Final Report

CM3203 – One Semester Individual Project – 40 Credits

A Machine Learning Approach for Memory Forensic Investigation

Author : Amna Mohd A Hameed

Supervisor : Michael Daley

Moderator : Luis Espinosa-Anke

Degree Programme: BSc Computer Science with Security and Forensics

School of Computer Science and Informatics

Cardiff University

2020

Abstract

Every process that executes in a digital system, has to run in memory at some point. Therefore, forensic analysis of memory is becoming increasingly important. The ability to detect memory images as benign or malicious immediately will enable analysts to prioritize their investigations. This will help to reduce the current backlog in forensic analysis caused by the current extraction procedures done on digital storage media. Despite numerous malware detectors being available to analyse digital storage media, they are often time consuming and not always efficient in detecting for obfuscation techniques used by malware or malicious activities. This project proposes a Machine Learning approach that applies to a memory forensics investigation procedure that utilizes an extracted log artefact from the memory image to classify memory. Different Machine Learning classification models were developed to identify the best performing classifier for the given dataset of psxview artefact log. Although there were project limitations and challenges, the proposed approach provided the initial results that show a proof of concept for a Machine Learning approach to resolve and reduce the need for additional processing, analsysing and investigating of memory images.

Acknowledgement

Alhumdulilah, I praise Almighty Allah for keeping me safe and in good health that was necessary to complete this project during the unprecedent time. I am grateful to my family and friends for their constant encouragement. support and prayers.

I am grateful to my supervisor, Michael Daley, for his ongoing engagement, motivation, and invaluable support and to my co-supervisor, Amir Javed for his guidance and sincere advice.

Thankful to both of my supervisors for keeping me in the right direction and for turning the initial concept I had into an exciting project.

Contents

Abstract

Acknowledgement

List of Figures

List of Tables

Chapter 1 : Introduction 12

1.1 Preface 12

1.2 Project Aims and Scope 13

1.3 Project Limitation and Constraints 13

1.4 Intended Audience 14

1.5 Document Layout 14

Chapter 2 : Background and Literature review 15

1. Introduction to Memory Forensics 15

1.1. Memory and its processing architecture 17

1.2. Memory Capture and Acquisition 20

1.2.1. Types of Memory Acquisition 21

1.3. Memory Analysis Methodologies 22

1.4. The Volatility Framework 24

1.5. Process and related main artefact logs 24

1.5.1. Process and Structure 24

1.5.2 Critical System Processes 26

1.6 Analsysing the Process using plugins 26

1.7 Limitation 27

2 Machine Learning 27
2.1 Machine Learning Techniques 28
2.2.1 Supervised Machine Learning 29
2.2.2 Unsupervised Machine Learning Model 30
2.3 Machine Learning Approach to Memory forensics investigation 30
2.3.1 Random Forest 31
2.3.2 Decision Tree 31
2.3.3 Support Vector Model 32
2.3.4 Naïve Bayes 33
2.3.5 Neural Networks Classification Model 35
2.4 Encounter possible outcomes of Machine Learning Models 36
2.5 Related Work 39
2.6 Programming Approaches 40
2.7 Research Question 41
Chapter 3 :Investigation and Analysis 42
3 Environmental Setup 42
3.1 Prerequisite 42
3.2 Analysis Approach 43
3.2.1 Acquisition of a compromised memory image 43
3.2.2 Main Analysis 44
3.2.3 Analysis of uncompromised image 57
3.3 Analysis Findings 59
3.3.1 Feature Selection 59
3.4 Data Collection 60
3.5 Conclusion 64

Chapter 4 : Specification and Design 65
4 Tool Requirements 65
4.1 Functional Requirements 65
4.1.2 Data Collection and Pre-processing requirement specification : 65
4.1.3 Machine Learning Classification requirement specification: 66
4.2 Non-Functional Requirements 67
4.3 System Implementation Architecture 67
4.3.1 System Design 68
4.3.2 Use Case Diagram: 68
4.4 Tool Development Methodology 69
Chapter 5: Implementation 71
5.1 Overview 71
5.2 Data Collection and Pre-processing Interface 72
5.2.1 Importing and Loading the Data 72
5.2.2 Data Pre-processing 74
5.2.3 Machine Learning Classification Models 77
5.3 Machine Learning Classification Interface 80
Chapter 6: Testing 83
Chapter 7 : Results and Evaluation 88
7.1 Classification Models Performance 88
7.2 Predication results analysis 89
Chapter 8: Future Work 91
Chapter 9 : Conclusion 92
9.1 Summary 92
Chapter 10: Reflection 93

References 95

List of Figures

Figure 1 : the main methodologies of digital forensics from [28] 16
Figure 2 CPU internal architecture structure from [3] 17
Figure 3 Illustration of multiple virtual address spaces sharing memory and secondary storage [3] 19
Figure 4 Virtual Address Paging from [14] 19
Figure 5 Process Structure from [2] 25
Figure 6 Process Organisation from [2] 25
Figure 7 General Machine Learning Structure from [12] 28
Figure 8 Machine Learning Classification 29
Figure 9 Unsupervised Learning Model 30
Figure 10 Decision Tree 32
Figure 11 Support Vector Classification 33
Figure 12 Bayes Theorem Equation from [35] 34
Figure 13 Bayes Theorem : instances from [35] 34
Figure 14 Human neuron cell from [37] 35
Figure 15 Artificial neuron from [37] 35
Figure 16 layers of Artificial neuron from [7] 36
Figure 17 Confusion Matrix from [7] 37
Figure 18 Accuracy Equation from [7] 38
Figure 19 Precision Equation 38
Figure 20 Recall Equation from [7] 38
Figure 21 F-Measure Equation from [7] 39
Figure 22 Decompressing memory image using Rekall 44
Figure 23 md5sum hash for acquired memory image 44
Figure 24 imageinfo output for compromised memory image 45
Figure 25 pslist output of compromised memory image 46
Figure 26 pstree output of compromised memory image 46
Figure 27 psscan output of compromised memory image 47
Figure 28 psxview output of compromised memory image 47

Figure 29 pslist output : System Protection and Anti-malware processes 50
Figure 30 pstree output : System Protection and Anti-malware processes 50
Figure 31 pstree output for suspected process 51
Figure 32 cmdscan and consoles command output 51
Figure 33 cmdline command output 52
Figure 34 malfind output for compromised memory image 53
Figure 35 Virus Total scanning for malwares 53
Figure 36 scanning a clean memory dump from compromised memory image 54
Figure 37 memory dump scan result of Virustotal 54
Figure 38 scanning other memory dumps from compromised memory image 55
Figure 39 last memory dump scan result from Virustotal 55
Figure 40 psxview output : initial suspected process from pslist 56
Figure 41 psxview output : reveal hidden process 56
Figure 42 Verified md5sum after analysis for compromised memory image 57
Figure 43 pslist output of uncompromised memory image 58
Figure 44 psxview output for uncompromised memory image 58
Figure 45 Use Case : Data collection and Pre-processing tool 68
Figure 46 Use Case : Classification tool 69
Figure 47 - Data Collection and Pre-processing Interface 72
Figure 48 select_N_folder() 73
Figure 49 select_N_folder() 73
Figure 50 load_pos_data() load_neg_data() 74
Figure 51 data_prep() 75
Figure 52 labelName() 76
Figure 53 save_data() 77
Figure 54 load_data() 78
Figure 55 set_features() 78
Figure 56 initialize machine learning model 79
Figure 57 evaluation metrics and output functions for Classifier Model 80
Figure 58 Classifier Prototype Tool Interface 81
Figure 59 getCSV () 81

Figure 60 classify () 82
Figure 61 Visualizer 82
Figure 63 Random Forest Confusion Matrix 89
Figure 64 Classification outcome of testing data against trained model 90

List of Tables

Table 1 Downloaded Benign memory images 62
Table 2 Downloaded Malicious memory images 63
Table 3 Test Case ID: A1 Creating Processed Dataset 84
Table 4 Test Case ID: A2 Support Vector Classifier 84
Table 5 Test Case ID: A3 Decision Tree Classifier 85
Table 6 Test Case ID: A4 Naive Bayes Classifier 85
Table 7 Test Case ID :A5 Random Forest Classifier 86
Table 8 Test Case ID :A6 Neural Networks Classifier 86
Table 9 Test Case ID: A7 Classifier Training and Testing 87
Table 10 Performance of various algorithms 88

This page intentionally left blank

Chapter 1 : Introduction

1.1 Preface

Nowadays, the field of digital forensic investigation is growing and advancing, as it plays a critical part in resolving cyber-crimes and tracing criminal's digital activities. With the rapid pace of technological change has enabled the law enforcement and forensic investigators to take into consideration of digital evidence to be used as evidence in criminal proceedings or to present in the court. However, various suspected devices are regularly collected from almost every crime scene as a source of intrusion; this has resulted in a backlog of devices and evidence to be examined forensically [9]. According to Goldberg's investigation news article, the police backlogs of numerous documented cases are being measured up to 12 months due to inconsistency delays of evidence examination and analysis with ongoing forensic workload [19]. In addition, obtaining evidences from the individual devices does not only require a lengthy process of both extraction and result analysis. But also, requires special knowledge and skills [28][19]. Hence, many forensic communities are nowadays more focused on prioritizing memory forensic analysis over the other forensics analysis areas. As memory dumps, or RAM, of the running system, preserves a significant number of volatile artefacts that provides relevant clues to investigators compare to non-volatile artefacts extracted from secondary storage devices [17]. Besides, many researchers are introducing new approaches to a digital forensic investigation that involves the use of machine learning (ML), a science branch of the artificial intelligence (AI) field, mainly to aid and assist investigators and analysts with advancing automation and detection for investigation procedures [20]. Therefore, to reduce the ongoing backlog of forensic investigation operations and avoid manual analysis and extraction process of unnecessary artefacts from memory images, there is a need to identify and classify acquired memory dumps and images as benign or malicious at the earlier examination and extraction stage.

1.2 Project Aims and Scope

The main aims of this project are to investigate and analyse memory dump for valuable hidden artefacts and using these extracted artefacts from a memory dump that can be applied to different Machine Learning models to classify a given artefact log of a memory image against trained Machine Learning Classifier Model. Also, identify the best performing Machine Learning model that can be used as a Memory Classifier.

The project scope mainly focuses on creating an automated assistant classification tool to examine and classify a memory artefact and investigate the performance of different classification Machine Learning Models acting as a Classifier for a memory artefact log. In addition, the expected outcome of this project is to have a set of results that shows a proof of concept of applying Machine Learning approaches to memory forensics investigation for classifying memory images. The main focus area of memory forensic investigation and analysis is investigating the system running process with sign detection of malicious activities using process-related logs and plugins.

1.3 Project Limitation and Constraints

As this project requires the digital forensics analysis it is important to consider a standard forensic workstation but due to limited resources an environment setup of Virtual Machine will be utilized as a suitable approach to conduct the investigation and analysis of a memory image. Also, it is important to maintain the data integrity of the evidence used to analyse as a forensic requirement to avoid any alteration and modification to original evidence during the analysis. Therefore, a cryptographic hash md5sum method will be used to ensure that the originality of the evidence data is not changed when memory image is analysed. Furthermore, the considerable encountering challenges from applying the Machine Learning approaches are data collection and pre-processing for Machine Learning Classification Models. As the necessity requirement of a medium-large sized dataset to pre-processed and used to train the Machine Learning models in order to retain effective results and useful insights. As this project will require creating a dataset utilizing extracted

log artefacts from memory images dataset, the dataset scale is expected to be limited size due to limited storage resources to store memory images dataset.

Considering all these limitation and constraints to the fixed timeframe of the project may have a medium impact on completing all the project stated aims and expected outcomes. As this project will be conducted as a university project and the initial results will be presented as a basis for further research.

1.4 Intended Audience

The intended audience and beneficiaries from this project are the researchers, analysts and investigators and individuals who are interested in developing an automate applications utilizing Machine Learning approaches for Memory classification and malware detection.

1.5 Document Layout

The rest of the document is structured as follows, **Chapter 2** outlines the background search and literature review of Memory forensics and related technical aspects of memory and analysis methodologies, Machine Learning basics and approaches, evaluation measures of Machine learning models as classifiers and related work. **Chapter 3** defines the approaches of the memory analysis methodologies with main artefact findings and data collection. **Chapter 4** layouts design and system specification requirements of the developing tool. **Chapter 5** details in programming level of the implementation of the developing tool and Machine Learning Models. **Chapter 6** features testcases that were performed for the developed tool , **Chapter 7** presents the initial results of the Classifier Models, **Chapter 8** addresses arising future work for the project ,**Chapter 9** : concludes the project main results and findings and **Chapter 10** details the learning experiences from undertaking this project.

Chapter 2 : Background and Literature review

This chapter provides an overview of technical concepts and the background context of the problem and approaches in more depth with reference to related literature. The chapter firstly introduces the principles and technical aspects related to memory forensics and outline memory's architecture and its processing including memory acquisition, analysis methodologies and main artefact. Furthermore, the chapter provides contexts and theory associated with Machine Learning, Machine Learning classification algorithms and address as an approach to memory forensics investigation with the awareness of related work. Finally, the chapter concludes with the approach to be adopted for the project and the research questions.

The theories and terminology reviewed in this chapter are referred to frequently throughout the remainder of the document.

1. Introduction to Memory Forensics

Over 15 years ago, the need of memory analysis in digital forensics world was primarily highlighted during the Digital Forensic Research Workshop (DFRWS) conference in 2005 as a challenge and the main objective was to motivate research of new techniques and advance tool developments in the field of memory forensic [16]. Subsequently, the constant researches on the topic provided a critical consideration of memory forensics and useful insights regarding the available volatile artefacts that are found in memory for the forensics investigation and analysis process [16][31].

Nowadays, memory forensics is arguably becoming one of the major areas focused for any forensic investigation. The main memory or RAM are used as primary storage to store the running system's executed programs, recent open files, threads and logs and the processed data [17]. These memory's volatile content is highly considered, as it provides valuable insights for forensics investigators and analysts of a compromised system to determine for any suspicious activities as malware or network intrusion [23][22]. In general, memory forensics is a domain of digital forensics and the process of memory forensic is mostly adapted and based

on the fundamental methodologies of digital forensics [11]. The process involves six main stages of identification ,collection and preservation, examination, analysis, documenting and recovery of evidential artefacts from any electronic devices that stores data that is relevant to an investigation. The sequence of the digital forensics' methodology is shown in the figure [29].

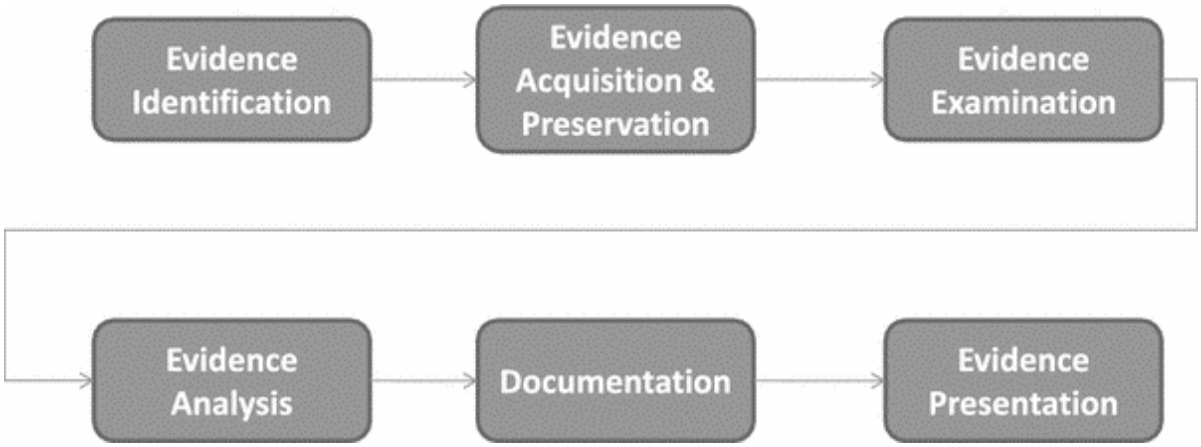

Figure 1 : the main methodologies of digital forensics from [28]

In terms of memory forensics, the set of digital forensic methodologies are relatively applied on the system's volatile physical memory with the consideration of the system state at time of identification and acquisition [1]. The main reason of the system state matters as it is the only accessible way to collect and acquire memory capture when system is powered on . As in the past, capturing memory images and memory dumps were often easily manageable as to limited size range of physical memory. However, the recent increase in size of physical memory of modern computers has impacted inefficiently for entire manual acquisition, examination and analysis of memory image as it requires specialized understanding knowledge of contextual information of memory and data structures and use of the associated tools that are compatible [22]. Most of the memory's artefacts are based on memory processes with virtual addressing associated to physical addressing. The basic principles related to memory data structure and how its function processes will be reviewed in the following sections.

1.1. Memory and its processing architecture

Basically, each computational device is composed with two principle components that performs computational processing and basic instructions of a system that are the physical memory and the processor [3]. These components are considered to hold forensic value, as the processor includes programs executions and the processes of central processing unit (CPU) of the whole computer system. Whereas the volatile physical memory, it consists of temporarily stored data related to the processor and executed programs of the active system. In term of the modern computer system architecture, CPU is often stated as a processor, that is indirectly accesses and requests the main memory (RAM) via Memory Controller Hub for instructional commands to execute and process the data. The following diagram Figure 2 shows the basic internal architecture of a computer system and how the system's controllers and processors are relatively interconnected to the main memory [3]

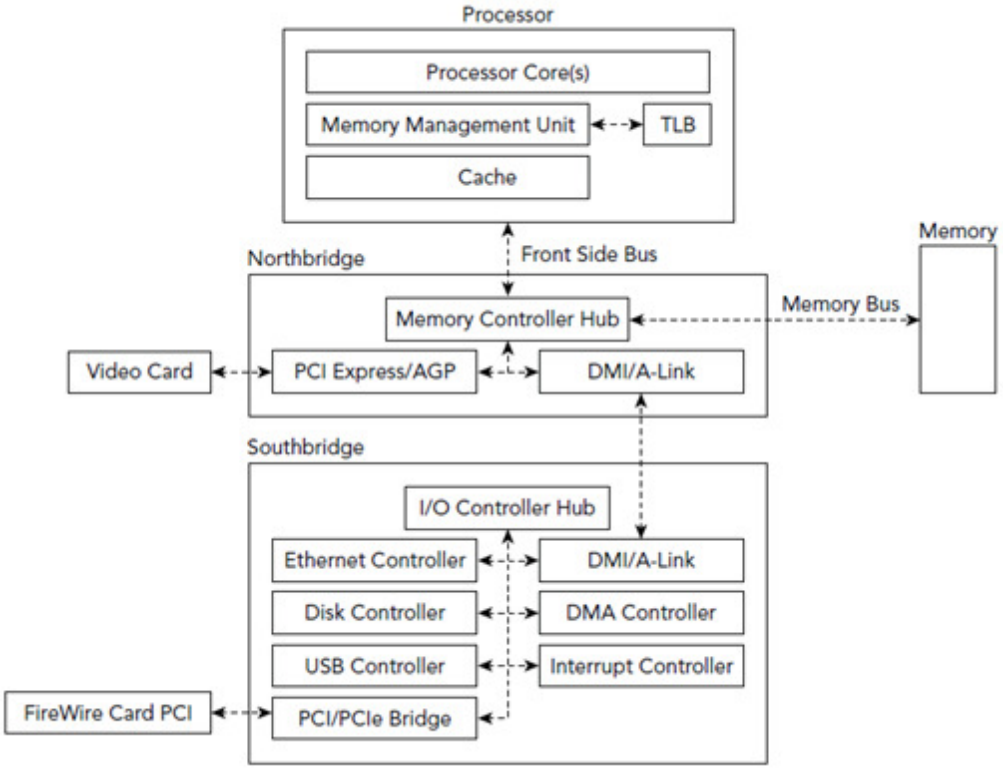

Figure 2 CPU internal architecture structure from [3]

Moreover, Memory is commonly known as random-access memory (RAM), particularly for its characteristic of random-access time in any order for the storage and location of the data. Memory also characterized as the most volatile data in a computer system, as its data and the content are lost when system state is off [22]. In addition, memory has the capability to collect, access and transfer data between the input/output (I/O) controllers and processor as shown in Figure 2 via the connected units of Northbridge and Southbridge. This indicates that the information regarding external connected devices and storage media resident in the main memory and can be acknowledged theses information for forensic investigation and analysis[3].

Besides, to manage processes and threads of the main memory are mainly performed by memory management unit (MMU) , virtual address translation and the memory data structures [26].The memory management is mainly the organization of the physical memory (RAM) for the allocation of the system's multitasking processes and operations. A special addressing scheme is used between the main memory and CPU for accessing the data and instruction to be executed. Virtual address is used, and it refers to address space in virtual memory for a process. Whereas the main memory has corresponding address for the virtual address which are addresses that the processor requests for accessing physical memory known as physical address space. The MMU is constantly used along with the memory manager for the translation of the processor's virtual address to the physical address using the translation lookaside buffer (TLB), also known as the MMU translation table for a given translation [3].

Each of the running processes is mapped with a private section of the virtual memory, that appears to provide more memory access than the actual physical memory space for the process [23]. Basically, virtual memory is used as allocation scheme for the process or the application when system is running short of the memory. The memory manager is responsible for transferring regions of memory to secondary storage to free up space in physical memory to allow the data to be exchanged between primary and secondary storage. If a thread accesses a virtual address that has been moved to secondary storage, that data is then brought back into physical memory [23]. This interaction is represented by the diagram Figure 3 that shows how

the virtual address are randomly mapped between the virtual memory, the main memory and secondary storage [3].

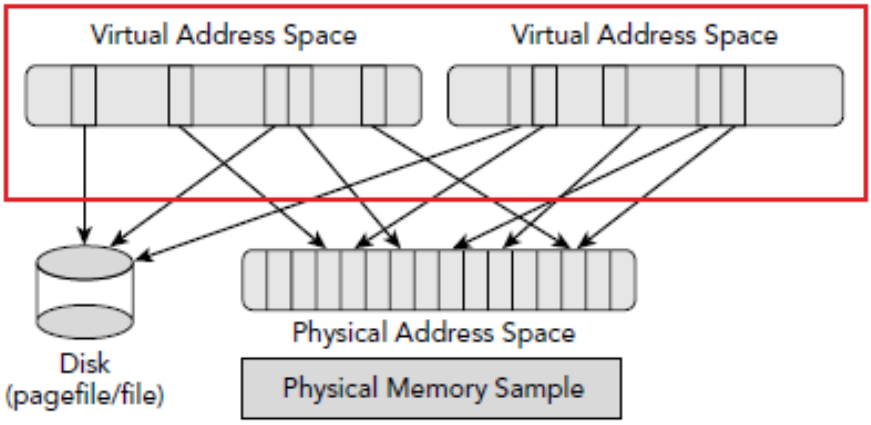

Figure 3 Illustration of multiple virtual address spaces sharing memory and secondary storage [3]

In terms of physical memory address data structure , it is consist of pages and table page that stored in main memory. In general, a memory page (page) is a fixed-length contiguous block of virtual memory, described by a single entry in the page table, that maps virtual pages to physical pages[27] Figure 4. [3][14].

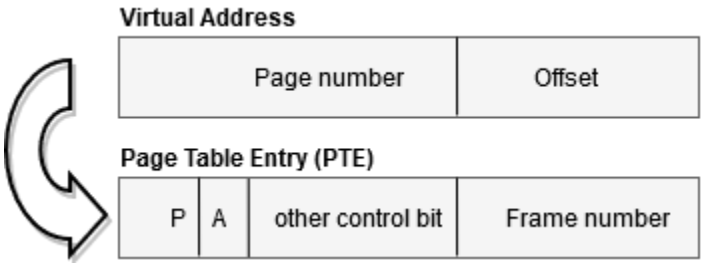

Figure 4 Virtual Address Paging from [14]

Paging often performed by the operating system for temporarily swapping memory contents out to secondary storage to free up the required space in physical memory. The page file or swap file store pages from the memory that have been swapped as not needed or set another

page [21] By default, the page file is stored in the operating system installation root, in a file called pagefile.sys is the pages once were in physical memory and might be active. This internal process of each application shows that all systematic operations are carried in main memory. With regards to forensics point of view, the list of memory addresses associated with the process are useful to reconstruct the whole memory space and get the data out that is missed [21]. This section has provided an overview of the main elements of memory management unit and how memory address is processed within Windows operating systems.

1.2. Memory Capture and Acquisition

The need for the memory acquisition has increased as more information are being stored on computer's memory that involves in cybercrimes and network attacks [11]. Memory acquisitions is highly prioritized for any identified live compromised computers as it contains extremely volatile data in memory which is ranked as top in order of the data volatility. The memory images and snapshots are only captured from a running system as once the system is turned off completely or rebooted the memory's content fades away [22][23]. Therefore, the first responders are trained and instructed by following the guidelines of the 'Association of Chief Police Officers (APCO)' to securely collect and handle evidences upon investigation. The four APCO guidelines [15] are listed as following :

- ✓ **Principle 1:** No action taken by law enforcement agencies or their agents should change data held on a computer or storage media which may subsequently be relied upon in court.

- ✓ **Principle 2:** In exceptional circumstances, where a person finds it necessary to access original data held on a computer or on storage media, that person must be competent to do so and be able to give evidence explaining the relevance and the implications of their actions.

- ✓ **Principle 3:** An audit trail or other record of all processes applied to computer based electronic evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result.

- ✓ **Principle 4:** The person in charge of the investigation (the case officer) has overall responsibility

These guidelines are essential to the first responders to carefully deal when acquiring memory images from the live compromised systems as any action performed as by clicking mouse or keyboard might activate any malicious code or malware that could erase the system automatically [15]. Therefore, it is essential requirement for first responders to secure the crime scene completely and conduct memory acquisition process initially before any other forensics collection and acquisition process[17]. In addition, avoid unnecessary interaction while acquiring memory image as the acquisition process is performed on live system in timely manner and any interruption would result in a corrupt version acquired image. Memory acquisition be performed based on two methodologies either Hardware-based or Software-Based. The main principle considered when acquiring the image is to use the least invasive approach possible as to less footprints on the running memory[1] [15].

1.2.1. Types of Memory Acquisition

Hardware based involves direct physical connection to the memory using hardware as write blocker and cable connections. These provide physical memory backup instead of logical. Whereas Software-based involves use of verified and tested toolkit to acquire memory bit-to-bit images and snapshots. Often these commercial and open-source tools are risky as it can alter the memory and affect its integrity [11]. As whatever actions performed by the investigator at the time of acquisition is firstly executed in the memory and can alter the system's current running processes that are likely pointed be forensically important [1]

In addition, one of the main advantages of memory acquisition of memory images are fairly smaller in size compared to the acquired images from non-volatile secondary storage and other storage devices as it requires less time for extracting and taking snapshot captures, whereas non-volatile devices require lengthy process only for acquiring the image as there size measures up to 16 GBs to 2TBs and more [1]. After acquiring and capturing the memory, a raw bit-to-bit memory image copy is produced and there are many acquired memory images formats. The main formats of acquired memory image are as dd file (.dd) , memory image

(.img) or as memory dump (.memdump). In addition, there are different types of memory dumps are executed by the operating system as Windows Crash Dump, which is created when system crashes and it is used to identify cause of the system crash e.g. blue screen crash and kernel crash. Another memory dump that is executed by system is Windows Hibernation File, which is created during the hibernation process and it stores as (hiberfil.sys) where copy of memory that the system dumps to disk and often these types of the memory dump images are considered for analysis [1]. As to commonly used memory acquisition tool for Windows operating systems is WinPmem, which is one and only open-source available to date. In addition, few analysis tools and frameworks that only supports memory images and dumps formats as common used raw formats are dd, img that are widely supported and there often several to acquisition tools that can be used to convert raw memory image format into another as The Volatility Framework [1][26].

When the memory captures are being collected and acquired, they are assigned with a fixed cryptographic hash code as MD5 hash for the integrity and data validation of the original evidences [17][22]. There are various hash calculating tools and most used is md5sum. It is a Linux sum tool that is used to create a checksum hash value based upon the content of the entire memory image using traditional algorithms. The hash is used before and after the analysis of memory images to identify for any alteration and modification to original memory image. If hash checksum returned same as before conducting the analysis, then integrity is maintained and validated through the analysis otherwise getting invalid MD5 which indicates that some alteration occurred to the original memory image during the analysis [22].Therefore ,it is important when investigating and analsysing the evidence without alternating the original content of the evidence and maintain perform the checks as the principles of the APCO[15].

1.3. Memory Analysis Methodologies

There are several ways to conduct the analysis of a memory dump as it consists range of different artefacts. But it is recommend for examiners and analysts to follow the basic methodology procedures when conducting memory examination and analysis in order ensure that all potential suspected artefacts and evidences are revealed for an incident investigation [11][17]. Most of the analysis and investigation procedures are conduct in specialized digital

forensics labs utilizing the forensics workstation and memory analysis tools. Besides, each analyst applies different analysis techniques depending on the investigation scenario [22]. Despite following a specific methodology, it is also important to consider covering different areas of memory analysis which ranges from System Process, Registry logs, Networking, Services, Kernel and Rootkits[6]. According to the SANS Computer and Incident response Institute guide cheatsheet [34] for the pure analysis of compromised memory is generally accomplished using the different plugins of Volatility Framework in following six steps that cover the major areas of for the analysis are listed in the following list :

A. Identify rouge process
    a. This area is mainly focuses to listing the system running process in the memory at time image was acquired .
B. Analyze process dlls and handles
    a. This area of analysis consists of revealing a list of related dlls, process security identifiers and handles for a selected running process
C. Review network artefacts
    a. This area presents the network related artefact as open and closed TCP connections, ports, and sockets as well as source and destination IP address
D. Look up Evidence of code Injection
    a. This area looks up for the areas in low level for signs of the code injection for specified process and offset address and dumps the infected areas to be analysed further
E. Check for signs of rootkits
    a. Looks up for hidden process and checks the memory process in cross validation view as well as looks up for the API dlls for specified process
F. Extract Processes, Drivers and Objects
    a. This area mainly used for extracting different artefacts as memory dumps for the specified process , driver, and objects that considered for further analysis.

In addition, Memory Forensics Practical guide from the Computing science and Mathematics of University of Sterling, where the main areas of the initial analysis of a compromised memory image includes inspecting the operating system versions, viewing process and

network connections, searching through memory process that can be performed using the Volatility Framework [13].

1.4. The Volatility Framework

The most widely used memory forensics platform for memory acquisition and analysis is known to be Volatility Framework. This tool is beneficial to analysis captured and imaged volatile memory for valuable information about the runtime state of the system, provides the ability to link artefacts from traditional forensic analysis [6][23] . Also, the tool provides range of plugins to analyse the memory artefacts of main 6 areas as mentioned earlier. In addition, this tool framework is python based and is also used as python library [4].

At the initial analysis of a memory image, it is important to distinguish the system running process. The following section will describe briefly about the system process as artefact along with process-related artefacts logs that are used for the analysis using the Volatility Framework plugins.

1.5. Process and related main artefact logs

1.5.1. Process and Structure

As to basic analysis methodology, it is important to acknowledge all the existed artefacts that can be found in a memory dump. All the artefacts and running processes in the system share a common origin that they all consist EPROCESS, which is the structure that Windows Operating System uses to represent or to call a process [2]. As each EPROCESS is consist of a main attribute that represent the allocation of the process in the memory region by the virtual memory space, that is unlinked from other system running processes. Additionally, the memory space is also consists of input actions performed by the system user for a process and it includes list of process executable, loaded modules and SIDs and user privileges and threads that Windows system organizes and distributes through Virtual Address Descriptors VAD, which is a paging method to load pages in memory [2]. The following diagram Figure 5 shows a basic process with associated attributes

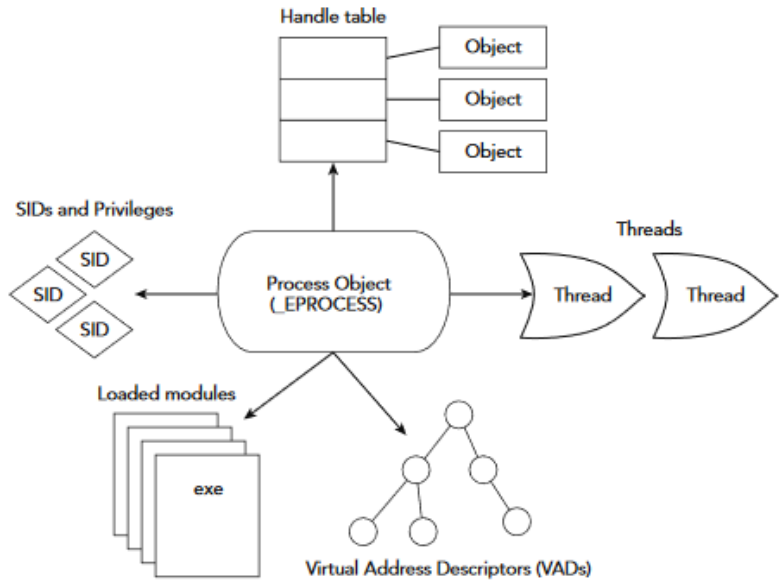

Figure 5 Process Structure from [2]

Generally, the process is organised with single linked which creates a double linked lists of processes structure. Where the a _LIST_ENTRY structure called ActiveProcessLinks (header), which contains two main elements: a Flink, forward link, that points to the header of the following process, and the Blink ,backward link, that points to the header of the former process. All Together, these linked structures build a double linking chain of processes as shown in Figure 6 [2].

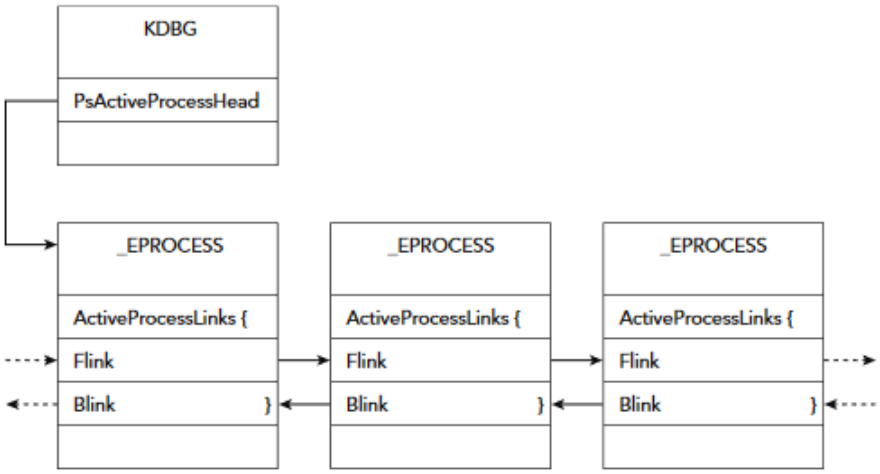

Figure 6 Process Organisation from [2]

1.5.2 Critical System Processes

Besides, it is important to distinguish the actual system running process with the applications running on the system memory RAM. Therefore, the following lists the essential system processes that runs normally in clean system memory:

- **System** – represents the default system process which includes threads that run-in kernel mode and it is usually represented as PID 4 [2].
- **csrss.exe** - The Client/Server Runtime Subsystem for creating and deleting processes and threads [2].
- **service.exe** – represents The Service Control Manager(SCM) process, manages Windows services and maintains a list of system services in a private reserved memory space [2].
- **svchost.exe**: represents a multiple shared host processes provides a space for DLLs that implement services [2] .
- **lsass.exe**: represents The Local Security Authority Subsystem process mainly responsible for security policy and verifying passwords and creating tokens [2].
- **winlogon.exe:** represents the interactive logon prompt process [2].
- **explorer.exe**: represents the Windows Explorer process and which represents a range of user interactions such as folder [2] .
- **smss.exe**: represents session manager process that is mainly responsible for managing and creating the sessions [2] .

There are more background system services and application processes other than the mentioned system process and it mainly differs from different versions of Windows Operating Systems.

1.6 Analsysing the Process using plugins

The main artefacts of process-related logs that are utilized to identify the system running processes are extracted using the Volatility Framework tool's plugins [2][4]. These plugins commands are listed below:

- **Pslist**, lists the processes and prints a summary which includes only active process and does not any terminated or hidden processes[2].
- **Pstree**, lists the **pslist** in a tree view, which reveals process relationships as the parent process and child processes [2].
- **Psscan**, lists terminated and hidden processes [2].
- **Psxview**, locates processes using alternate process listings, using the cross-reference different sources of information and reveal malicious discrepancies.[2]

Further context and details of each of these plugins and logs will be detailed in Chapter 3.

1.7 Limitation

Some of the considerable challenges of conducting the Memory forensic analysis is that it requires a standard forensic workstation for acquiring, analysing and extracting artefacts from memory images. Also, when conducting acquisition and analysis ,if any incorrect process of extraction occurred while acquiring a memory image due multiple command runs or technical errors mapping of the binary representation data then inaccurate reading of the memory dump's data results in to incorrect extraction and processing of the outcome results [22][11].

2 Machine Learning

In recent years, the Machine Learning approaches are significantly becoming a high demand in many industries and businesses for the purpose of obtaining meaningful data insights and automation analysis [8]. Machine Learning (ML) is one of the emerging domains that is highly associated with the research field of Artificial Intelligence (AI). The concept of Machine Learning in conjunction with AI, is referred as field of study that gives computers the ability to learn without being explicitly programmed [33] and according to definition of Tom M. Mitchell for ML is refers as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E ."[39][7].

Clearly, Machine Learning can be defined as the ability of a computer program based on computational algorithms that can automatically learns the underlaying patterns from given information and data to provide useful insights [8]. Besides, Data Knowledge Discovery processes which includes Data Mining is crucial in Machine Learning programs, as the knowledge extraction of known and unknown data from the large-scale data source are utilized as basis of data insights and further exploration for key decisions from the given data. Numerous applications are widely adapting Machine Learning techniques such as stock prediction, credit scoring, smart medical checks, malware detection, and many more as the applications are beneficial in delivering useful predictive analysis [7]. The following diagram shows as a general scheme for Machine Learning as classification approach Figure 7

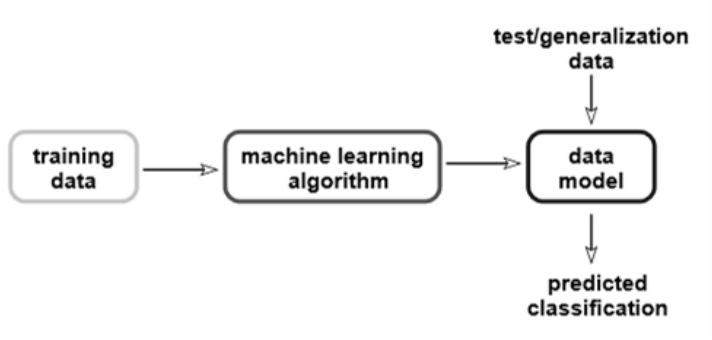

Figure 7 General Machine Learning Structure from [12]

2.1 Machine Learning Techniques

Machine Learning provides various approaches as classification, regression ,pattern recognition and many more that are constructed based on mathematical and statistical algorithms which processes the outcome knowledge from a given data of a sample dataset [8] .In general ,there are several types of Machine Learning models that ranges from Supervised Learning, Unsupervised Learning, Semi-supervised Learning, Reinforcement Learning and Deep Learning [7]. Theses Machine Learning models are used for different purposes and each type of learning model is used to perform either descriptive or predictive analysis depending on the chosen algorithm, type of

analysis required to solve the problem with consideration of type of dataset used for the analysis. But the most popular Machine Learning models that are used commonly : Supervised learning model and Unsupervised learning models. In the following section will discuss the two main categories of machine learning models with their uses [5][8].

2.2.1 Supervised Machine Learning

Supervised model refers to an algorithmic learning model that infers the underlaying patterns and insights relationship between the labelled data and target values of unlabelled data that is subject to predication outcome [7]. Considering a malware detection example based on the Machine Learning classification approach as shown in the Figure 8 , Where a labelled training dataset of the files is used with labels of benign and malicious for learning and training task of the model. The labels are used to identify each data of the dataset. As model is trained and adapted the generalized pattern and feature knowledge from the given dataset's data. The model applies classification function on the test unseen data, which is unlabelled data, where it classifies and predicts according to the supplied labels and trained dataset and produce possible outcome prediction [7][25]

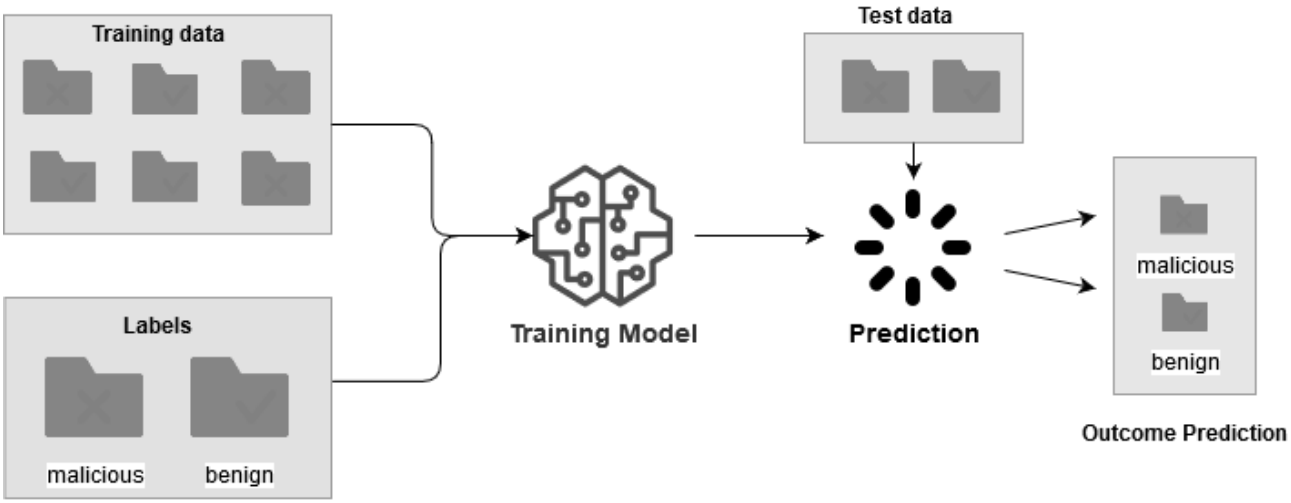

Figure 8 Machine Learning Classification

2.2.2 Unsupervised Machine Learning Model

In terms of the Unsupervised learning model, it is only requiring unlabelled input dataset. This learning model utilizes clustering and grouping algorithms that can automatically find regularity from the unlabelled data without human interference and it filters and groups the unlabelled data into small clusters of similar features and provides each cluster with a suitable label based on the acknowledge similarity patterns from the dataset as shown in Figure 9 [5]. Clearly, The Unsupervised learning model is considered to be useful when labelling large dataset [7] Some uses of the Unsupervised model are found in the areas of data compression, outlier detection, classification, and human learning [7].

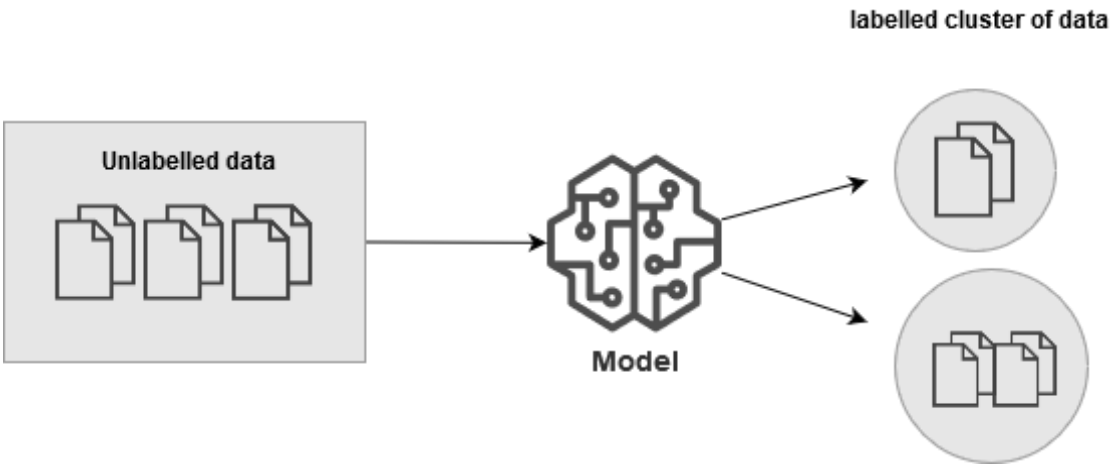

Figure 9 Unsupervised Learning Model

2.3 Machine Learning Approach to Memory forensics investigation

With the consideration of the project's aim to identify and classify a memory image using extracted artefact log instances as benign or malicious is clearly requires a classification approach. Therefore, Supervised Machine Learning models based on the classification algorithms are been appointed to address the problem stated. In addition, several researchers and recent studies have adapted the Machine Learning classification approaches for unknown malware detection and

malware related classification problems [21][25]. The most commonly used classification algorithms of Machine Learning models for detection purposes are Random Forest (RF) , Decision Tress (DT) ,Support Vector (SV) . Naïve Bayes (NB) , and Neural Networks (NN) . The following sections will briefly disuses about each of the classifiers mentioned above .

2.3.1 Random Forest

Random Forests (RF) model is one the popular machine learning predicative classification algorithm that is commonly used in detection and filtering applications [21]. This model is an ensemble learning model for classification , where model constructs a strong learner by employing a collection of decision trees that are formed by weak decorrelated decision tree classifiers [7]. In general, the model applies a bootstrap sample for the given dataset, and it creates individual classification trees for each sample randomly, where each decision tree outputs the classification class based on most frequently occurred values and feature in the class of given sample. The bagging approach applied by the model to aggregate individual decision trees and outputs average estimation of the classification class [7].As example of input data given to be classified , the model will classify based on the majority estimation of all the decision tress subtrees [7]. The main reason is that it takes the average of all the decision tree estimations that it explicitly estimate relative importance of a variable of classification for given data without biases [5]. In terms of the model properties, it runs efficiently on large datasets and provides a good accuracy as well as effective method for estimating missing data [7] .

2.3.2 Decision Tree

Decision Tree is also consider as the most popular Machine Learning classifiers. This model is based on the ensemble learning method of divide-and-conquer [7]. The model is constructed as a hierarchical structure in the form of a tree structure , where it intakes dataset and gradually trains and splits into smaller subsets to a certain limit is reached. This results in tree's internal decision nodes and leaves, and each decision node represents possible features labeling as branch outcomes and each leaf represent output classification class for a depending branch decision node [7] [5]. The model process starts at root node to split in sequence-manner and classifies with corresponding

branches that represents values connecting to given data features until a decision leaf is identified and points the output value to label unseen data [7]. The following Figure 11, shows an example of the decision tree for known and unknow files types, where an unseen data unknow file given to the model as input, the model will determine the classification of unknown file by locating the path of nodes and branches that matches to given data features as Unknown -> Application -> and then identifies class leaf of benign or malicious based on whether the unknown file is hidden or unhidden [7][5].

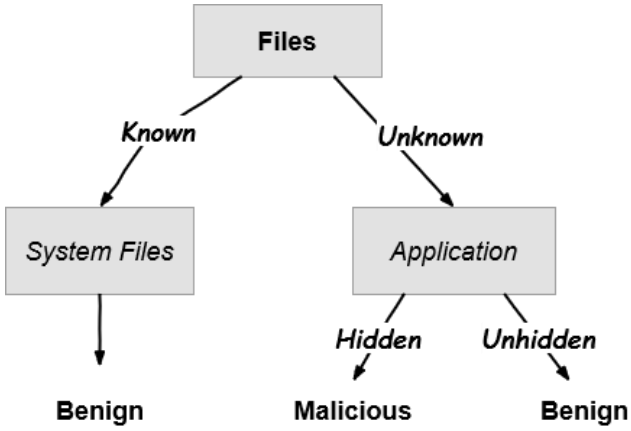

Figure 10 Decision Tree

The Decision tree classifier allows interpretability and fast allocation for the classification as its method of divide and conquer that efficiently reduces half of the unnecessary features . In addition, decision tree classifier is also known as C4.5[5].

2.3.3 Support Vector Model

Support Vector Model (SVM), is supervised learning model that is used mainly as linear classification and regression model for data analysis, pattern recognition and detection [7]. The model is based on linear and kernel methods [5], where given input (training data) is split into two-class learning task that are called support vectors. The SVM algorithm structures a classification model method that classifies unseen data to one of the two support class vectors on either side of an optimal hyper plane which splits two-classes and making it a simple binary linear classifier [7] . The model represents the training data as points when classifying and it creates a

partition of data into two-classes which is divided by the major distance to the adjacent training data point of any class. Any unseen data given will be predicted and classified either of the two-classes as shown in Figure [7]. In terms of the main properties of this model , it provides higher accuracy in detection and classification outcomes with minimal true error rate. The model yields a hyperplane margin that is useful for any small and large sample as it handles complexity of given data [7][35] Besides, the model's kernel function is highly correlated to Neural Networks, as it is able to function as a non-linear model as well [25].

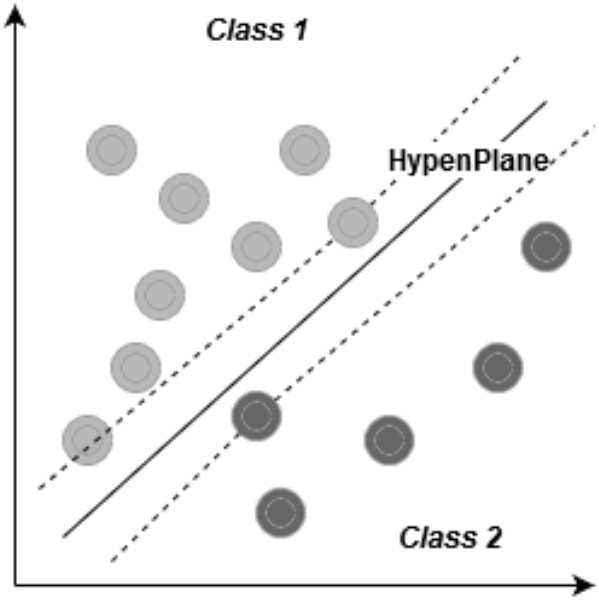

Figure 11 Support Vector Classification

2.3.4 Naïve Bayes

Another popular predication classification model is called the Naïve Bayes [25]. It is a simple supervised learning model based on Bayesian algorithms and theorem for predicting the conditionally independent possibility of classes of various features for a given data sample [7]. The principle of the Naïve Bayes classifier is a probabilistic and assumption method, where it calculates input training data with the assumption of conditionally independent of each other for a given class label [7] . For given unseen data to Naïve Bayes classifier, it will calculate the predication outcome based on the maximum likelihood probability of feature instances of training

data sample [35]. The algorithmic equation of Naïve Bayes calculating the conditional possibility is shown in Figure 12

$$P(l/f) = \frac{P(l) * P(f/l)}{P(f)}$$

Figure 12 Bayes Theorem Equation from [35]

The **P (l/f)** represents the Naïve Bayes posterior probability and it includes main elements that are **P(l)** , **P(f)** and **P(f/l)** [35].

- Both **P(l)** and **P(f)** represent the prior probabilities without regard to each other
  - **P(l)** is the previous likelihood of class label.
  - **P(f)** is the previous possibility that a given feature set appeared.
- **P(f/l)** is the previous likelihood and predicator prior possibility.

Additionally, Naïve Bayes stated that individual feature instances are not dependent as shown in Figure 13 [35].

$$P(l/f) = \frac{P(l) * P(f1/l) * \ldots * P(fn/l)}{P(f/l)}$$

Figure 13 Bayes Theorem : instances from [35]

Obviously, Naïve Bayes classifier is based on a simple equation that does not require complex parameters and functions. It is appealing that the model is relatively easy to create and use as to its properties of simplicity and robustness. Some of the popular application of Naïve Bayes algorithm are used in text classification and spam filtering [7].

2.3.5 Neural Networks Classification Model

Numerous Deep Learning approaches are being utilized nowadays in detection and classification for malwares and one of these popular approaches is Neural Network as Classification model [25].This model is based on artificial neural network algorithms, which consist of multiple artificial neurons connected in a neural network layers similarly to human's neural cell [7][25].The following diagrams illustrates a human neuron cell and artificial neuron Figure 14 and Figure 15.

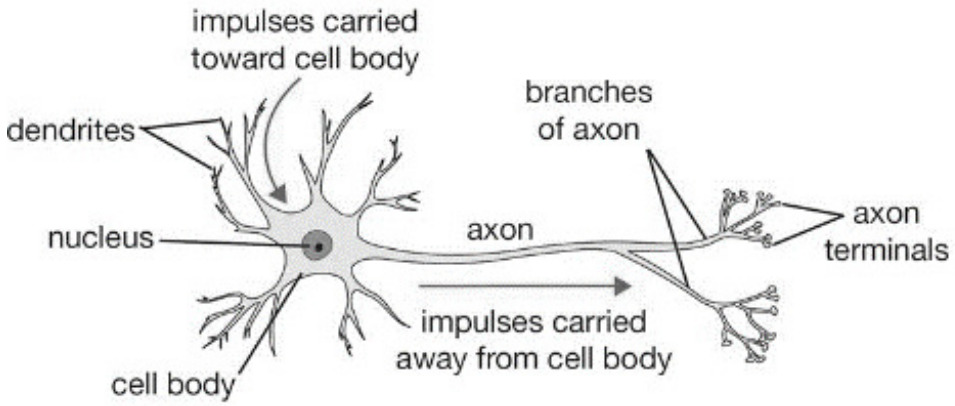

Figure 14 Human neuron cell from [37]

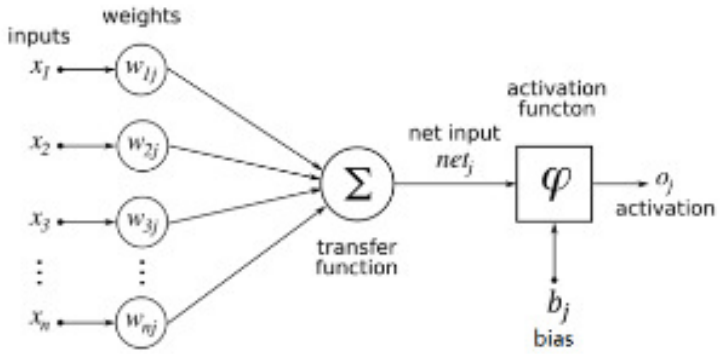

Figure 15 Artificial neuron from [37]

The NN model intakes the inputs of training data as a layer of given neurons set , and each neuron represents a single input of data. The input neuron is associated with cascaded layers of artificial

neuron which consist of weight, transfer, bias, and activation functional layers[35][7]. The model obtains each neuron and it passes through the functional layers to provide an output classification, which is formed by the individual outputs of neurons as presented in following diagram 16 [37]

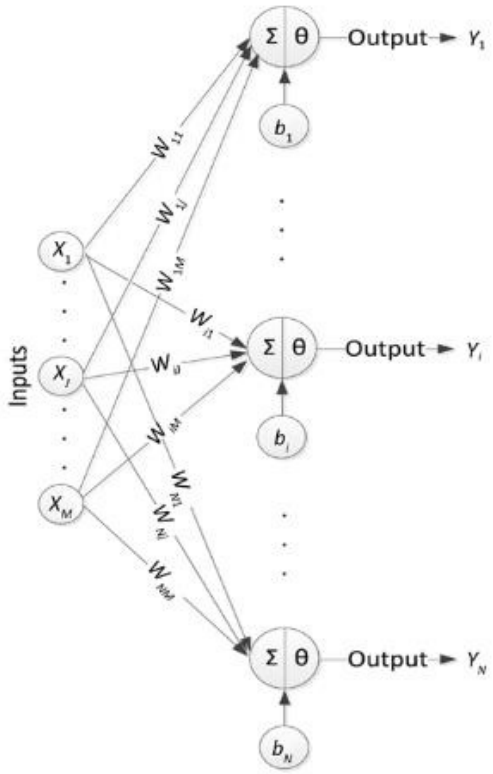

Figure 16 layers of Artificial neuron from [7]

In nutshell, a deep learning neural network classifier model utilizes a range of algorithms that endeavors to identify underlying interactions and patterns of a dataset through a method layers that mimics the actions of the human neural network functionality [7][37]. In addition, one of the major issues that effects the model outcome is the inefficacy to process Big data to provide a subsequent outcome. Also, it provides data nonlinearity in which it does not function classification properly as it output the predication based on high classification value [7].

2.4 Encounter possible outcomes of Machine Learning Models

It is essential to determine a proper classification machine learning algorithm for the proposed binary classification problem. There several performance measures that are utilized to evaluate the

machine learning algorithms in term of how well is the learning capacity of a model in achieving a correct difference between classification classes and how well classifier model constructs properly to process classification task for testing data [10] .In addition, to ensure that obtaining results from models represents accurate insights, these measures help to avoid modelling error or Overfitting, which represents poor generalization and inaccurate classification of the outcome predication [7]. Hence, it is crucial to evaluate the learning algorithms of Machine Learning models for assessing and expressing the success of a binary classification study [10].

In general, there are four main types of binary classification outcome of Machine Learning models that are :

1. True Positive (TP), means predication is correctly identified as positive.
2. True Negative(TN), means predication is correctly identified as negative.
3. False Positive (FP), means predication is incorrectly identified as positive.
4. False Negative (FN), means predication is incorrectly identified as negative.

These direct outputs of classification predication are represented in 2-dimensions as 'Confusion Matrix' table as shown in Figure 17.

Confusion Matrix		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Figure 17 Confusion Matrix from [7]

The Confusion Matrix is a representation of Machine Learning performance for a classification algorithm based on the given test data in matrix. The matrix contains two classification against each other that are the predicted classification and the actual classification in form of four types of

predication outcomes as mentioned earlier [7]. If Confusion matrix had higher number of predication values for both TP and TN, it indicates for good performing learning algorithm and relative to the total correctly classified for the testing set [25][10]. Overall, the Confusion matrix is useful as accuracy indicator for the model's outcome classification results. Also, there are other performing measures that are used as performance comparison metrics for the learning model with testing dataset which includes Accuracy, Precision, Recall and F-measure [5][7].

- **Accuracy** represent the rate of the true and false predication made by the model and it calculated using the true values (TP & TN) by the total number of predications [7] as shown in Figure 18.

$$\text{Accuracy (AC)} = \frac{TP + TN}{TP + TN + FN + FP}$$

Figure 18 Accuracy Equation from [7]

- **Precision also known as Predicative value** represent the rate of true positive predicted cases and positive samples that are correctly classified [25][5]

$$\text{Precision (P)} = \frac{TP}{TP + FP}$$

Figure 19 Precision Equation

- **Recall** represent the rate of actual positive cases which are correctly identified [25][5]

$$\text{Recall (R, true positive rate)} = \frac{TP}{TP + FN}$$

Figure 20 Recall Equation from [7]

- **F-measure or F score** represent the rate of overall accuracy performance and it is calculated via precision and recall with equal weights [10].

$$\text{F} - \text{Measure} = \frac{\left(\beta^2 + 1\right) \cdot P \cdot R}{\beta^2 \cdot P + R},$$

Figure 21 F-Measure Equation from [7]

All these measures have the accurate performance rate 1 and the least worst performance rate is 0 [5][25].

2.5 Related Work

The need for a reducing the consistent backlog of memory data and analysis of digital evidence is crucial as it can provide useful hidden artefacts and links that can be used to proof claims of innocent victims [22][44]. Tool as volatility framework can be used for the memory inspection and indeed, it provides investigators with useful artefacts to be analysed for anomalies ,malware or string search that are obtained using the traditional manual procedure. Recently, an automation approach was proposed called Quincy: Detecting Host-Based Code Injection Attacks in Memory Dumps [44] –, where a tool employed a machine learning approach to detect and filter memory dumps from the compromised machine for the malware attacks that are either as injected code or sent over the network [44]. The tool functioned to classify the infected memory region and predict the possible damage that can malware cause with malware classification. Possibly an approach to prioritize malware analysis from the volatile memory data to overcome the manual analysis. Although Quincy machine learning algorithms focus more utilizing Malfind and Hollowfind artefacts for detection of malware injections, but these plugins require considerable time to scan the memory image and extraction the dumps for the suspected areas which is not yet efficient [44]. With a similar motivation of automation approach for memory forensics investigation, aiming to develop a classification tool that is specifically fast detection and classify memory dump's by

utilizing process-related artefact log that help forensic analysts and investigator for memory investigation procedures prior analysis.

2.6 Programming Approaches

Python programming language is being considered for the projects tool implementation. There are many python libraries and modules available that commonly supports the programming of Machine Learning models and algorithms. Scikit-learn library is the most popular library that supports to develop different Machine learning models. Scikit-learn consist of two basic libraries of Python, NumPy and SciPy, which adds a set of algorithms for common machine learning and data mining tasks, including clustering, regression, and classification. Even tasks like encoding and labelling data, feature selection and ensemble methods can be implemented, and a comprehensive documentation is available with tutorials and code examples [36][24]. In addition, Scikit-learn library provides functionality of evaluating the classifier models by the supported evaluation metrics utilities. The reason of considering the Scikit-learn library as it has previously used in developing Machine Learning approaches and it enables for quicker implementation of the prototype programs [24. In terms of memory analysis, open-source Volatility library [4] that can be utilized as its python-based tool and most commonly used for analysing memory raw images and dumps and extracting artefacts. Some complexity expected from the utilizing the Volatility as library as not all plugin is fully supported, and it is important to consider the python build version when developing the tool as Volatility library is supported by python 2.7 version [41]. Another useful tool for memory acquisition and analysis is Rekall Forensic and Incident Response Framework which is also considerable for the project as it is python-based tool. In terms of the tool interface [30], python provides a built-in GUI library **Tkinter** that can be utilized for the tool interface implementation to have a better user graphical interface compare to traditional command line interface [38].

2.7 Research Question

What are the useful artefacts can be extracted from a compromised memory dump for any memory forensics investigation ?

How does the Machine Learning approach would help in classifying extracted artefact log from memory dump and how does the approach resolve the problem of forensic backlog ?

Chapter 3 :Investigation and Analysis

This chapter of the document details the prerequisite and environment setup, analysis techniques used to investigate and analyse a memory image for process-related artefacts that are likely to be hidden in a memory with consideration of other memory artefacts. In addition, the data collection approach for the Machine Learning models.

3 Environmental Setup

Prior to conducting the analysis and investigation procedure of the memory snapshots for artefacts, a Linux environment based Virtual Machine was prepared and installed as a forensics analysis platform to perform fair investigation and analysis of memory snapshots. The following section will detail the equipment and tool resources as well as the defined dataset of memory images that were utilized for conducting the investigation and analysis of memory image.

3.1 Prerequisite

Equipment and Setup:

1. Lenovo G50-70 (15 inches, 2016)
2. Host Operating System (Version: Windows 8.1, 64-bit, Build 9600 ,6.3.9600)
3. Hardware Processing CPU Intel i7 -4510U @ 2.00GHz 2.60.GHz , RAM 12 GB
4. VMware Workstation 15 Player (Version: 15.5.2 Build-15785246)
    1. Installed latest version of **LinuxMint Mate (19.3)** Operating system as Virtual Machine VM
    2. VM Specifications :
        - **RAM Size = 2 GB**
        - **Internal Hard Disk 64 GB** of free available space used
    3. Two most commonly used open-source Memory Forensics analysis and acquisition Toolkits were installed in VM:
        - **Rekall-core 1.6** [30] utilized as decompressor for memory snapshots from **.aff4** format into memory images standard format **.img.**

- **Volatility Framework with Windows 10 Memory Compression Version 2.6** [40] is utilized to perform analysis and investigation of decompressed memory snapshots to retrieve useful artefacts and information and to execute outputs of memory artefact logs .

Memory image dataset : a research dataset was utilised for this project experimental study of analysing the process-related artefacts of memory image. The dataset consists of (4300 positive and 300 negatives) acquired realistic memory snapshots of an uncompromised and compromised Windows 10 virtual machines. Memory images were acquired using Rekall's WinPmem acquisition tool and were stored as compressed images in Advanced Forensics Format (AFF4). Briefly, The memory images were compromised using several malware based on obfuscation evasion techniques. All the datasets were collected between 2017 and 2019 at ST Engineering Electronics-SUTD Cyber Security Laboratory, Singapore University of Technology and Design in Singapore [32].

3.2 Analysis Approach

Investigating a compromised memory image requires deep understanding of different memory forensics investigation and analysis techniques in order to uncover intrusion source with all associated artefacts. Despite the limited timeframe of the project, some areas of the investigation and analysis of a compromised memory will be limited and will mainly focuses on the process related artefacts that could be found in a memory in order to identify memory's process activities and behaviours as benign or malicious. Yet, the basic memory analysis methodology will be adapted as mentioned in the Background Search **Chapter 2** as well as from the practical guide for conducting the analysis of a compromised memory image [13] .

3.2.1 Acquisition of a compromised memory image

After a successful installation of compatible analysis resources of VM and memory analysis software Rekall and Volatility , a compromised sample of memory image from malicious dataset was downloaded then was decompressed using Rekall to acquire image (.**img**) format of the memory as shown in Figure 22.

```
root@amna-vmware:/home/amna/Desktop/dataset# rekall imagecopy -f /home/amna/Desktop/dataset
/posimg1.aff4 -O posimg1.img
Range 0x1000 - 0x9e000
Range 0x100000 - 0x2000
Range 0x103000 - 0xdfeed000
Range 0x100000000 - 0x20000000
```

Figure 22 Decompressing memory image using Rekall

Once memory image was decompressed ,an md5 hash code is generated and associated with acquired memory image. Then, utilized Volatility tool to process, analyse, and extract meta-features and artefact logs from the volatile memory image. The further detailed analysis of compromised image and extracted artefacts will be presented in the following section as guided steps.The compromised memory from positive/malicious dataset was randomly selected and utilized for the analysis .The memory image file that was used for the main analysis is **snapshot-alpha_mixed-1-meterpreter-reverse-tcp-2019-01-02_09-53-57.aff4** .

3.2.2 Main Analysis

The main areas of analysis and investigation of compromised memory image is accomplished using Volatility Framework Tool and is summarised as follows:

1. **Create a data integrity hash code**

   Initially to ensure that original memory is not altered during the analysis a md5sum hash file is created once the memory file is decompressed as acquired memory image and ready to be analysed as show in the in Figure 23

```
root@amna-vmware:/home/amna/Desktop/dataset# md5sum /home/amna/Desktop/dataset/posimg1.img
33e398d3a85df4bbe25be6ab12cc19c6  /home/amna/Desktop/dataset/posimg1.img
root@amna-vmware:/home/amna/Desktop/dataset# md5sum /home/amna/Desktop/dataset/posimg1.img
> checkposimg.md5
root@amna-vmware:/home/amna/Desktop/dataset# cd /home/amna/win10_volatility
```

Figure 23 md5sum hash for acquired memory image

2. **Ensure to identify the raw memory OS profile prior the analysis of the artefacts [13].**

As initial step of memory investigation and analysis, the command **imageinfo** is utilized to determine the memory OS (operating system) and the profile system from which memory image was acquired with additional high level details of hardware architecture type and format of the memory system service pack. It is considered very important for memory analysts to select correct profile upon the analysis to determine if it was correctly acquired or is corrupted memory image [13].

(NOTE: It is important to correctly input the memory profile as commands are case sensitive and may issue error. Additionally, specify the file location as the command will only work with files that are located in the appropriate directory as mentioned in the command and be Patience for the process to display the output) [13]

```
root@amna-vmware: /home/amna/win10_volatility
File  Edit  View  Search  Terminal  Help
root@amna-vmware:/home/amna/Desktop/dataset# cd /home/amna/win10_volatility
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/p
osimg1.img imageinfo --profile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win10x64_17134
                     AS Layer1 : Win10CompressedAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/amna/Desktop/dataset/posimg1.img)
                      PAE type : No PAE
                           DTB : 0x1aa002L
                          KDBG : 0xf800a8e2f520L
          Number of Processors : 1
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0xfffff800a7fc4000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2019-01-02 01:53:59 UTC+0000
     Image local date and time : 2019-01-02 09:53:59 +0800
root@amna-vmware:/home/amna/win10_volatility#
```

Figure 24 imageinfo output for compromised memory image

3. **Identify rouge and running process**

Number of plugins can be applied with Volatility command when examining memory process related artefacts and one of the main plugins is **pslist**, which is used to list running process and identify for any unusual and rouge running process [2] .

```
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/posimg1.img pslist --profile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0xffffe189686c4040 System                    4      0    127        0 ------      0 2019-01-02 01:44:17 UTC+0000
0xffffe18968758040 Registry                 68      4      3        0 ------      0 2019-01-02 01:44:14 UTC+0000
0xffffe18969e5b580 smss.exe                320      4      4        0 ------      0 2019-01-02 01:44:17 UTC+0000
0xffffe1896ad24580 csrss.exe               408    400     11        0      0      0 2019-01-02 01:44:31 UTC+0000
0xffffe1896adb3080 wininit.exe             480    400      5        0      0      0 2019-01-02 01:44:31 UTC+0000
0xffffe1896adbd580 csrss.exe               492    472     12        0      1      0 2019-01-02 01:44:31 UTC+0000
0xffffe1896ade3080 winlogon.exe            556    472      6        0      1      0 2019-01-02 01:44:31 UTC+0000
0xffffe1896adf3080 services.exe            580    480     10        0      0      0 2019-01-02 01:44:31 UTC+0000
0xffffe1896adb0080 lsass.exe               616    480     12        0      0      0 2019-01-02 01:44:32 UTC+0000
0xffffe1896b064580 fontdrvhost.ex          704    556      6        0      1      0 2019-01-02 01:44:32 UTC+0000
0xffffe1896b0ea500 fontdrvhost.ex          712    480      6        0      0      0 2019-01-02 01:44:32 UTC+0000
```

Figure 25 pslist output of compromised memory image

As to Figure 25, it presents the high level of the running process of both system and non-system application with memory Virtual Address offset (also in Physical Address Offset with parameter when using command), Process ID and name, handles and threads, session and start time. PID is important to consider during the analysis to suspect any suspicious process. As further analysis can be investigated related to a suspicious process in terms of related running process, dlls and other artefacts are linked mostly with PID and their offset processing locations in the memory. One of the disadvantages of pslist plugin is that it does not provide any details of any hidden or previously terminated processes [13][2].

In addition, the process can be either inactive or active as certain processes can be hidden and are not always presented in **pslis**t or **pstree** , theses can be revealed using **psscan** to display previously terminated (inactive) processes and hidden or unlinked by a rootkit running processes as well as using psxview [18][2]

```
. 0xffffe1896b0ea500:fontdrvhost.ex          712    480      6        0 2019-01-02 01:44:32 UTC+0000
 0xffffe1896ade3080:winlogon.exe             556    472      6        0 2019-01-02 01:44:31 UTC+0000
. 0xffffe1896ba0c580:userinit.exe           3596    556      0 ------ 2019-01-02 01:44:38 UTC+0000
.. 0xffffe1896ba26080:explorer.exe          3632   3596    102        0 2019-01-02 01:44:38 UTC+0000
... 0xffffe1896be2b580:VBoxTray.exe         5576   3632     13        0 2019-01-02 01:44:54 UTC+0000
... 0xffffe1896be9a580:OneDrive.exe         5684   3632     17        0 2019-01-02 01:44:54 UTC+0000
... 0xffffe1896bd17580:MSASCuiL.exe         5484   3632      4        0 2019-01-02 01:44:53 UTC+0000
... 0xffffe1896c2044c0:payload-x86-al       4888   3632      4        0 2019-01-02 01:52:34 UTC+0000
... 0xffffe1896a03c580:cmd.exe              3580   3632      2        0 2019-01-02 01:53:57 UTC+0000
.... 0xffffe1896a026580:conhost.exe         4272   3580      6        0 2019-01-02 01:53:57 UTC+0000
.... 0xffffe1896a010580:winpmem-2.1.po      6348   3580      2        0 2019-01-02 01:53:57 UTC+0000
. 0xffffe1896b064580:fontdrvhost.ex          704    556      6        0 2019-01-02 01:44:32 UTC+0000
. 0xffffe1896b272080:dwm.exe                 920    556     12        0 2019-01-02 01:44:33 UTC+0000
 0xffffe1896adbd580:csrss.exe                492    472     12        0 2019-01-02 01:44:31 UTC+0000
root@amna-vmware:/home/amna/win10_volatility#
```

Figure 26 pstree output of compromised memory image

pstree log, does provides useful insights about the processes hierarchy to determine and reveal related processes and their relationships [2] as Figure 26.

```
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/posimg1.img psscan --profile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Name                 PID   PPID PDB                 Time created                   Time exited
------------------ ---------------- ------ ------ ------------------ ------------------------------ ------------------------------
0x0000e189686ba080 svchost.exe        1052    580 0x000000001b700002 2019-01-02 01:44:33 UTC+0000
0x0000e189686bc080 svchost.exe        1148    580 0x000000001d200002 2019-01-02 01:44:33 UTC+0000
0x0000e189686c4040 System                4      0 0x00000000001aa002 2019-01-02 01:44:17 UTC+0000
0x0000e189686e7080 svchost.exe        1096    580 0x000000001ad00002 2019-01-02 01:44:33 UTC+0000
0x0000e18968745080 svchost.exe         424    580 0x0000000027100002 2019-01-02 01:44:33 UTC+0000
0x0000e18968758040 Registry             68      4 0x0000000000700002 2019-01-02 01:44:14 UTC+0000
0x0000e18969c0c080 svchost.exe        6936    580 0x0000000008e00002 2019-01-02 01:53:32 UTC+0000
0x0000e18969c18340 svchost.exe        6988    580 0x0000000017100002 2019-01-02 01:53:32 UTC+0000   2019-01-02 01:53:39 UTC+0000
0x0000e18969c6d380 svchost.exe        7124    580 0x0000000022d00002 2019-01-02 01:53:34 UTC+0000
0x0000e18969d5e080 MpCmdRun.exe       6404   6064 0x0000000107200002 2019-01-02 01:54:39 UTC+0000
0x0000e18969e5b580 smss.exe            320      4 0x000000003d8d0002 2019-01-02 01:44:17 UTC+0000
0x0000e18969eb7580 svchost.exe        6580    580 0x0000000019400002 2019-01-02 01:53:32 UTC+0000
0x0000e18969f42580 svchost.exe        5444    580 0x0000000111800002 2019-01-02 01:54:44 UTC+0000
0x0000e18969fa2580 WmiPrvSE.exe       6836    784 0x0000000000d100002 2019-01-02 01:53:32 UTC+0000
```

Figure 27 psscan output of compromised memory image

psscan log, lists processes (PID) with parent process ID and scans for additional processes in the system that might to hidden or terminated [2] as shown in Figure 27.

psxview log, also list processes that can be hidden while running on the memory where it displays as cross-view for running process and hidden process which includes pslist and psscan [2] .

```
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/posimg1.img psxview -
-profile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
------------------ -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x000000002f648400 svchost.exe            3228 True   True   True     True   True  True    False
0x000000001b041500 fontdrvhost.ex          712 True   True   True     True   True  True    True
0x0000000019c42580 svchost.exe            1332 True   True   True     True   True  True    True
0x000000011baba080 svchost.exe            1052 True   True   True     True   True  True    True
0x00000000274b8080 services.exe            580 True   True   True     True   True  True    False
0x00000000519c6580 svchost.exe            3864 True   True   True     True   True  True    False
0x000000001d888580 svchost.exe            1192 True   True   True     True   True  True    True
0x00000000428db080 ShellExperienc         4512 True   True   True     True   True  True    False
0x000000011babc080 svchost.exe            1148 True   True   True     True   True  True    False
0x000000000c74a580 SearchFilterHo         2436 True   True   True     True   True  True    True
0x0000000040a6d580 audiodg.exe            3296 True   True   True     True   True  True    True
0x000000001cf62580 svchost.exe             356 True   True   True     True   True  True    True
```

Figure 28 psxview output of compromised memory image

psxview is consist of following useful attributes related to running process in the system as cross-validation view :

- pslist
- psscan
- thrdproc, Thread scanning and it essential for every process to have at least one active thread. As if process manipulated with a rootkit and process tries to be hidden it is process' pool scans the for the process threads [2].
- CSRSS handle table, it is a critical system process description and represents the creation of every process and thread [2] .
- Pspcid table: This is a special handle table located in kernel memory that stores a reference to all active process and thread objects [2]
- Session processes: represents associates all processes that belong to a particular user's logon session [2].
- Desktop threads: represents the threads attached to each desktop process [2].

4. Analyse suspected process and related process artefacts

From an analyst's point of view any process noticed as suspicious would require further analysis. The following plugins were used for further analysis of a suspected process using PID to reveal different artefacts as network artefacts, code injections, rootkits . Almost all process-related plugins perform with a –OFFSET and -p/--PID parameter that allow to track and uncover hidden and malicious suspected processes locations.

After the reviewing the **pslist**, **psscan**, **pstree** plugin output, the following lists created was of running process in memory that are divided in order to distinguish process as Application process and related System process.

<u>Application process</u>

- OneDrive.exe,
- winpmem-2.1.po – memory acquisition tool
- cmd.exe – command line prompt
- SkypeBackgroun

- SkypeApp.exe
- VBoxService.ex
- OfficeHubTaskH
- Payload-x86-al – unknown

System Services, Processes and Applications [42][43]

- Critical System Process [2]
  - System,svchost.exe,services.exe,lsass.exe ,explorer.exe ,winlogon.exe
- smss.exe - Session Manager Subsystem, System file
- Registry - System file
- winint.exe - Windows Initialize , System file
- fontdrvhost.exe - Usermode Font Driver Host , System file
- dwm.exe - Desktop Windows Manager, System file
- spoolsv.exe - Spooler Service, System files
- SecurityHealth - Windows Security System file
- MemCompression - Memory Compression, System file
- MsMpEng.exe - Microsoft Malware Protection Engine, System protection file
- dasHost.exe - Device Association Host, System file
- sihost.exe - Shell Infrastructure Host, System file
- userint.exe - User Initialization, System file
- ctfmon.exe - Text input service support, System file
- taskhostw.exe - Tasks Host for Windows, System file
- SearchIndexer - Indexeur Microsoft Windows Search, System file
- cohost.exe -Console Application Host, System file
- audiodg.exe - Windows Audio Device Graph Isolation, System file
- WmiPrvSE.exe -Windows Management Instrumentation Provider Host Service, System file
- NisSrv.exe - Network Realtime Inspection Service, System protection file
- MSASCuiL.exe, Microsoft Antivirus Security Centre User Interface Logo, system protection file

- RuntimeBroker - Permissions manager for the Windows Store, System file
- SgrmBroker.exe - System Guard Runtime Monitor Broker Service , System file
- Shellexperience - Application Frame Host, System file
- dllhost.exe - Dynamic Link Library Host, System file
- SerachUI.exe - Search User Interface , System file
- smartscreen.exe - Antimalware and anti-phishing , System file
- sedsvc.exe - Windows Remediation Service,System update file
- wuauclt.exe - Windows Update AutoUpdate Client ,System update file
- SearchFilterHo - Windows Search Indexer, System file
- SearchProtocol - Windows Search Protocol Host, System file
- AM_DELTA_Patch - Anti-Malware Signature Delta Update Package , System file
- MpSigStub.exe - Microsoft Protection Signature Stub , System Update file

The initial indication of a compromised memory image is expected to be infected with a malware is that the System Protection and Anti-malware files are appeared to be disabled or terminated in the system as of the Exit Time as well the attributes as shown in Figure 29.

```
0xffffe1896a02e080 svchost.exe          6792    580     17        0 ------      0 2019-01-02 01:54:43 UTC+0000

0xffffe18969f42580 svchost.exe          5444    580      5        0      0      0 2019-01-02 01:54:44 UTC+0000

0xffffe1896a13b580 wuauclt.exe          4228   3500      0 -------- ------      0 2019-01-02 01:54:45 UTC+0000   2019-01-02 01:54:54 UTC+000
0
0xffffe1896a1dd580 AM_Delta_Patch       5092   4228      0 -------- ------      0 2019-01-02 01:54:45 UTC+0000   2019-01-02 01:54:54 UTC+000
0
0xffffe1896a1e3580 MpSigStub.exe        5396   5092      0 -------- ------      0 2019-01-02 01:54:45 UTC+0000   2019-01-02 01:54:54 UTC+000
0
```

Figure 29 pslist output : System Protection and Anti-malware processes

```
.. 0xffffe1896b8f6400:svchost.exe                3228    580      8      0 2019-01-02 01:44:37 UTC+0000
.. 0xffffe1896c073580:svchost.exe                3500    580     23      0 2019-01-02 01:46:37 UTC+0000
... 0xffffe1896a13b580:wuauclt.exe               4228   3500      0 ------ 2019-01-02 01:54:45 UTC+0000
.... 0xffffe1896a1dd580:AM_Delta_Patch           5092   4228      0 ------ 2019-01-02 01:54:45 UTC+0000
..... 0xffffe1896a1e3580:MpSigStub.exe           5396   5092      0 ------ 2019-01-02 01:54:45 UTC+0000
.. 0xffffe18969c0c080:svchost.exe                6936    580     11      0 2019-01-02 01:53:32 UTC+0000
.. 0xffffe18969eb7580:svchost.exe                6580    580      6      0 2019-01-02 01:53:32 UTC+0000
```

Figure 30 pstree output : System Protection and Anti-malware processes

5. **Suspected process to be analysed are the following process as to the pstree plugin :**

After acknowledging the system protection files ,considerably **cmd .exe** process is noticed as a suspected process to look at in memory image along with other unknow processes as payload-x86-al . **cmd.exe** is commonly observed as a starter point for initializing malicious activities or suspicious activation commands

```
.. 0xffffe1896ae8c580:svchost.exe                1892    580     10      0 2019-01-02 01:44:34 UTC+0000
. 0xffffe1896adb0080:lsass.exe                    616    480     12      0 2019-01-02 01:44:32 UTC+0000
. 0xffffe1896b0ea500:fontdrvhost.ex               712    480      6      0 2019-01-02 01:44:32 UTC+0000
 0xffffe1896ade3080:winlogon.exe                  556    472      6      0 2019-01-02 01:44:31 UTC+0000
. 0xffffe1896ba0c580:userinit.exe                3596    556      0 ------ 2019-01-02 01:44:38 UTC+0000
.. 0xffffe1896ba26080:explorer.exe               3632   3596    102      0 2019-01-02 01:44:38 UTC+0000
... 0xffffe1896be2b580:VBoxTray.exe              5576   3632     13      0 2019-01-02 01:44:54 UTC+0000
... 0xffffe1896be9a580:OneDrive.exe              5684   3632     17      0 2019-01-02 01:44:54 UTC+0000
... 0xffffe1896bd17580:MSASCuiL.exe              5484   3632      4      0 2019-01-02 01:44:53 UTC+0000
... 0xffffe1896c2044c0:payload-x86-al            4888   3632      4      0 2019-01-02 01:52:34 UTC+0000
... 0xffffe1896a03c580:cmd.exe                   3580   3632      2      0 2019-01-02 01:53:57 UTC+0000
.... 0xffffe1896a026580:conhost.exe              4272   3580      6      0 2019-01-02 01:53:57 UTC+0000
.... 0xffffe1896a010580:winpmem-2.1.po           6348   3580      2      0 2019-01-02 01:53:57 UTC+0000
. 0xffffe1896b064580:fontdrvhost.ex               704    556      6      0 2019-01-02 01:44:32 UTC+0000
. 0xffffe1896b272080:dwm.exe                      920    556     12      0 2019-01-02 01:44:33 UTC+0000
```

Figure 31 pstree output for suspected process

pstree was used to display the parent process related to suspicious processes, to identify the suspicious source . In our case**, explorer.exe** represents the main parent process PPID of the suspected processes of **cmd.exe** and **payload-x86-al**. Therefore, to suspect these related processes the cmdscan and consoles plugins were used to reveal for any last commands and command history of **cmd.exe** process. The following displayed figure shows that these commands are not supported for the memory image OS profile without executing any error message Figure 32.

```
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/posimg1.img consoles --
profile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/posimg1.img cmdscan --p
rofile=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
root@amna-vmware:/home/amna/win10_volatility#
```

Figure 32 cmdscan and consoles command output

Another command-line related plugin cmdline was available to be utilized that displays process command-line arguments and it did provide interesting information of the suspected processes as details of where is process is running from with PID as shown in

Figure 33. Notice the process (PID 48888) with a directory named malware fat rat and it consists of executable file. This clearly shows that the memory is compromised with a malware through the executable **payload-x86-al** file. Two further plugins were used to detect signs of a malicious activities in the compromised memory with regards to the suspected process 4888.

```
svchost.exe pid:   3892
Command line : c:\windows\system32\svchost.exe -k netsvcs -p -s Appinfo
**************************************************************************
payload-x86-al pid:   4888
Command line : "C:\Users\useradmin\Desktop\malware_fatrat_09102018\final-payload-08Dec2018\
payload-08Dec2018-192168128\payload-x86-alpha_mixed-1-meterpreter-reverse-tcp-08-12-18.exe"
```

Figure 33 cmdline command output

6. Sign of Code injection

As we have identified malware executable file, the following malfind plugin is used to lookup for infected areas and dumps out areas for signs of code injection for the given PID of the suspected process ,the plugin executed 18 memory dump sections. Later the extracted dumps were scanned using the virus total in order to reveal information about the code injection and malwares and the whole process of plugin scan for memory , dumping and scanning against the malware detector took about more than 1 hour , all the steps are presented by Figure 34-39.

```
oot@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/datase
/posimg1.img malfind --dump-dir ./output_dir --profile=Win10x64_17134
olatility Foundation Volatility Framework 2.6.1
rocess: MsMpEng.exe Pid: 2320 Address: 0x21047790000
ad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
lags: PrivateMemory: 1, Protection: 6

x21047790000  56 57 53 55 41 54 41 55 48 83 ec 28 48 8b e9 48   VWSUATAUH..(H..H
x21047790010  8d b1 88 38 00 00 ff e2 48 83 c4 28 41 5d 41 5c   ...8....H..(A]A\
x21047790020  5d 5b 5f 5e c3 00 00 00 00 00 00 00 00 00 00 00   ][_^............
x21047790030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

x47790000 56               PUSH ESI
x47790001 57               PUSH EDI
x47790002 53               PUSH EBX
x47790003 55               PUSH EBP
x47790004 41               INC ECX
x47790005 54               PUSH ESP
x47790006 41               INC ECX
x47790007 55               PUSH EBP
x47790008 48               DEC EAX
x47790009 83ec28           SUB ESP, 0x28
x4779000c 48               DEC EAX
x4779000d 8be9             MOV EBP, ECX
x4779000f 48               DEC EAX
x47790010 8db188380000     LEA ESI, [ECX+0x3888]
```

Figure 34 malfind output for compromised memory image

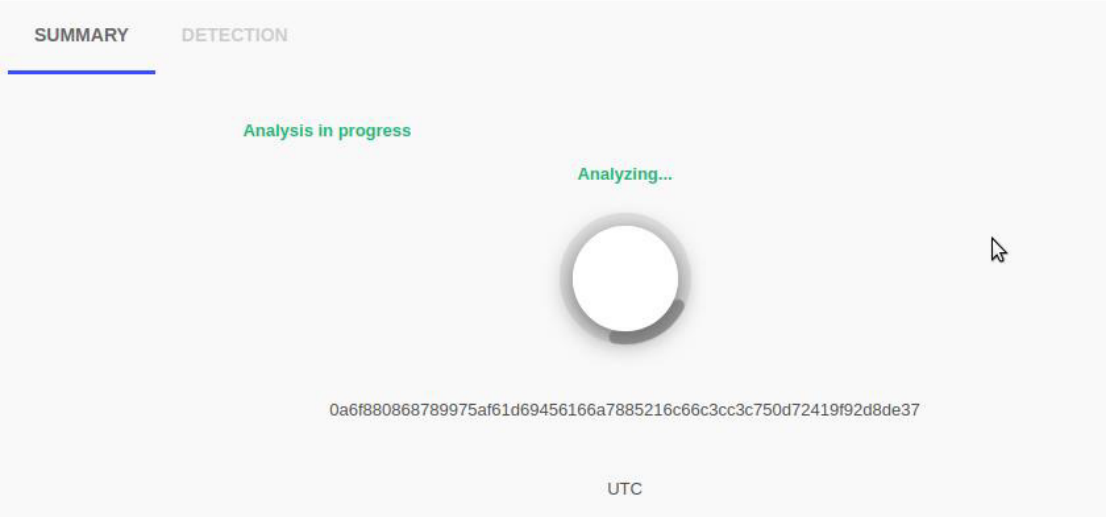

Figure 35 Virus Total scanning for malwares

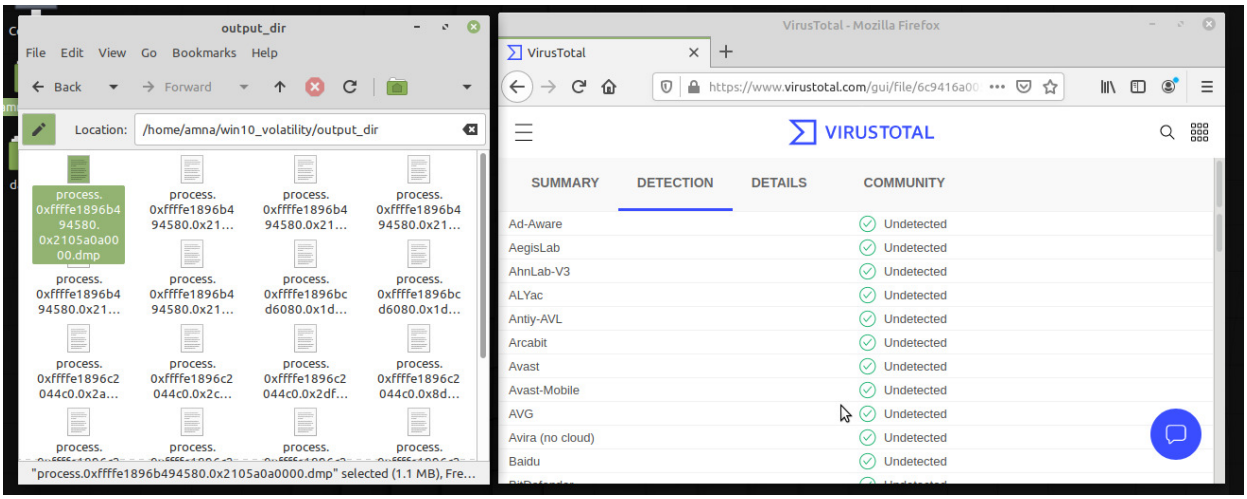

Figure 36 scanning a clean memory dump from compromised memory image

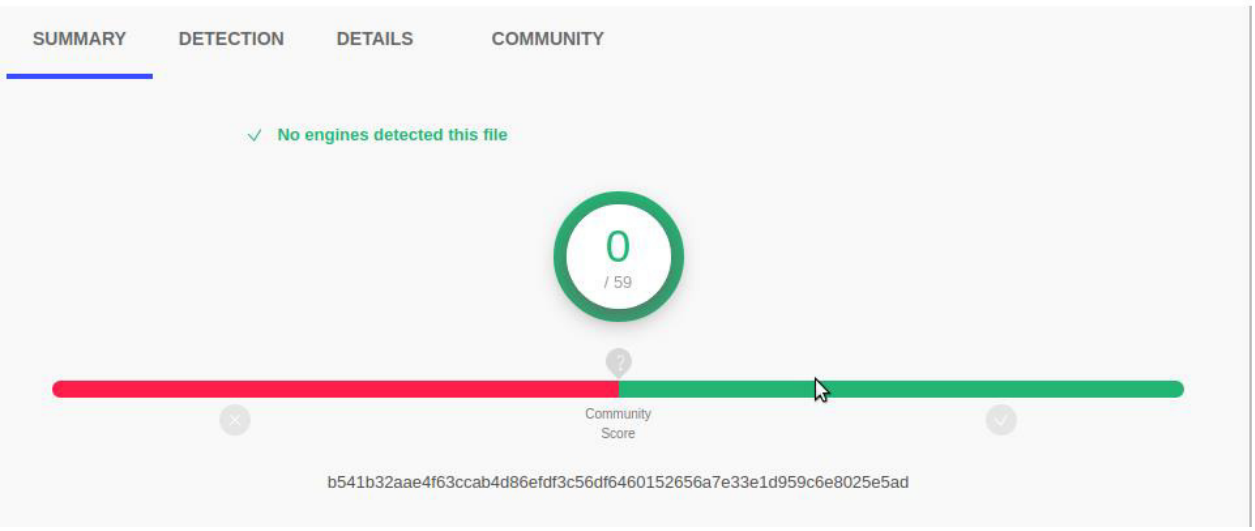

Figure 37 memory dump scan result of Virustotal

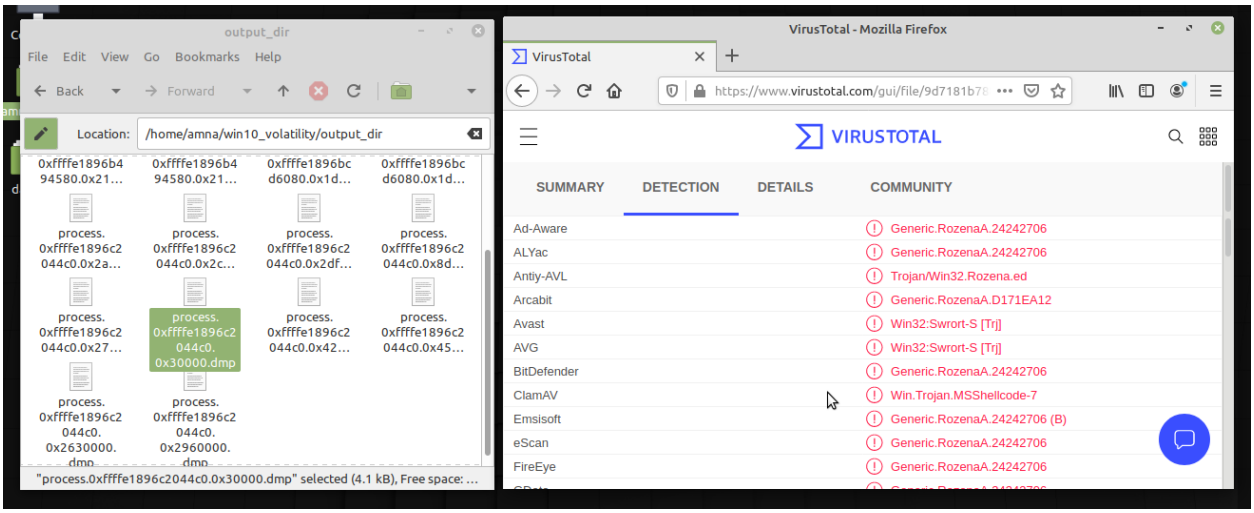

Figure 38 scanning other memory dumps from compromised memory image

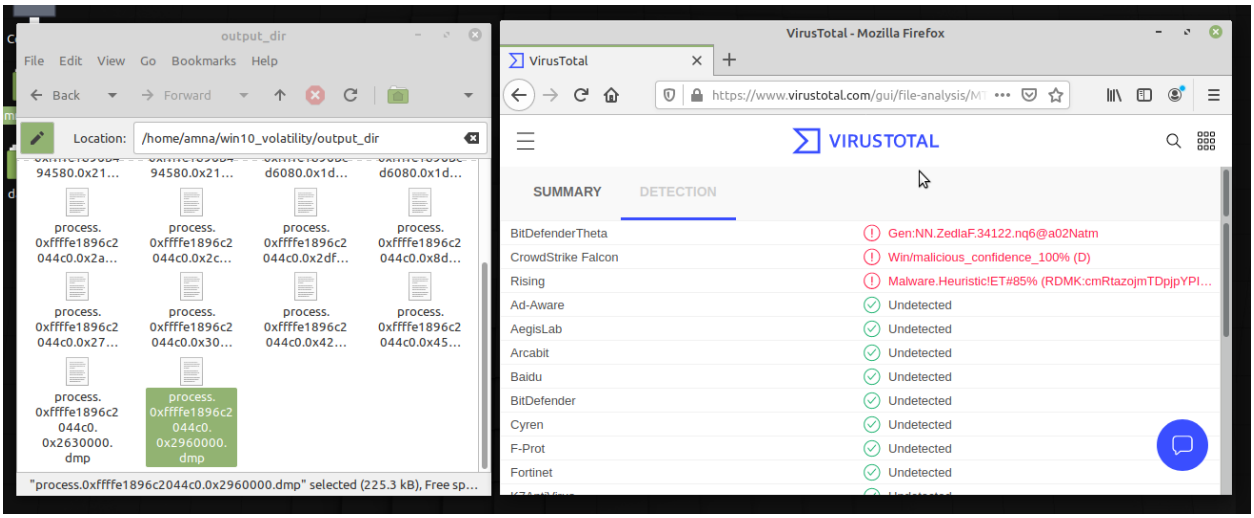

Figure 39 last memory dump scan result from Virustotal

7. Sign of Rootkits

In addition, psxview plugin used as to detect for sign of rootkits and for other hidden processes. It showed the suspected process as normal running process but the psxview log reveals some additional processes are that hidden and existed process compare to pslist.

0x000000003c9c1580	svchost.exe	3080	True	True	True	True	True	True	False
0x000000006955b4c0	payload-x86-al	4888	True	True	True	True	True	True	False
0x000000003e064080	SearchUI.exe	4620	True	True	True	True	True	True	False
0x0000000021726580	svchost.exe	2220	True	True	True	True	True	True	False
0x000000000f4d7580	svchost.exe	864	True	True	True	True	True	True	False

Figure 40 psxview output : initial suspected process from pslist

0x00000000176ba580	csrss.exe	492	True	True	True	True	False	True	True	
0x0000000037fa8580	userinit.exe	3596	True	True	False	True	False	True	False	2019-01-02 01:44:59 UTC+0000
0x0000000108a2e080	svchost.exe	6792	True	True	True	True	False	True	False	
0x000000011bac4040	System	4	True	True	True	True	False	False	False	
0x00000000201ba580	csrss.exe	408	True	True	True	True	False	True	True	
0x0000000101a18340	svchost.exe	6988	True	True	False	True	False	True	False	2019-01-02 01:53:39 UTC+0000
0x0000000108bc6580	smartscreen.ex	6500	True	True	True	True	False	True	False	
0x000000000f32e2f0	@G?????@G?????	21...8	False	False	False	False	False	False	True	
0x0000000108b3b580	wuauclt.exe	4228	True	False	False	False	False	True	False	2019-01-02 01:54:54 UTC+0000
0x0000000108bdd580	AM_Delta_Patch	5092	True	False	False	False	False	True	False	2019-01-02 01:54:54 UTC+0000
0x0000000108be3580	MpSigStub.exe	5396	True	False	False	False	False	True	False	2019-01-02 01:54:54 UTC+0000
0x0000000101d42580	svchost.exe	5444	True	True	True	False	False	True	False	
0x0000000108b2b580	conhost.exe	6484	False	False	False	False	False	True	False	2019-01-02 01:54:54 UTC+0000
0x0000000108a2a580	MpCmdRun.exe	6756	False	False	False	False	False	True	False	2019-01-02 01:54:54 UTC+0000
0x0000000101b5e080	MpCmdRun.exe	6404	False	True	True	False	False	True	False	

Figure 41 psxview output : reveal hidden process

It appears that some system security services as AM_DELTA_Patch and SmartScreen are being terminated and being as hidden process as previously acknowledged about certain system protection processes were disabled. In addition, some more suspected processes are revealed that appears to be hidden but in active state running the memory. These additional processes indicate as suspicious and malicious as their behavior of hiding themselves through the false values across the attributes while being active. This would require more memory dumps extraction for each of the hidden process for deep analysis and malware detection .

The further extraction of the suspected processes and hidden active artefacts existed in the physical memory is achievable by given the PID though utilizing the following **Extraction of Process, driver, and objects** plugin:

- procdumps checks for the executable dumps that can be extracted
- dlldumps for suspected dlls to be extracted
- moddump for the suspected kernel drivers

- memdump for extracting the physical memory section with mapping of physical and virtual addresses of areas where the suspected processes running

The area of extraction plugins was not utilized as part of the analysis as it requires deep analysis and malware scan for each extracted memory dump artefact and space to store extraction. Hence, it is considered as further analysis of section once the memory is identified as malicious.

Once the analysis of compromised memory was completed a Md5um is checked to ensure the data integrity as shown in Figure 42.

```
root@amna-vmware:/home/amna/Desktop/dataset# md5sum posimg1.img
33e398d3a85df4bbe25be6ab12cc19c6  posimg1.img
root@amna-vmware:/home/amna/Desktop/dataset#
root@amna-vmware:/home/amna/Desktop/dataset#
```

Figure 42 Verified md5sum after analysis for compromised memory image

3.2.3 Analysis of uncompromised image

In addition, another analysis was carried out for uncompromised memory image in order understand the behaviors of process in a clean normal memory image. A negative image file was utilized was **snapshot-2017-12-11_21-38-20.aff4 ,** with similar analysis steps were followed but mainly centered analyse the pslist and psxview artefact logs as presented in Figure 43-44.

```
root@amna-vmware:/home/amna/win10_volatility# python vol.py -f /home/amna/Desktop/dataset/negimg1.img pslist --profil
e=Win10x64_17134
Volatility Foundation Volatility Framework 2.6.1
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Ex
it
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ --
----------------------------
0xffffc18ad2a2a040 System                    4      0    103        0 ------      0 2017-12-12 21:32:44 UTC+0000

0xffffc18ad40c7040 smss.exe                292      4      5        0 ------      0 2017-12-12 21:32:44 UTC+0000

0xffffc18ad4053640 csrss.exe               376    364     10        0      0      0 2017-12-12 21:32:45 UTC+0000

0xffffc18ad4cc0080 smss.exe                448    292      0 --------      1      0 2017-12-12 21:32:45 UTC+0000   20
17-12-12 21:32:46 UTC+0000
0xffffc18ad4cbd080 wininit.exe             456    364      4        0      0      0 2017-12-12 21:32:45 UTC+0000

0xffffc18ad4cc1080 csrss.exe               468    448     11        0      1      0 2017-12-12 21:32:45 UTC+0000

0xffffc18ad4d69080 winlogon.exe            528    448      6        0      1      0 2017-12-12 21:32:46 UTC+0000

0xffffc18ad4dab500 services.exe            572    456     16        0      0      0 2017-12-12 21:32:46 UTC+0000

0xffffc18ad4d82080 lsass.exe               580    456      9        0      0      0 2017-12-12 21:32:46 UTC+0000

0xffffc18ad4c48640 svchost.exe             660    572      2        0      0      0 2017-12-12 21:32:46 UTC+0000
```

Figure 43 pslist output of uncompromised memory image

```
Offset(P)          Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
------------------ -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
WARNING : volatility.debug    : NoneObject as string: Buffer length 11887 for _UNICODE_STRING not within bounds
0x000000010f414500 VBoxService.ex       1232 True   True   True     True   True  True    False
0x0000000119e7b640 MsMpEng.exe          2468 True   True   True     True   True  True    False
0x000000011cada080 svchost.exe          1028 True   True   True     True   True  True    False
0x0000000077099080 svchost.exe           820 True   True   True     True   True  True    False
0x000000008e5db080 svchost.exe          2392 True   True   True     True   True  True    False
0x0000000074dbc080 svchost.exe          1732 True   True   True     True   True  True    False
0x000000011a676480 svchost.exe          2624 True   True   True     True   True  True    False
0x00000000187a6640 WmiPrvSE.exe         5312 True   True   True     True   True  True    False
0x000000000d287640 RuntimeBroker.       4844 True   True   True     True   True  True    False
0x000000008c91d640 svchost.exe          2316 True   True   True     True   True  True    False
0x000000010f43c640 svchost.exe          1280 True   True   True     True   True  True    False
0x000000010f582080 lsass.exe             580 True   True   True     True   True  True    False
0x000000010f43a640 svchost.exe          1320 True   True   True     True   True  True    False
0x000000010f42a640 svchost.exe          1624 True   True   True     True   True  True    False
0x000000008b89c640 svchost.exe          2164 True   True   True     True   True  True    False
0x000000010f42e640 WUDFHost.exe         1492 True   True   True     True   True  True    False
0x000000010f424640 svchost.exe          1744 True   True   True     True   True  True    False
0x0000000072951640 svchost.exe          3356 True   True   True     True   True  True    False
0x000000010f432640 svchost.exe          1420 True   True   True     True   True  True    False
0x000000001434c080 WmiPrvSE.exe         4360 True   True   True     True   True  True    False
0x0000000098213640 svchost.exe          3892 True   True   True     True   True  True    False
0x00000000080f4640 SearchIndexer.       4392 True   True   True     True   True  True    False
```

Figure 44 psxview output for uncompromised memory image

In comparison to the compromised image that the uncompromised image has normal system running application and service process and only one terminated and unlinked process which is considered to be as clean process as memory image is already labeled benign and does not required for further analysis.

3.3 Analysis Findings

Hence, to avoid the reaching to the conclusion of dumping all the considerable infected areas of physical memory dumps using the suspected hidden process PIDs and scanning against the Virustotal or any Malware Detector to reveal whether these memory processes are malicious and consist of malwares or code injections is time consuming as these steps require a scanning all extracted dumps in which some dumps are resulted as cleaned dumps. Also, it requires additional space to store all extracted dumps of the suspected processes in which some of cleaned dump sections are ineffective to be stored.

3.3.1 Feature Selection

Although malfind and psxview are two main plugins that can be used to detect signs of malicious activities from simple analysis of a compromised image other than the pslist and psscan that are used to identify the unknow processes for the initial analysis. Hence, that malfind is a useful for the extraction of the code injection signs but not the main source of the code injection dump output as one dump and also malfind artefact log requires understanding of the low-level memory structure compare to psxview log artefact which is high level.

1. **Psxview** can give some hints of any process trying to duplicate system files when running as process as it reveals all process including system related process is that trying to be appeared as hidden while being an active process. As some artefacts can disguise a system process in number of ways; but it is important to distinguish essential system processes of actual machine and consider their reserved area of processing that should not be involved with other non-systemic processing files. As certain system files run as one process but with multiple services as process, dlls, threads and handles.

2. With consideration of psxview artefact log to be utilized for applying the machine learning approach. As the Volatility Framework plugin does provides outputs for artefacts in some formats that are **xlxs**, **text** , **html** and **json** but not all plugins support all of the formats. But, in terms of psxview log is obtainable and as machine learning approaches the .**xlsx** format is easily readable and manageable format. Additionally, the obtainable logs psxview is

considered to require less pre-processing as most of its attribute features are categorical and numerical that can be prepared and preprocessed to be used for machine learning.

3. Psxview log is useful as it has a high-level cross validation data that helps to understand and identify for the hidden process behaviours.

4. All attribute features of the psxview artefact log excluding the Exit time attribute are considered as often terminated process still remains in the system. In addition, the defined attributes are considered for identifying a process as hidden or unhidden as if any attribute showed false indicates that the process is missing. Besides, comparing the pattern of the system process behavior from the psxview , that is distinguished and revealed from analysis of both malicious and begin memory image that the clean memory image of benign processes are consist of false values for the deskthrd attribute comparing to the compromised memory image processes which consist of both false and true in deskthrd .

3.4 Data Collection

As the requirement of the Machine learning of the training and testing data a dataset will be created using a collection of extracted psxview logs from the memory image dataset. Number of the memory images were downloaded from the defined dataset and each of the malicious and benign memory images were stored in separate folder to distinguish prior creating the dataset for labelling them as positive and negative. There was no automation process for the extracting logs from all memory image dataset at once it required manual process which includes downloading, decompressing each image using Rekall, and then extracting directly the psxview log pull-outs and storing as .**xlxs** using the Volatility Framework. After all pull-out were extracted an archived folder was created with md5sum to ensure the originality of psxview logs dataset.

In order to have useful insights and results about 20 positive memory images were randomly download from the dataset which consist of 3 different groups of compromised images with different techniques and 20 negative memory images were download randomly from the benign dataset. Despite 40 images were used as each of downloaded memory image sized around 698MB to 1.3 GB and decompressed acquired memory image sized around 4.2-5 GB.

3.4.1 Creating Dataset

In order to create the pre-processed dataset using the extracted psxview logs , a programming approach will be utilized using the python programming language to implement a specialized tool that will allow importing and creating dataset for the raw psxview logs. The further details will be described in the next **Chapter 4 Design and Specification.**

The tables Table 1 and Table 2, presents the downloaded images of both negative and positive memory images with their extracted psxview logs that were utilized to create the dataset with.

Original Filename	Extracted image filename	Log output as xlxs
snapshot-2017-12-11_21-38-20.aff4	neg1	psx_neg1
snapshot-2017-12-13_11-44-49.aff4	neg2	psx_neg2
snapshot-2017-12-13_10-45-52.aff4	neg3	psx_neg3
snapshot-2017-12-12_15-23-26.aff4	neg4	psx_neg4
snapshot-2017-12-12_15-39-17.aff4	neg5	psx_neg5
snapshot-2017-12-12_15-43-44.aff4	neg6	psx_neg6
snapshot-2017-12-13_14-15-4.aff4	neg7	psx_neg7
snapshot-2017-12-13_15-33-14.aff4	neg8	psx_neg8
snapshot-2017-12-13_16-7-32.aff4	neg9	psx_neg9
snapshot-2017-12-13_11-21-50.aff4	neg10	psx_neg10
snapshot-2017-12-13_12-35-7.aff4	neg11	psx_neg11
snapshot-2017-12-13_14-31-2.aff4	neg12	psx_neg12
snapshot-2017-12-13_14-33-39.aff4	neg13	psx_neg13
snapshot-2017-12-13_15-51-27.aff4	neg14	psx_neg14
snapshot-2017-12-13_15-56-1.aff4	neg15	psx_neg15
snapshot-2017-12-12_17-38-2.aff4	neg16	psx_neg16
snapshot-2017-12-13_11-28-37.aff4	neg17	psx_neg17
snapshot-2017-12-13_13-31-10.aff4	neg18	psx_neg18

snapshot-2017-12-14_12-42-41.aff4	neg19	psx_neg19
snapshot-2017-12-14_14-49-33.aff4	neg20	psx_neg20

Table 1 Downloaded Benign memory images

Original Filename	Extracted image filename (.img)	Log output (.xlsx)
snapshot-alpha_mixed-1-meterpreter-reverse-tcp-2019-01-02_09-53-57.aff4	pos1	psx_pos1
snapshot-alpha_mixed-1-meterpreter-reverse-tcp-2019-01-02_09-55-37.aff4	pos2	psx_pos2
snapshot-alpha_mixed-6-meterpreter-reverse-tcp-2019-01-02_12-10-56.aff4	pos3	psx_pos3
snapshot-alpha_upper-3-meterpreter-reverse-tcp-2019-01-03_10-14-33.aff4	pos4	psx_pos4
snapshot-alpha_upper-4-meterpreter-reverse-tcp-2019-01-03_10-41-34.aff4	pos5	psx_pos5
snapshot-hyperionPesScrmbler-alpha_mixed-1-meterpreter-reverse-tcp-2019-02-25_09-27-58.aff4	pos6	psx_pos6
snapshot-hyperionPesScrmbler-alpha_mixed-1-meterpreter-reverse-tcp-2019-02-25_09-29-25.aff4	pos7	psx_pos7
snapshot-hyperionPesScrmbler-alpha_mixed-4-meterpreter-reverse-tcp-2019-02-25_10-15-33.aff4	pos8	psx_pos8
snapshot-hyperionPesScrmbler-alpha_upper-2-meterpreter-reverse-tcp-2019-02-25_13-23-6.aff4	pos9	psx_pos9
snapshot-hyperionPesScrmbler-bloxor-1-meterpreter-reverse-tcp-2019-02-26_09-36-5.aff4	pos10	psx_pos10

snapshot-shellter-payload-alpha_mixed-1-meterpreter-reverse-tcp-2019-01-29_16-0-16.aff4	pos11	psx_pos11
snapshot-shellter-payload-alpha_mixed-1-meterpreter-reverse-tcp-2019-01-29_16-2-35.aff4	pos12	psx_pos12
snapshot-shellter-payload-alpha_mixed-5-meterpreter-reverse-tcp-2019-01-30_12-27-0.aff4	pos13	psx_pos13
snapshot-shellter-payload-alpha_mixed-8-meterpreter-reverse-tcp-2019-01-30_14-8-40.aff4	pos14	psx_pos14
snapshot-shellter-payload-cmd-brace-3-meterpreter-reverse-tcp-2019-01-15_09-50-28.aff4	pos15	psx_pos15
snapshot-alpha_upper-1-meterpreter-reverse-tcp-2019-01-02_15-55-2.aff4	pos16	psx_pos16
snapshot-hyperionPesScrmbler-alpha_upper-3-meterpreter-reverse-tcp-2019-02-25_13-45-22.aff4	pos17	psx_pos17
snapshot-shellter-payload-alpha_upper-3-meterpreter-reverse-tcp-2019-01-30_15-23-9.aff4	pos18	psx_pos18
snapshot-call4_dword_xor-2-meterpreter-reverse-tcp-2019-01-07_10-37-34.aff4	pos19	psx_pos19
snapshot-cmd-generic_sh-1-meterpreter-reverse-tcp-2018-12-13_12-31-17.aff4	pos20	psx_pos20

Table 2 Downloaded Malicious memory images

3.5 Conclusion

Overall, the analysis was focused on more process related artefacts that can be used to identify suspicious process and use plugins that can reveal some signs of malicious activities in a compromised images. Despite there are other areas of where artefacts of the process-related logs are obtainable through the extraction, but it is best to extract artefacts when a memory identified as malicious. The Volatility Framework does allow extraction and inspection using main parameters of the malicious process which are PID and offset address as these allow to locate any infected source that resident in physical memory sections with executables for the identified suspicious process. In terms of the artefact log that can be utilized to apply the machine learning approach is the psxview.as it provide more context of the running process as active and inactive state with sign of process behaviors including behavior detection of rootkits that includes signs of malwares behavior. Therefore, the main finding of this analysis was to reveal an artefact that can be used to identify memory as benign and malicious and used to apply in machine learning model.

Chapter 4 : Specification and Design

This chapter of the document outlines an implementation and design structure of the classifier tool to be developed, including initial design specification, system requirements and development strategy.

4 Tool Requirements

As a basic principle of any development of a software or tool a set of functional and non-functional requirements are produced prior the programming and implementation of the software. These specified requirements are beneficial to develop the tool in a proper direction and flow to produce a tangible working product.

4.1 Functional Requirements

The main functional specifications and sub requirement components of the developing tool are described as the following list:

4.1.2 Data Collection and Pre-processing requirement specification :

The tool system must have a feature that allows the importing and loading data from the raw artefact log files from user input with their labels to create a preprocessed dataset to be used for training and testing the classification models.

1. The system must be able to obtain all raw logs files data directly from user input for the specified labeled directories
2. The system selects the required features and labels the raw data based on their obtained and specified labeled directories
3. The system creates a dataset by merging and preprocessing the labelled raw data
4. The system outputs a preprocessed dataset in an appropriate format to be used for the Machine Learning Classification models.

4.1.3 Machine Learning Classification requirement specification:

The tool system must have a feature to import and load processed data from the preprocessed dataset file that is provided by user input for training and testing the Machine Learning Classification models.

1. The system must be able to load and read processed data from the dataset file from user input by providing the dataset location directory path
2. The system should define and set the features and labels of the imported dataset
3. The system should have the best performing Machine Learning classification model
   3.2 Develop and create Machine learning models using different classification algorithms which are Random Forest , Decision trees, Neural Network , Naïve Bayes, Support Vector Machines.
   3.3 Train, test and evaluate each model and select best performing classification model
4. The system is able to train the selected classification model with provided dataset
5. The system is able to test the selected classification model to predict and classify unseen data
6. The system should be able to provide an output of a classification document of trained and tested model with prediction outcomes.

Some additional functional requirements for the tool if time permitted:

1. Provide a feature that allow a system to output a short summary that categories and sort memory dump for investigation via weighting number of benign and malicious process existed in a memory log artefact.
2. Provide a feature that allows to import and load data directly from the raw memory dump from the user input with their label to extract a preprocessed log file to test against trained Machine Leaning classification model.
3. Provide a feature that allows system to save and load the trained classification models for testing model with unseen log.

4.2 Non-Functional Requirements

The non-functional specifications of the developing tool are described in the following list:

- ✓ Usability
  - A Graphical User Interface will be developed that will allow ease for user interaction with the system.
- ✓ Reusability
  - The tool will be implemented as simple program that can be reusable in another system.
- ✓ Reliability
  - The tool will be performing the system functions without any expected failures or errors.
- ✓ Speed and Performance
  - the process of data collection, preprocessing and outputting dataset should be fairly quick as well as training and testing the model should be fast with results output and files

4.3 System Implementation Architecture

The system will be implemented as two separate program interface prototypes. The main reason of dividing the system implementation is mainly to avoid the complexity and allow handlining any modification easily for performing a direct specified task. In addition, this implementation architecture considered to be as suitable approach with consideration of the project time frame. Therefore, as the requirement of the dataset to train and test the ML model a **Data Collection and Pre-processing** tool will be developed. This tool will be used in order to automate the process of the data collection and preprocessing to provide a preprocessed dataset with labels. The second tool that will be developed as **Classifier**, where best performing classification Machine Learning model will be implemented, and tool classification model will be supplied with processed dataset to train the model and test model for the classification document outcome .

System Structure will be based on following two separate units:

1. Data collection and Preprocessing - mainly responsible to create a preprocessed dataset with labels
2. Machine Learning Classifier – mainly responsible to train the model with selected features and training dataset and test unseen data with classification trained model.

4.3.1 System Design

In order to understand the implementation structure and behavior of the developing system, a software design modelling solution Unified Modelling Language (UML) diagram is used. This approach helps developers to visualise the design architecture of the program prototype and understand the context of the program functionality. In addition, UML diagrams drive the implementation and assist developers in implementing the system requirements. Therefore , an Use case Diagram is created for the **Classifier** tool to understand the behavior interaction between the user and system as well as system actions that holds the functional requirements and is shown below Figure 45 -46.

4.3.2 Use Case Diagram:

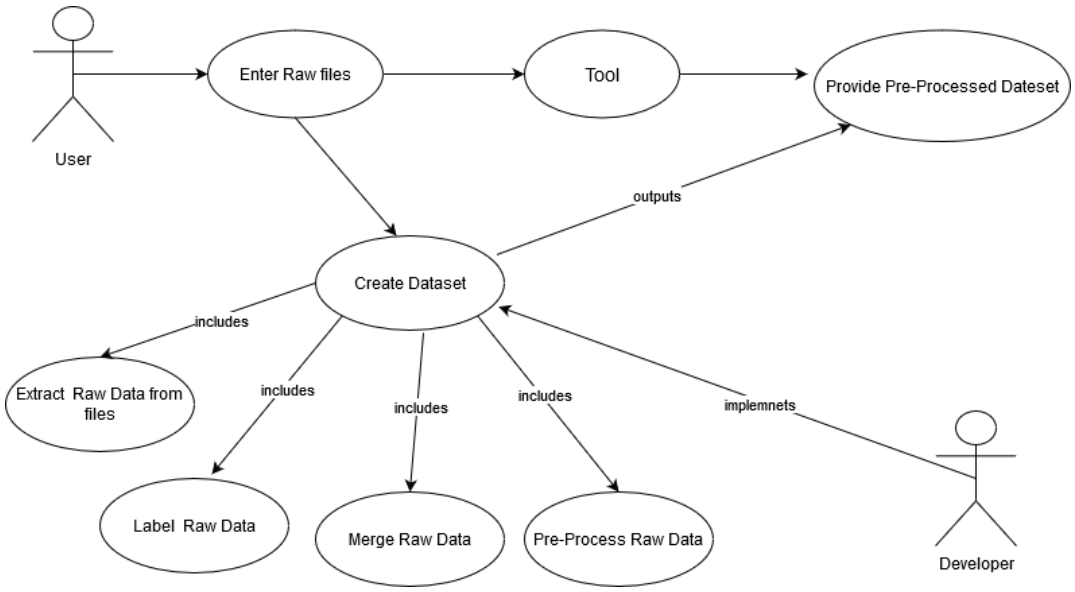

Figure 45 Use Case : Data collection and Pre-processing tool

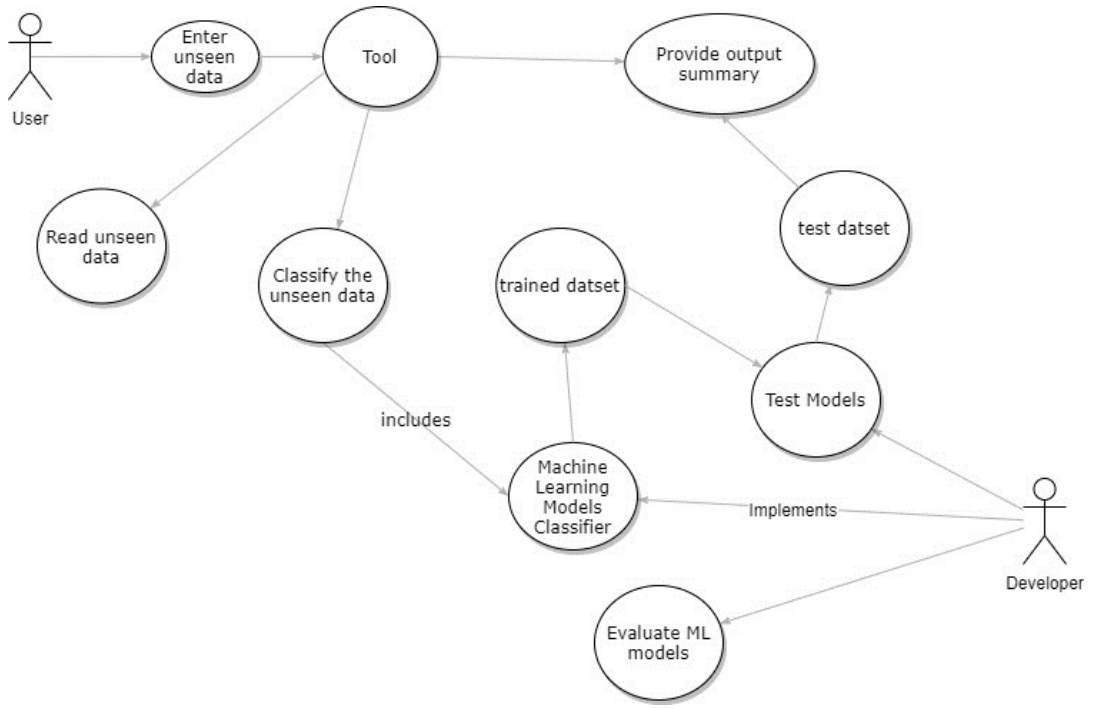

Figure 46 Use Case : Classification tool

4.4 Tool Development Methodology

To deliver a working prototype program of the tools, a development strategy has been set out for the implementation and testing of the tools with consideration of timeframe of the project. The tools will be developed following the Agile Software Development Methodology. This strategy allows dividing the project tasks into smaller increments to deliver a tool feature iteratively. Once each of the project increments or iterations are completed then easily combined and presented to the customer as a working product. The main reasons of adapting this development strategy for the project is that Agile methodology allows continues milestone deliverables of the tool requirements as well as easy to handle any changes to the tool requirements at the start or during the implementation phase and even later stage of the development without effecting other operating functions of the tool. As this project initial tools, the functional specifications are expected for alteration and modification as it depended on the analysis and data collection phase before the implementation phase. Also, this methodology allows flexibility to the development

process as it allows to prioritize in delivering tool functional requirements to have at least a working prototype that provides initial results if not fully functional**.** Besides, presenting a working prototype to the customer allows to receive feedback for improvement to achieve optimal project results. Therefore, this project will have three main four iterations to develop the prototype tools as following:

- **1st iteration**: the implementation of tool for the data collection and preprocessing functions which includes loading memory dump's log artefacts and executing preprocessed log file.

    - Deliverable :Labeled processed dataset and logs files with Name encoded labels

- **2nd iteration**: pre-modelling of the different Machine Leaning Classifiers using the different classification algorithms of Machine Leaning .

    - Deliverable : pre-processed dataset and at least 5 models to be developed

- **3rd iteration:** train, test and evaluate the Machine Learning Classifiers with preprocessed dataset
    - Deliverable :Evaluation and predication outcome documents of each Machine Learning Classifier

- **4th iteration:** developing Classifier Interface with best performing Machine Learning Classifier
    - Deliverable : Desired functional requirements and improve tool usability and improve user interaction

Chapter 5: Implementation

This section of the document will details the implementation of the tools including the main implementation code of data collection and preprocessing, Machine Learning models and Classifier with regards to used programming tools.

Constraints

The time constraint and challenges of this university project has impacted and limited the implementation of the tool desired functional requirements and specification. The initial requirements of the tool are prioritized for the implementation and the tool will be developed as a working prototype following a simple implementation method.

5.1 Overview

The main implementation of the prototype tool is divided into two separate interface implementation and development and are presented in the list below :

1. Data Collection and Preprocessing Graphical User Interface implementation, mainly responsible for importing, loading, and preprocessing the raw data of psxview logs and creating a validated preprocessed labeled dataset file as csv. In addition, all the input raw logs are collected from the user input specified labeled directories "data-input" .Also, the output dataset is saved and stored within a directory called "data-output" that is held in Data Collection and Preprocessing directory.

2. Different Machine Learning Classification Models were implemented and developed as simple python scripts in order to train, test and evaluate for best performing model for the classification of the given dataset. Classification models that were developed :Random Forest , Decision trees, Neural Network , Naïve Bayes, Support Vector Machines. Each of these models provides a report as text file which includes the evaluation and performance metrics and prediction outcomes of the model for given dataset.

3. Classifier Graphical User Interface implementation, mainly responsible for importing processed dataset, training the model, testing, and classifying unseen data. In addition, this implementation includes the development of the best performing Machine Learning classification model that is used as memory classifier.

5.2 Data Collection and Pre-processing Interface

The initial approaches towards the development of machine learning models is collecting and importing the data with their labels. Thus, a prototype tool is implemented to load the raw psxview log data from two specified directories from user input. Each importing data directory is a label for the data obtained.

5.2.1 Importing and Loading the Data

Data Collection and Preprocessing module is initialized as GUI interface userwindow.mainloop(), using the python built in library tkinter .The interface is consist of main functions of taking user input for importing malicious and benign of raw psxview artefact data .

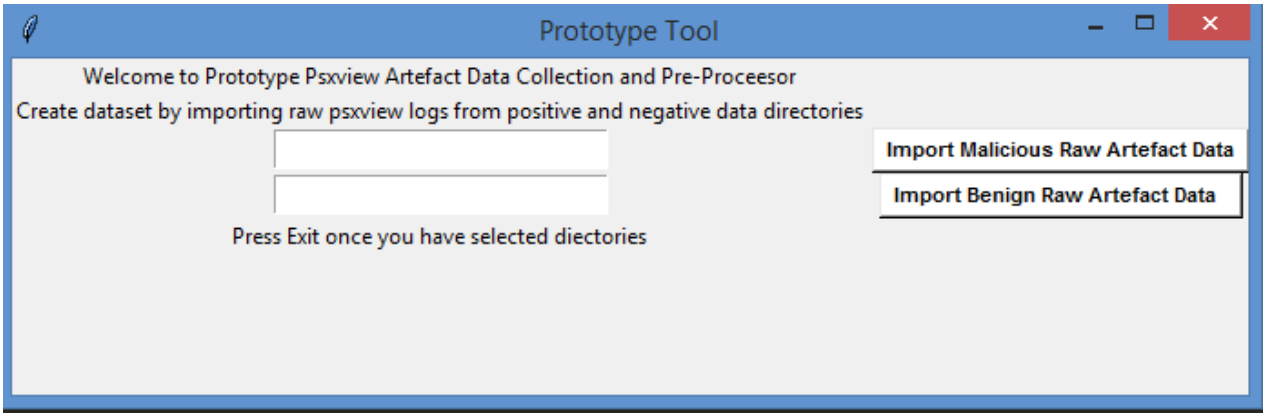

Figure 47 - Data Collection and Pre-processing Interface

The library tkinter provides useful function of asking user for the selecting the directories. Once the user selects the directories of where labeled raw artefact logs are located, the paths for each of these directories are collected as global variable by two functions select_N_folder() and select_N_folder().

```
#select positive dir to load files for creaing dataset
def select_P_folder ():
    global pos_dir_path

    import_P_path = fd.askdirectory(title='Select Positive Data Folder')
    #pos dir path
    pos_dir_path = (import_P_path)
    ent1.insert(tk.END, pos_dir_path)
```

Figure 48 select_N_folder()

```
#to select negative dir to load files for creaing dataset
def select_N_folder ():
    global neg_dir_path
    #neg dir path
    import_N_path = fd.askdirectory(title='Select Negative Data Folder')
    neg_dir_path = (import_N_path)
    ent2.insert(tk.END, neg_dir_path)
```

Figure 49 select_N_folder()

Next, functions load_pos_data(pos_dir_path) is called. This function takes the directory path of that holds the positive/malicious raw psxview log and uses python built-in function oslistdir(), returns a list of all the excel files of **.xlxs extension** in selected directory and append these files with their path into listfile_in_dir[] .To ensure that data read as excel file structure, the required libraries pandas and xlrd were imported. Once all files paths are loaded and stored in listfile_in_dir, an empty dataframe is initialized and created to load, read, and store the data from each of these imported files and then added to dataframe by loop as shown in Figure 50.

As to cleanse the raw dataset from unnecessary data and useless features that are not be utilized for modelling are eliminated. In our case, the last column is dropped (Exit Time) and added a new label column for the raw data classification (Class). For malicious imported files are labeled with as 1 as they imported from positive directory. A similar function is implemented called load_neg_data (neg_dir_path), for the benign files as they are labeled with 0 as they are imported from the negative directory. Each of these functions return a positive dataframe called pdf and negative dataframe called ndf .

```python
#--------------------------------      raw data input section    --------------------------------#
def load_pos_data(pos_dir_path):
    files_in_dir = []
    #load all files pos raw .xlsx in list with filename and path
    for files in os.listdir(pos_dir_path):
        if (files[-4:] == "xlsx"): # only selects xlxs
            files_in_dir.append(os.path.join(pos_dir_path,files))
    pos_dir_path = pos_dir_path +"/" +files
    # Initialize empty dataframe
    df = pd.DataFrame()
    # loop for each file in the list and then process the data and merge all files together
    for f in files_in_dir:
        raw_data = pd.read_excel(f) # read each file
        df = df.append(raw_data) # each file data is added to df
        df= df[df.columns[:-1]] # removed last ( Exit Time ) column
        #---Add Malicious Classification Column---#
        df['Class'] = 1
        pdf = df
    #return positive dataset
    return pdf

def load_neg_data(neg_dir_path):
    files_in_dir = []
    #load all files neg raw .xlsx in list with filename and path
    for files in os.listdir(neg_dir_path):
        if (files[-4:] == "xlsx"): # only selects xlxs
            files_in_dir.append(os.path.join(neg_dir_path,files))
    neg_dir_path = neg_dir_path +"/" +files
    # Initialize empty dataframe
    df = pd.DataFrame()
    # loop for each file in the list and then process the data and merge all files together
    for f in files_in_dir:
        raw_data = pd.read_excel(f) # read each file
        df = df.append(raw_data) # each file data is added to df
        df= df[df.columns[:-1]] # removed last ( Exit Time ) column
        #---Add Benign Classification Column---#
        df['Class'] = 0
        ndf = df
    #return negative dataset
    return ndf
```

Figure 50 load_pos_data() load_neg_data()

5.2.2 Data Pre-processing

In order to process data for machine learning model for the training, testing and the classifier operations, the raw data should be in a numerical values and formats as strings and categorical data are useless to Machine Learning models.

Function data_pre(pos_dir_path, neg_dir_path) is implemented to merge using the function concat() for both benign and malicious log dataframe as one dataset and return a pre-partial processed dataset as data. The dataframe rows are randomized using shuffle() from sklearn.utils

and rest index to create a standard format of dataset for Machine Learning models without the order pattern of classification labels. For the accessibility of the column names of dataframe, (Offset (P)) column is renamed to (Physical_Offset) and the values of this column are converted using python built in function of converting hex values in decimal values using lambda and int., In addition, as most of the dataset features contain categorical data, the dataframe columns of categorical data of True and False are mapped as 1 and 0 Boolean values. Additionally, PID numerical column, is also transferred as Boolean values as this feature is used to identify whether there is any null values or values less than 1 for the process ID.

```
#--------------------------------      data pre-processing section    ------

def data_pre(pos_dir_path,neg_dir_path):
    # gather data together into one dataframe
    pdf= load_pos_data(pos_dir_path)
    ndf=load_neg_data(neg_dir_path)
    data = pd.concat([pdf, ndf])
    #print(data.info())

    #randomize the rows
    data = shuffle(data)

    #reset index  after rows shuffuled
    data.reset_index(drop=True, inplace=True)

    #---Rename Column---#
    data.rename(columns={'Offset (P)' : 'Physical_Offset'}, inplace = True)

    #---- Convert Physical_Offset hex values into decimal values ---#
    data['Physical_Offset'] = data['Physical_Offset'].str.rstrip('L')
    hex_values= pd.Series(data['Physical_Offset'])
    b16 = lambda hex_values: int(hex_values,16)
    data['Physical_Offset']=hex_values.apply(b16)

    #--- Converts numerical values into 1/0 ---#
    data['PID'] = (data['PID'] >= 1).astype(int)

    ## 1 = True , 0 = False //  Malicious = 1 , Benign = 0

    #---- Convert Boolean values into 1/0  ---#

    data['pslist']=data['pslist'].map({True: 1, False: 0})
    data['psscan']=data['psscan'].map({True: 1, False: 0})
    data['thrdproc']=data['thrdproc'].map({True: 1, False: 0})
    data['pspcid']=data['pspcid'].map({True: 1, False: 0})
    data['csrss']=data['csrss'].map({True: 1, False: 0})
    data['session']=data['session'].map({True: 1, False: 0})
    data['deskthrd']=data['deskthrd'].map({True: 1, False: 0})

    return data
```

Figure 51 data_prep()

labelName(pos_dir_path,neg_dir_path), function used to fill null values as null string and encode the strings of the (Name) column into numerical labels, which is Process Name string that are encoded with number labels. The preprocessing function Label Encode() and fit.transform is utilized from sklearn library .In addition, the function also fills null values as null string using fillna method and at the end the function returns a fully processed dataset with encoded process name labels.

```
def labelName(pos_dir_path,neg_dir_path):
    #---- Create labels for Name  ---#
    data=data_pre(pos_dir_path,neg_dir_path)
    #fill null values with unknown
    data["Name"].fillna("null", inplace = True)
    data['Name']=data['Name']

    # Encode Labels with numbers starting from 0
    le = preprocessing.LabelEncoder()

    #Replace original label values in dataframe with encoded labels
    data['Name'] = le.fit_transform(data['Name'])

    # #Returns original labels
    # # data['Name']= le.inverse_transform(data['Name']) to get back original
    # # print(data.tail(10))

    return data,le
```

Figure 52 labelName()

The last function is the save_data(pos_dir_path,neg_dir_path) , which execute two csv output files that are the preprocessed dataset and the other file is consist of the encodings of the Name column and the encoded numerical values to be used later for identify Process Name. Once the function save_data(pos_dir_path,neg_dir_path) is called after the user selects directories it run all the functions and save the outputs of the dataframe as **.csv file** in the “test folder” via the absolute path for the directory.

```
#-------------------------------      dataset output section    -------------------------------#

def save_data(pos_dir_path,neg_dir_path):

    data,le=labelName(pos_dir_path,neg_dir_path)
    file_dir = os.path.dirname(os.path.abspath(__file__))
    #output directory
    csv_folder = 'test'
    file_path = os.path.join(file_dir, csv_folder, 'Testdataset1.csv')

    #Store Dataset as csv output file
    output = data.to_csv(file_path, index = False, header=False)

    #Store Process Labels as df
    Name_Labels=list(zip( le.transform(le.classes_),le.classes_))
    # print(Name_Labels)
    df = pd.DataFrame(Name_Labels, columns = ['Name', 'ID'])
    # print (df.tail(5))
    #labels output file
    file_path2 = os.path.join(file_dir, csv_folder, 'labels1.csv')
    output = df.to_csv(file_path2, index = False, header=True)
    messagebox.showinfo("Completed ","Dataset created in dataset-output folder ")

save_data(pos_dir_path,neg_dir_path)
```

Figure 53 save_data()

5.2.3 Machine Learning Classification Models

Five different types of machine learning classifier models were developed using classification algorithms imported from Scikitlearn classifier library which includes Random Forest , Decision trees, Neural Network , Naïve Bayes, Support Vector Machines. Each of these models are constructed as simple python program and as initial development for the classifier models. The main aim of these models to be developed is to evaluate the performance of the machine learning classification models against the dataset.

Each of the models, have a similar implementation pattern that consist of a main function load_data(filename), that intakes user input to import a processed dataset using dataset location path via the function input() .The program only accepts **csv file** as the program will read the imported file **as csv** using the pandas and creates a dataframe called dataset with assigned column names and return dataset.

```
print("Welcome")
filename=input('Enter the psxview dataset file path of .csv extension :')

def load_data(filename):
# load the dataset
    col_names = ['Physical_Offset','Name','PID','pslist','psscan','thrdproc','pspcid','csrss','session','deskthrd','Class']
    dataset= pd.read_csv(filename, header=None, names=col_names, delimiter=',')

    return dataset
```

Figure 54 load_data()

The dataset passed is to following function which is the set_feature(filename). This function sets the dataset as array and sets the features and the target selection of the dataset. All of the first features ['Physical_Offset', 'Name', 'PID', 'pslist',' psscan', 'thrdproc', 'pspcid', 'csrss', 'session', 'deskthrd'] from dataset are considered and the last column of the **[Class]** is used as target to classify features against it. As function set_feature(filename) is called it will execute two variables that are the features and labels of the dataset.

```
def set_features(filename):
    dataset = load_data(filename)
    # dataset as array
    array = dataset.values
    # Features Selection  take columns 0 to 9 from( Physical_Offset .... Class)
    features = array[:,0:10]
    # Target Selection Class = Label
    labels = array[:,10]

    return features, labels

features, labels=set_features(filename)

X_train, X_test, Y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=17) # 70% training and 30% test
```

Figure 55 set_features()

As one of the machine learning model requirements is to fit model with training and testing dataset. Therefore, a common method of data splitting is utilized from the Scikit-learn library which is train_test_split() to split dataset features and labels into two subsets that are training, and testing dataset. The portion of the dataset split is based on the test size parameter of the function which is set as 70% for training and 30% for testing and this split is used as standard split of machine learning models to avoid the overfitting and modelling error.

The training subset of features are used to train the model with their labels and the testing subset features are used to test against trained model for model prediction outcomes for the labels. Once

the dataset is split , the model is initialized as function for classification algorithms as SVC() and then assigned variables of the training set are passed to fit and train the model and predication outcomes of unseen testing set against the trained model predict() is used with the testing set the assigned variables are passed as parameter to fit model.

```
#Time at Start of Model Generating
start_time = time.time()
#----- SVC Classifier ------#
model = SVC( gamma='auto')
model.fit(X_train, Y_train)
```

Figure 56 initialize machine learning model

And this part of the program implementation consists of executing an output of document **.text file** using the open() with text parameter. The executed **.text** document includes the evaluation results of the training model and evaluation of testing unseen set against trained model with evaluation metrics of the Machine Learning models as mentioned in background. Scikit-learn metric library was utilized to import the evaluation metrics function s which includes Accuracy, Precision, and Recall ,Confusion Matrix, classification report. The list below shows the evaluation metric with the function used from Scikit-learn and Figure 57 shows how they called in the code

```
print("\n**************** Training Model Evaluation Report  ****************\n",file=f)
preds_train = model.predict(X_train)
print("Trained Model Accuracy of model: {}".format(round(accuracy_score(preds_train, Y_train),2)*100),file=f)
print("Trained Model Precision: {}".format(round(precision_score(preds_train, Y_train),2)*100),file=f)
print("Trained Model Recall :{}".format(round(recall_score(preds_train, Y_train),2)*100),file=f)
print("Trained ModelF1 Score: {}".format(round(f1_score(preds_train, Y_train),2)*100),file=f)

print("\n****************** Classification Report:  ******************\n",file=f)
print(classification_report(preds_train, Y_train),file=f)
print('Confusion Matrix:',file=f)
#cmtx table
cmtx_label = np.unique([preds_train, Y_train])
cmtx = pd.DataFrame(
    confusion_matrix(preds_train, Y_train ,labels=cmtx_label),
    index=['Actual :{:}'.format(x) for x in cmtx_label],
    columns=['Prediction:{:}'.format(x) for x in cmtx_label])
print(cmtx,file=f)
print("\n 0 = Benign , 1 = Malicious \n",file=f)
print("--- Time : %s seconds took to generate model with prediction ---\n" % (time.time() - start_time),file=f)

print("\n**************** Testing Model Evaluation Report  ****************\n",file=f)
preds_test = model.predict(X_test)
for i in range(5):
    print('%s => %d (expected %d)' % (X_test[i].tolist(), preds_test[i], y_test[i]),file=f)
print("",file=f)
print("Test accuracy of model: {}".format(round(accuracy_score(preds_test, y_test),2)*100),file=f)
print("Test Precision: {}".format(round(precision_score(preds_test, y_test),2)*100),file=f)
print("Test Recall :{}".format(round(recall_score(preds_test, y_test),2)*100),file=f)
print("Test F1 Score: {}".format(round(f1_score(preds_test, y_test),2)*100),file=f)

print("\n****************** Classification Report:  ******************\n",file=f)
print(classification_report(preds_test, y_test),file=f)
print('Confusion Matrix:\n',file=f)
#cmtx table
cmtx_label = np.unique([preds_test, y_test])
cmtx = pd.DataFrame(
    confusion_matrix(preds_test, y_test ,labels=cmtx_label),
    index=['Actual :{:}'.format(x) for x in cmtx_label],
    columns=['Prediction:{:}'.format(x) for x in cmtx_label])
print(cmtx,file=f)
print("\n 0 = Benign , 1 = Malicious \n",file=f)
print("--- Time : %s seconds took to generate model with prediction ---\n" % (time.time() - start_time),file=f)
f.close()
```

Figure 57 evaluation metrics and output functions for Classifier Model

Other classification models are based on similar implementation, but they consist of different classification algorithm function as MLPClassifier(), RandomForesetClassifier(), DecisionTreeClassifier() and GaussianNB().

5.3 Machine Learning Classification Interface

After the implementation and evaluation of the different Machine learning models for the best performing classification model the following Interface is developed as Classifier. The evaluation and results outcome of the different models as classifiers are discussed in the **Chapter 7** Results of this document.

This interface consists of the implementation of the main functions of importing processed dataset and provides classification outcomes of the training dataset and prediction outcome of unseen dataset against the trained model **as document text file** and a **visual diagram of the classification**.

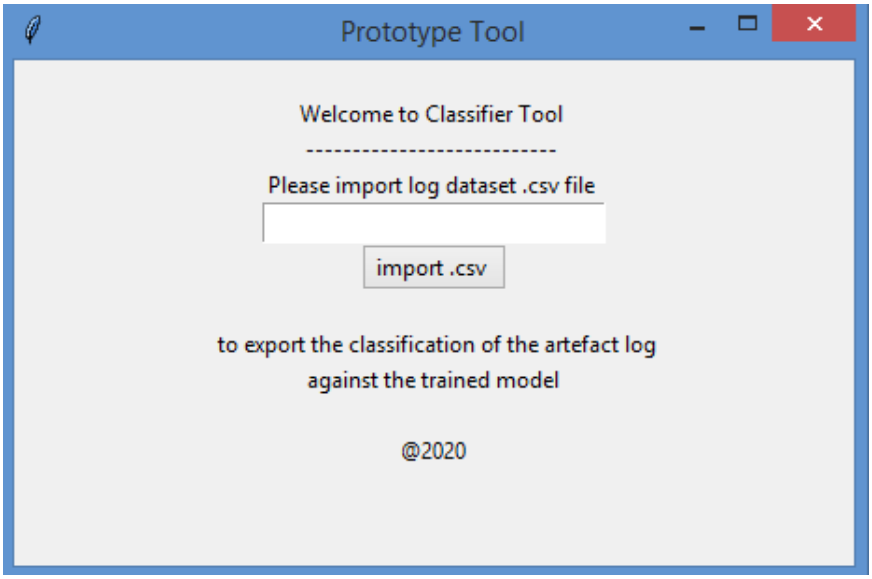

Figure 58 Classifier Prototype Tool Interface

The Classifier interface is initialized with userwindow.mainloop(), The main function of this Classifier is the importing the dataset using the getCSV (), where a small prompt will be displayed to the user to select the preprocessed **.csv file.**

```
# import csv and import csv
def getCSV ():
    global df

    import_file_path = fd.askopenfilename(title='Select .csv',)
    col_names = ['Physical_Offset','Name','PID','pslist','psscan','thrdproc','pspcid','csrss','session','deskthrd','Class']
    df = pd.read_csv (import_file_path, header=None, names=col_names, delimiter=',')
    csv_path = (import_file_path)
    ent1.insert(tk.END, csv_path)
    return df
```

Figure 59 getCSV ()

Once the preprocessed dataset is imported the tool will run through the functions set_feature() for setting the features and target. Then function classify(), split dataset into train and test sets as 80 % for training and 20% testing and fit to the Random Forest Model to output the document of the classification. The Classier tool implementation is based on the Machine Learning Random Forest Classier Algorithm with similar implementation pattern as shown in Figure 60.

```
#Create model and output txt report for classification
def clasify():

    features, labels=set_features()
    X_train, X_test, Y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=17) # 70% training and 30% test

    #Time at Start of Model Generating
    start_time = time.time()

    #Bulid RandomForestClassifier
    model=RandomForestClassifier(n_estimators=100)
    model.fit(X_train, Y_train)

    f = open("Classifier Report.txt","a")
    print("----------------------------------------------\n",file=f)
    today = date.today()
    d3 = today.strftime("%m/%d/%y")
    t = time.localtime()
    current_time = time.strftime("%H:%M:%S", t)
    print("Date: ",d3," || Time: ",current_time,file=f)
    print("Data Split size is .20",file=f)
    print("Train data  ",X_train.shape,Y_train.shape,file=f)
    print("Test data  ",X_test.shape, y_test.shape,file=f)
    preds_train = model.predict(X_train)
    print("---- Classifying of 5 cases of trained data----\n",file=f)
    for i in range(5):
        print('%s => %d (expected %d)' % (X_train[i].tolist(), preds_train[i], Y_train[i]),file=f)
    print("\n----     Confusion Matrix :   ---- \n",file=f)
    #cmtx table
    cmtx_label = np.unique([preds_train, Y_train])
```

Figure 60 classify ()

Once the output document text file is created a notification will be displayed and a visual diagram is also displayed a function visualizer () was utilized from the yellowbrick.classifier library.

```
messagebox.showinfo("Completed ","classification report is created in existed folder ")

#output a visualized table of classification
visualizer = ConfusionMatrix(model, classes=[0,1])
visualizer.fit(X_train, Y_train)        # Fit the visualizer and the model
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()                       # Finalize and show the figure
```

Figure 61 Visualizer

Chapter 6: Testing

This section of the document consists of the testcases that were used to test the prototype tools in order to test if they consist the main functional requirements as mentioned in design and specification. Each testcase is tested on Host Machine Windows 8.1 Operating System and using the command line prompt to run the prototype tools and for the initial machine learning scripts.

Test Case Id: A1	
Test Case Tittle	Create Processed Dataset
Precondition	Extracted psxview raw logs from malicious and benign memory images are placed in separate directories as positive and negative directories
Test Case Description	Importing, loading raw psxview logs and output a preprocessed dataset as **Testdataset.csv** file
Test Case Steps:	1. User need to select a malicious/positive directory that consist of malicious/positive psxview logs to be imported 2. A small prompt will open that allow user to navigate and browse to select the desired directory 3. User need to select a benign/negative directory that consist of benign/negative psxview logs to be imported 4. A small prompt will open that allow user to navigate and browse to select the desired directory 5. Once the correct directories inputted the user need to exit the interface 6. The output file is saved locally in predefined directory.
Test Case Outcome: Preprocessed Dataset	
Test Case Passed with mentioned sequenced steps	
Related Test : -	

Table 3 Test Case ID: A1 Creating Processed Dataset

Test Case Id: A2	
Test Case Tittle	Support Vector Classifier
Precondition	Must have a previously created preprocessed dataset
Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the training and testing data
Test Case Steps:	1. User need to provide a preprocessed dataset **csv file** path to be imported 2. Once the correct csv file is imported the program appends evaluation and classification results to a text file 3. The output file is saved in the directory
Test Case Outcome : Classification and predication outcome document	
Related Test : Test Case Id: A1	

Table 4 Test Case ID: A2 Support Vector Classifier

Test Case Id: A3	
Test Case Tittle	Decision Tree Classifier
Precondition	Must have a previously created preprocessed dataset
Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the training and testing data
Test Case Steps:	1. User need to provide a preprocessed dataset **csv file** path to be imported 2. Once the correct csv file is imported the program appends evaluation and classification results to a text file 3. The output file is saved in the directory

Test Case Outcome : Classification and predication outcome document
Related Test : Test Case Id: A1

Table 5 Test Case ID: A3 Decision Tree Classifier

Test Case Id: A4	
Test Case Tittle	Naïve Bayes Classifier
Precondition	Must have a previously created preprocessed dataset
Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the training and testing data
Test Case Steps:	1. User need to provide a preprocessed dataset **csv file** path to be imported 2. Once the correct csv file is imported the program appends evaluation and classification results to a text file 3. The output file is saved in the directory
Test Case Outcome : Classification and predication outcome document	
Related Test : Test Case Id: A1	

Table 6 Test Case ID: A4 Naive Bayes Classifier

Test Case Id: A5	
Test Case Tittle	Random Forest Classifier
Precondition	Must have a previously created preprocessed dataset
Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the training and testing data
Test Case Steps:	1. User need to provide a preprocessed dataset **csv file** path to be imported

	2. Once the correct csv file is imported the program appends evaluation and classification results to a text file 3. The output file is saved in the directory
Test Case Outcome : Classification and predication outcome document	
Related Test : Test Case Id: A1	

Table 7 Test Case ID : A5 Random Forest Classifier

Test Case Id: A6	
Test Case Tittle	Neural Networks Classifier
Precondition	Must have a previously created preprocessed dataset
Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the training and testing data
Test Case Steps:	1. User need to provide a preprocessed dataset **csv file** path to be imported 2. Once the correct csv file is imported the program appends evaluation and classification results to a text file 3. The output file is saved in the directory
Test Case Outcome : Classification and predication outcome document	
Related Test : Test Case Id: A1	

Table 8 Test Case ID : A6 Neural Networks Classifier

Test Case Id: A7	
Test Case Tittle	Classifier Training and Testing
Precondition	Must have a previously created preprocessed dataset

Test Case Description	Importing a preprocessed dataset and provide classification document and predication outcome of the testing data
Test Case Steps:	1. User need to select a **csv file** of the preprocessed dataset to be imported **2.** A small prompt will open that allow user to navigate and browse to select the desired **csv file** 3. Once the correct csv file is imported the program executes a text file and a visual diagram of the prediction classification of the unseen data 4. A notification message displayed once the outcome is executed 5. The output file is saved in the directory
Test Case Outcome : Classification and predication outcome document and Visual Diagram	
Related Test : Test Case Id: A1	

Table 9 Test Case ID: A7 Classifier Training and Testing

Chapter 7 : Results and Evaluation

This chapter of the document evaluates the different Machine Learning classification models to identify the best performing Machine learning model as classifier against the given dataset.

7.1 Classification Models Performance

Evaluating the performances of the Machine learning classifier model is crucial to for ensuring that the selected model is working optimally and outputting effective insights and results as intended classifier. Therefore, after each of the 5 machine learning classification models (Random Forest , Decision trees, Neural Network , Naïve Bayes, Support Vector Machines) were implemented. The models were trained with 70 % of training dataset and were tested with 30% of unseen testing dataset and tested models were evaluated for their performances and effectiveness. The following comparison table of model evaluation was created to compare and evaluate the performances of the different classifier models for the classification task. The tables consist of the evaluation results that are based on utilizing the main classification metrics focused on Accuracy, Precision ,Recall and F1 with execution time for the results outcome of the Machine Learning classification models.

Testing Models/Metric Scores	Accuracy	Precision	Recall	F1	Time /s
Support Vector	89.0	100.0	84.0	91.0	0.0013
Naïve Bayes	67.0	68.0	70.0	69.0	3.9
Decision Tree	89.0	87.0	92.0	90.0	4.2
Random Forest	93.19	95.27	91.85	93.53	3.9
Neural Networks	78.0	76.0	82.0	79.0	0.00135

Table 10 Performance of various algorithms

According to the evaluation results obtained of testing dataset against the trained model is presented in the Table 10, The Random Forest Classifier model ranked higher in the overall accuracy, precision and F-measure with 93.19 %, 95.27% and 93.53 with in 3.9 seconds compare Naïve Bayes and other models. Besides, Random Forest Classifier model is the only model that achieved and scored high for the majority of evaluation metrics and considerably represent the best performing classifier model to be for the psxview log dataset comparing to other Machine Learning classification models in the Table 10.

7.2 Predication results analysis

Since we are interested on how the best performing model is classifying the benign and malicious process behaviour with consideration of all the features against the target selection. Further evaluation metric of confusion matrix was applied to get better insights of the model predication performance relative to the baseline of the dataset.

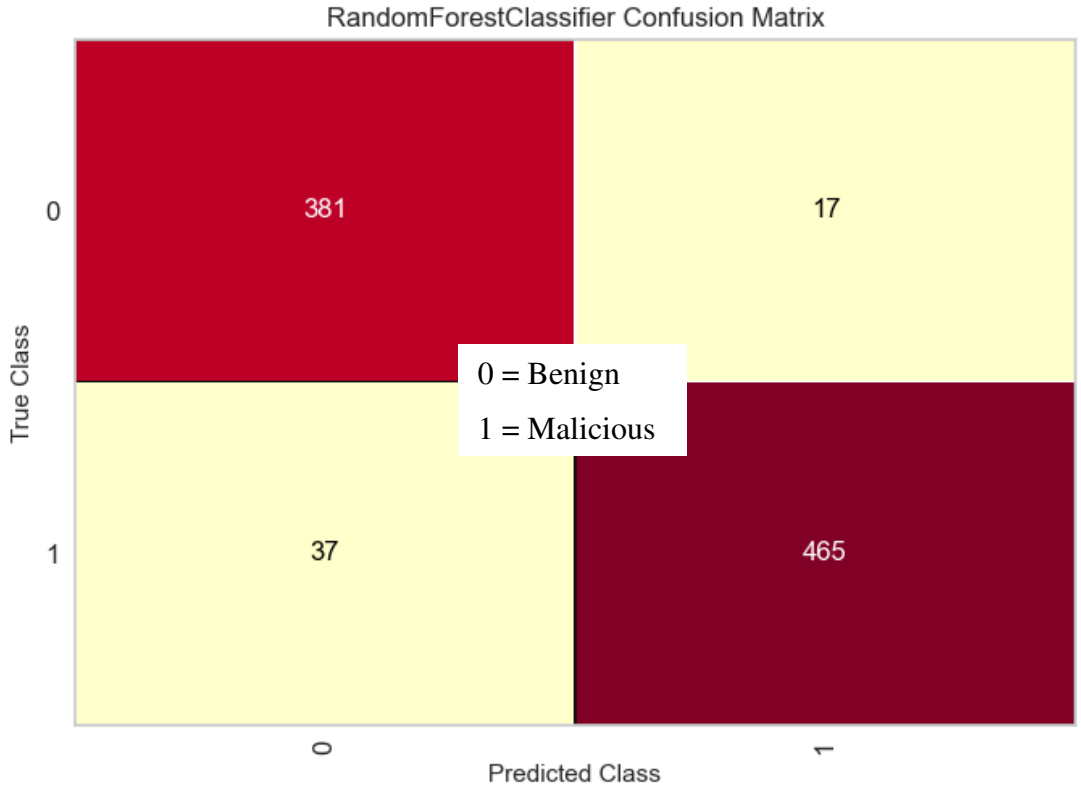

Figure 62 Random Forest Confusion Matrix

In terms of the confusion matrix table which represents the classification prediction of the 20% testing data against trained model with 80 % training data. Coherently, the accuracy of the model as classifier is satisfiable as to the predication outcome results as indicating high for TP and TN and low for the FP and FN . Overall, it is clear from the results that the Random Forest classifier model has adapted the patterns of the both malicious and benign of active and inactive processes behaviour and had proven to be best performing classifier model to be considered for the classification tool.

```
----  Classifying of 10 cases of testing data ----

[615806336, 76, 1, 1, 1, 1, 0, 1, 1, 0] => 1 (expected 1)
[2429392448, 76, 1, 1, 1, 1, 1, 1, 1, 0] => 0 (expected 0)
[3540825664, 76, 1, 1, 1, 1, 1, 1, 1, 0] => 0 (expected 0)
[1375114112, 46, 1, 1, 1, 1, 0, 1, 1, 1] => 1 (expected 1)
[3540825664, 76, 1, 1, 1, 1, 1, 1, 1, 0] => 0 (expected 0)
[4515844224, 80, 1, 1, 1, 1, 0, 1, 1, 0] => 0 (expected 0)
[4640994688, 58, 1, 1, 1, 1, 0, 1, 1, 1] => 1 (expected 1)
[22935104, 76, 1, 1, 1, 1, 1, 1, 1, 0] => 0 (expected 0)
[4296676928, 73, 1, 1, 1, 1, 0, 0, 0, 0] => 1 (expected 0)
[644199808, 76, 1, 1, 1, 1, 1, 1, 1, 1] => 1 (expected 1)
```

Figure 63 Classification outcome of testing data against trained model

Chapter 8: Future Work

This chapter defines some sustainable work and assumptions that can might addressed in further research for the project.

Some of the project remaining sub-cores aims of implementation, tests and experiments have been left for the future work due to limitations and time constraint of the project . In addition, this project requires further analysis and extraction of other artefacts data from memory image which is considerable and very time consuming .

Some of the sub aims and functional requirements that could developed as future work are listed below :

- Testing and improving the results of the Random forest classifier model with better understanding and validating the feature attributes as prioritizing features of pslist and psscan over the other features when classifying.

- Implementing the desired functional requirements mentioned in Design and Specification **Chapter 4**, of the classifier tool that could show more results to the initial outcomes of the project.

- It would be interesting for constructing an automate predicator using volatility plugins for importing memory images directly and looking up for the vital artefact that can retrieved easily and utilized to distinguish memory as benign or malicious .

- Improving the data collection and preprocessing tool to include options of preprocessing multiple process related artefacts and testing against the classifier.

- Considerably important for the future research to include other investigation and analysis areas of memory forensics including dlls, handles and threads.

Chapter 9 : Conclusion

This chapter of the document summarises the main findings and outcomes of the project's stated aims.

9.1 Summary

Overall, this project was aimed at investigating and utilizing a Machine Learning approach for memory forensics investigation in order to assist and automate investigation procedures using an artefact to reduce the ongoing backlog problem.

From the experimental point of view, the new approach based on ML is beneficial for the automation of memory forensic investigation procedures. One of the main findings of this project demonstrates that psxview is a useful artefact to discover malicious behavior of both active and inactive running process in memory image and it can be applied to ML in order to classify a memory image as either benign or malicious.

Unfortunately, there was not enough time for the implementation of the desired functional requirements of the classification tool that could show more results to the initial results. As some of desired outcomes of this project have not fully achieved due to the technical challenges of setting a forensic analysis environment which includes installation and version clashes of open source analysis tools of both Rekall and Volatility in VMware. In addition, time constraint and patience to conduct the analysis of memory image and obtain the pull-out the logs for the data collection. The project was accessible to limited resources for dataset storage of memory images which has impacted on data collection for machine learning models and with consideration of time limitation of the project the remaining sub-core aims considered as part of the future work and for further research. Despite the faced challenges and limitations during the project's fixed timeframe, the concept of the applying machine learning approach has been proven by the initial result and findings of this project, where the best performing Machine Learning Classification model is utilized to be modelled as classifier tool that can applied to automate in assisting and identifying a memory image as malicious or benign using a psxview log artefact prior dumping memory sections and malware detection that can reduce manual analysis of memory and extraction.

Chapter 10: Reflection

"You learn something valuable from all of the significant events and people, but you never touch your true potential until you challenge yourself to go beyond imposed limitations."
— Roy T. Bennett

Throughout this project, I have learned and gained more insights about memory forensics analysis and investigations as a learner in the field. In general ,I had an interest in digital forensics, I proposed the initial project concept to have a learning opportunity to explore new artefacts that I have not analysed before and adapt to new forensics analysis and investigation techniques. I believe that I started to understand more of the memory forensics to a standard but still need to learn more to understand the lower-level behaviors of memory artefacts as APIs and malfind. Also, through setting up the Virtual machine as forensics platform which was new to me as I had no prior experience but I have learned it and its uses on how it isolates analysis platform from the host machine as well as familiarised myself with the different versions of Linux operating system as it was challenging to get the correct version and resources for the analysis tool due to their compatibility to certain Linux environment. In addition ,discovered how VM manages the drag and drop of the files from the host machine as I discovered that it saves duplicates of each dragged files in separate hidden folder which utilized by the VM , which impacted the specified hardware storage for the data collection when acquiring the memory images. In terms of Machine Learning, I had very minimal understanding of the concept and through background research, I learned useful knowledge about different techniques and algorithms of Machine Learning as classifier and it was fascinating to learn and understand its benefits to resolve different problems including memory forensics investigation and malware detection . Another lesson I learned is managing the project with ongoing challenges to deliver achievable results despite of going through unprecedented times and experiencing new changes I kept myself determined to prove the concept even through the project limitations. Although it was challenging to combine two fields to fit into a single approach in a limited timeframe but utilizing the basic implementation approach to automate data collection and developing classifier was beneficial for this project as it can be utilized in future and for further research. Finally, documenting and writing the final document

was another challenge for me as I wrote my report close to deadline along faced challenges, I had to go back and refresh myself with uncompleted document draft sections and rewrite properly for the report. Overall, learning experience from undertaking this project, I have developed valuable skills in different aspects of analysis, problem-solving, utilizing new approaches and handling technical challenges as well as I have gained researching skills that are necessary to prove a concept and all these developed skills will beneficial for myself and for the future projects .

References

1. (2014) 'Memory Acquisition ', in Ligh, M.H., Case, A., Levy, J. and Walters, A., (ed.) The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Indianapolis, Indiana: John Wiley & Sons, pp. 69-114.
2. (2014) 'Process, Handles and Tokens ', in Ligh, M.H., Case, A., Levy, J. and Walters, A., (ed.) The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Indianapolis, Indiana: John Wiley & Sons, pp. 149-187.
3. (2014) 'Systems Overview ', in Ligh, M.H., Case, A., Levy, J. and Walters, A., (ed.) The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Indianapolis, Indiana: John Wiley & Sons, pp.1-26.
4. (2014) 'The Volatility ', in Ligh, M.H., Case, A., Levy, J. and Walters, A., (ed.) The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Indianapolis, Indiana: John Wiley & Sons, pp. 45-67.
5. Alpaydin, E. (2014), Introduction to Machine Learning . [Online]. Available at: http://dl.matlabyar.com/siavash/ML/Book/Ethem%20Alpaydin-Introduction%20to%20Machine%20Learning-The%20MIT%20Press%20(2014).pdf [Accessed: 07-03-2020].
6. Amari, K . (2009). Techniques and Tools for Recovering and Analyzing Data from Volatile Memory.[ONLINE]. Available at: https://www.sans.org/reading-room/whitepapers/forensics/paper/33049 [Accessed 3 February 2020].
7. Awad, M. and Khanna, R.(2015). Efficient learning machines: theories, concepts, and applications for engineers and system designers. [Online]. Available at: https://link.springer.com/content/pdf/10.1007%2F978-1-4302-5990-9.pdf [Accessed: 07-03-2020].
8. Bhatnagar, R., (2018). February. Machine Learning and Big Data processing: a technological perspective and review. In International Conference on Advanced Machine Learning Technologies and Applications (pp. 468-478). Springer, Cham. DOI: 10.1007/978-3-319-74690-6_46
9. Boast, K. and Harriss, L. (2016). Digital Forensics and Crime – POSTNOTE Number 520 March 2016 - UK Parliament. [online] Available at:

https://researchbriefings.parliament.uk/?ContentType=&Topic=Science+and+technology&SubTopic=Internet+and+cybercrime&Year=2016&SortByAscending=false [Accessed 30 Jan. 2020].

10. Canbek, G., Sagiroglu, S., Temizel, T.T. and Baykal, N.(2017). Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights. In 2017 International Conference on Computer Science and Engineering (UBMK).pp. 821-826. IEEE.
11. Case, A. and Richard III, G.G.(2017). Memory forensics: The path forward. Digital Investigation, 20, pp.23-33.
12. Ciortuz, L .(2017), MACHINE LEARNING .[PDF] .Available at https://profs.info.uaic.ro/~ciortuz/SLIDES/2017s/ml0.pdf [Accessed : 1 March 2020].
13. Computing Science and Mathematics (2014), ITNP2A Computer Security & Forensics, Forensics Practical Two : Volatility, University of Stirling, Stirling, Scotland
14. CS 241 Staff University of Illinois.(2014) 'Virtual Memory and Paging' [PDF]. CM3111 :Forensics Available at https://courses.engr.illinois.edu/cs241/sp2014/lecture/09-VirtualMemory_II_sol.pdf [Accessed : 1 March 2020].
15. Daley,M.(2019) 'ACPO Guidlines ' [PowerPoint presentation]. CM3111 :Forensics Available at https://learningcentral.cf.ac.uk/webapps/blackboard/execute/content/file?cmd=view&content_id=_5252938_1&course_id=_393280_1&framesetWrapped=true[Accessed : 1 March 2020].
16. DFRWS.(2006).DFRWS 2005 Forensics Challenge. [ONLINE] Available at: http://old.dfrws.org/2005/challenge/. [Accessed 3 February 2020].
17. Fichera, J. and Bolt, S. (2013). Volatile Data Analysis. Network Intrusion Analysis, pp.71-117.
18. Fortuna, A . (2017). Volatility, my own cheatsheet (Part 2): Processes and DLLs. . [ONLINE]. Available at: https://www.andreafortuna.org/2017/07/03/volatility-my-own-cheatsheet-part-2-processes-and-dlls/ [Accessed : 1 March 2020].

19. Goldberg, A. (2015). Child Abuse Cases Delayed by Police Backlog. BBC News. [online] Available at: http://www.bbc.co.uk/news/uk-34713745 [Accessed 29 Jan. 2020].

20. Iqbal, S. and Abed Alharbi, S. (2019). Advancing Automation in Digital Forensic Investigations Using Machine Learning Forensics. Digital Forensic Science [Working Title]. [online] Available at: https://www.intechopen.com/online-first/advancing-automation-in-digital-forensic-investigations-using-machine-learning-forensics [Accessed 29 Jan. 2020].

21. Kumara, M.A. and Jaidhar, C.D., 2017. Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digital Investigation, 23, pp.99-123.

22. Malin, C.H., Casey, E. and Aquilina, J.M., (2012). Malware forensics field guide for Windows Systems: Digital Forensics Field Guides. Waltham ,MA, Syngress

23. Messier, R,(2016). 'Memory Forensics' Operating system forensics. Waltham ,MA, Syngress,pp 95-127

24. Müller, A.C. and Guido, S.(2016). Introduction to machine learning with Python: a guide for data scientists. O'Reilly Media, Inc.

25. O. Olowoyo and P. A. Owolawi.(2019).Detection of Malware using Artificial Neural Networks," 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC),pp. 1-6. IEEE.

26. Osbourne, G.,(2013). Memory forensics: review of acquisition and analysis techniques (No. DSTO GD 0770). DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) CYBER AND ELECTRONIC WARFARE DIV.

27. Page.(2020).Wikipedia.[Online].Available at https://en.wikipedia.org/wiki/Page_(computer_memory) [Accessed : 07-03-2020].

28. Quick, D. and Choo, K. (2014). Impacts of increasing volume of digital forensic data: A survey and future research challenges. Digital Investigation, 11(4), pp.273-294.

29. Raghavan, S.(2013). Digital forensic research: current state of the art. CSI Transactions on ICT, 1(1), pp.91-114.

30. Rekall Forensic (2017). [Online].Available at: http://www.rekall-forensic.com/releases [Accessed : 07-03-2020].
31. Reust, J. and Friedburg, S.(2006). DFRWS 2005 Workshop Report. [ONLINE] Available at: http://www. dfrws. org/2005/download/2005final. pdf. [Accessed 3 February 2020].
32. Sadek, I., Chong, P., Rehman, S. U., Elovici, Y., & Binder, A. (2019). Memory snapshot dataset of a compromised host with malware using obfuscation evasion techniques. Data in brief, 26, 104437. https://doi.org/10.1016/j.dib.2019.104437 [Accessed 25-02-2020]
33. Samuel, A.L.(1959). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of research and development, 3(3), pp.210-229.
34. SANS Institute. (2020). Memory Forensics Cheat Sheet v2.0. [ONLINE]. Available at: https://digital-forensics.sans.org/media/volatility-memory-forensics-cheat-sheet.pdf [Accessed : 16-04 2020].
35. Saravanan, R. and Sujatha, P.(2018). A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification. 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS),pp. 945-949. IEEE.
36. Scikit-learn (2017). [Online]. Available at: http://scikit-learn.org/stable/index.html [Accessed: 27-04-2020].
37. Sun,X .(2020), Computer Vision with Deep Learning,.[PowerPoint presentation] .CM3202: Emerging Technologies .Available at https://learningcentral.cf.ac.uk/bbcswebdav/pid-5308222-dt-content-rid-14272389_2/xid-14272389_2 [Accessed : 07-04-2020].
38. Tkinter(2020). [Online].Available at: https://docs.python.org/3/library/tkinter.html [Accessed : 15-05-2020].
39. TURING, I.B.A., 1950. Computing machinery and intelligence-AM Turing. Mind, 59(236), p.433.
40. Volatility Framework with Windows 10 Memory Compression(2020). [Online].Available at: https://github.com/fireeye/win10_volatility [Accessed : 12-03-2020].

41. Volatility2.6(2020).[Online].Available at: https://github.com/volatilityfoundation/volatility/wiki [Accessed : 5-03-2020].
42. Windows File Information. (2005 [Online].Available at https://www.file.net/process/[Accessed : 28-05-2020].
43. Windows process and task list. (1993) [Online].Available at https://www.neuber.com/taskmanager/process/ [Accessed : 28-05-2020].
44. Barabosch, T., Bergmann, N., Dombeck, A. and Padilla, E.(2017). Quincy: Detecting host-based code injection attacks in memory dumps. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 209-229. Springer, Cham.