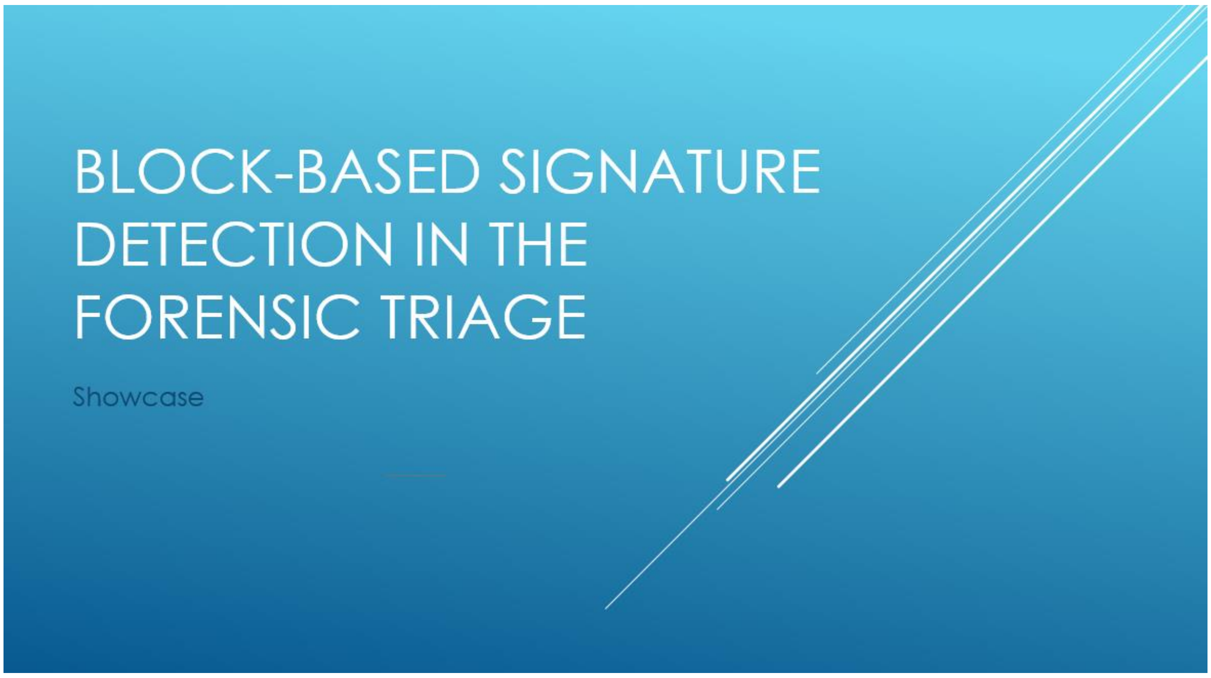
O. Weis, 2019. “Pros and cons of the virtual machines – How to access serial devices in VMs” [Online] [Accessed on 6th April 2020].
https://www.serial-server.net/virtual-machine/

The Linux Information Project, 2007. “The cd Command” [Online] [Accessed on 7th April 2020].
http://www.linfo.org/cd.html

Office of Energy Efficiency & Renewable Energy, 2020. “Program Evaluation: Why, What, and When to Evaluate” [Online] [Accessed on 9th April 2020].
https://www.energy.gov/eere/analysis/program-evaluation-why-what-and-when-evaluate

## Appendix (Showcase Materials)

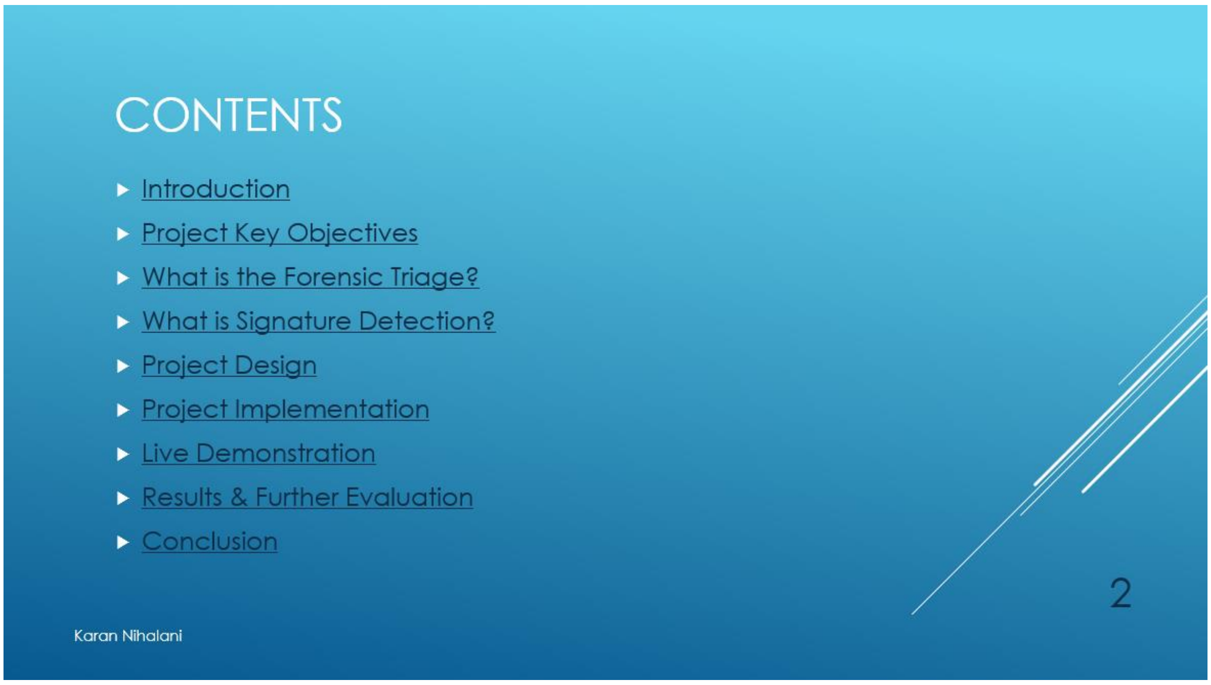
### Slide 1



BLOCK-BASED SIGNATURE DETECTION IN THE FORENSIC TRIAGE

Showcase

### Slide 2



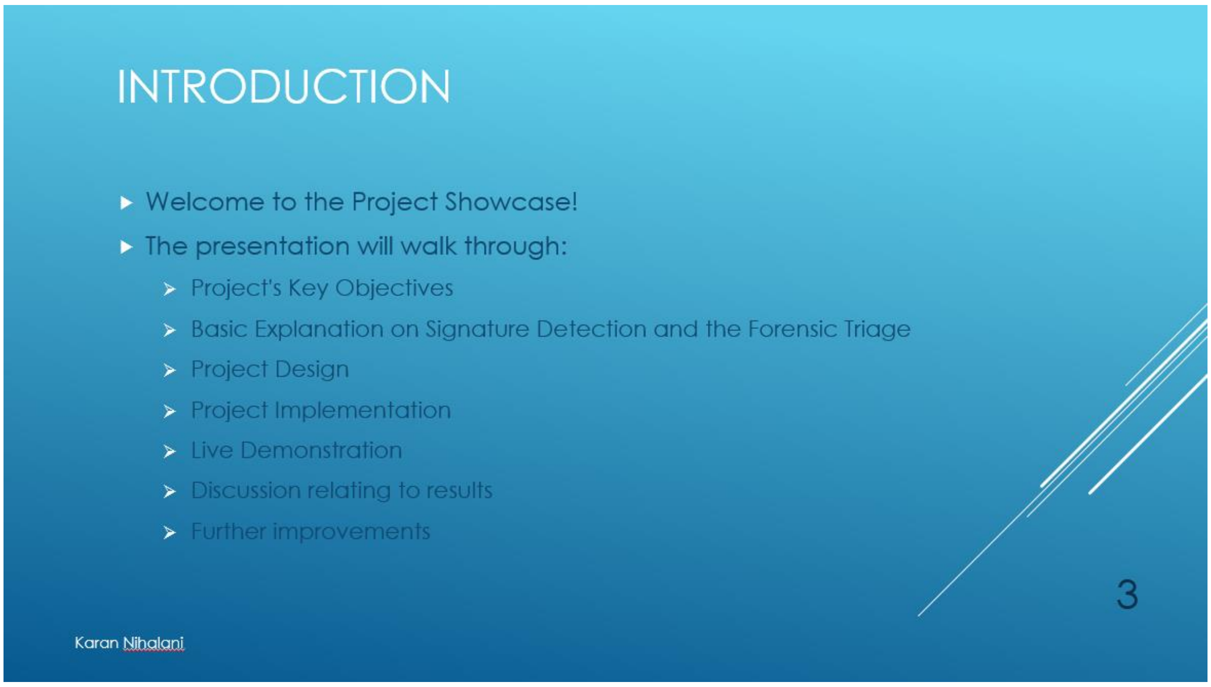
CONTENTS

- Introduction
- Project Key Objectives
- What is the Forensic Triage?
- What is Signature Detection?
- Project Design
- Project Implementation
- Live Demonstration
- Results & Further Evaluation
- Conclusion

Karan Nihalani

2

**Slide 3**



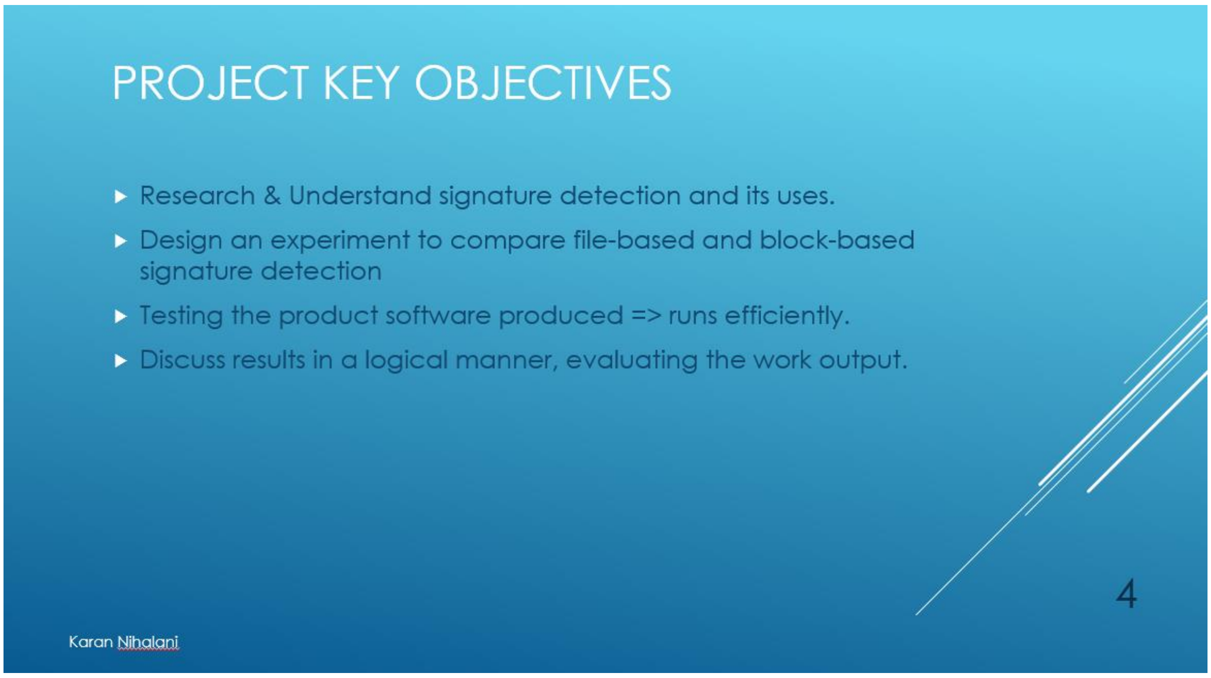
## INTRODUCTION

- Welcome to the Project Showcase!
- The presentation will walk through:
  - Project's Key Objectives
  - Basic Explanation on Signature Detection and the Forensic Triage
  - Project Design
  - Project Implementation
  - Live Demonstration
  - Discussion relating to results
  - Further improvements

3

Karan Nihalani

**Slide 4**



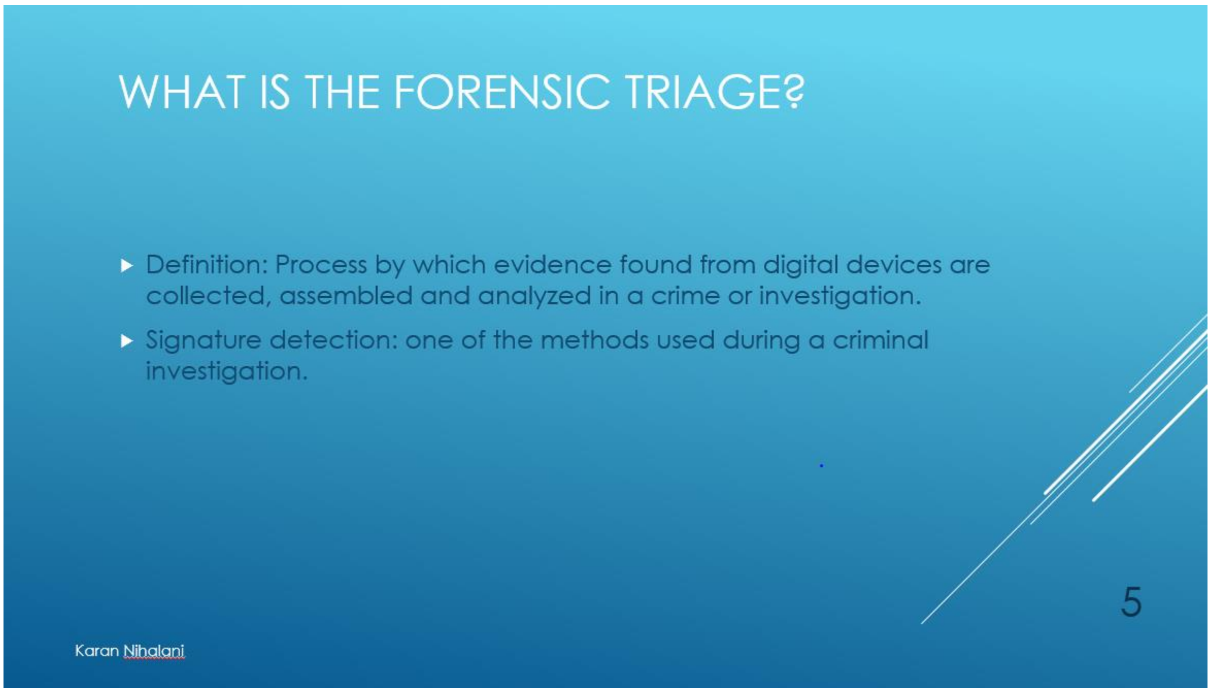
## PROJECT KEY OBJECTIVES

- Research & Understand signature detection and its uses.
- Design an experiment to compare file-based and block-based signature detection
- Testing the product software produced => runs efficiently.
- Discuss results in a logical manner, evaluating the work output.

4

Karan Nihalani

**Slide 5**



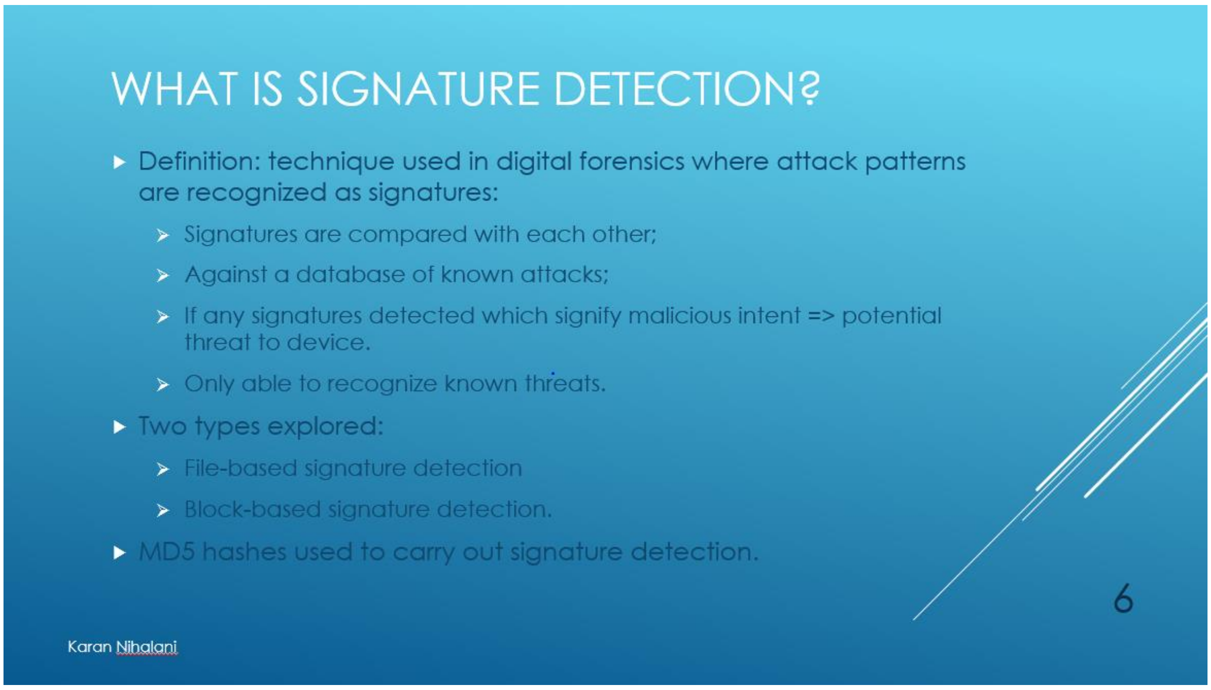
## WHAT IS THE FORENSIC TRIAGE?

- Definition: Process by which evidence found from digital devices are collected, assembled and analyzed in a crime or investigation.
- Signature detection: one of the methods used during a criminal investigation.

Karan Nihalani

5

**Slide 6**



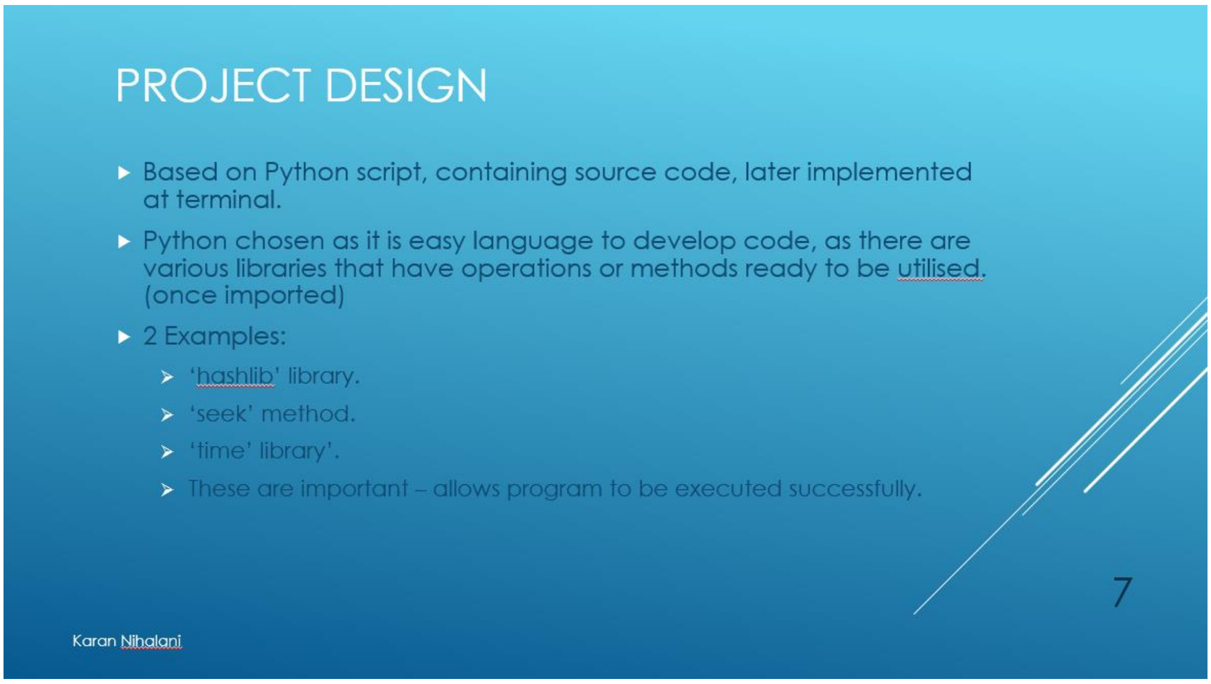
## WHAT IS SIGNATURE DETECTION?

- Definition: technique used in digital forensics where attack patterns are recognized as signatures:
  - Signatures are compared with each other;
  - Against a database of known attacks;
  - If any signatures detected which signify malicious intent => potential threat to device.
  - Only able to recognize known threats.
- Two types explored:
  - File-based signature detection
  - Block-based signature detection.
- MD5 hashes used to carry out signature detection.

Karan Nihalani

6

**Slide 7**



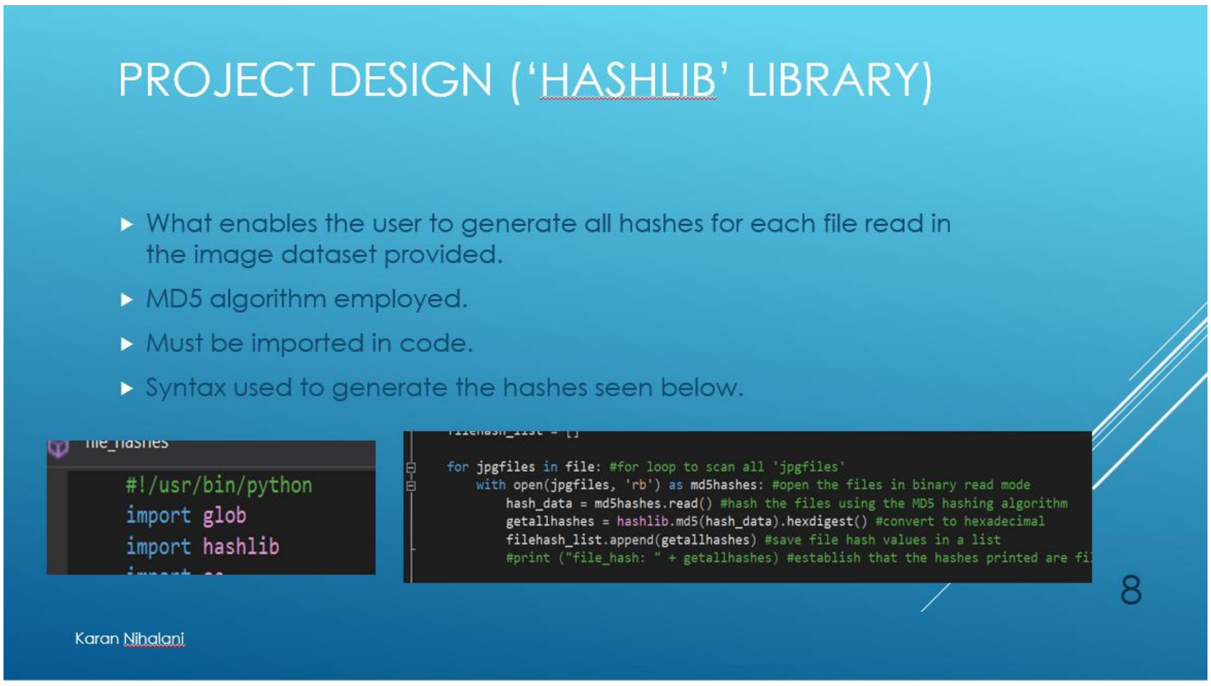
## PROJECT DESIGN

- Based on Python script, containing source code, later implemented at terminal.
- Python chosen as it is easy language to develop code, as there are various libraries that have operations or methods ready to be utilised. (once imported)
- 2 Examples:
  - 'hashlib' library.
  - 'seek' method.
  - 'time' library'.
  - These are important – allows program to be executed successfully.

7

Karan Nihalani

**Slide 8**



## PROJECT DESIGN ('HASHLIB' LIBRARY)

- What enables the user to generate all hashes for each file read in the image dataset provided.
- MD5 algorithm employed.
- Must be imported in code.
- Syntax used to generate the hashes seen below.

file_hashes

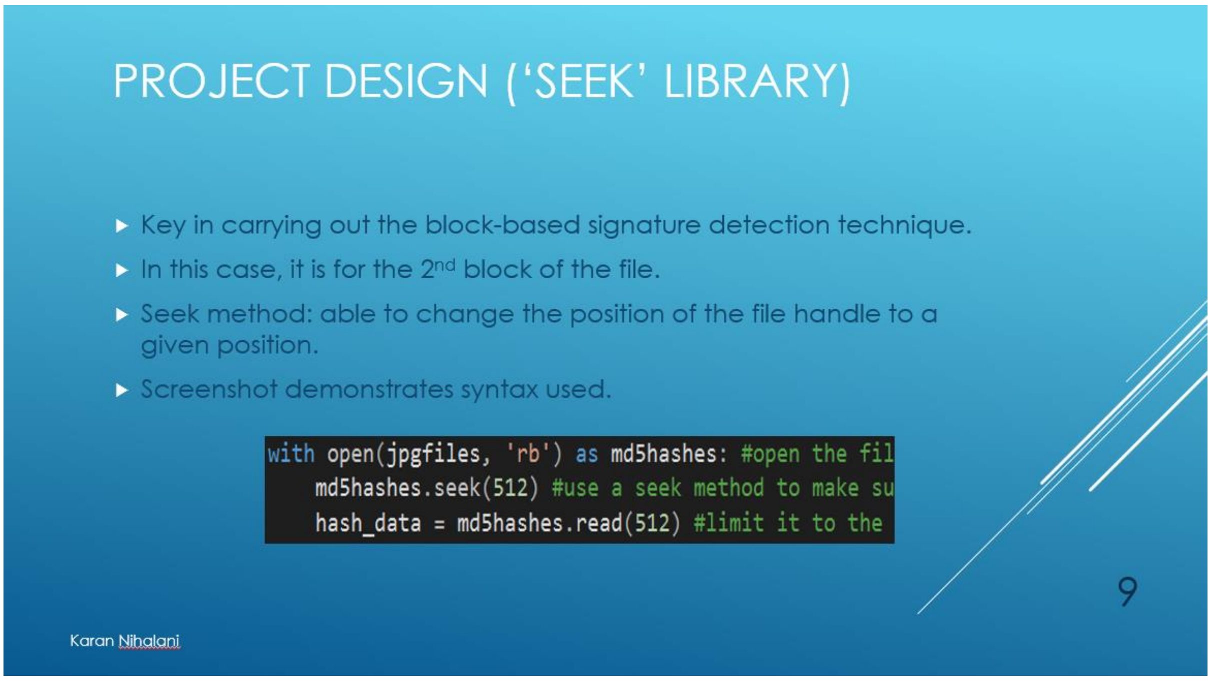
```
#!/usr/bin/python
import glob
import hashlib
import os
```

```
filehash_list = []

for jpgfiles in file: #for loop to scan all 'jpgfiles'
    with open(jpgfiles, 'rb') as md5hashes: #open the files in binary read mode
        hash_data = md5hashes.read() #hash the files using the MD5 hashing algorithm
        getallhashes = hashlib.md5(hash_data).hexdigest() #convert to hexadecimal
        filehash_list.append(getallhashes) #save file hash values in a list
        #print ("file_hash: " + getallhashes) #establish that the hashes printed are fi
```

8

Karan Nihalani

**Slide 9**



## PROJECT DESIGN ('SEEK' LIBRARY)

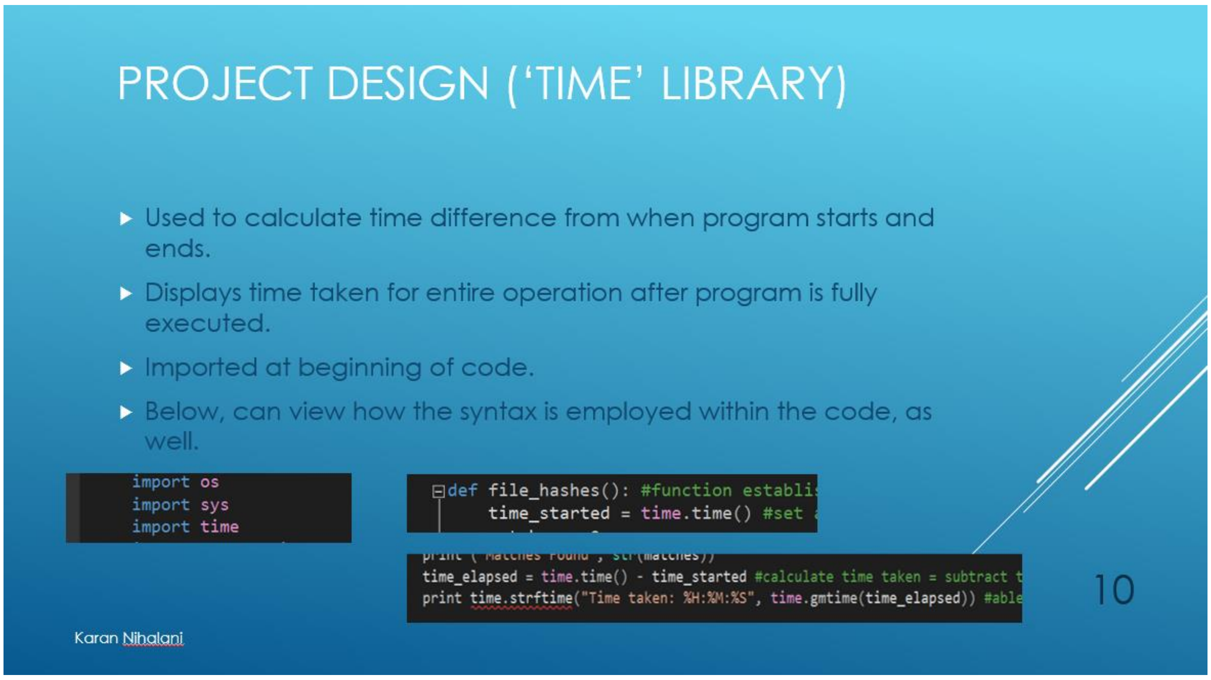
- Key in carrying out the block-based signature detection technique.
- In this case, it is for the 2nd block of the file.
- Seek method: able to change the position of the file handle to a given position.
- Screenshot demonstrates syntax used.

```
with open(jpgfiles, 'rb') as md5hashes: #open the fil
    md5hashes.seek(512) #use a seek method to make su
    hash_data = md5hashes.read(512) #limit it to the
```

Karan Nihalani

9

**Slide 10**



## PROJECT DESIGN ('TIME' LIBRARY)

- Used to calculate time difference from when program starts and ends.
- Displays time taken for entire operation after program is fully executed.
- Imported at beginning of code.
- Below, can view how the syntax is employed within the code, as well.

```
import os
import sys
import time
```

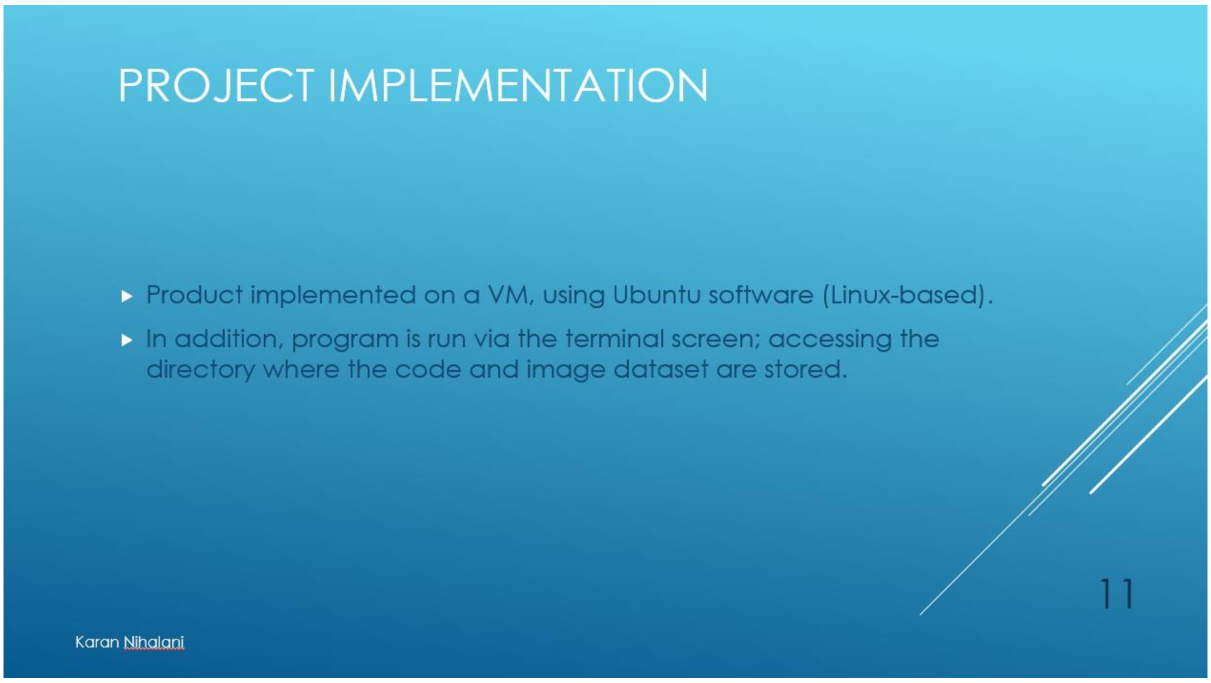
```
def file_hashes(): #function establi
    time_started = time.time() #set
```

```
print ("Matches Found", str(matches))
time_elapsed = time.time() - time_started #calculate time taken = subtract t
print time.strftime("Time taken: %H:%M:%S", time.gmtime(time_elapsed)) #able
```

Karan Nihalani

10

**Slide 11**



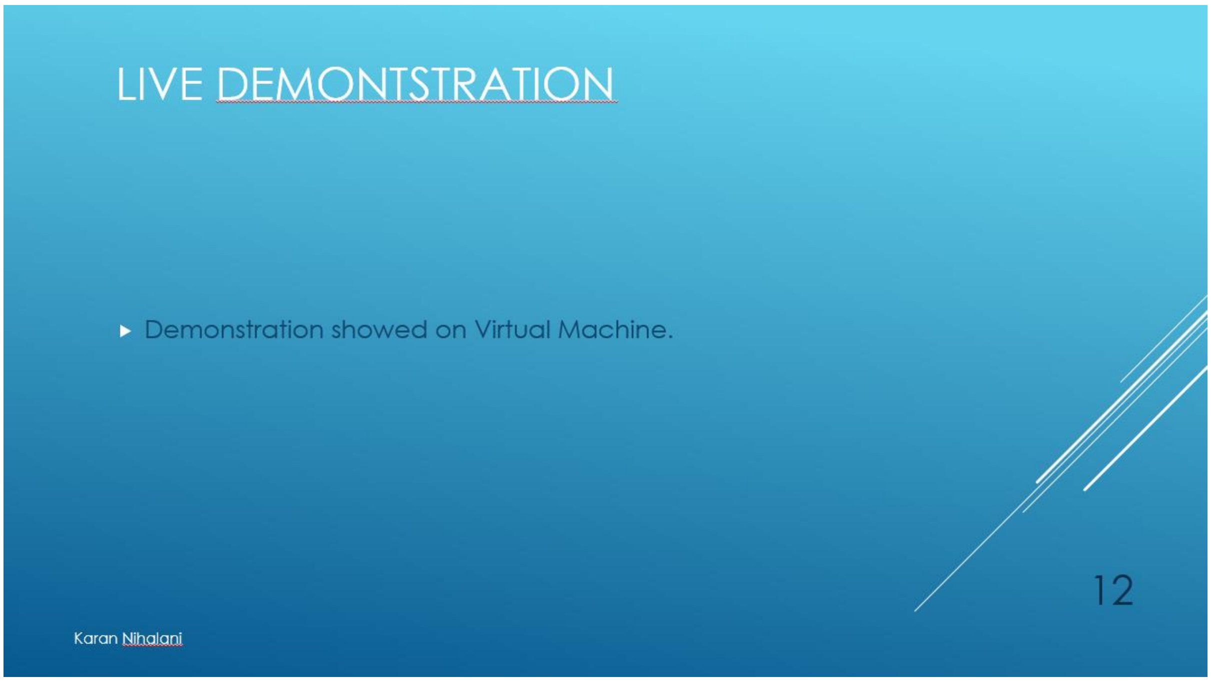
## PROJECT IMPLEMENTATION

- Product implemented on a VM, using Ubuntu software (Linux-based).
- In addition, program is run via the terminal screen; accessing the directory where the code and image dataset are stored.

11

Karan Nihalani

**Slide 12**



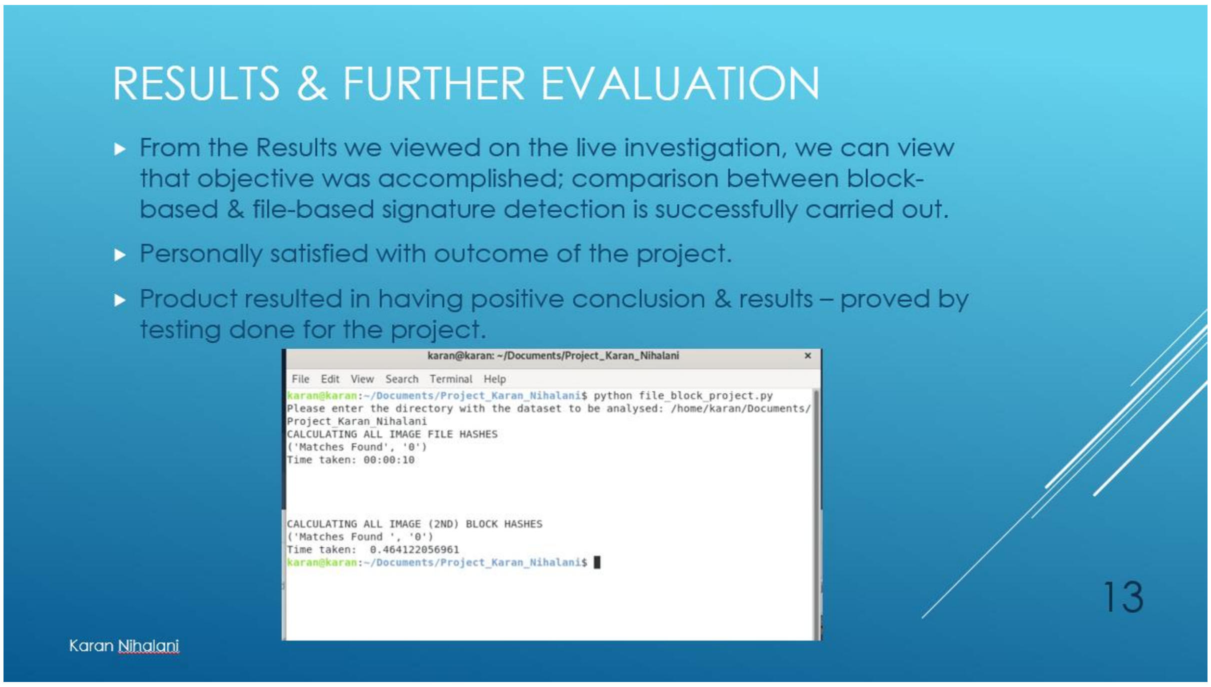
## LIVE DEMONTSTRATION

- Demonstration showed on Virtual Machine.

12

Karan Nihalani

## Slide 13



## RESULTS & FURTHER EVALUATION

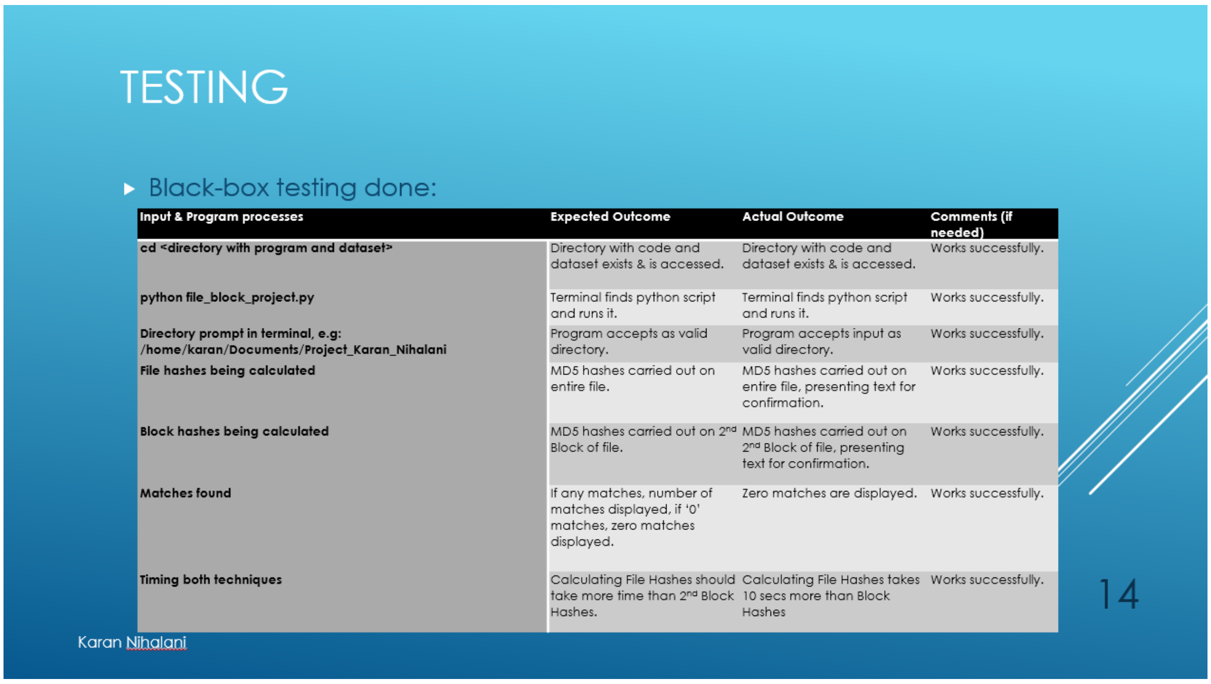
- From the Results we viewed on the live investigation, we can view that objective was accomplished; comparison between block-based & file-based signature detection is successfully carried out.
- Personally satisfied with outcome of the project.
- Product resulted in having positive conclusion & results – proved by testing done for the project.

```
karan@karan: ~/Documents/Project_Karan_Nihalani
File  Edit  View  Search  Terminal  Help
karan@karan:~/Documents/Project_Karan_Nihalani$ python file_block_project.py
Please enter the directory with the dataset to be analysed: /home/karan/Documents/
Project_Karan_Nihalani
CALCULATING ALL IMAGE FILE HASHES
('Matches Found', '0')
Time taken: 00:00:10



CALCULATING ALL IMAGE (2ND) BLOCK HASHES
('Matches Found ', '0')
Time taken:  0.464122056961
karan@karan:~/Documents/Project_Karan_Nihalani$ 
```

Karan Nihalani

13

## Slide 14



## TESTING

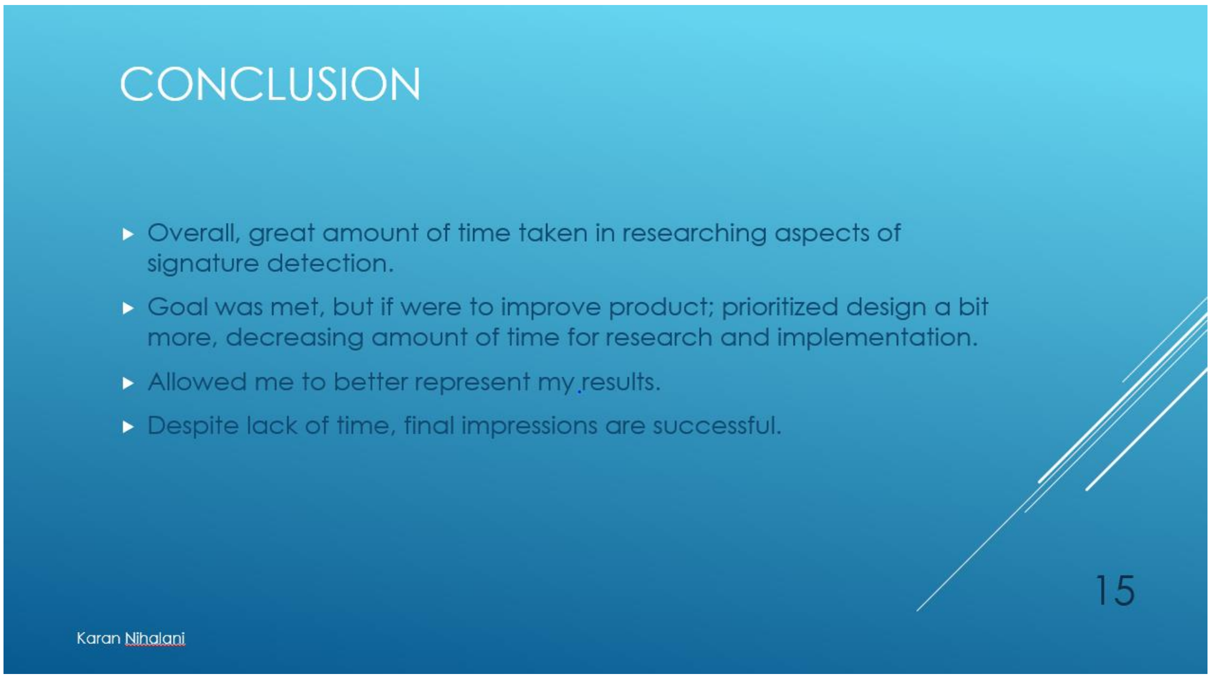
- Black-box testing done:

| **Input & Program processes** | **Expected Outcome** | **Actual Outcome** | **Comments (if needed)** |
|---|---|---|---|
| **cd <directory with program and dataset>** | Directory with code and dataset exists & is accessed. | Directory with code and dataset exists & is accessed. | Works successfully. |
| **python file_block_project.py** | Terminal finds python script and runs it. | Terminal finds python script and runs it. | Works successfully. |
| **Directory prompt in terminal, e.g: /home/karan/Documents/Project_Karan_Nihalani** | Program accepts as valid directory. | Program accepts input as valid directory. | Works successfully. |
| **File hashes being calculated** | MD5 hashes carried out on entire file. | MD5 hashes carried out on entire file, presenting text for confirmation. | Works successfully. |
| **Block hashes being calculated** | MD5 hashes carried out on 2nd Block of file. | MD5 hashes carried out on 2nd Block of file, presenting text for confirmation. | Works successfully. |
| **Matches found** | If any matches, number of matches displayed, if '0' matches, zero matches displayed. | Zero matches are displayed. | Works successfully. |
| **Timing both techniques** | Calculating File Hashes should take more time than 2nd Block Hashes. | Calculating File Hashes takes 10 secs more than Block Hashes | Works successfully. |

Karan Nihalani

14

**Slide 15**



## CONCLUSION

- Overall, great amount of time taken in researching aspects of signature detection.
- Goal was met, but if were to improve product; prioritized design a bit more, decreasing amount of time for research and implementation.
- Allowed me to better represent my results.
- Despite lack of time, final impressions are successful.

15

Karan Nihalani

## Appendix (Code)

```
#!/usr/bin/python
import glob
import hashlib
import os
import sys
import time
import matplotlib
import matplotlib.pyplot as plt
#import matplotlib.pyplot as plt

#def file_hashes():
       #path = "/home/karan/Project"
       #dirs = os.listdir(path)
       #BLOCKSIZE = 65536
       #hashing = hashlib.md5()

       #for file in dirs:
              #buff = afile.read(BLOCKSIZE)
              #while len(buff) > 0:
                     #hashing.update(buff)
                     #buff = afile.read(BLOCKSIZE)

       #print (hashing.hexdigest())

def file_hashes(): #function established for scanning all image files in the directory
       time_started = time.time() #set a timer - time started - used later on to
calculate difference in time later on
       matches = 0
       file = glob.glob("/home/karan/Project_Karan_Nihalani/*.jpg") #scans for all
".jpg" image files in a directory, using "glob" import
       print ("CALCULATING ALL IMAGE FILE HASHES")

       filehash_list = []

       for jpgfiles in file: #for loop to scan all 'jpgfiles'
              with open(jpgfiles, 'rb') as md5hashes: #open the files in binary read
mode
                     hash_data = md5hashes.read() #hash the files using the MD5
hashing algorithm
                     getallhashes = hashlib.md5(hash_data).hexdigest() #convert to
hexadecimal
                     filehash_list.append(getallhashes) #save file hash values in a
list
                     #print ("file_hash: " + getallhashes) #establish that the hashes
printed are file hashes

              with open('stored_hashes.txt', 'rb') as hashes_file:
                     f = hashes_file.read()
                     for i in filehash_list:
                            if i in f:
                                   matches = matches + 1

       print ("Matches Found", str(matches))
       time_elapsed = time.time() - time_started #calculate time taken = subtract the
time started from the actual time taken
       print time.strftime("Time taken: %H:%M:%S", time.gmtime(time_elapsed)) #able to
make it readable for user - represented in hours, minutes and seconds


       print("""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""")
```

```
        print(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""")
        print(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""")
        print(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""")

def block_hashes(): #function established for scanning all image files in the
directory
        matches = 0
        time_started = time.time() #set a timer - time started - used later on to
calculate difference in time later on
        file = glob.glob("/home/karan/Project_Karan_Nihalani/*.jpg") #scans for all
".jpg" image files in a directory, using "glob" import
        print ("CALCULATING ALL IMAGE (2ND) BLOCK HASHES")

        blockhash_list = []

        for jpgfiles in file: #for loop to scan all 'jpgfiles'
                with open(jpgfiles, 'rb') as md5hashes: #open the files in binary read
mode
                        md5hashes.seek(512) #use a seek method to make sure to start
scanning the hash from the 512th byte
                        hash_data = md5hashes.read(512) #limit it to the 2nd block of the
file
                        getallhashes = hashlib.md5(hash_data).hexdigest() #convert to
hexadecimal
                        blockhash_list.append(getallhashes) #save block hash values in a
list
                        #print("2nd_block_hash: " + getallhashes) #establish that the
hashes printed are file hashes

                with open ('stored_hashes.txt', 'rb') as hashes_file: #open the txt file
with stored hashes in binary read mode
                        f = hashes_file.read() #establish the read file as f
                        for i in blockhash_list: #for loop, looping through every block
hash stored in the 'blockhash_list' list
                                if i in f: #condition - if there are any matches in the
list
                                        matches = matches + 1 #keep on incrementing
        print ("Matches Found ", str(matches)) #print out message indicating that
matches have been found

        time_elapsed = time.time() - time_started #calculate time taken = subtract the
time started from the actual time taken
        print time.strftime("Time taken: %H:%M:%S", time.gmtime(time_elapsed)) #able to
make it readable for user - represented in hours, minutes and seconds

#def plot_graph():
        #x = [1000, 2000, 3000, 4000, 5000]
        #y = [1,2,3,4,5,6,7,8,9, 10]
        #y2 = [1,2,3,4,5,6,7,8,9]

        #plt.title("Time taken to hash blocks versus files")
        #plt.xlabel("Amount of Files")
        #plt.ylabel("Time")
        #plt.show()

def main():
        file_hashes()
        block_hashes()
        #plot_graph()


main()
```

# Ethics Form

## START HERE - Basic Information

This form must be completed for all student projects.

**Before you proceed**

Some activities inherently involve increased risks or approval by external regulatory bodies, so a proportional ethics review is not recommended and a full ethical review may be required.

These may include:

i. Approval from an external regulatory body (including, but not limited to: NHS (HRA), HMPPS etc.);
ii. Misleading participants;
iii. Research without the participants' consent;
iv. Clinical procedures with participants;
v. The ingestion or administration of any substance to participants by any means of delivery;
vi. The use of novel techniques, even where apparently non-invasive, whose safety may be open to question;
vii. The use of ionising radiation or exposure to radioactive materials;
viii. Engaging in, witnessing, or monitoring criminal activity;
ix. Engaging with, or accessing terrorism related materials;
x. A requirement for security clearance to access participants, data or materials;
xi. Physical or psychological risk to the participants or researcher;
xii. The project activity takes place in a country outside of the UK for which there is currently an active travel warning issued by the authorities (see info button);
xiii. Animals, animal tissue, new or existing human tissue, or biological toxins and agents.

**If any of these activities are fundamental to your project, please contact your supervisor to determine if a full application is required.**

This form must be completed for each research project which you undertake at the University. It must be approved by your supervisor (where relevant) PRIOR to the start of any data collection.

In completing this form, please consult the University's ACADEMIC ETHICAL FRAMEWORK for ethical research.

A1 Please confirm that you will abide by the University's Academic Ethical Framework in relation to this project.

[✓ Yes]
No

A2 Are you submitting this application as a learning experience, for a unit which already has ethical approval? (please confirm with your supervisor)

Yes
[✓ No]

A3 Student details

| Title | First Name | Surname |
|---|---|---|
| | Karan | Nihalani |

Email karan.nihalani@stu.mmu.ac.uk

A3.1 Manchester Metropolitan University ID number

17023122

A4 Supervisor

| Title | First Name | Surname |
|---|---|---|
| Mr | Robert | Hegarty |

| | |
|---|---|
| Faculty | Science and Engineering |
| Telephone | +44 (0)01612471541 |
| Email | r.hegarty@mmu.ac.uk |

A5 Which Faculty is responsible for the project?

Science and Engineering

A6 Course title

Computer Forensics and Security

A7 Project title

Block Based Signature Detection for Forensic Triage

A8 What is the proposed start date of your project?

01/10/2019

A9 When do you expect to complete your project?

30/04/2020

A10 Please describe the overall aims of your project (3-4 sentences). Research questions should also be included here.

- I will research how signature detection is employed during forensic triage
- Identify the key requirements of the triage process
- Evaluate different approaches to signature detection (Block and file based)
- Recommend the most appropriate approach based on the results of my evaluation.

A11 Please describe the research activity

- Carry out research and read about signature detection and its uses in today's world.
- Also, I will explain key concepts and different types of signatures that can be detected in the early stages of an investigation.
- Design an experiment to compare block and file based
- Identify appropriate software libraries to develop the experiment/software
- Identify the requirements for signature detection in digital forensics
- Design an experiment to evaluate and compare file and block based approaches to signature detection for digital forensic triage.
- To able to implement the appropriate and correct software to be able to conduct my experiment.
- Test the software to ensure it functions correctly
- Evaluate your software using a standardised data set, perform comparison between block and file-based approaches to demonstrate the benefits of block-based triage
- To be able to evaluate my project against the aims and objectives presented above.
- To document any future work I will have identified from working during this project.

A12 Please provide details of the participants you intend to involve (please include information relating to the number involved and their demographics; the inclusion and exclusion criteria)

No participants

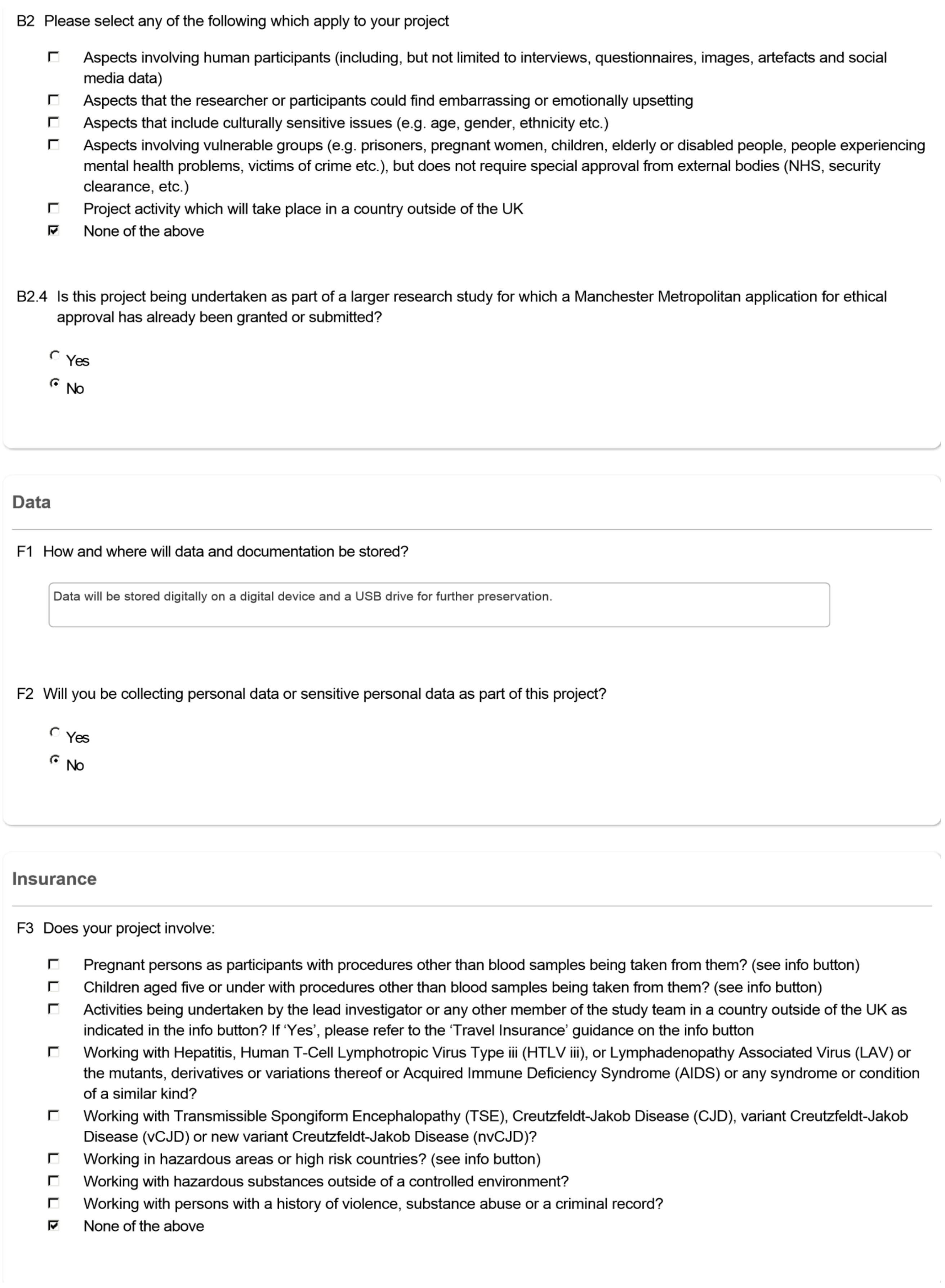
A13 Please upload your project protocol

| Tipo | Document Name | Nombre del archivo | Version Date | Versión | Tamaño |
|---|---|---|---|---|---|
| Project Protocol | Feasibility Study Draft3 | Feasibility Study Draft3.docx | 17/10/2019 | 3 | 74,4 KB |

## Project Activity

B1 Are there any Health and Safety risks to the researcher and/or participants?

Yes [✓ No]



B2 Please select any of the following which apply to your project

- ☐ Aspects involving human participants (including, but not limited to interviews, questionnaires, images, artefacts and social media data)
- ☐ Aspects that the researcher or participants could find embarrassing or emotionally upsetting
- ☐ Aspects that include culturally sensitive issues (e.g. age, gender, ethnicity etc.)
- ☐ Aspects involving vulnerable groups (e.g. prisoners, pregnant women, children, elderly or disabled people, people experiencing mental health problems, victims of crime etc.), but does not require special approval from external bodies (NHS, security clearance, etc.)
- ☐ Project activity which will take place in a country outside of the UK
- ☑ None of the above

B2.4 Is this project being undertaken as part of a larger research study for which a Manchester Metropolitan application for ethical approval has already been granted or submitted?

Yes [✓ No]

## Data

F1 How and where will data and documentation be stored?

Data will be stored digitally on a digital device and a USB drive for further preservation.

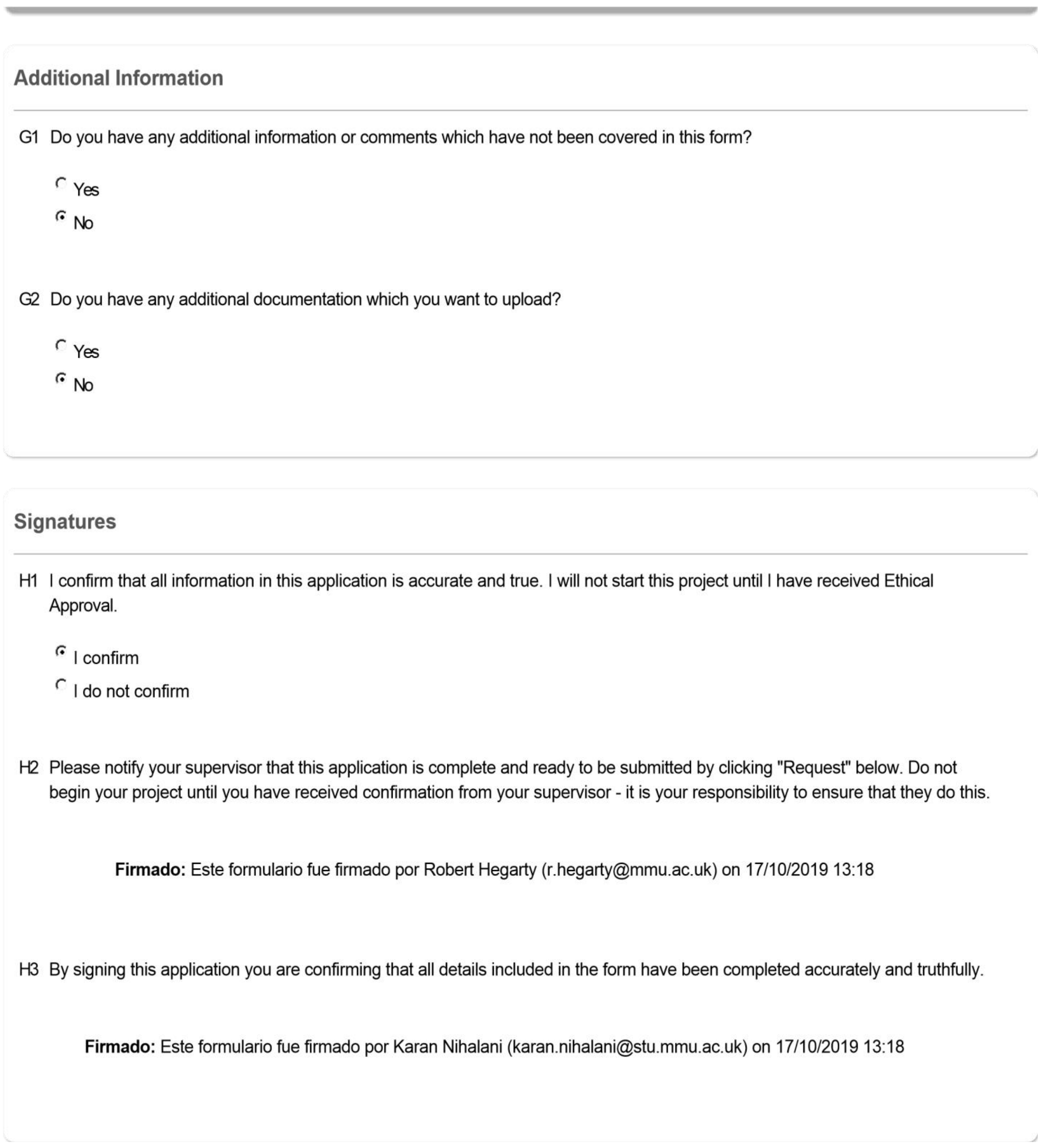
F2 Will you be collecting personal data or sensitive personal data as part of this project?

Yes [✓ No]

## Insurance

F3 Does your project involve:

- ☐ Pregnant persons as participants with procedures other than blood samples being taken from them? (see info button)
- ☐ Children aged five or under with procedures other than blood samples being taken from them? (see info button)
- ☐ Activities being undertaken by the lead investigator or any other member of the study team in a country outside of the UK as indicated in the info button? If 'Yes', please refer to the 'Travel Insurance' guidance on the info button
- ☐ Working with Hepatitis, Human T-Cell Lymphotropic Virus Type iii (HTLV iii), or Lymphadenopathy Associated Virus (LAV) or the mutants, derivatives or variations thereof or Acquired Immune Deficiency Syndrome (AIDS) or any syndrome or condition of a similar kind?
- ☐ Working with Transmissible Spongiform Encephalopathy (TSE), Creutzfeldt-Jakob Disease (CJD), variant Creutzfeldt-Jakob Disease (vCJD) or new variant Creutzfeldt-Jakob Disease (nvCJD)?
- ☐ Working in hazardous areas or high risk countries? (see info button)
- ☐ Working with hazardous substances outside of a controlled environment?
- ☐ Working with persons with a history of violence, substance abuse or a criminal record?
- ☑ None of the above



## Additional Information

G1 Do you have any additional information or comments which have not been covered in this form?

Yes [✓ No]

G2 Do you have any additional documentation which you want to upload?

Yes [✓ No]

## Signatures

H1 I confirm that all information in this application is accurate and true. I will not start this project until I have received Ethical Approval.

[✓ I confirm] I do not confirm

H2 Please notify your supervisor that this application is complete and ready to be submitted by clicking "Request" below. Do not begin your project until you have received confirmation from your supervisor - it is your responsibility to ensure that they do this.

**Firmado:** Este formulario fue firmado por Robert Hegarty (r.hegarty@mmu.ac.uk) on 17/10/2019 13:18

H3 By signing this application you are confirming that all details included in the form have been completed accurately and truthfully.

**Firmado:** Este formulario fue firmado por Karan Nihalani (karan.nihalani@stu.mmu.ac.uk) on 17/10/2019 13:18