ANTI-MALWARE SOFTWARE

DELIVERABLE 1:

FINAL YEAR DISSERTATION

ARJUN RAJEEV NEDUNGADI

SUPERVISOR: DR. HANI RAGAB

B.Sc. (HONS) COMPUTER SYSTEMS

DECLARATION

I, Arjun Rajeev Nedungadi confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Arjun Rajeev Nedungadi

Date: 30/10/2017

ABSTRACT

The world is moving to a new-era. A phenomenon like no other brings itself to challenge the very core of humanity – technology. It is now absolutely ubiquitous. The rapid advancements that present themselves to humanity increases the number of *connected* people. Humungous amounts of data, information and knowledge is generated every single second and are stored in machines. A serious and imminent threat today is malware. Software built to cause damage exists and can present themselves at anytime and anywhere. The level of destruction is unfathomable. The best remedy is to always be prepared and ready for a guaranteed attack.

We encounter malware everywhere. From that random movie that was downloaded last week to that annoying advertisement on the local news website. Malicious programs await a chance to enter the machine with an aim to propagate to a massive audience. However, things are not as bad as they seem. There do exist various anti-malware software – packed with hardcore features and easy-to-use user interfaces. The truth is – we should all be scared. No anti-malware software can offer a 100% detection rate. Zero-day attacks are imminent. This is attributed to the cleverness of malware writers. These artists clearly know how to confuse anti-malware software and bypass strict detection mechanisms.

Traditional anti-malware software rely on signature-based techniques which are static, outdated and ineffective towards polymorphic malware. Thus, behaviour-based techniques surfaced to increase the optimism on anti-malware products. However, malware writers are quickly catching up. The only thing left to do is utilise existing technology and build strong defences. Currently, the world is in a cognitive era fuelled by artificial intelligence.

This project involves building an anti-malware system utilising the concepts of machine learning and data mining to increase malware detection and classification rates. The idea is to present a usable piece of software with magnanimous capabilities and potential for growth. The inspiration stems from attempts from other researchers to achieve the best malware detection and classification rates. Further, this project aims to critically compare and contradict the usability, performance and overall effectiveness of commercial anti-malware software and the proposed anti-malware software.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 5

1.1. CONTEXT 5
1.2. AIM 5
1.3. OBJECTIVES 5
1.4. DOCUMENT OVERVIEW 5
1.5. ABOUT MALWARE ANALYSIS 6
1.5.1. Malware 6
1.5.2. Malware Analysis 6

CHAPTER 2: LITERATURE REVIEW 7

2.1. ANTI-MALWARE & MACHINE LEARNING 7
2.1.1. Anti-Malware Software 7
2.1.2. Machine Learning 8
2.1.3. Classification 8
2.2. RELATED WORK 9
2.2.1. Malware Detection 9
2.2.2. Malware Classification 11
2.2.3. Anti-Malware Design 13
2.2.4. Critical Analysis 15

CHAPTER 3: REQUIREMENTS ANALYSIS 17

3.1. PROJECT REQUIREMENTS 17
3.1.1. Functional Requirements 17
3.1.2. Non-Functional Requirements 19
3.2. USE CASE DESCRIPTION 20

CHAPTER 4: DESIGN 22

4.1. ARCHITECTURE 22
4.2. TECHNICAL CHOICES 23
4.3. INITIAL DESIGN 24

CHAPTER 5: EVALUATION STRATEGY 25

CHAPTER 6: PROJECT MANAGEMENT 28

6.1. METHODOLOGY 28
6.2. PROJECT PLAN 29
6.3. RISK MANAGEMENT PLAN 30
6.3.1. Risk Identification & Analysis 30
6.3.2. Risk Planning & Monitoring 32
6.4. PROFESSIONAL, LEGAL, ETHICAL & SOCIAL ISSUES 34

REFERENCES 35

CHAPTER 1: INTRODUCTION

1.1. CONTEXT

Traditional malware detection involves using simple signature-based static analysis techniques to determine the maliciousness of a suspicious file. However, these approaches are heavily outdated. Sophisticated malware are polymorphic and are capable of camouflaging themselves from detectors. Thus, the current state of malware requires a cognitive solution. Applying the concepts of Machine Learning is an attempt to *teach* the detector to adapt to the ever-changing landscape of malware.

1.2. AIM

The main aim of this project is to design and develop an Anti-Malware system driven by a combination of traditional and heuristic styles of detection. The proposed Anti-Malware system will use a Machine Learning based engine to increase detection rates of malicious files.

1.3. OBJECTIVES

The objectives of this project are:

- To learn about existing malware detection techniques and anti-malware software.
- To incorporate a signature-based detection engine to the anti-malware system.
- To use an existing machine learning model to enable learning in the anti-malware system.
- To design and develop a robust, user-friendly and functional GUI application.
- To test the usability and accuracy of the proposed anti-malware software.
- To critically compare the proposed anti-malware software alongside existing anti-malware software.

1.4. DOCUMENT OVERVIEW

This paper introduces the holistic concept of malware analysis in subchapter 1.5.1. This is followed by a literature review in chapter 2 where anti-malware and relevant machine learning techniques are discussed. Further, subchapter 2.2 elaborates on the related work with a focus on the methodologies, results and a critical analysis for each research paper. Subchapter 2.2.4 summarises the criticism which serves as an inspiration for this project. Chapter 3 introduces the project requirements and elaborates on the *functional* and *non-functional* requirements for the development of the project. Subchapter 3.2 elaborates on a particular use-case mechanism accompanied by an *activity diagram*. In chapter 4, an initial architecture diagram and user-interface mock-up is presented to visualise the mechanisms of the anti-malware system. Chapter 5 outlines the evaluation strategy for the various modules and components of the project. The project methodology and project plan are expressed in chapter 6. Subchapter 6.3 elaborates on risks linked directly to the project requirements. Whereas, subchapter 6.4 introspects on any professional, legal, ethical and social issues associated with the project.

1.5. ABOUT MALWARE ANALYSIS

In this section, we will discuss the foundational concepts of malware and the ideas of malware analysis.

1.5.1. Malware

Malware simply refers to *malicious software* i.e., software built with an intention to cause harm to users, data, systems or networks. [1]

This broad statement brings about a plethora of questions. The fact is that *malware* is a broad term. Hence, to be more specific, a taxonomy of malware exists to truly distinguish the capabilities, damage and structure of a specific malicious program.

According to Kaspersky Lab, malware can be classified into viruses and worms, trojans, suspicious packers, and malicious tools. [2] However, there are a variety of different styles of classification. For example, F-Secure includes backdoors, exploits and rootkits as part of their malware classification tree. [3]

1.5.2. Malware Analysis

A malware is essentially a threat that must be captured, dissected and rigorously analysed to truly understand its sophistication. The process of dissecting malware and understanding how it works is called *malware analysis*. [1] This process is an art which encompasses a variety of techniques, tools and ideologies. A *malware analyst* must be knowledgeable about the latest breakthroughs, technologies and advancements in the world of malicious software. It is a race to the finish line; a war between the cracker and the malware analyst. Only time will reveal the winner.

As with any form of analysis, the main purpose of malware analysis is to garner more information about a malicious software. This involves determining what a malicious software can do, how to detect it, and how to measure and preclude its damage. [1] There are two fundamental approaches to malware analysis – static malware analysis and dynamic malware analysis.

From a holistic perspective, *static malware analysis* involves an examination and analysis of the malicious software without executing it. Static analysis is a powerful technique for reconnaissance. The plethora of tools and techniques contained in static analysis allows the analyst to uncover a vast array of information about the malware. In technical terms, the main idea is to reverse-engineer the malware. This can be done by loading the malicious program into a *disassembler*; which provides the analyst with a series of instructions encompassed in the program. All this is accomplished without executing the malicious software.

On the other hand, *dynamic malware analysis* involves running the malicious software and scrutinising its behaviour at runtime. Similar to the process of static analysis, the analyst is expected to observe any changes made to the system by the malicious software. These changes could include registry entries, file input/output, network access or installation of new programs amongst many others. The only caveat to dynamic analysis is the risk of infecting the system. Hence, it is the duty of the analyst to set up a safe, sandboxed and isolated environment before executing the malware. Some commonly used tools for dynamic analysis include: debuggers, process monitors, registry monitors, network analysers and network simulators. Dynamic analysis is typically highly rewarding and can reveal a plethora of information about the malware's payload.

CHAPTER 2: LITERATURE REVIEW

2.1. ANTI-MALWARE & MACHINE LEARNING

In this subchapter, we will discuss the basic ideas behind anti-malware software and introduce machine learning concepts with a focus on supervised learning.

2.1.1. Anti-Malware Software

An *anti-malware software* is essentially a tool used to protect the system against the evil effects of malware. There are various features, functionalities and techniques used by anti-malware software to protect the user, system and network from different threats.

The basic functionality of an anti-malware software is to identify a malicious program and stop it as soon as it enters the system. However, if a malicious program does bypass the security measures in-place, the anti-malware must strive to minimise further damage and remove it before it propagates to other systems on the network. [4]

However, the primary function of an anti-malware software is – malware detection. The ability of an anti-malware software to detect malicious software as soon as it enters the system is one of the most important metrics on which a solution is judged on. An anti-malware software typically employs two detection techniques – signature-based detection and behaviour-based detection.

Essentially, an anti-malware software attempts to detect a malicious program by using its signature as a reference. If the suspected program contains a byte sequence matching a known-threat's byte sequence (or signature), then the anti-malware software considers the program as a risk and quarantines it. In many cases, the anti-malware deploys a *heuristic* strategy of detection. This is a technique where the anti-malware inspects program logic to identify specific suspicious behaviours. [5] This process is accomplished by the *signature-based detection engine* of the anti-malware software. Ideally, an anti-malware software maintains a *signature database* (sometimes known as *virus database*) which contains a list of known malicious signatures. This database is frequently updated by the vendor and in many cases, by the community. However, signature-based detection is a double-edged sword. While techniques such as *heuristic signature detection* help unravel previously unidentified malware, they can generate false-positives. Similarly, signature-based detection is not resilient to evasion mechanisms such as – code obfuscation, packing and delayed decryption of the payload. [5]

Another technique employed by anti-malware software is *behaviour-based detection*. This is similar to dynamic malware analysis – the *detection engine* dynamically investigates the suspected program's execution behaviour. [1] This investigation ends in a classification as – malicious or benign. This is done by monitoring the status of the system, network communications and any other dynamic transposition. Behaviour-based detection offers the powerful primacy of detecting obfuscated malicious programs. The behaviours of such programs remain unchanged even after code obfuscation. [5] However, the caveat of detecting false-positives still exists in this technique. This attributes to the fact that legitimate benign programs might *behave* like malicious programs. Researchers are investigating behaviour-based approaches to accurately identify malicious behaviours while effectively reducing false positives. [5]

2.1.2. Machine Learning

The subject of machine learning evolved from the field of artificial intelligence with an aim to emulate human intelligence in machines. [6] The classic machine learning approach adapts the scientific paradigm of *induction* and *deduction*. The induction process involves learning from a specified collection of data (called as *training set*). This *learning* is applied to a test in the deduction process on another collection of data (called as *testing set*) with an aim to predict the behaviour of the new data. [7] There are three basic classifications of machine learning algorithms.

Supervised learning represents a learning scenario driven by the phenomenon of inductive inference. The process is supervised in the sense that samples are *taught* to the algorithm. This information is used as a reference point by the algorithm to produce a relevant output. In technical terms, the algorithm is *labelled* with correct answers to assist the process of prediction. [7]

On the other hand, *unsupervised learning* represents a scenario where we effectively rely on observations made by the machine learning algorithm. This process, by nature, is unsupervised in the sense that we allow the algorithm to *construct* the underlying patterns and processes. [6]

Lastly, *reinforcement learning* follows a disparate strategy where the algorithm aims towards reaching an end-goal and maximising the *reward*. The scenario involves an environment where the *agent* learns based on experience. This occurs through the technique of trial-and-error where several iterations push the agent towards its goal. [6]

Under these two varieties, there exist are vast array of algorithms utilising different approaches to solve different problems.

Classification is a supervised learning problem which uses the underlying concept of *pattern-recognition* to make relevant predictions. These predictions are discrete classes. The outcome of classification can either be binary (for example – a transaction is either *fraudulent* or *authorised*) or multi-class (for example – a tree can either be *yellow birch*, *pin cherry* or *chestnut oak*). The process involves a simple mapping derived from the training set of data consigned to the classification algorithm. The goal of classification is to accurately make a prediction for any unseen input. [6]

Regression is a supervised learning problem where the goal is to predict a real-valued output for a given pattern. If the outcome is a *continuous* value, this is known as *linear regression*. If the outcome is a *discrete*, this is known as *logistic regression*. A great example of regression is – trying to predict the cost of a house given its size in square feet. The outcome (cost) is a continuous real-value. [7]

Clustering is an unsupervised learning problem which deals with an attempt to find any *structure* within a collection of unlabelled data. The broad idea of clustering algorithms is to group objects together based on some form of similarity between them. These groups are also known as *clusters*. [8] Examples of clustering algorithms include – K-means clustering and Fuzzy C-means clustering.

2.1.3. Classification

Classification is one of the most important problems in machine learning. The core concept of pattern-recognition is a viable utility to solve a plethora of real-world problems. To discuss classification in detail, it is pertinent to understand that a pattern is solely described by its *features*. For example – in the scenario of face recognition, a possible feature is the distance between the eyes. [6]

A typical classification problem involves three processes. [6]

Collection of data involves acquiring data *relevant* to the scenario at hand.

Feature selection may subsequently involve feature reduction, where the aim is to reduce the *dimensionality* of the features. The goal is capture *invariant* features to truly differentiate the individuality of each class.

Lastly, the *classification* process attempts to perform the actual task of mapping labels with patterns to produce an output. The outcome of classification can be *binary* (binary classification) or *arbitrary* (multi-class classification).

Let us outline three traditional classification algorithms –

k-Nearest Neighbour Classification is an intuitive technique where *k* points of the training data which are nearest to the testing points are found. A majority vote between the *k* points determines the label to be given to the testing point. [6]

Decision Trees are a technique which involves constant partitioning of the input space to build a *tree*. The process of classification involves traversing the tree from the *root node* until a *terminal node* is reached. [6]

Linear Discriminant Analysis is a technique which computes a *hyperplane* in the input space with an aim to minimise the variance within a class and maximise the distance between classes. [6]

There exist a variety of other intuitive classification algorithms such as – Neural Networks, Support Vector Machines (SVMs) and Naive Bayes. These algorithms work well in different domains and scenarios.

2.2. RELATED WORK

In this subchapter, we will discuss the research done towards the fields of malware detection and malware classification utilising the core concept of machine learning.

2.2.1. Malware Detection

U. Baldangombo et al. [9] proposed a static malware detection system which utilises the concepts of data mining and machine learning to detect malicious software. Their aim was to accurately detect *new* (or unseen) malware binaries. The dataset used consisted of 247348 Windows PE binary files of which 236756 were malicious and 10592 were benign. The malicious PE files considered were non-packed. Packing is a technique which involves compressing the code with an aim to obfuscate it from anti-malware software. API function calls from these DLLs were amalgamated and grouped based on malicious behaviours. The *flagged* behaviours include: file-related, network-related, memory-related, registry-related, windows service and other behaviours. A feature vector, with the PE header, DLL and API function calls, was constructed for each program. The authors utilised these a hybrid feature set (HFS), which combined the three different types of features into one set. These features were used to train three classifiers – Naive Bayes, Support Vector Machines and J48 decision tree.

The results show that the J48 decision tree classifier offered the highest detection accuracy. Whereas, in most cases, the Naive Bayes classifier gave the worst detection accuracy. The classifiers which used the DLL name feature yielded the lowest detection rates while the classifiers which used a combination of PE header and API function features gave slightly better detection accuracy than classifiers which solely relied on PE header features. In fact, the best detection rate was 99.6% which was offered by the J48 classifier which utilised hybrid features consisting of PE header and API functions.

Critical Analysis

An interesting caveat in this approach is the lack of support for packed PE files. In real-life, malicious programs are bound to be packed with atypical packers. Detecting and decompressing packers requires a different heuristic approach. Unpacking tools used by authors – such as UPX and ASPack work on known packers. Further, the approach used here is static is nature. The features extracted – PE headers, DLLs and API function calls, are still skewed towards the concept of *static malware analysis*. The only shortcoming in their approach is that malicious programs can employ *obfuscation* and *polymorphism* techniques such as code obfuscation to easily bypass the anti-malware detection system. Packing is a lighter form of code obfuscation.

Meanwhile, ***M. Siddiqui et al.*** [10] proposed a novel idea of using critical program instruction sequences to make a binary classification between malicious and benign programs based on the concept of data mining. The problem was formulated as a binary classification problem and built models using logistic regression, neural networks and decision trees. In their approach, all features were instruction sequences which were aptly extracted by the disassembly. The dataset used consisted of 820 Windows PE files with an equal number of malicious and benign programs. The malicious programs contained in the dataset were either Win32 viruses, worms or trojans. The binaries underwent transformation to a disassembly representation from which the instruction sequences were extracted as the primary feature. In the process of feature extraction, 62608 unique instruction sequences were identified in which sequences with less than 10% of frequency of occurrence were discarded. Thus, 98.2% of the dataset vanished and only 1134 instruction sequences were selected as features. The dataset was further split into 70% training and 30% testing data. Three models – a logistic regression model, a neural network model and a decision tree model were used for classification as malicious or benign. In their approach, the logistic regression model was used for the calculation of the probabilities that a given instruction sequence belonged to a malicious or clean class. The neural network model was taught to classify between malicious and benign classes by repeatedly being presented instruction sequences from both classes. The results of this approach showed that the decision tree model yielded the best detection rate of 98.4% with a false alarm rate of only 4.9%. The overall accuracy of this model amounted to 96.7%. Whereas, the *neural network* yielded a detection rate of 97.6% with a false alarm rate of 4.1%. The overall accuracy for this model amounted to 97.6%. The logistic regression model yielded the smallest detection rate of 95% with a staggering false alarm rate of 17.5%. The overall accuracy for this model amounted to only 88.6%.

Critical Analysis

Like the approach in [9], the authors here used a static methodology for classification of binaries. The approach of using instruction sequences from the disassembly of malicious and clean programs as the primary classification feature is simpler and more straightforward. However, a caveat lies in the fact that the dataset used did not account for new breeds of malware with mutation capabilities. For example – polymorphic viruses. A static approach is not sufficient to accurately classify such types of malware. In fact, a combination of static and dynamic approaches is deemed necessary. While this detection mechanism does not work with packed malware, the authors are optimistic about utilising dynamic techniques to first allow the malicious program to decrypt itself and then obtain its disassembly representation containing the instruction sequences for the purpose of feature extraction. The authors conclude that using a decision tree model works best for classification and the next step is to use a random forest approach.

2.2.2. Malware Classification

J. Z. Kolter et al. [11] described the use of machine learning and data mining to acutely enable detection and classification of malicious executables. Unlike the work of M. Siddique et al. [10] where the authors disregarded the use of the *n-grams* technique to extract a sequence of instructions, the approach here evangelises this technique to capture n-grams of byte codes as features. The work here builds on previous work done by the same authors [12] where they built an anti-malware prototype using machine learning and data mining techniques. The system was called MECS (Malicious Executable Classification System), capable of detecting unknown malicious programs in the wild. Their dataset consisted of 1971 benign executables and 1651 malicious executables which consolidated a variety of attack vectors and payloads. The learning methods used included: Instance-based Learner, Naive Bayes, Support Vector Machines (SVMs) and a decision tree. The concept of *boosting* was used to build a stronger model by combining average-performance models together using a specific cost function.
The authors initially conducted an experiment with a small collection of samples. This collection produced 68744909 distinct n-grams. Further, ten-fold cross validation was used for evaluation. The top 500 n-grams were selected and various classification methods were applied. The results of this experiment showed that the boosted methods, the instance-based learner (IB*k*) and the Support Vector Machine (SVM) performed well. Whereas, the Naive Bayes classified did not perform well. With a larger collection of samples, containing 1971 benign executables and 1651 malicious executables, the same procedure described above was repeated. The top 500 n-grams were selected out of over 255 million distinct n-grams. In this experiment, the boosted J48 decision tree classifier outperformed all the other methods. IB*k* and SVMs performed with comparable results. The authors also performed *multi-class classification* after detection. To achieve this, they used the concept of classifying executables based on payload functions. For the purpose of experimentation, only three functional families of malware were considered – mass-mailers, backdoors and executable viruses. The experimental methodology remained the same as above. The boosted J48 decision tree classifier proved to be the most accurate in binary classification applied to detect whether an executable is malicious or benign. The classifier achieved a true-positive rate of 0.98 and a false-positive rate of 0.05.

Critical Analysis

The approaches mentioned above appear to be thorough and well-designed. The authors have achieved sound results in the arena of malware detection especially for obfuscated executables. They propose that a *boosted J48 decision tree* classifier is the best classifier for detection of malicious executables. However, a caveat in their approach was the limited performance in multi-class classification of malware. They do not attribute this to their methodology, but believe that it is because of fewer training samples and obtaining properly labelled data. In fact, their approach was limited to only three functional types of malware – mass-mailers, backdoors and viruses. To achieve appropriate classification results, it is pertinent to ensure sufficient training samples. This stems from the fact that a mass-mailer might utilise function calls and DLLs similar to a legitimate e-mail client. Further, we can consider using more than three classes unlike the authors who only assumed mass-mailers, backdoors and executable viruses.

K. Rieck et al. [13] exploited the *behavioural patterns* of malware families which are typically shared amongst the variants of each class to classify malicious binaries. The behavioural patterns are used to reflect the origin and intent of the malicious program.

The authors collected a corpus of 10072 unique malware binaries from the wild using honeypots and spam traps. They applied the Avira AntiVir anti-virus engine to allow identification of *known* malware instances. The anti-virus engine yielded 14 malware families based on the most common labels assigned by the engine. The malware binaries were executed on *CWSandbox* - a safe sandboxed environment under a Windows operating system. A behaviour-report was generated for each binary consisting of changes to the file system, registry, processes, mutexes, network and services. Features which reflect common malicious behavioural patterns were extracted from the report and was used to create a high-dimensional vector space. The authors made use of *Support Vector Machines* (SVM) for the purpose of multi-class classification. Lastly, the discriminative model for each malware family was analysed to reveal any insights into the classification model and relationships between other malware families.
To evaluate, the malware corpus consisting of 10072 samples was randomly split into three sets – training, validation and testing sets. On an average, 88% of the binary samples in the testing set were correctly assigned to the right malware families. Further, the initial experiment was extended to test the prediction of malware families. On average, 69% of the malware behaviour were correctly classified. Malware belonging to 9 additional malware families were used to experiment the agility of their methodology. They were found to be correctly identified as *unknowns*. Although, when this mechanism was implemented, the overall accuracy dropped from 88% to 76% due to a certain amount of *confusion* caused within the classifier.

Critical Analysis

While the authors pursued an important aspect of malware detection involving multi-class classification, they themselves identified certain limitations to their approach. The technique used within the sandboxed environment assumes and observes only one single execution path of the malicious binary. However, sophisticated malware bypass these analysis techniques. Hence, an appropriate mechanism is the *multi-path execution* which provides a holistic view on the malicious sample and its behaviour. Similarly, sophisticated malware often detect the presence of a sandboxed environment and acts as a benign program. Techniques such as delayed code obfuscation and *mimicry* of other malware families is of concern to us and must be appropriate handled. While the proposed methodology correctly identifies a malicious sample belonging to an unknown malware family using a rejection mechanism, it is not enough to deal with the dynamic nature and advances in the world of malware.

R. Islam et al. [14] presented a novel approach to detect and classify malware using the concepts of *pattern-recognition* and other statistical methods at various levels of the malicious analysis process. In their approach, the feature extraction process involved selecting a combination of both – *function length frequency* (FLF) and *printable string information* (PSI) as features. In other methods, for example in [9], the approach involved extracting a range of static features from each malicious sample. In FLF, the length, or number of bytes in a function is extracted. With this information, the authors determined the *frequency* with which the function lengths occur in other malware samples. PSI, on the other hand captures the strings inside each unpacked malware sample. These strings contain a lot of valuable information about the malware from a static point-of-view. In fact, certain strings could indicate API function calls and imported DLL names. The authors collected all such *strings* to form a global database of strings. For each malware sample, a vector is created in which values are true (1) or false (0) corresponding to whether the respective string in the global database was present in the malware's PSI or not. Finally, the authors combine both the FLF and PSI vectors to form a single entity representing the features for a particular malware sample. The dataset consisted of 1521 malware samples with 13 malware families. The authors proceeded to build 5 sets

of 80% training and 20% testing data. The authors then performed k-fold cross validation on the malware along with 151 benign files. The classification mechanisms used were: Naive Bayes, SVM, Random Forest, Decision Table and IB1 (Instance-based Learner). In addition to these, a meta-classifier and booster called AdaboostM1 was added to each classifier.
The results show that the Decision Table base classifier gave the best accuracy with a result of 98.15%. The other base classifiers gave equally comparable results. However, with the meta-classifier enabled alongside the base classifiers, the best accuracy was once again given by Decision Table with a result of 98.86%. The authors conclude that using a combination of both strings and function lengths in the process of static feature selection forms the basis of effective malware classification.

Critical Analysis

The approach taken by the authors simply utilises the concept of static malware analysis. Identifying the function length frequencies and printable string information for each malware sample is a straightforward task. While the results are commendable (a classification accuracy of 98%), there are some obvious caveats in the methodology. The authors did not consider *packed* malware samples. Their defence states that unpacking packed malware is a complicated and slow task which requires different techniques as compared to classification. However, it is pertinent to note that the concept of acquiring printable string information will typically work after the malware has been unpacked. Strings inside a packed malware are jumbled and encrypted. Similarly, the concept of function length frequencies and identifying patterns is only suitable for unpacked malware. Sophisticated malware are typically packed with unique packers which makes it difficult to perform any sort of static analysis on them. Polymorphic malware can exhibit multiple behavioural patterns. This is bound to *confuse* the classifier and potentially correlate a sample to an incorrect malware family. However, the concept of using a meta-classifier is an interesting approach. The authors' decision to use k-fold cross validation is also commended.

2.2.3. Anti-Malware Design

J. A. Morales et al. [15] presented an approach to evaluate the effectiveness of commercial anti-malware software. In their approach, they equally considered both – detection and treatment of malicious programs by the method of partitioning the *true positives* into *automatic treatment*, *user-option treatment* and *no treatment* to assist detection and treatment. According to their evaluation methodology, a commercial anti-malware program which maximises *automatic treatment* and minimises *false negatives*, *user-option treatment* and *no treatment* will achieve the highest level of effectiveness with regards to the detection and treatment mechanisms.
The authors used this new form of measurements to conduct three tests with the purpose of evaluating four commercial anti-malware programs. The four chosen commercial anti-malware products included – Kaspersky Internet Security 2010, ESET Smart Security V4.2, ZoneAlarm Extreme Security V9.1 and BitDefender Total Security 2010.
The authors conducted four tests. *Test 1* involved calculating the effectiveness of the commercial anti-malware program to detect and treat a folder containing a set of *known* malware samples. *Test 2* involved calculating the effectiveness of the commercial anti-malware program to detect and treat a system with an infected state and an unknown number of malware samples. *Test 3* involved calculating the effectiveness of the commercial anti-malware program's ability to be installed successfully and then detect and treat a system with an infect state and an unknown number of malware samples.
According to the results, all the four commercial anti-malware programs performed the best in *test 2*, followed by *test 1* and then *test 3*. Kaspersky Internet Security 2010 yielded an overall *automatic treatment*

rate of 99.3%. This shows that Kaspersky performs the best detection and treatment mechanisms as compared to the other commercial anti-malware programs. Meanwhile, all the commercial anti-malware programs displayed *false negatives*, *user option treatment* and *no treatment*. Further, it is seen that all the commercial anti-malware programs detected and treated malware samples in an inconsistent manner. This is because of the environment and scenario under which they operate. This is particularly visible in *test 3* where the commercial anti-malware program was installed post-infection of the system. The authors believe that the malware sample can *impair* the effects of the anti-malware program.

Critical Analysis

One caveat regarding the approach here is the fact that benign files were not included in the detection process. The entire sample set given to the commercial anti-malware programs consisted of malicious objects. As addressed by the authors, another limitation exists in the methodology itself. All the tests were carried out in a safe, isolated and virtual environment using a virtual machine. However, sophisticated malware can detect the presence of a virtual environment and can behave differently (or as a benign program). Their approach is an interesting take to understand the effectiveness of a commercial anti-malware program from a holistic perspective.

O. Sukwong et al. [5] conducted an empirical study on the effectiveness of commercial antivirus software. Over a period of five months, the study automatically examined 1115 unique malicious samples. The dataset of malicious samples included Windows-compatible files with extensions – EXE, SYS, DLL, DOC, XLS and PDF. 80% of the samples included EXE files while only 10% were PDF files. All benign files were removed from the dataset. The malicious files were of different types such as worms, viruses, trojans, rootkits and spyware. The authors installed six commercial offerings from Avast, Kaspersky, McAfee, Norton, Symantec and Trend Micro. The methodology in this study involved subjecting each of the acquired malware samples to each antivirus offering. If a malicious sample bypasses the signature-based detection engine, then the malware is executed for three to five minutes. This is done to examine the *behaviour-based* claims made by the commercial offerings. The authors also provided unrestricted Internet access to ensure that sophisticated polymorphic malware do not detect the experimental environment and block their original behaviours. The results show that the chance of the malicious program to cause damage increases as the time of detection increases. Avast, with no behaviour-based detection capabilities detected and eliminated approximately the same percentage of malware as Norton, Kaspersky and McAfee – which utilise behaviour-based detection engines. A study was conducted to determine how the antivirus offerings protect the machine when a malware bypasses the signature-based detection engine. It was seen that the antivirus offerings with behaviour-based capabilities, ensured a *block* on certain system-wide changes such as modification of DLL and EXE files. It was found that McAfee actively blocked 11 programs from adding themselves to run at system start-up. Further, it was noticed that Avast and Norton sometimes did not initiate a quarantine on bypassed *root* programs which enabled further infection of the system. Symantec features remediation of the Windows registry if a malicious program alters the network and system keys. This characteristic was also found to exist in other offerings with behaviour-based capabilities. On the other hand, none of the antivirus offerings completely blocked code injection by malware. McAfee, Norton and Trend Micro offer browser plugins to protect against browser based malicious attacks and websites. Similarly, all offerings are compatible with e-mail clients to filer malicious emails such as spam. However, Avast, McAfee and Symantec severely lack protection against automatic malicious downloading behaviours. 13.18% of the malware samples with downloading capabilities went undetected. The authors conclude that antivirus offerings cannot effectively detect all current forms of malware despite behaviour-based detection.

Critical Analysis

The approach here completely relied on malicious program samples. While the aim of the research was to determine the effectiveness of commercial antivirus offerings, the authors could have parallelly studied the detection and treatment mechanisms (if any) for benign software. In real-life, users are prone to scan benign files with antivirus products. Further, the entire study of scanning took place in an isolated virtual environment. Sophisticated malware are capable of detecting such environments and can behave as benign programs. However, in praise of the methodology, it was an interesting decision to enable completely unrestricted Internet access to the antivirus software. The defence towards this approach is the fact that certain new antivirus offerings do not maintain a static or on-premise virus database. In fact, many offerings rely on cloud-based approaches to *pull* the latest signatures and malware definitions. Also, this would reduce the probability that the malware sample might detect a virtual environment. The authors also attempted to execute the malware sample to initiate an understanding on the changes to the registry. The focus was limited to the network and system keys. However, this is insufficient. Other keys within the registry such as Microsoft Word's key which controls macro execution and Excel's key which controls add-ins are potential vulnerabilities which can be exploited by the malware.

2.2.4. Critical Analysis

In conclusion, we have seen researchers attempt to detect malware using machine learning and data mining concepts succeed with classification algorithms such as the *J48 decision tree* model. Other models such as SVMs, Naive Bayes, neural networks and logistic regression produced comparable results. [9] [10] [11]

Further, we have discussed an attempt to perform multi-class classification to identify and classify malware samples based on their families. The research shows that detection is an important mechanism which needs to be performed accurately before classification. Further, it was seen that classification requires a malware corpus consisting of diverse samples. The ideology behind this is to account for the plethora of labels available for each malware family. Studies show that classification is a necessity which comes after detection. Hence, it is pertinent for an anti-malware system to perform accurate detection to enable sound classification of malware based on their family. This was attempted based on the API function calls present in DLLs inside the malicious programs. However, in criticism, the methodology focused on a single path of execution and used a rejection mechanism to enable identification of malware classes. The methodology must also consider sophisticated malware which contain multiple paths of execution and other complex behaviours. [11] [13]

An interesting approach was mentioned in [14] where the authors attempted to detect and classify malware based on a combination of static features. The resultant vector consisted of 2 features – function length and strings for each malware sample. Interestingly, the accuracy of this approach was 98.86%. However, an important caveat was that fact that they employed only unpacked malware samples. Static analysis works best with unpacked malware. Otherwise, the system is required to first unpack the malware and then subsequently perform detection (by static techniques) and classification. The use of k-fold cross validation appeared to be a successful technique for the used dataset.

We also discussed the relevance of commercial anti-malware programs available with respect to their detection and treatment mechanisms. The research shows that an anti-malware program should be equally judged based on its ability to treat and detect malware samples. A variety of tests done indicate that success is highly dependent on the *environment* and *scenario*. Further, it was seen that a commercial anti-malware program is highly effective when it has a high *automatic treatment* rate and low *false negatives*, *user-option treatment* and *no treatment* rates. However, the research was only focused on detection and treatment of *malicious* programs. Users frequently scan legitimately benign files and documents. The research done does not account for the behaviour exhibited in such a scenario. Further, the behaviour-based detection mechanisms in commercial anti-malware offerings appear to be inadequate in detecting sophisticated dynamic malware. While the registry is an important entity of the operating system, only limited keys within it were scanned and remediated if infected. [15] [5]

The research done here serves as an inspiration to move forward with this project. We can confirm that using machine learning and data mining techniques is the right approach to detect and classify previously unseen malware. The problem of malware detection is a supervised learning problem which can be solved using the technique of classification.

CHAPTER 3: REQUIREMENTS ANALYSIS

The Project Requirements subchapter is of vital importance to the development and planning of the Honours Project.

To develop a robust, usable and effective Anti-Malware software, it is pertinent to focus on truly understanding the requirements of the system. By explicitly recording these requirements, we can proceed with better clarity.

The requirements discussed below are linked to the aims and objectives as detailed in subchapters 1.1 and 1.2.

3.1. PROJECT REQUIREMENTS

Requirements can essentially be classified as:

1. Functional Requirements (FRs)
2. Non-Functional Requirements (NFRs)

The Functional Requirements (FRs) describe the core functionalities, offerings and behaviours of the system. A priority describes the importance of these functionalities and a description offers a better understanding of the requirement.

The Non-Functional Requirements (NFRs) describes *how* the system should achieve its goals. These requirements focus on the operation of the system and are further grouped based on performance, accessibility, availability, usability, etc.

We will describe the FRs and NFRs for the system:

3.1.1. Functional Requirements

FR 1. Anti-Malware System

ID	FR 1.1
REQUIREMENT	The system shall scan all types of Portable Executable (PE) files
PRIORITY	Must Have
DESCRIPTION	The system will take a PE (EXE or DLL) file as input and classify it as either malicious or benign.

ID	FR 1.2
REQUIREMENT	The system shall use an existing signature-based malware detection engine to scan PE files
PRIORITY	Must Have
DESCRIPTION	The system will feature a traditional signature-based engine to attempt detection of malicious PE files.

ID	FR 1.3
REQUIREMENT	The system shall use an engine which will accept external Machine Learning models/methods to classify PE files
PRIORITY	Must Have
DESCRIPTION	The system will pursue 2 levels of classification: (a) Classifying a PE file as malicious or benign (b) Classifying a malicious PE files into a category (Trojan, Adware, Virus, Ransomware, Worm, etc.)

ID	FR 1.4
REQUIREMENT	The system shall notify the user of any detected malicious activity
PRIORITY	Must Have
DESCRIPTION	The system will alert the user of the detection of any malicious activity and (optional) quarantine/remove the malicious PE file from the machine.

FR 2. User

ID	FR 2.1
REQUIREMENT	The user shall provide a PE file to be scanned with the system
PRIORITY	Must Have
DESCRIPTION	The user can select a PE file (by drag-and-drop, file dialog or hash) to the system.

ID	FR 2.2
REQUIREMENT	The user shall create, read, update and delete contents of the Virus Database
PRIORITY	Must Have
DESCRIPTION	The user can perform CRUD (Create-Read-Update-Delete) operations on the Virus Database which contains known malware signatures. (a) Create – user can add a new signature (b) Read – user can view an existing signature, date added & other data (c) Update – user can update data related to an existing signature (d) Delete – user can remove an existing signature from the database

ID	FR 2.3
REQUIREMENT	The user shall take further action on a detected malicious PE file
PRIORITY	Should Have
DESCRIPTION	The user can choose to either: (a) Delete – permanently remove from system (b) Ignore – do nothing (c) Quarantine – isolate infected file into a safe environment

3.1.2. Non-Functional Requirements

NFR 1. Hardware

ID	NFR 1.1
REQUIREMENT	Environment
PRIORITY	Must Have
DESCRIPTION	The system shall be deployed on modern machines equipped with the Windows Operating System.

NFR 2. Usability

ID	NFR 2.1
REQUIREMENT	User-Interface
PRIORITY	Should Have
DESCRIPTION	The system shall be presented with a User-Interface, which is: (a) User-friendly – simple & easy to use GUI (b) Responsive – user actions given right responses (c) Consistent – similar design-patterns to improve memorability of system (d) Intuitive – detect wrong inputs & other incorrect user actions (e) Familiar – known design-patterns to ensure quick request/response

NFR 3. Extensibility

ID	NFR 3.1
REQUIREMENT	Software Extensibility
PRIORITY	Could Have
DESCRIPTION	The system shall be open to extensions in the form of new features, bug-fixes and performance improvements. This is achieved by: (a) Writing modular code following good programming practices (b) Creating proper documentation for architecture/code (c) Publishing code on a Source Code Repository (such as GitHub)

3.2. USE CASE DESCRIPTION

To better understand the Functional Requirements (FRs), we will describe a use case using a Textual Description. The flow of the use case can be visualised using an Activity Diagram (part of UML family).

This is the Textual Description of **Scan PE File** use case:

Use Case: Scan PE File

ID: 1

Brief Description: Anti-Malware system classifies a PE file provided by the user

Primary Actors: User

Pre-Conditions:

1. The Anti-Malware system must be up and running
2. File provided by user is a valid PE (EXE or DLL) file

Main Success Scenario:

1. User inputs/selects a PE file
2. Anti-Malware system loads PE file into context
3. Anti-Malware system runs PE file against existing Virus Database
4. PE file is classified as malicious or benign
5. Malicious PE file is categorised as Trojan, Adware, Worm, Ransomware, Virus, etc.
6. Anti-Malware system reports results to user

Alternate Flows:

3.a. User provides appropriate model/method to system
3.b. Anti-Malware system runs PE file on a Machine Learning engine with given model

Exceptions:

4.a. Anti-Malware is unable to classify PE file
4.b. PE file is quarantined for further action by user
4.c. The use case ends

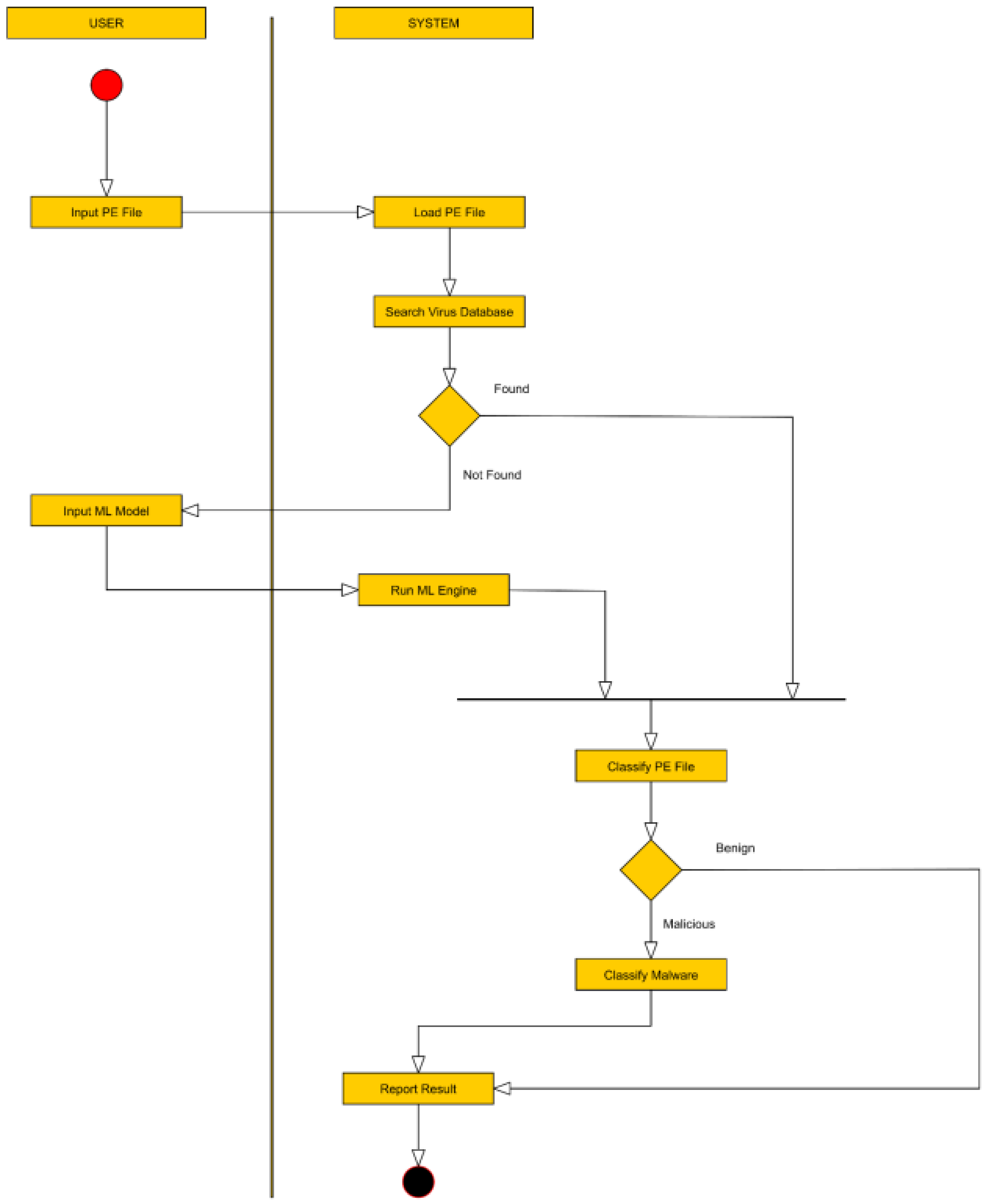


Figure 1. Activity diagram for Scan PE File use case

Figure 1 illustrates the Activity Diagram for the **Scan PE File** use case specifying the start state, activities and end state of the use case.

CHAPTER 4: DESIGN

The design of the Anti-Malware system aims to be simple, intuitive and rich in functionality.

The idea is to develop a software which works effectively while maintaining a robust User-Experience.

During the early stages of the project – mock-ups, sketches and wireframes were created to visualise the system. These activities help understand the depth of the problem at hand and shifts perspective to the bigger picture.

4.1. ARCHITECTURE

The Anti-Malware system consists of the following modules:

- Signature-based engine which scans against a Virus Database (existing)
- ML-based engine which uses a provided model for classification

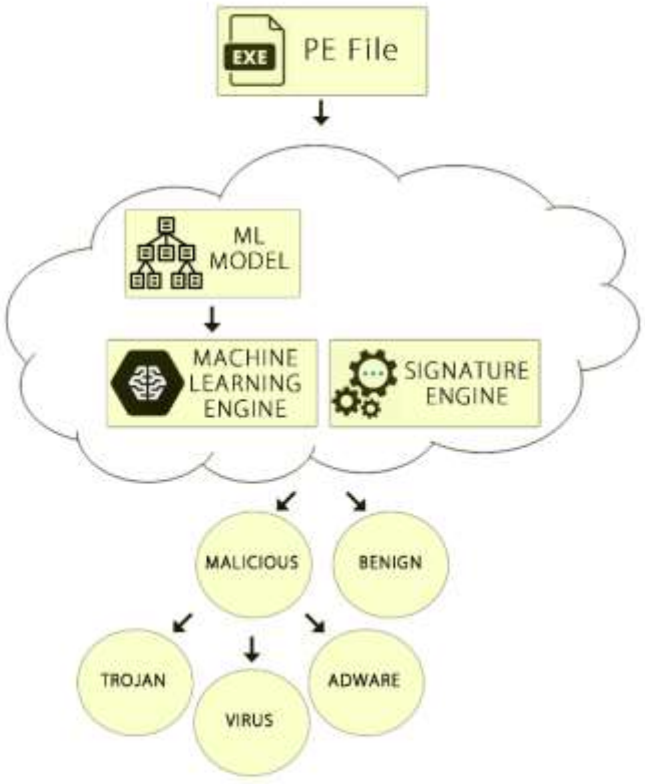


Figure 2. Architecture diagram of the proposed anti-malware system

Figure 2 illustrates the high-level architecture diagram of the proposed anti-malware system showing its various components and possible outputs.

4.2. TECHNICAL CHOICES

An Anti-Malware system can be considered as a module consisting of multiple micromodules.

This kind of architecture gives a programmer appropriate freedom in selecting the right stack of technologies to produce a functional, robust and scalable solution.

The main logic of the software will be written in **Python 2.7**.

Python is a modern language which is fast and provides access to a plethora of powerful libraries. These libraries act as an interface to functions written in traditional languages such as C and C++.

Due to its vast array of libraries and packages, we can utilise the following packages for the development of the system:

- pandas
- scikit-learn
- numpy
- scipy
- graphviz
- matplotlib

Further, Python offers a simple syntax with magnanimous capabilities. Being an interpreted language, it is also portable between different operating systems. As a scripting language, Python offers the programmer a chance to quickly test out implementations with its various shorthands (such as *lambda expressions*) and other programming tricks.

The Graphical User Interface (GUI) component of the software will be written in **Visual C#**.

Since the Anti-Malware system is proposed to run on a Windows machine, it is recommended to choose the .NET framework. Visual C# provides access to a WinForms GUI application which allows the programmer to create responsive, user-friendly and visually appealing User-Interfaces.

A similar option in Python is by using tkinter – Python's standard GUI package.

The idea is to allow modularity to flourish. This is a good programming practice.

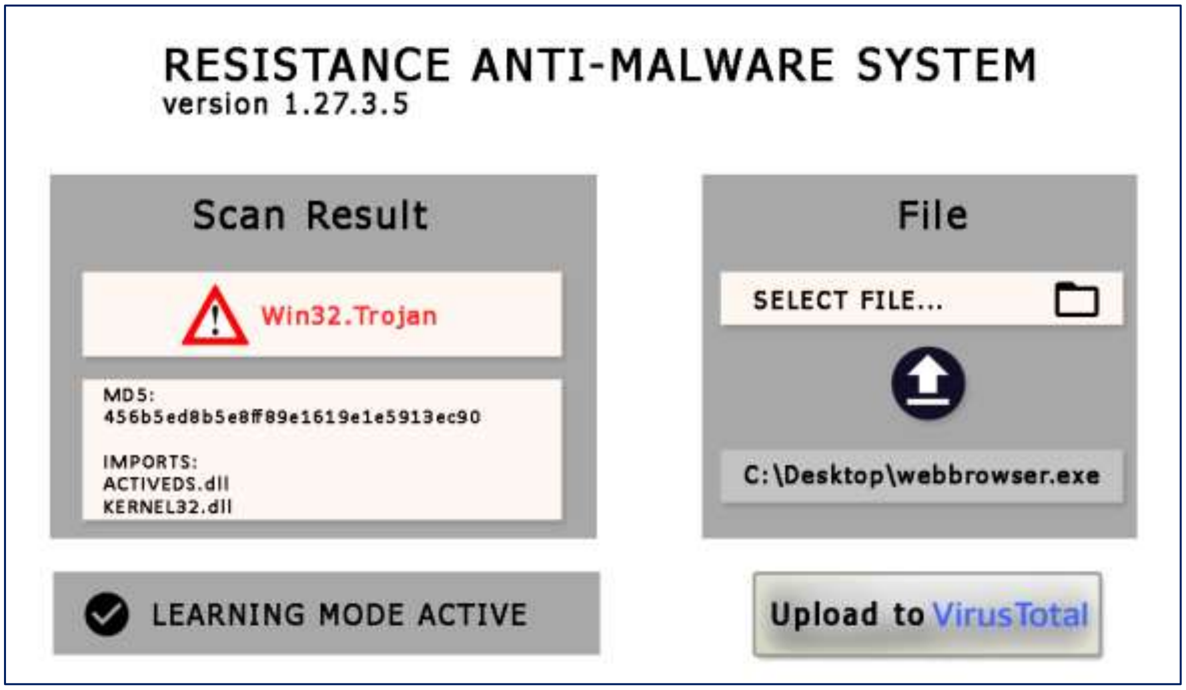
Keeping the software loosely coupled and highly cohesive makes extensibility, maintainability and scalability much easier.

For deployment, Visual C# provides an easy functionality to *package* the project into an installable executable. This is a packaged version of the application with all its resource and dependencies. This increase portability and allows the dissemination of the software to the community.

A GitHub repository will be initialised to maintain version control for the project. Upon the completion of the project, the repository will be made open to the community as open-source software (OSS).

The idea is to allow the concept of **open collaboration** to flourish. While the production of bugs is inevitable, the community can help identify them and (potentially) fix them. Users are also free to suggest extensions or additional features to the software.

4.3. INITIAL DESIGN



Win32.Trojan

MD5:
456b5ed8b5e8ff89e1619e1e5913ec90

IMPORTS:
ACTIVEDS.dll
KERNEL32.dll

C:\Desktop\webbrowser.exe

Figure 3. Mock-up of the proposed anti-malware system

An initial mock-up of the Anti-Malware system is illustrated in figure 3 to visualise the various modules and activities involved in the malware detection and classification process. A sample malicious file is illustrated to show the outcome of detection and classification.

The **File** module allows the user to upload a PE file from the machine. A file dialog will pop-up which will permit the user to browse local directories and choose a PE (EXE or DLL) file.

The **Scan Result** module displays the result of the *detection* and *classification* (if applicable) processes in a human-readable format. The first line indicates that the PE file is **malicious** and belongs to the **Win32.Trojan** malware family. The next line displays other relevant information such as corresponding MD5 hash and imported DLL file names.

The **Upload to VirusTotal** button will allow the user to upload the *signature* of the detected malicious PE file to an online virus database service such as VirusTotal.

The **Learning Mode Active** radio button indicates whether the machine learning based detection engine is recording the result and incorporating the outcome into the model.

This illustration serves as a starting point to understand the usability of the system from a holistic view-point. From the perspective of design, the overall aesthetic of the User-Interface makes use of the concept of *visual representation* which involves iconography, images, colours and other visual elements to capture the user's attention and enable ease of use.

CHAPTER 5: EVALUATION STRATEGY

Evaluation is an important stage for the success of the Honours Project.

The goal of this project is to build a robust software. Hence, it is required to test and evaluate the usability, functionality and robustness of the Anti-Malware system.

The purpose of doing this is to understand the following metrics:

- Completeness: Does the Anti-Malware system satisfy all the functional requirements?
- Soundness: Does the Anti-Malware system function as expected?
- Quality: Does the Anti-Malware system perform better than traditional and similar systems?
- Usability: Is the Anti-Malware system user-friendly and intuitive?
- Extensibility: Is the Anti-Malware system open to extensions/change?

For example: –

A possible scenario is:

ABC is a PE file known to be a Win32/Trojan. Does the Anti-Malware system successfully detect ABC?

The answer to this question is binary – either yes or no.

Another possible scenario is:

The Anti-Malware system boasts a modern User-Interface. Are users able to effectively scan a PE file ABC and determine its classification – malicious or benign?

The answer to this question is also binary – either yes or no.

However, for a question such as: **How easy is it to scan a PE file using the Anti-Malware system?**

The answer is quite subjective. Specifically, we aim to measure *subjective quantitative data*.

Two methods used to achieve this are:

1. Rating Scale: a scale from 0-10 where 0 symbolises a *non-intuitive* and 10 symbolises a highly *intuitive* system.

2. Likert Scale: a scale typically from 1-5 or 1-7 with categories such as:
    a. Strongly agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly disagree

To proceed with these evaluations and extract useful results, we can employ several objective and subjective methods. These methods and strategies are selected in accordance with the **Functional Requirements** and **Non-Functional Requirements** which were identified earlier.

The goal is to ensure a systematic way to evaluate the status of each requirement.

Functional Requirements:

The Functional Requirements actively specify the *functionalities* of the Anti-Malware system.

1. Scan all types of PE files
   To evaluate this, we will ensure a wide variety of file samples. The file samples will contain a variety of PE files and non-PE files. The system must accept the PE files and disregard the non-PE files. The result of this evaluation is binary – accept or reject for an input file.

2. Classify PE files as malicious or benign
   In Machine Learning problems, there is typically a set of samples. This set is divided into 2 separate entities – training and testing data. The classifier is *trained* with the training set provided with expected outcomes. The classifier is *tested* against the testing set. The result of this evaluation is binary – file classified as malicious or benign.

3. Classify malicious PE files based on category (Trojan, Adware, Virus, Worm, etc.)
   Similar to the above evaluation strategy, we will provide appropriate training and testing sets. The outcome of this requirement is arbitrary and depends on the *classes* or categories of malware specified. As an example – for a malicious PE file, the outcome can be *Trojan*, *Worm*, *Adware* or *Ransomware*. However, the outcome of the evaluation is binary – whether the category mapped is correct or incorrect.

4. User action on detected malicious PE file
   This requirement holds various outcomes. There are 3 choices to the user:
   a. Delete
   b. Ignore
   c. Quarantine
   To evaluate these, we must take a different approach for each user choice.
   Delete – completely remove the PE file from the system. The result of this evaluation is binary – malicious file deleted or not deleted.

   Ignore – do nothing on the malicious PE file. This can be evaluated by measuring certain characteristics of the PE file such as size, number of system calls, files targeted, etc. before and after being detected by the Anti-Malware system. The result of this evaluation is binary – malicious file ignored or not ignored.

   Quarantine – isolate the PE file in a safe location within the system. In this mode, the malicious PE file **must not** harm the system. To evaluate this mechanism, the system's characteristics must be thoroughly recorded before and after the detection of the PE file by the Anti-Malware system.
   The result of this evaluation is binary – safely quarantined or not. This evaluation method requires the need of a safe, isolated environment (such as a virtual machine) to protect the machine from any leaks/attacks.

Non-Functional Requirements:

The Non-Functional Requirements are essentially grouped into categories as: Usability, Hardware and Extensibility.

Usability:

In Usability, we aim to evaluate the User-Interface (UI) of the Anti-Malware system. We will measure 5 different metrics. To evaluate these, we can use both – *qualitative* and *quantitative* methods.

1. **User-friendliness** describes the simplicity and ease-of-use of the Graphical User Interface (GUI). The best way to evaluate this is by means of open ended questions. This technique is subjective and hopes to capture the true thoughts of the user for the given system.
2. **Responsiveness** describes the process of receiving the right response for a particular user action. The outcome of this evaluation is binary – either a right response is received or not received. The evaluation strategy is to provide the user with a scenario with an achievable outcome. This process must be repeated for all critical user actions. To be successful, all user actions must be responsive.
3. **Consistency** describes the use of similar design patterns throughout the system. Consistency can potentially clash with the concept of aesthetic design. Having an overly consistent design pattern can lead to a *boring* or *repetitive* design. The best way to evaluate this metric is by using a Likert Scale or Rating Scale. The user is given the opportunity to *rate* the consistency of the system.
4. **Intuitiveness** describes the ability of the system to detect wrong inputs and other incorrect user actions. This is a broad statement. We are concerned about intuitiveness from a functionality and design point of view. A state of high intuitiveness is achieved by correct software testing. Test Driven Development (TDD) is a practice of writing tests during the development of the system. Defining appropriate unit tests can diminish the number of bugs in the program. Both Python and C# provide useful frameworks for implementing unit tests. The best way to evaluate this metric is by using a Likert Scale or Rating Scale. The user can provide a *rating* on the intuitiveness of the system.
5. **Familiarity** describes the ability of the system to follow the de-facto trends associated with its offerings. This metric is extremely subjective. A tech-savvy individual is more likely to appreciate a familiar design. Whereas, an occasional user is more concerned with the instantaneous usability of the system. A possible way to evaluate this metric is by employing open ended questions. This serves as a method to capture the true essence of familiarity for the user.

Hardware:

The system will run on machines under the Windows Operating System (OS). The idea is to evaluate the system under popular Windows OS versions such as Windows XP, Windows 7, Windows 8.1 and Windows 10. The outcome of this evaluation under each OS version is binary – running or not running.

Extensibility:

Developing extensible software is the right way to go. Future development involves addition of new features, fixing bugs, adapting to new standards or simple re-factoring the program. The system will feature extensive technical and non-technical documentation. The documentation will follow industry standards and will be hosted on a public repository. Surveys and questionnaires can be used to evaluate the documentation. The outcome of this evaluation is subjective and can vary on a user-to-user basis.

CHAPTER 6: PROJECT MANAGEMENT

To plan the ideation, development and assessment phases of the Honours Project, it is pertinent to chart a well-thought out, feasible and achievable project plan.

6.1. METHODOLOGY

The first step is to map out the methodology used to accomplish this task. Since this project involves building a software, it is viable to choose an industry-standard and rewarding methodology. A popular, yet feasible approach is SCRUM.

The main idea behind SCRUM is to build software iteratively with frequent deployments to enable regular feedback from the supervisor. A structured approach of producing functionalities and modules in an iterative manner greatly reduces the risk of *misunderstanding* the functional requirements. This is perhaps the underlying advantage of SCRUM over other development models such as the Waterfall model. However, SCRUM orients itself towards small-to-mid sized development teams. [16] We plan to adapt SCRUM and rekindle its processes to benefit the development of our project.

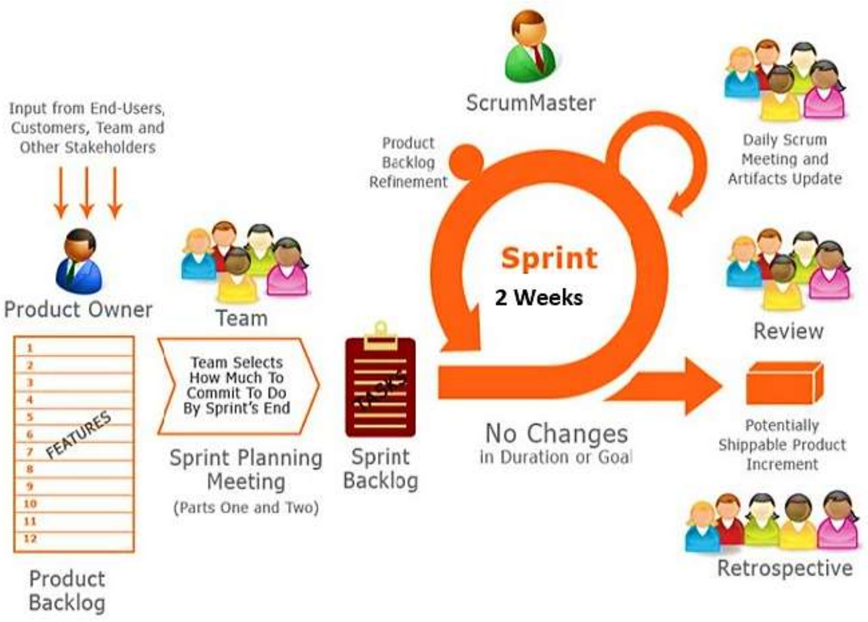
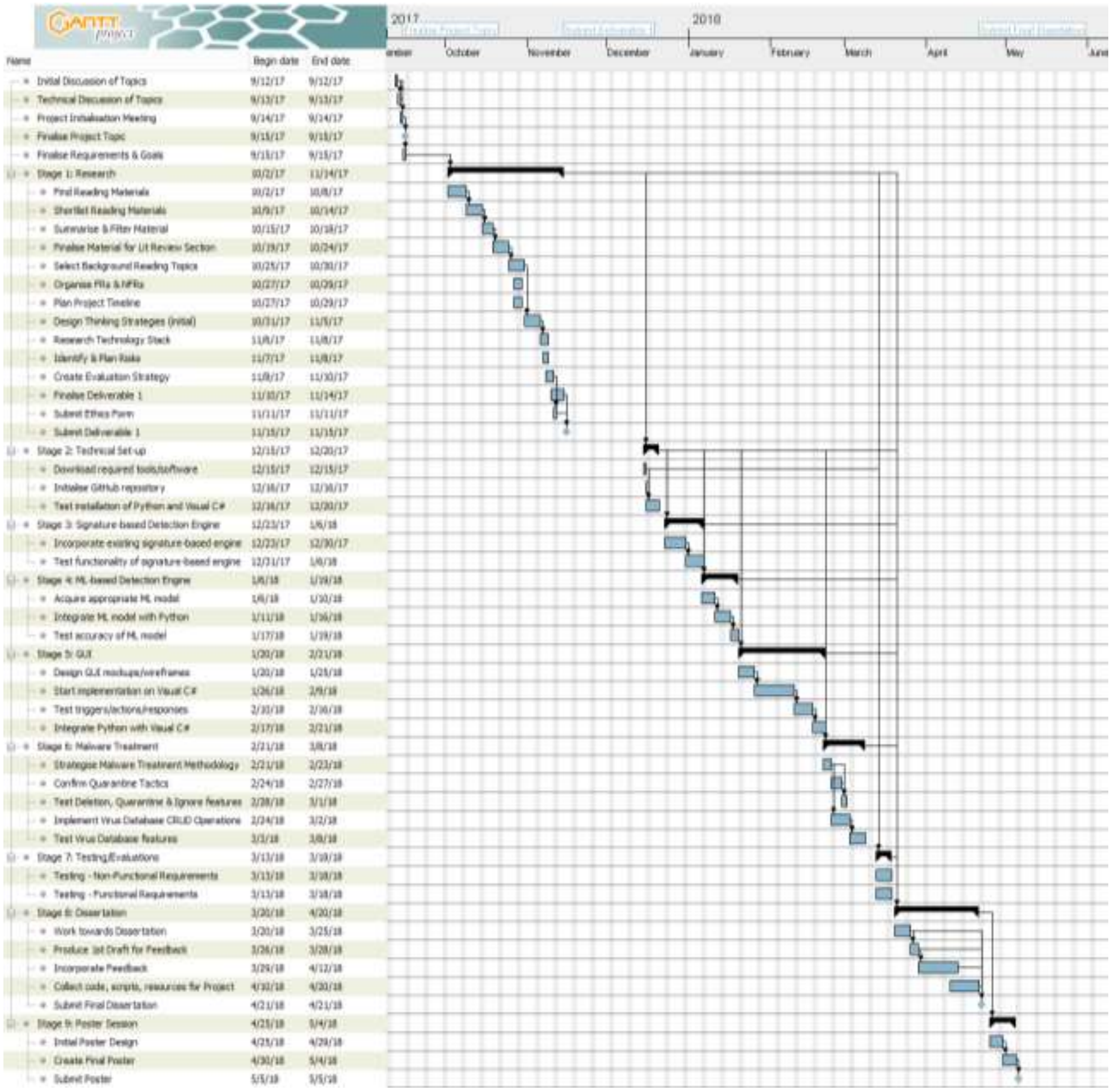

Input from End-Users, Customers, Team and Other Stakeholders
Product Owner
Team
ScrumMaster
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
2 Weeks
Review
Team Selects How Much To Commit To Do By Sprint's End
FEATURES
Sprint Planning Meeting
(Parts One and Two)
Sprint Backlog
No Changes
in Duration or Goal
Potentially Shippable Product Increment
Retrospective
Product Backlog

Figure 4. Outline of the SCRUM methodology [17]

As illustrated in figure 4, we plan to implement the project requirements in an iterative fashion. An iteration represents one development cycle. In SCRUM, an iteration is called as a *sprint*. The concept of feedback stems from regular meetings between the development team and product owner. In our case, the supervisor and student will meet weekly to discuss the *completed sprint* and *upcoming sprints*. These meetings are known as *SCRUM meetings* and involve presentation of the functionalities/modules completed so far. Further, all parties maintain a *product backlog* and a *sprint backlog*. The *product backlog* is a document which contains a prioritised list of requirements to be completed. Whereas, the *sprint backlog* represents a smaller list of tasks for the duration of a particular *sprint*. [16]

6.2.PROJECT PLAN



Name	Begin date	End date
Initial Discussion of Topics	9/12/17	9/12/17
Technical Discussion of Topics	9/13/17	9/13/17
Project Initialisation Meeting	9/14/17	9/14/17
Finalise Project Topic	9/15/17	9/15/17
Finalise Requirements & Goals	9/15/17	9/15/17
Stage 1: Research	10/2/17	11/14/17
Find Reading Materials	10/2/17	10/8/17
Shortlist Reading Materials	10/9/17	10/14/17
Summarise & Filter Material	10/15/17	10/18/17
Finalise Material for Lit Review Section	10/19/17	10/24/17
Select Background Reading Topics	10/25/17	10/30/17
Organise FRs & NFRs	10/27/17	10/29/17
Plan Project Timeline	10/27/17	10/29/17
Design Thinking Strategies (initial)	10/31/17	11/5/17
Research Technology Stack	11/6/17	11/8/17
Identify & Plan Risks	11/7/17	11/8/17
Create Evaluation Strategy	11/8/17	11/10/17
Finalise Deliverable 1	11/10/17	11/14/17
Submit Ethics Form	11/11/17	11/11/17
Submit Deliverable 1	11/15/17	11/15/17
Stage 2: Technical Set-up	12/15/17	12/20/17
Download required tools/software	12/15/17	12/15/17
Initialise GitHub repository	12/16/17	12/16/17
Test installation of Python and Visual C#	12/16/17	12/20/17
Stage 3: Signature-based Detection Engine	12/23/17	1/6/18
Incorporate existing signature-based engine	12/23/17	12/30/17
Test functionality of signature-based engine	12/31/17	1/6/18
Stage 4: ML-based Detection Engine	1/6/18	1/19/18
Acquire appropriate ML model	1/6/18	1/10/18
Integrate ML model with Python	1/11/18	1/16/18
Test accuracy of ML model	1/17/18	1/19/18
Stage 5: GUI	1/20/18	2/21/18
Design GUI mockups/wireframes	1/20/18	1/25/18
Start implementation on Visual C#	1/26/18	2/9/18
Test triggers/actions/responses	2/10/18	2/16/18
Integrate Python with Visual C#	2/17/18	2/21/18
Stage 6: Malware Treatment	2/21/18	3/8/18
Strategise Malware Treatment Methodology	2/21/18	2/23/18
Confirm Quarantine Tactics	2/24/18	2/27/18
Test Deletion, Quarantine & Ignore features	2/28/18	3/1/18
Implement Virus Database CRUD Operations	2/24/18	3/2/18
Test Virus Database features	3/3/18	3/8/18
Stage 7: Testing/Evaluations	3/13/18	3/18/18
Testing - Non-Functional Requirements	3/13/18	3/18/18
Testing - Functional Requirements	3/13/18	3/18/18
Stage 8: Dissertation	3/20/18	4/20/18
Work towards Dissertation	3/20/18	3/25/18
Produce 1st Draft for Feedback	3/26/18	3/28/18
Incorporate Feedback	3/29/18	4/12/18
Collect code, scripts, resources for Project	4/10/18	4/20/18
Submit Final Dissertation	4/21/18	4/21/18
Stage 9: Poster Session	4/25/18	5/4/18
Initial Poster Design	4/25/18	4/29/18
Create Final Poster	4/30/18	5/4/18
Submit Poster	5/5/18	5/5/18

Figure 5. Gantt chart outlining project activities, milestones and deadlines

Figure 5 illustrates the Gantt chart for the duration of the project. It conveys the activities, deliverables and deadlines. The Gantt chart serves as a timetable and outlines the activities in a sequential manner to target the relevant milestones within the given deadlines.

6.3. RISK MANAGEMENT PLAN

This subchapter focuses on identifying and understanding risks that could sabotage the success of the Honours Project. The saying *"prevention is better than cure"* is an important one and this project aims to have a proper, well-thought out strategy in place to combat any risks.

The goal is to complete all the activities and produce the necessary deliverables within their deadlines.

Risk Management is comprised of 4 different processes:

1. Risk Identification
2. Risk Analysis
3. Risk Planning
4. Risk Monitoring

6.3.1. Risk Identification & Analysis

There are certain metrics associated with a risk. They are as follows:

The **Probability** of a risk can be articulated based on the following scale:

Very Low	Low	Moderate	High	Very High

The **Impact** of a risk can be articulated based on the following scale:

Tolerable	Serious	Catastrophic

The **Priority** of a risk can be articulated based on the following scale (and colour scheme):

🟥 High	🟨 Medium	🟩 Low

The **Type** of a risk can be articulated based on the following scale:

Technology	People	Organisation	Tools	Estimation	Requirements

Table 1. List of risks accompanied with an analysis for each risk

ID	RISK	TYPE	PROBABILITY	IMPACT	INDICATION
🟥 R1	Malware leakage from virtual machine onto the host machine	Technology	High	Catastrophic	Unorthodox behaviour of machine
🟨 R2	Incompatibility of signature-based	Tools	Moderate	Serious	Difficulty in integration of given signature-

🟨	malware detection engine				based engine on the anti-malware system
🟥 R3	Corruption of given machine learning model	Technology	Moderate	Serious	Testing set giving erroneous answers for given model
🟨 R4	Failure to completely remove a detected malicious program from machine	Technology	Low	Catastrophic	Persistence or detection of given malware post-deletion
🟨 R5	Corruption of safe quarantine location for the anti-malware system	Technology	Low	Catastrophic	Failure to accurately quarantine detected malware in a safe environment
🟨 R6	Addition or drastic change of functional requirements for the anti-malware system	Requirements	Very Low	Serious	Discovery of legitimate functionalities which are needed by the system
🟩 R7	Wrongful estimation of time to develop anti-malware system	Estimation	Low	Serious	Milestones not being met within desired timeframe
🟨 R8	Confusing, non-intuitive and bug-filled User-Interface	Tools	Low	Tolerable	Users facing difficulty while using the anti-malware system
🟨 R9	Unable to meet deadlines for deliverables	Organisation	Low	Serious	Incompletion of various aspects of the project

6.3.2. Risk Planning & Monitoring

Now that the risks have been identified and categorised based on type, probability, impact and priority in table 1, the next step is to consider each risk and develop a strategy to *manage* that risk.

However, it is pertinent to ensure that each risk is thoroughly monitored. The idea is to regularly assess each identified risk to understand whether its probability of occurrence will increase or decrease.

There are three types of strategies which can be used for a risk:

1. Avoidance – to reduce the probability that the risk might arise
2. Minimisation – to reduce the impact of the risk on the project
3. Contingency – to describe how to deal with the risk if it arises

Table 2. List of avoidance, minimisation, contingency strategies and monitoring mechanisms for each risk

ID	AVOIDANCE	MINIMISATION	CONTINGENCY	MONITORING
R1	Ensure appropriate settings on virtual machine software such as snapshots, network isolation, no shared folders, etc.	Ensure readily available images of Windows operating system suitable for host machine and virtual machine.	Perform a complete re-installation of the host operating system and setting up a new virtual machine instance.	Perform regular checks on running processes, services and registry integrity on host machine.
R2	Ensure a readily available, functional and open-source signature-based malware detection engine.	Ensure sufficient options of signature-based engines – both proprietary and open-source. Also, conduct research on implementation strategies of such a detection engine.	Consider using a proprietary solution or write an implementation of the signature-based engine.	Perform regular research for updates about engines on open-source websites, blogs and repositories such as GitHub and StackOverflow.
R3	Ensure appropriate testing mechanisms involving the testing set of data for the given model to ensure that the	Ensure appropriate research on other potentially viable models. Also, maintain an	Make use of another existing model or create a new model by performing feature selection, feature	Perform regular rigorous testing of the machine learning model

🟥	accuracy is always within a certain threshold.	understanding of the features and patterns.	reduction and classification.	with a diverse testing set.
R4	Ensure rigorous testing mechanisms to be certain that a malicious file is completely removed from the machine.	Ensure readily available mechanism to suspend anti-malware system and automatically perform critical checks on the machine and network.	Suspend anti-malware system and perform rigorous checks on the operating system, registry, network and other critical environments.	Perform regular checks on persistence of malware post-deletion.
R5	Ensure a reliable storage point for the anti-malware system to store quarantined files.	Ensure the reliability of the deletion mechanism of the anti-malware system and another safe quarantine location.	Perform immediate deletion of malicious files and switch to the backup quarantine location.	Perform regular reliability checks on current and backup quarantine locations.
R6	Ensure a thorough initial analysis of the proposed anti-malware system and confirm the requirements with the supervisor.	Discuss with supervisor to produce a plan on incorporating changes/additions if and when they arise.	Use a plan (discussed with supervisor) to tackle any additions/changes in a sustainable manner which will not affect the production of other deliverables.	Discuss the functionalities and features of the proposed anti-malware system with supervisor and carefully capture his/her feedback.
R7	Ensure a well-thought and realistic project plan along with a Gantt chart outlining the activities, deliverables and deadlines.	Discuss with supervisor to produce a plan utilising a ranking-based mechanism to arrange the list of tasks to be completed.	Use a plan (discussed with supervisor) to complete tasks based on priority and importance.	Ensure that all activities, tasks and deliverables are completed as per the dates given on the Gantt chart.

R8	Ensure a suitable usability testing strategy and thoroughly analyse results regarding the overall UI and UX. Make changes iteratively based on feedback.	Ensure that a comprehensive user-guide is written and published for the system. Also, ensure that the code is maintainable with sufficient comments and documentation.	Deploy the user-guide to the users of the anti-malware system and simultaneously work on necessary changes on the UI and UX as soon as possible.	Regularly capture feedback on the anti-malware system from the supervisor and its other users.
R9	Ensure a suitable mechanism to split a large deliverable into smaller sub-units. Accomplish the smaller sub-units in an organised manner.	Discuss with supervisor to create a series of alternate activities for each deliverable to meet the goals of the project.	Attempt an alternative which meets the goals of the project and discard the incomplete deliverable activity (in the worst case).	Ensure regular meetings with supervisor to confirm the deliverable and all its corresponding sub-units, tasks and activities involved.

6.4. PROFESSIONAL, LEGAL, ETHICAL & SOCIAL ISSUES

The activities conducted towards the success of the project may result in certain professional, legal, ethical and social issues.

From an ethical perspective, the *evaluation strategy* involves usability testing with users. The purpose of usability testing is to capture the thoughts of the user towards the anti-malware system. As detailed in chapter 5 ,the main aim of the usability study is to measure 5 different metrics – User-friendliness, responsiveness, consistency, intuitiveness and familiarity.

This project requires permission for *interface only screening* where participants are given questionnaires to elaborate on their user experience with the anti-malware system. To alleviate this issue, an *ethical approval form* was submitted on the Honours Project System.

From a professional perspective, the users' host machine may be susceptible to a malware leakage from the anti-malware system. This was outlined as a *high priority* risk as mentioned in subchapters 6.3.1 and 6.3.2. However, this professional issue may quickly escalate to a potential legal issue as well. Sophisticated malware may potentially delete valuable data, corrupt networks and machines which may cause monetary loss to users. Similarly, all open-source, proprietary and other software, packages and libraries used towards the development of the anti-malware system will be respected and explicitly acknowledged. The code written to build the anti-malware system will be made publicly available on an online platform like GitHub for further development, bug-fixing and general feedback. Users are legally allowed to download, share, modify and distribute the code under a Creative Commons licence.

REFERENCES

[1] M. Sikorski and A. Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 2012.

[2] Kaspersky, “Malware Classification Tree,” Kaspersky Lab, [Online]. Available: https://securelist.com/threats/the-classification-tree/. [Accessed 13 November 2017].

[3] F-Secure, “Classification,” F-Secure, [Online]. Available: https://www.f-secure.com/en/web/labs_global/classification-types. [Accessed 13 November 2017].

[4] J. Andress, “Chapter 11 - Operating System Security,” in *The Basics of Information Security*, Elsevier Inc., 2014, pp. 171-187.

[5] O. Sukwong, H. S. Kim and J. C. Hoe, “Commercial Antivirus Software Effectiveness: An Empirical Study,” *IEEE Computer,* vol. 44, no. 3, pp. 63-70, 2011.

[6] R. Gunnar, “A Brief Introduction into Machine Learning,” Friedrich Miescher Laboratory of the Max Planck Society, Tübingen, Germany.

[7] W. Hämäläinen, “Machine learning,” [Online]. Available: http://www.cs.joensuu.fi/~whamalai/skc/ml.html. [Accessed 20 November 2017].

[8] M. Matteucci, “A Tutorial on Clustering Algorithms,” Politecnico di Milano, [Online]. Available: https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/. [Accessed 21 November 2017].

[9] U. Baldangombo, N. Jambaljav and S.-J. Horng, “A Static Malware Detection System Using Data Mining Methods,” *International Journal of Artificial Intelligence & Applications,* vol. 4, 2013.

[10] M. Siddiqui, M. C. Wang and J. Lee, “Data mining methods for malware detection using instruction sequences,” *Artificial Intelligence and Applications,* pp. 358-363, 2008.

[11] J. Z. Kolter and M. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” *Journal of Machine Learning Research,* vol. 7, pp. 2721-2744, 2006.

[12] J. Z. Kolter and M. Maloof, “Learning to Detect Malicious Executables in the Wild,” *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 2004.

[13] K. Rieck, T. Holz, C. Willems, P. Düssel and P. Laskov, “Learning and Classification of Malware Behavior,” *International Conference on Detection of Intrusions and Malware, and Bulnerability Assessment DIMVA,* pp. 108-125, 2008.

[14] R. Islam, R. Tian and L. Batten, “Classification of Malware Based on String and Function Feature Selection,” *Second Cybercrime and Trustworthy Computing Workshop (CTC),* vol. 2, 2010.

[15] J. A. Morales, R. Sandhu and S. Xu, “Evaluating Detection and Treatment Effectiveness of Commercial Anti-malware Programs,” *5th International Conference on Malicious and Unwanted Software (MALWARE)*, 2010.

[16] ScrumAlliance, “The Scrum Guide,” 2017. [Online]. Available: https://www.scrumalliance.org/why-scrum/scrum-guide. [Accessed 25 November 2017].

[17] *Agile BI Development Methodology*. [Art]. The George Washington University.