**Malware Categorization using Machine Learning**

A dissertation submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science

In

**The Queen’s University of Belfast**

By

**Thomas Pickup**

**29/04/2019**

# Declaration

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC3002 – COMPUTER SCIENCE PROJECT**

**Dissertation Cover Sheet**

A signed and completed cover sheet must accompany the submission of the Computer Science dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Thomas Pickup Student Number: 40145342

Project Title: Malware Categorization using Machine Learning

Supervisor: Dr Philip O’Kane

**Declaration of Academic Integrity**

Before signing the declaration below, please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook

2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)

3. Does not exceed the specified page limit

4. Is clearly presented and proof-read

5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**By submitting your dissertation, you declare that you have watched the video on plagiarism at http://www.qub.ac.uk/directorates/sgc/learning/WritingSkillsResources/Plagiarism/ and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment, and you have acknowledged all written and electronic sources used.**

*Student’s signature – Thomas Pickup Date of submission – 29/04/2019*

# Acknowledgements

I want to thank:

* My project supervisor **Philip O’Kane**, for his guidance, expertise and patience throughout this dissertation.
* The online community at **VirusShare** for their help in providing me with access to their online malware repositories used in this dissertation.
* My **Family** for their time and support throughout this final year dissertation.

# Abstract

The purpose of this dissertation is to explore how Machine Learning can assist with predicting a Malware’s Classification and Family using Machine Learning. The project created used a Cuckoo Environment to run the dynamic analysis; this produced a programme execution trace which was exported as a JSON file. A parser was created to extract features to be compiled into a dataset which had a Machine Learning algorithm performed on it to predict its family. The Machine Learning algorithm categorized each sample and used automatic feature selection to optimize the datasets performance. The final model was able to achieve an overall accuracy of 81.21% correct classification.

The report complements this project, expanding on how the project was implemented and designed. Research was also conducted that delves into the area of Malware Analysis and Machine Learning as well as a Literature review of previous work done into this area.

The project code repository can be found on the Queen’s University EEECS GitLab [1].

# Contents

[Declaration i](#_Toc7391850)

[Acknowledgements ii](#_Toc7391851)

[Abstract iii](#_Toc7391852)

[Contents iv](#_Toc7391853)

[1 Introduction and Problem Area 1](#_Toc7391854)

[1.1 Malware Types 2](#_Toc7391855)

[1.1.1 Trojan 2](#_Toc7391856)

[1.1.2 Worm 3](#_Toc7391857)

[1.1.3 Ransomware 4](#_Toc7391858)

[1.1.4 Keylogger 6](#_Toc7391859)

[1.2 Malware Analysis Techniques 7](#_Toc7391860)

[1.2.1 Static Analysis 7](#_Toc7391861)

[1.2.2 Dynamic Analysis 8](#_Toc7391862)

[1.2.3 Comparison 8](#_Toc7391863)

[1.2.4 Cuckoo Environment 9](#_Toc7391864)

[1.3 Machine Learning 10](#_Toc7391865)

[1.3.1 Supervised Learning 11](#_Toc7391866)

[1.3.2 Unsupervised Learning 11](#_Toc7391867)

[1.4 Previous Work 11](#_Toc7391868)

[2 Solution Description and System Requirements 12](#_Toc7391869)

[2.1 Solution Description 12](#_Toc7391870)

[2.2 System Requirements 12](#_Toc7391871)

[3 Design 14](#_Toc7391872)

[3.1 Architectural Design 14](#_Toc7391873)

[3.2 Software System Design 14](#_Toc7391874)

[3.2.1 Cuckoo Subsystem 15](#_Toc7391875)

[3.2.2 Parser Subsystem 17](#_Toc7391876)

[3.2.3 Machine Learning Subsystem 17](#_Toc7391877)

[3.3 User Interaction 18](#_Toc7391878)

[3.4 Key Design Decisions 20](#_Toc7391879)

[3.4.1 Machine Learning Algorithm 21](#_Toc7391880)

[3.4.2 Malware Chosen for Analysis 23](#_Toc7391881)

[3.4.3 Error Handling and Validation 25](#_Toc7391882)

[4 Implementation 27](#_Toc7391883)

[4.1 Use of Supporting Tools 27](#_Toc7391884)

[4.1.1 Languages Used 27](#_Toc7391885)

[4.1.2 Development Environment 27](#_Toc7391886)

[4.1.3 Version Control 28](#_Toc7391887)

[4.2 Key Implementation Decisions 28](#_Toc7391888)

[4.2.1 API Calls Dataset Structure 28](#_Toc7391889)

[4.2.2 Automatic Feature Selection 29](#_Toc7391890)

[4.2.3 Distribution 30](#_Toc7391891)

[4.3 Use of Software Libraries 30](#_Toc7391892)

[4.3.1 Python Libraries 30](#_Toc7391893)

[4.3.2 R Libraries 31](#_Toc7391894)

[4.4 Important Processes 32](#_Toc7391895)

[4.4.1 Parsing of JSON Reports 32](#_Toc7391896)

[4.4.2 Model Creation 33](#_Toc7391897)

[4.4.3 Unseen Sample Submission 34](#_Toc7391898)

[4.5 Implementation of Components 35](#_Toc7391899)

[4.5.1 Cuckoo Subsystem 35](#_Toc7391900)

[4.5.2 Parser Subsystem 36](#_Toc7391901)

[4.5.3 Machine Learning Subsystem 36](#_Toc7391902)

[4.5.4 Graphical User Interface 36](#_Toc7391903)

[5 Testing 38](#_Toc7391904)

[5.1 Testing Approach 38](#_Toc7391905)

[5.1.1 Black Box 38](#_Toc7391906)

[5.1.2 White Box 38](#_Toc7391907)

[5.1.3 Model Testing 38](#_Toc7391908)

[5.2 Testing Results 39](#_Toc7391909)

[5.2.1 Test Case 01 39](#_Toc7391910)

[5.2.2 Test Case 02 39](#_Toc7391911)

[5.2.3 Test Case 03 40](#_Toc7391912)

[5.2.4 Test Case 04 40](#_Toc7391913)

[6 System Evaluation and Experimental Results 41](#_Toc7391914)

[6.1 Evaluation of Results 41](#_Toc7391915)

[6.2 Technical Conclusion 42](#_Toc7391916)

[6.3 Personal Conclusion 43](#_Toc7391917)

[Appendices i](#_Toc7391918)

[User Manual i](#_Toc7391919)

[References ii](#_Toc7391920)

[Table of Figures vi](#_Toc7391921)

[Table of Tables vii](#_Toc7391922)

[Table of Equations viii](#_Toc7391923)

# Introduction and Problem Area

Malware: any malicious program or code that is harmful to computers.

With an expected, 20 Billion Internet-connected devices to come online by 2020 [2], protection against and identification of Malware is becoming more critical by the day. The average cost of a malware attack on a company is $2.4million and the cost in time of a malware attack is 50 days [3].

It is clear that Malware is becoming even more varied, with the NHS Ransomware attack spanning off multiple clones of the WannaCry virus, G Data Software estimates that in 2017 alone there were 7.41 million new malware specimens [4].

The cost and rate at which malware is growing make this field one of the most important in the Computer Science industry. Current methods of analysing malware are not proving useful for this ever-changing field. Static Analysis is prevalent, but the major downside is that it does not protect against zero-day attacks or new strains. Even polymorphic code can fool malware detectors that rely upon Static Analysis.

Machine Learning algorithms using dynamic analysis provide a viable alternative to this limitation, by basing their result on the behaviour of the specimen, the model theoretically can predict not only whether it is benign or malicious but could also be used to classify what family of malware the sample belongs to.

The goal of this dissertation to provide an in-depth look into how we could use machine learning in the future to classify malware. The dissertation will look into different methods of malware analysis techniques; it will then document the process of setting up a Cuckoo Sandbox environment that will allow us to analyse the behaviour of the specimen. This dataset will be used with a machine learning algorithm to predict what type of malware (or benign) a sample is. This process will be repeated to determine the optimal category definition.

In Section 1, a detailed introduction into the field of Malware Analysis and Machine Learning will be given. Section 2 will specify the key aims for this project as well as the system requirements.

Section 3 will provide a detailed look into the design process, giving and explaining the critical design decisions that were made. Expanding on this, Section 4 will look at the Implementation process and what further considerations were made.

In Section 5, will test the system, looking into the overall performance via key metrics as well as ensuring that the functional requirements are met. Finally, in Section 6 an evaluation of the results will be given, and both technical and personal conclusions will be provided.

## Malware Types

As this dissertation will focus on identifying malicious files, an enhanced overview of different types of Malware is required, in this section is a detailed overview of the main types of Malware that are active today. Later on, in this dissertation will be an overview of the specific malicious families that will be tested.

### Trojan

A Trojan Horse, commonly referred to as a Trojan, is a virus that is often disguised as legitimate software. It is called a Trojan due to the method of attack used by the Greeks in the Trojan war; the Greeks gifted a huge wooden horse that concealed an army. This parallels the attack method used by the Trojan Virus, the payload that would do the damage is hidden in a legitimate program [5].

A cybercriminal would often use social engineering to spread a Trojan Virus. Usually, a victim would have to click on a fake link or email that would redirect the user to a webpage often designed in the style of the legitimate counterpart. Once the user downloads and launches the file, the Trojan may execute [6].



Figure 1. Example of Flash Player Trojan Webpage

In Figure 1above you can see a webpage that looks identical to the Adobe Flashplayer download page, however in this example, the webpage is from a website called flesh-updates-max.com, this would be set up by the cybercriminal. This attack method would catch out most unsuspecting users.

The Trojan horse, when on your system can do many things, most would be designed not to alert the user to the Trojan Horses presence. For example, Trojans can work as spyware when they’re working like this; they would capture the user's credit card details, passwords and other information that could be valuable to sell on or exploit. Another way a Trojan Horse can cause damage is by using your computer as part of a botnet; this can cause damage to other users, mainly when performing a Distributed Denial of Service attack. The DDoS attack would work with other computers and devices on the botnet to overload a target's network. The target of this attack could be anyone from a single person up to a multinational company, recent examples of this attack are the Boxing Day attacks on online game services such as Xbox Live and PlayStation Network. Being part of a botnet can put the user in great trouble as cyber-attacks are often detected through the originators IP address [6] [7].

### Worm

A Worm is a virus that works by copying multiple instances of itself and infecting other computers within the network. The key indicator of a system becoming infected by a Worm is the when System Resources start to consume a large number of resources; this slows down the system. The reasoning for this is that the worm will often infect parts of the operating system itself meaning that to an untrained eye all that would be seen is a system resource being used excessively [8].

The Morris worm is often credited as becoming the first widespread use of a Worm virus. Created accidentally by Robert Tappan Morris in November 1988, it was meant to be research into understanding how a Worm could spread quickly. When a Worm looks for a new system, it sees if there is or was a Worm previously on that system; System Administrators realised they could respond to the Worm’s requests with a simple “Yes” and the Worm would not try to infect the system. Morris programmed the worm to infect, even if a “Yes” was issued one in seven times. This is the reason why the Worm infected so many computers; around 10% of the internet is often suggested. This is an example of how quickly a Worm can infect systems as well as how a Worm can use vulnerabilities in system functions to propagate [9].

The most common infection method of a Worm is via a software vulnerability. For example, the Morris Worm mentioned above used a weakness in the Unix Sendmail program as well as weak passwords on systems as its attack vector [10].



Figure 2. "Self-Retweeting Tweet"

Less sinister Worms, like the “Self Retweeting Tweet” in Figure 2, used Cross-Site Scripting on Twitter as an attack vector. This exploited a vulnerability in where Twitter would display the *<Script>* HTML tag as code rather than text. This resulted in a JavaScript code snippet being run that searches for the retweet button and presses it. As the JavaScript code was embedded in the Tweet, this would be executed whenever it appeared on a user’s feed. Although this was not used for a serious offence, it highlighted that this form of Worm could be used in a more severe manner, performing any browser function or even downloading files as a user without them knowing [11].

Worms can often be used as transportation methods for other types of Malware; for example, the WannaCry Ransomware attack used a Worm as its primary transportation method [12]. By exploiting a vulnerability in the way, Windows handles the SMB Protocol. It was able to propagate across wide networks.

### Ransomware

Ransomware is any type of Malware that attempts to stop a user from using their system and demands payment in exchange for the release of this system [13].

There are various ways this can be achieved [13] [14]:

**Locker Ransomware** or **“Law Enforcement” Ransomware** often forces the user into paying out a fee to use their computer. WinLock, created circa 2007, would lock the user out of their computer by displaying pornographic images in full screen, the method of paying this fee was via SMS text message.

Another variant of this type of Ransomware is Reveton which would fool victims into thinking their computer had been took over by the FBI or Interpol and the only way to get access back was to pay via a prepaid card.



*Figure 3. Reveton Malware Screen*

This payment could range from $100 to $3000. It was successful as the average user would not know what to make of the message and would genuinely believe they are being investigated.

**“Scareware”** Ransomware used the simple tactic of telling users their computer is infected.

A typical example is to pose as a legitimate software company and tell the user there is a fee for getting rid of the malicious files.



*Figure 4. Example of Scareware Ransomware*

If the user decides against protection the ransomware is offering; then they are bombarded with pop-ups until they decide to pay. Usually, the files are safe. However, it gets the user through the annoyance of pop-ups and the inability to use their computer.

**Encrypting Ransomware** uses an encryption technique to lock the user out of their files. It then demands money for the unlock of these files.



*Figure 5. Screenshot of WannaCry virus and background*

An example of this type of malware is the WannaCry attack. When infected, the WannaCry virus would work through all the user’s files encrypting them with a hybrid of RSA and AES encryption [15]. This meant that a user’s files were encrypted and irretrievable without a decryption key. A message would then be displayed, telling the user that their files were encrypted and that they had to pay a fee to get them unlocked, seen in Figure 5. Handily, the creators of WannaCry were willing to help by providing the encryption key for $300 in BitCoin. Due to BitCoin’s anonymity, it meant the creator was difficult to track down [12]. The damage from this Ransomware was insurmountable, with companies having to decide whether the data was worth the risk of losing or they should risk paying the fee in BitCoin without the substantial promise of a decryption key being provided.

### Keylogger

A keylogger is any tool or function that captures a user’s input and then sends it or stores it unbeknownst to the user. With a bad actor behind a Keylogger, it can be used to syphon off users passwords, credit card details, and other personal information. A fake phishing email could be used as a primary point of infection with the user clicking on a malicious link that downloads the executable file and runs it [16].



Figure 6. Example KeyLogger Interfaces

There are consumer-facing Keyloggers that can be bought on a monthly subscription. One keylogger shown in Figure 6 can be used to record KeyStrokes automatically, Websites Visited, as well as the Microphone and Webcam [17]. Keyloggers may also be used as part of physical hardware, as one news outlet has reported, Keyloggers are being used by at least one student to record exam and test questions typed into an unsuspecting teachers’ computer. These devices can be bought for as low as $40 and look exactly like a regular USB thumb drive. Sometimes they can be installed into the keyboard itself [18]. This attack vector is the most open to consumers with products being directed towards the curious.

## Malware Analysis Techniques

This project will require analysis of the Malware Files to compile the Dataset used for Machine Learning. There are two main types of analysis, Dynamic and Static; these are then tied together and utilised in an open source product called Cuckoo.

### Static Analysis

Static Analysis is a conventional technique used by most Antivirus software; it looks at the properties of the sample file to determine whether a file seems malicious. This does not run the file. Instead, this technique looks at the file itself.

Basic indicators of whether a file is malicious or not could range from the name of the file right through to the MD5 Hash of a file [19]. A common way to try and piece together what a program is doing is by using the Microsoft Strings utility; this allows a user to pass in an executable, the utility will then extract any embedded Unicode and ASCII strings that are contained within the executable [20]. This allows a researcher to try and understand what a program is doing based upon what strings are included, for example, if a program returns an IP address as a string, this could indicate that the program is trying to scan or connect to an external device.

An MD5 Hash can allow a researcher or Antivirus to detect whether a program is a malicious file. Each file will have a unique string given to it that is derived from the data in the file. This means that the same malicious file if distributed to many computers, will still have the same MD5 Hash. This hash can be compared with a database of known malware hashes to determine whether or not that file is malicious or not.

A Packer works to compress an executable, thus minimizing storage and bandwidth, this was useful in the early days of computing and the internet, where storage and bandwidth were an issue. Legitimate companies will often use packers to bundle executables; however, with packers commercially available, this has opened their use up to bad actors [21]. The problem with packers for malware researchers is that they obfuscate the code, in a way to conceal the functionality, this can make it harder to analyse its functionality statically. For example, a packed program would have its own MD5 Hash meaning that an AntiVirus based purely on the Hash of a file would allow a packed malicious file to run if it hadn’t been previously discovered in that packed form. A security researcher could use a tool like PEiD to check whether a packer has been used, this is not always a sign that a program is malicious, but is a vital tool into figuring out how whether a program is trying to conceal its actual functions.

An advanced static analysis method is to reverse engineer the executable, using tools like IDA Pro, a researcher can see the Assembly Code for a file. This can expose what the code is trying to do. For example, if a program is calling the subroutine *GetInternetConnectedState* from Kernel32.dll, we could assume it is trying to initiate an internet connection. The Assembly code gives us a line by line overview of what the malicious program will execute. It is unlikely an Antivirus would use this information and would most likely be used for further research on a potentially malicious file.

### Dynamic Analysis

Dynamic Analysis works by running a sample file, and recording what it does. This would typically be run, when the sample is being researched, on a Virtual Machine that would create a Sandbox Environment in which the Malware can’t spread and can be studied.

Tools like ProcMon, allow a researcher to see what an executable is doing by logging all of the system calls. From this data, a researcher could see if something is written to the registry, or if a sample creates or deletes files. It can even tell if a malicious file is listening to a network port. This allows the researcher to see precisely what the program has done and when it did it. It does not, however, check for network activity.

For Network Activity, a researcher could use Wireshark; this would be set up inside the sandbox or a separate machine on a virtual network. Wireshark is a packet capture tool that uses PCap to log all network activity. On a virtual network, this allows the researcher to see what websites or external IP Addresses an executable is trying to access, as well as the protocol it is trying to access them over. The WannaCry virus can be seen trying to access the SMB Port 445 as well as looking up to killswitch domain. The packet capture data largely assisted researchers in stopping the spread of the WannaCry virus as the data showed only after a failed attempt to resolve the killswitch domain name would the virus attempt to run [22].

### Comparison

Static Analysis’ primary benefit is that the potentially malicious file does not have to be executed, thus allowing the program to do what it wants, this makes it a lot safer than the alternative of Dynamic Analysis.

Dynamic Analysis, however, looks at how the program behaves, rather than basing the decision entirely on the signature. A bad actor may have changed the source code, leading to an incorrect signature classification, however as longs as it functions relatively the same, a tool that uses dynamic analysis may be able to flag it up.

### Cuckoo Environment

The Cuckoo Environment is an open source automated tool that runs both static and dynamic analysis on samples passed to it. It utilises several subsystems that allow it to examine malware samples effectively passing on the results in a JSON format.

It does this with the use of a Virtual Machine. This Virtual Machine is created by the user and allows for customization to suit the user’s needs, for example, if the user wants to diagnose only Java files the Virtual Machine would have a basic copy of Windows 7 with just Python and Java Run Time Environment installed. In section 3.2.1, more detail will be provided on the design considerations on the Virtual Machine needed for this project as well as how Cuckoo will be set up and interfaces with the other subsystems.

#### Cuckoo Web

The Cuckoo Web UI provides a graphical user interface that the user can use to submit samples as well as check and download the report generated once the analysis has been completed. As well as this, a REST API is also provided that allows for the remote submission of samples as well as the downloading of reports as above [23].



Figure 7. Cuckoo Web UI

#### Cuckoo Scoring System

The Cuckoo scoring system is a calculated decimal that can roughly describe how malicious or potentially dangerous a sample is. Once the Virtual Machine has finished it’s run of the sample, Cuckoo compiles a report of what it has done and assigns a threat level to events of suspicion. In Figure 8 you can see a list of alerts; the threat level ranges from one (Blue) to three (Red).

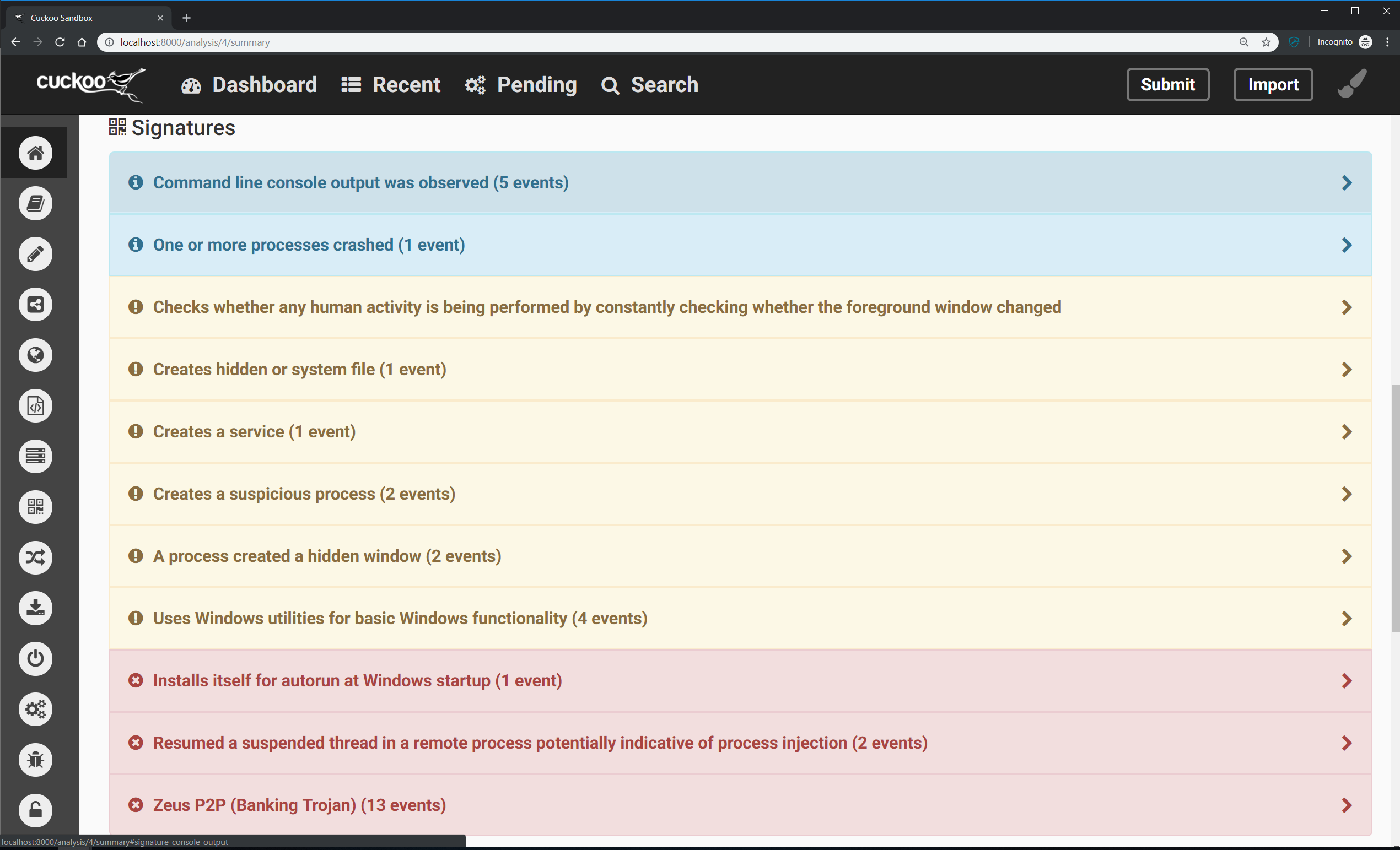


Figure 8. Example of Cuckoo Signature Alerts

The Sample’s Cuckoo Score is calculated by adding up all the threat levels and then dividing them by 5.0. In the case above, this comes to:

Equation 1. Equation for Cuckoo Score

The scores are then ranked with the following boundaries:

* 0 – 4 is Low Risk
* 4 – 7 is a Medium Risk
* 7 – 10+ is a High Risk

This number doesn’t necessarily prove that a program is malicious, however like the previous analysis techniques, when pieced together with other evidence it can be useful to find out what malicious activities the sample is trying to do.

## Machine Learning

Machine Learning will form a fundamental part of this project. It allows a program to form decisions based on previous data. There are two main sections to Machine Learning Algorithms; supervised and unsupervised learning; each of these techniques has its own merits and disadvantages.

### Supervised Learning

Supervise learning is when a known dataset is used to make predictions. A training dataset consists of input data and outcomes, for example, a dataset predicting the outcome of a student’s degree might include previous students A Level and High School Grades as the input data as well as a degree classification as an outcome. The Machine Learning algorithm attempts to produce a model that can be used to make predictions on unseen examples; this model can be optimized by using only the variables that produce a high amount of correlation with the output.

Supervised learning can be used for both *Classification* and *Regression* problems; the nature of this project calls for a classification.

### Unsupervised Learning

Unsupervised Learning techniques draw their conclusion from a dataset of unlabelled outcomes; the most common problem this solves is clustering issues. This is where an exploratory technique will be used to find hidden patterns and group data based upon those patterns. An outcome is then predicted based upon the similarity of the test data and the model.

## Previous Work

dddd

# Solution Description and System Requirements

Over the course of the project, a set of key aims and requirements will have to be agreed, in this section, those aims will be discussed. The definition of these aims and requirements is to ensure that the project is completed successfully and to allow for key metrics that will allow for better evaluation.

## Solution Description

The overall aim of this project is to produce a system that will take in a repository of Cuckoo Report files and produce a model capable of categorizing a sample file into a malware family based upon its behaviour. A GUI will be developed to allow the user to complete this task.

The general aims of this project are:

* Gain an understanding through current publications and online resources about malware analysis techniques and machine learning algorithms to design and implement the system.
* Set up an Environment that can run sample files and creating a report of exactly what the program has executed as well as other metrics such as whether the process was successful.
* Research into the current usage of Malware Analysis using Machine Learning and determine the metric on what to make decisions on.
* Gain an understanding of the data manipulation process, parsing the data from a report to make it machine friendly.
* Learn about machine learning algorithms with the goal of categorizing malware into a cluster that correlates its family, then proceed to data mine the dataset for multiple families.
* Gain an understanding of the GUI Creation process and good user interface design processes.

## System Requirements

To achieve the aims set out in Section 2.1, a number of functional and non-functional requirements must be clearly defined. These requirements will be used to benchmark whether the aims have been achieved throughout the project.

The functional requirements are:

* The program is able to read a repository of samples and output a list of those samples.
* The program is able to read the folder names those samples are in as the malware type and then output the folders as a list of malware types.
* The program is able to copy the “Reports.json” files from Cuckoo saving them in a repository of reports files.
* The program is able to parse those report files making a dataset of API Calls.
* The project is to include a Cuckoo Environment that is able to process samples passed to it.
* The Cuckoo Environment should include a virtual machine that is based on the Windows 7 OS to run the analysis on.
* The program should have a “Machine Learning Core” that is able to import the dataset and output a model that best represents the dataset.
* The program should be able to take in an unseen sample and evaluate it using a generated model.
* The Machine Learning Core should be able to produce accuracy metrics to allow for evaluation of how well the model works.
* The project should be run via a Graphical User Interface that will allow a user to interact and run the project in its entirety.
* The GUI should allow for customization of the program’s preferences, including the ability to change where the model is saved and read from.
* The GUI should provide an interface for the user to view statistics on the Model’s performance as well as the Dataset’s properties.

The non-functional requirements are:

* The program should be easy to use and distribute with little set up from the end user.
* The project should be robust and handle errors clearly, reporting back to the user where possible.
* The program code should be reusable and common functions should be accessible from subclasses to reduce code redundancy.
* The code should be clearly documented.
* The program, of course, should provide an accurate prediction that can identify malicious samples.

# Design

The design of the system is crucial to making sure all of the goals and requirements are met, this next section will provide a foundation for how the system is to be built and what challenges and decisions are to be made before the implementation begins.

## Architectural Design

The system will be built of various subsystems to achieve the overall aims and requirements set in Section 2. The GUI will be the frontend of the system; it allows the user to perform all of the actions that will make up the backend. Figure 9 shows the UML Diagram for the backend of the system, showing how the user might go about executing each of the actions.

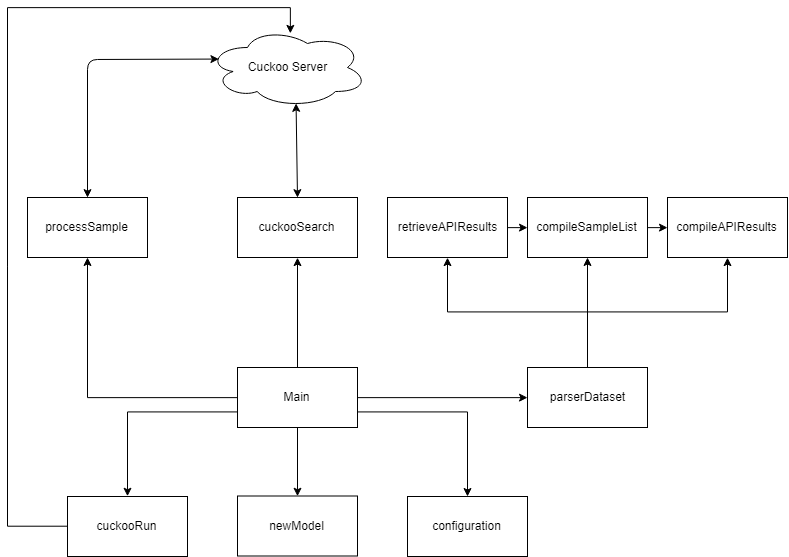


Figure 9. UML For the System

## Software System Design

There are three main parts to the system that will be developed; a flow diagram can be shown that demonstrates how these systems will interact in Figure 10. The first part is a cuckoo environment; this will take in an executable file and run it, making a report of the API Calls amongst other pieces of information. The next part of the system is a parser; this will take in the JSON file that has been created by the cuckoo environment and will extract features from this creating the training dataset. The final part of the system is the actual Machine Learning mind. It will be trained using the dataset created previously and will be able to take in a set of API Calls and predict the family of malware the executable belongs to.

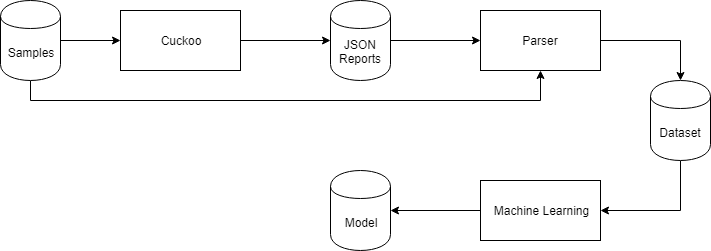


Figure 10. Model Generation Process

These three subsystems will be tied together by the Graphical User Interface that will allow the user to run these subsystems as well as submitting unseen examples, more detail about how to GUI will work can be seen in Section 3.3. The unseen submission process will have a separate process as seen in Figure 11; this will require several of the files outputted by the Model Generation Process.



Figure 11. Unseen Sample Submission

### Cuckoo Subsystem

The Cuckoo Subsystem runs the Dynamic Analysis on samples passed to it. It is responsible for generating the report containing the API Calls that is used for the parsing process. This subsystem takes in the Sample Repository and outputs amongst other things the JSON Report files that contain the API Calls. To accomplish this, two parts of the Cuckoo Subsystem will be needed; the Cuckoo Environment will be needed for both the model generation as well as the unseen sample submission. The Cuckoo API will only be used in the process of predicting an unseen sample.

#### Cuckoo Environment

The Cuckoo Environment will consist of a host machine and an analysis guest machine as shown in Figure 12. The samples will be uploaded to the Cuckoo Host by utilizing the web interface as shown in Section 1.2.4.1.



Figure 12. Example of Cuckoo Environment

The analysis machine will be a Windows 7 virtual machine; it will have nothing else installed other than Python 2.7 and Pillow. Pillow will be used to take screenshots of the Virtual Machine and could then be used to verify whether a program has run successfully and what the visual indicators are.

The Windows 7 virtual machine will be used to run the specimen file. This will allow the Cuckoo Host to perform dynamic analysis, looking at what that file is doing when activated. After it has finished executing, the Cuckoo host will generate the JSON report an example of this can be seen in Figure 13.



Figure 13. An example of the JSON reports collection.

Out of this report, we are interested in the API Calls. API stands for Application Programming Interface; these are functions in DLL files that an executable would run to perform various system tasks.



Figure 14. Example of API Call in JSON File

In the JSON report at Figure 14, we can see a whole host of details about what the process, in this case, a sample file called “stats.exe” is trying to do on the computer. In this case, it is running an API called NTAllocateVirtualMemory.

These JSON files will be passed on to the parser subsystem.

#### Cuckoo API

The Cuckoo API will be utilised to automatically submit and retrieve the reports for the unknown sample submission process described in Figure 11. The creators of Cuckoo document the API fully and this documentation will be used to assist with the creation of a separate subsystem that can interface with the Cuckoo Server [24].

### Parser Subsystem

As shown in the Figures Figure 10 and Figure 11, the JSON files will be fed into the parser subsystem that along with other items, such as the sample repository and in the case of unseen examples the original dataset, will be used to create a list of API Calls that can be used to train the model or evaluate an unseen sample. The subsystem will be incorporated into the Applications front end.

### Machine Learning Subsystem

This subsystem will be responsible for producing the model as well as utilizing the model to create an informed decision on the category of malware that a sample belongs in. It will take in the dataset of API Calls provided by the parser subsystem and perform a machine learning algorithm resulting in a dataset. For the unseen sample submission process, it will use the API Call dataset for that individual sample as well as the model to predict whether or not the sample is malicious as well as its category.

## User Interaction

The Graphical User Interface is essential in making sure that the project is user-friendly. It will require a simple layout that is easy to grasp. In this subsection, there is a wireframe design for each of the screens the user will see, as well as this there will be a paragraph explaining the design and how the user might interact with these screens.

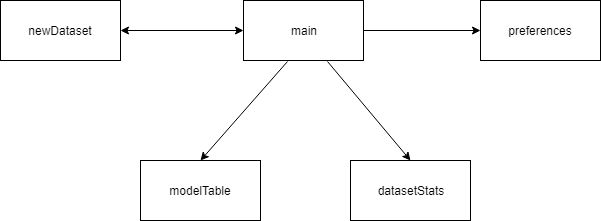


Figure 15. Relation Diagram of GUI Screens

Figure 16 is the Main Screen; this will be the first screen that the user sees when they launch the program. It will be a hub for the program, providing access to all the important functions as shown in the Architectural Design Diagram. The main feature of the Main Screen will be a console window in which data about the current progress of any functions will be shown. For example, the output on whether a sample is malicious or not and what family of malware it belongs to will be displayed there.

To the left of the console will be a sidebar containing various actions split up into four different sections. This will include various functions, from submitting an unseen sample, to creating new Datasets and new Models.

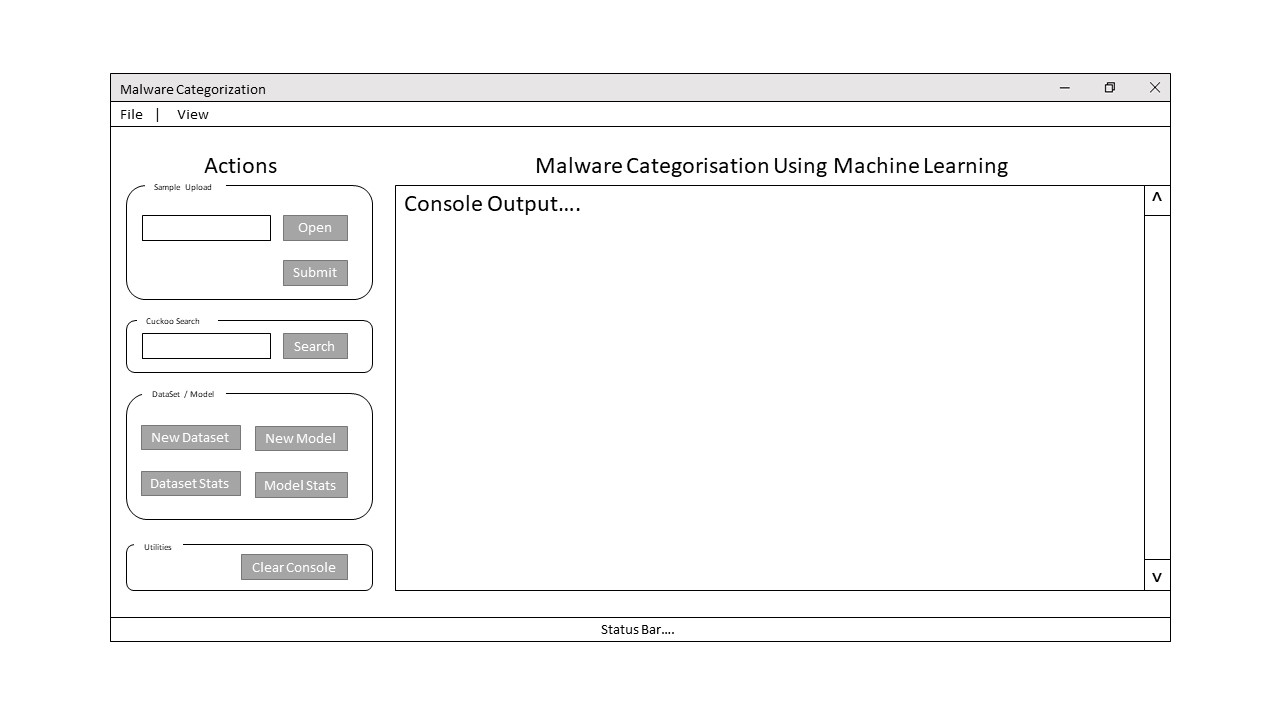


Figure 16. Main Screen

Figure 17 is the Model Statistics Screen; this will display to the user the overall accuracy ratings and breakdown of test cases, displaying the correct and incorrect number of samples tested. Underneath this will be the Confusion Matrix for the model. It will show a breakdown per malware family and paired with the F Score; this will provide a good insight into how accurate the categorisation truly is.



Figure 17. Model Statistics Screen

Figure 18 is the Dataset Statistics Screen; it will be another way in which the user can analyse the accuracy of the dataset. It will display the name of the family as well as the total number of items. As well as this, the average cuckoo score across the family will be displayed.



Figure 18. Dataset Statistics Screen

Figure 19 is the Preferences Screen; this will allow the user to change the core settings of the program. For example, the user will have the option to change where the dataset is read and saved to, as well as changing settings related to the machine learning environment and how it operates. When the user presses *Save* the program will check for any validation errors with the inputted settings.



Figure 19. Preferences Screen

Figure 20 is the New Dataset Options Screen; it will be launched when the user presses *New DataSet* on the Main Screen. The main reason for including this is that it will allow the user to customize what they want to do, during the dataset creation process. If the Reports are already in the Reports Directory, then the user won’t need to run the first step. Likewise, if the user has recently regenerated the dataset, they may just want to extract the Reports from the Cuckoo Directory. This screen will allow the user to customize this process. Once *Run New Dataset* has been pressed, a function in the main screen will be run, and this screen would be closed.



Figure 20. New Dataset Options Screen

## Key Design Decisions

To achieve the key aims and requirements set out in Section 2, several design considerations have to be made; this will range from the types of Malware Chosen for Analysis to how the program is going to provide Error Handling and Validation.

### Machine Learning Algorithm

The Machine Learning Algorithm is one of the most important design decisions it determines not only how accurate the Samples can be categorised, but also how quickly a model can be generated. The first decision to make is on what technique should be used, Supervised, or Unsupervised, based on the dataset containing labelled data and the issue is a classification problem, a *Supervised Learning* technique should be selected.

Supervised Learning techniques encompass a plethora of Machine Learning Algorithms, namely *kNN* (K Nearest Neighbours) and *SVM* (Support Vector Machines).

The SVM Algorithm is best used when there are only two different classes, if the project called for just a classification of whether something is malicious or not this would be ideal, however, the project requires a dataset with multiple classes, one for each type of sample submitted. This would rule out the SVM Algorithm.

Another reason to rule out the SVM Algorithm is that it is processor intensive when generating a large dataset; the time taken to process this data is higher.

The kNN Algorithm is one of the simplest Machine Learning Algorithms. However, it is also a highly accurate algorithm. It is used for multi-class problems; this is perfect due to the dataset having multiple malware families. A kNN Model is simple compared to other Machine Learning algorithms; all that is stored is the training data and their categories. This makes for an easy to distribute model.

The kNN Algorithm is the best algorithm for this dataset. Given this, a detailed look into how the algorithm works is shown in Section 3.4.1.1.

#### kNN Algorithm

The kNN Algorithms aim is to find a predefined (*k*) number of training samples that are closest to a test sample and use those points to predict a class for the new test sample. In this project, the Euclidean Distance is the measurement between the test sample and all other training samples that will be used.

Equation 2 shows how the distance is worked out between two points, *a* and *b*.

Equation 2. The formula for Euclidean Distance between Two Points

This equation provides the distance of a straight direct line between two points; it is best to visualise this as the hypotenuse of a right-angled triangle. Figure 21 shows this with the two points being corners of the triangle.

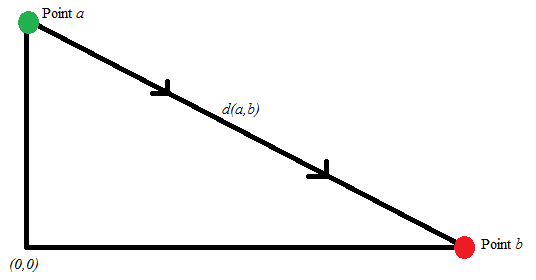


Figure 21. Euclidean Distance Visualised

This method of calculating the distance is the most common way of representing it as it always gives a straight line directly between two points, in the case of the kNN Algorithm this allows for an accurate representation of how a new test sample aligns with the training samples.

The final step once the distance between samples is calculated is to select the *k* closest to the test sample and tally up the number of samples in each class.

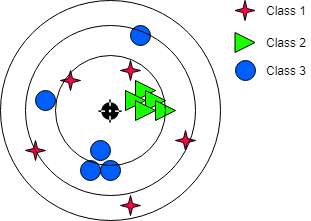


Figure 22. kNN Algorithm Visualised

In the example shown in Figure 22, with a value of *k* being equal to 5, approximately four *green triangles* classes and one *blue circle* classes would be selected. Given that there are more *green triangle* classes in the selected training set points, the algorithm would assume that the new test sample would be of the *green triangle* class type.

This type of algorithm rewards clusters, assuming that if a new test sample aligns with a cluster of training samples, then it would be of that class type. The nature of the solution requirements calls for a classification solution that rewards clusters, making the kNN Algorithm the recommended fit.

### Malware Chosen for Analysis

In Section 1.1, a brief overview of the most prominent malware types that are affecting users today, the following section describes the reasoning for choosing the specific Malware Families for analysis based upon their traits as well as known execution process.

#### Benign Files

Benign files will provide a classification for when a sample is not malicious. As the purpose of this project is not only to categorise malicious files into Malware Families but also to recognise Malware from safe (or Benign) files, this type is required for analysis.

The executables selected for analysis will be sourced from verified software vendors, it will be a mix of portable executables (requiring no installation) as well as installation files, the likes sourced from reputable vendors such as Adobe, and Oracle.

This coverage of different types of Benign files will hopefully lead to a successful ability to classify whether a sample is malicious or not.

#### CryptoRansom

The CryptoRansom family is the encrypting ransomware talked about it Section 1.1.3, it searches through the files on the user’s computer and proceeds to encrypt them with a random key, the random key is then sent back to the creator's server, and a message is shown alerting the user that their files have been encrypted.

Although this type ranges across multiple strains, they all follow a set process that should be representative of the family. This will hopefully provide a high accuracy rate.

#### InstallCore

InstallCore is a mixture of a KeyLogger and AdWare; however, for the purpose of this project, it will be used to represent a KeyLogger. It is installed, under the guise of a legitimate piece of software, in Figure 23 this can be seen as a PDF Converter. The user would download the installer from a malicious site, and then the installer would instead set up a KeyLogger and fail to install the legitimate application.



Figure 23. Example of Install Core Installation Screen

This means that the user would potentially not notice the KeyLogger being installed. The KeyLogger would then transmit the user's keystrokes to a central server that the creator of the Malware owns.

The data transmitted could potentially contain banking details as well as other sensitive information such as passwords and personal data. Given the fact that the installer follows the same process but with different images and guises, it should provide a solid cluster and thus a high accuracy rate.

#### Mediyes

Mediyes is a Trojan that redirects web traffic to a bad actor's servers. Due to the web traffic redirection, it means that unless a website uses SSL Encryption, a user’s personal data could be stolen. Even if the website does not use SSL Encryption, the websites a user is visiting could be valuable in conducting further attacks, for example, a targeted phishing email.

The Trojan copies a file *hidywpbr.sys* to the *system32* folder and then sets itself to run at startup [25]. Due to this a clear process is defined and should allow for the API Calls to form into a cluster.

#### Zeus / ZBot

Zeus, commonly referred to as ZBot, is a Trojan that is specifically designed to steal financial data. It was first discovered in 2007, and in 2011 the source code was released, this has led to an increased amount of other bad actors creating copycat programs that function relatively the same as the original. As a result of this, the newer strains should be categorized as Zeus.

The Zeus virus creates a Botnet; this is a network of controlled machines and a control server that allows the creator of the strain to control the machine. It also allows for massive data collection and for the creator to start a distributed attack using the infected computers as the attackers [26] [27].

The nature of this attack vector gives a clear process on where the malware is both looking for financial records as well as when it is injecting itself into other processes for the purpose of the botnet. This clear process should be beneficial in giving a cluster of API Calls.

#### Generic WinPE

Generic WinPE is, in this case, a mixture of all unknown families as well as the families stated in the Sections above (not including Benign files). The entire purpose of including this type in the dataset is to provide a catch-all in the case that a sample isn’t categorized as one of the above, meaning that completely unknown samples will not be simply categorized as Benign. By its design, it will mean that this type should have a low accuracy rating, in a similar way to Benign files, due to it being a diverse mix of different types of malware.

### Error Handling and Validation

To ensure that the system is as robust as possible and that any eventuality is considered, extensive error handling must be implemented. As well as this, to verify that the training set is complete and effective, Cross-Validation must be implemented.

#### Error Handling

Due to the application needing the ability to process files as well as allow user input, Error Handling with having to be implemented to ensure that any data passed through is valid. In the preferences screen, the user will be able to choose a folder location; the program should check if this location exists and is accessible. Similarly, before submitting an unseen sample or before performing a search, checks should be made to ensure the Cuckoo Server is online and running.

Error Handling will also be implemented to ensure that the flow of data has not been interrupted. For example, if the API Extension cannot download a report, it should try to retrieve it again alerting the user that there was an error. Basic checks should also be made to ensure that the system has the required libraries installed to make sure that it can proceed without impacting the flow of data.

#### Cross-Validation

Cross-Validation is a resampling technique used to evaluate the effectiveness of a limited dataset. For the purpose of this project, *k*-fold Cross Validation will be used. The *k*-fold methodology splits a dataset into *k* separate sections called folds; the algorithm then uses one fold as the test data and the remaining folds as the training data.

It is used to estimate the performance of a model on data that is not used in the model or unseen data; it results in a lower level of bias than other methods such as the train/test split.

The value of *k* can be any number from two to the number of items in the dataset. However, the value of *k* is commonly 10 due to a general model that has a low bias. The book Applied Predictive Modelling stated that “The choice of *k* is usually 5 or 10, but there is no formal rule”. This is due to the increase in *k* resulting in the difference between the training set and amount of items in the fold getting smaller; this results in a higher bias [28].

Based on this information, the default value of *k* will be set at 10, but the user will also have the ability to both turn on and off cross-validation as well as change the value of *k*. This can give limited control of the model creation process over to the user.

# Implementation

The implementation of the project will be discussed in this section, it will cover the tools used, the processes used, and the key decisions made throughout the process of implementing this project.

## Use of Supporting Tools

This project needed the use of plenty of Supporting Tools and Languages to develop it. Below is listed the tools and languages that were used in the making of this project as well as the development environment used to create the code.

### Languages Used

For the development of this project, there was a need for both a scripting language to automate the different subsystems as well as a language aimed towards data science. For this purpose, Python and R were chosen.

Python was chosen as the scripting language because of its ease of use as well as the fact that it was built with scripting in mind. It also contains multiple libraries for GUI development, chief among these is Tkinter, making the development of an effective UI possible. For the Machine Learning portion of the project, R was chosen; this is because of its advantages when manipulating large datasets as well as the inbuilt libraries that are provided with R.

As a note, Python also contains great support from pre-existing libraries to be used for Machine Learning, however my experience with R in the CSC3060: Artificial Intelligence and Data Analytics module during my final year gave R a slight advantage over Python.

### Development Environment

The project was developed on a Windows System that leverages Microsoft and Canonical’s recent partnership to run Ubuntu as a subsystem on Windows 10 [29]. This allowed for Cuckoo to be installed as part of this Ubuntu without the need for a separate Virtual Machine to become the Cuckoo Host. As the

For the development of the code, the Atom text editor was used. It is a text editor created by GitHub that has built-in support for Python Syntax highlighting. As it was developed by GitHub, it also has enhanced support for the Git Version Control System [30]. It was chosen for its simple layout as well as powerful enhancements.

### Version Control

During the course of the practical part of the dissertation, the Queen’s EEECS Gitlab service was used to host a Git Repository containing the code used for this project. A private Github Repository was also used as a secondary backup location.

The use of Git meant that both myself and the supervisor could access the project’s codebase at any time. By using the commit system to upload incremental changes, it gave an itemised report of the features and changes that had been made, down to the line of code. This meant that in the event of a bug or issue, it was easy to investigate where and when the bug was created. In the event of a catastrophic failure, a full rollback could be carried out by using the Git system.

## Key Implementation Decisions

Over the course of the implementation, numerous decisions had to be made to ensure that the project kept closely to the Design and Solution Requirements. In this subsection are the main decisions made to ensure this.

### API Calls Dataset Structure

Admittedly, the most important key decision to make during the implementation process was on how best to represent the API Call structure. The best way would be using a dataset of *n* samples long, with a row of *m* APIs long. The integer that would be used is the total amount of times the specific API was called during the process trace. This data structure should preserve as much information as possible, making sure that there is no loss of information. An example of this dataset structure can be seen in Figure 24.

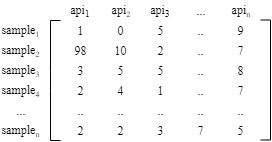


Figure 24. API Call Dataset Structure

Along with this dataset, to assist with the Machine Learning section, a sample list and malware types should also be exported. The sample list will comprise of the MD5 Hash as well as the Malware Type ID. This Malware Type ID will correspond to the Malware Types list, the layout of this will simply be the name of the Malware, taken from the Sample Repository and a numeric ID. Linking this together will give an MD5 Hash and Malware Type for each sample used in the dataset.

The Sample Repository is laid out as shown in Figure 25. Example Sample Repository. In the example below “Malware 1” and “Malware 2” correspond to two different malware types, and Sample A-E correspond to the MD5 Hashes.

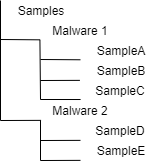


Figure 25. Example Sample Repository

### Automatic Feature Selection

Minimizing loss of data is important in helping the machine learning algorithm make as informed decision as possible. However, giving too much redundant information can not only slow down that decision-making process but hinder it entirely. To alleviate this issue, feature selection can not only cut down on the number of factors that are considered but also increase the accuracy of the model.

The feature selection algorithm that was chosen is the Boruta Algorithm. It is a novel feature selection algorithm for finding important variables that are built upon the Random Forest Classification algorithm [31]. It is relatively quick to implement and can be run without tuning parameters, making it truly automatic.

It works by running a Random Forest classifier computing a *Z* score that represents the average loss over its standard deviation. This can be used as a measure of importance. It works out how important a feature is based upon whether that feature scores above or below the average *Z* score. It removes the unimportant features and keeps the average and important features running the Random Forest classifier again until all the attributes have been assigned either average or important rating.

Once this has been completed, the algorithm leaves behind a dataset of features that have been deemed worthy. For this project, the set of important features will be used; this will ensure that the machine learning program is only using the features that have a high level of correlation to the output.

### Distribution

To successfully complete the requirement of the application being user-friendly, the application has to be easy to set up. Throughout the implementation process, changing from computer to computer to continue developing was a problem due to different libraries being installed on different computers. This meant that downloading and setting up of libraries was the first task that had to be completed each time a new computer was used for development. Putting that in the eyes of the user, this would be unacceptable and provide a barrier for this requirement to be fulfilled.

Due to this, the application has to be packaged into an executable that can be distributed easily and contains all the libraries needed for execution as a dependency in it. The solution to this is a Python Library called Py2Exe.

Py2Exe allows a developer to provide a Python script and along with specified additional files, output an executable that can be run on any users machine that has .NET 3.5 installed. Given the fact that every Windows PC has this installed as default since Windows XP, this should allow the user to run the application with little set up needed. It even bundles the required version of Python and all the libraries needed, meaning that any reliance on a specific version of Python with specific libraries installed is negated.

The use of Py2Exe means that a successful distribution can be created and thus the requirement of User Friendliness can be marked off from a portability standpoint.

## Use of Software Libraries

To implement this system, the use of a mixture of software libraries was needed, both in the Python Language and the R Language. Detailed in this subsection are the main ones that were used.

### Python Libraries

There are three key Libraries that were used within the Python code; these are Tkinter, Py2Exe and Threading.

**Tkinter** is the Graphical User Interface Library; it is a Python Interface to the Tcl/Tk library written in C. It provided an easy and quick way of creating a GUI given that the majority of common user interface controls are created as objects, for example, there is an object for textboxes and labels. The inclusion of this library allowed for rapid prototyping as well as the linking of the Python functions that were written to objects on the screen.

**Py2Exe** was responsible for providing this distribution executable of the program. A setup script was created that included additional files like the R Scripts and the location for the main python script. Running the setup script would compile these files as well as other libraries into its package. This package would then be converted into a *.NET* executable. The benefit of doing this is that the entire program can be run with little dependencies as Python 2.7.2 is included as a DLL file. This means that a user would be able to download the executable and have everything that they need to run to program right there.

**Threading** was responsible for running processor intensive tasks without impacting the user’s ability to use the UI. One of the problems that were first encountered when creating the UI was that when a long process was executed, it would freeze up the display. This was quickly narrowed down to the fact that python operated on one thread meaning that if a big task is executed it would wait for that to finish before continuing on, this impacted the UI because clicks would be handled in the same thread. The solution to this was to create a multithreaded application using the threading package; this meant that any long running process wouldn’t freeze up the main application screen.

Other smaller libraries were used, for example, the OS library was used to interface with the file system; saving files and checking for directories. As well as this the CSV library was used to open and close the datasets and corresponding files as CSVs.

### R Libraries

The Machine Learning sub-sections required the use of multiple libraries to perform the *k*NN Algorithm as well as other data manipulation

Caret & E1071 are used for the Machine Learning functions; Caret provides a simple way of implementing the *k*NN Algorithm and running predictions, rather than manually performing that process it automatically runs the operation providing a model to be exported. The E1071 package is used for data manipulation including the splitting of data for cross-validation, it is also used by Caret for similar operations.

Boruta automates the feature selection process; it implements the Boruta algorithm in R. As stated in Section 4.2.2, it is an important function to selecting only variables that have a high level of correlation to the output. It takes in the training dataset and outputs the features that have high importance. This is then used to select only the important features, or in our case API Calls.

## Important Processes

The program has several key processes that provide various functions; the main processes are the Parsing of JSON Reports, the Model Creation, and the Unseen Sample Submission processes. In this section is a discussion of those processes as well as flow diagrams to describe how they work.

### Parsing of JSON Reports

Figure 26 gives a detailed flow chart of the Parsing of JSON Reports Process after the figure is a detailed overview of each step of the process.

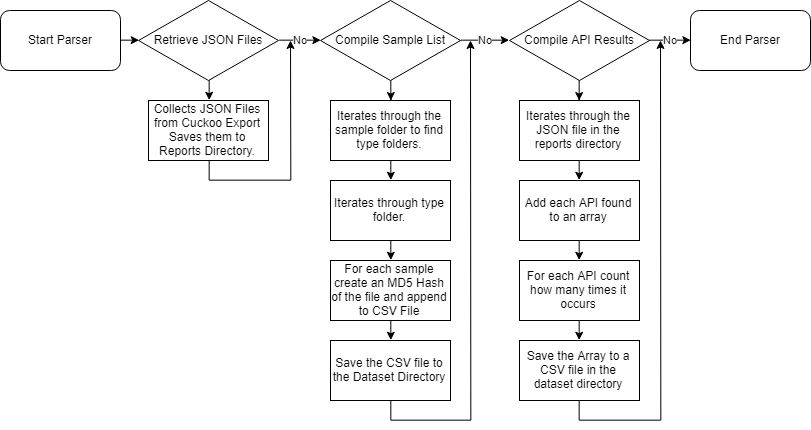


Figure 26. Parsing of JSON Reports Process

The Parser process is effectively the Dataset Creation process; it produces a dataset in a machine-readable format that can have machine learning performed on it. In the implementation process, the size of the repository went from 248 GB to a dataset of only 2.20 MB; this was due to only keeping the information that was necessary to produce informed decisions. The dataset as shown in Section **Error! Reference source not found.**, only keeps the total API calls.

To achieve this, the process first starts off by retrieving the JSON Reports from the Cuckoo Directory and copies them to the reports directory. As with each of these main steps, the user can decide whether they want to skip or carry it out as well as configuring the directories it looks through. These JSON Report files are used in the next steps.

Next, the process compiles a sample list, to do this it iterates through the sample folder, and then iterates through the malware type folders that it finds, it adds the name of which to an array that is exported as the *malware\_types* CSV file. In each of the malware type folders, there are multiple samples; the program iterates through each of these producing an MD5 Hash of the file. This MD5 Hash, as well as a sample ID that is an arbitrary number just incremented from zero, is added to a CSV file. This sample list is used in the model creation step.

The final sub-process is to produce the actual API Dataset; this step iterates through each of the JSON Report files, previously copied over from Cuckoo in the last step. At each iteration, the program looks through to find each API called by the sample; it stores this information in an array. The program then loops back for a second time on each JSON report, this time it iterates through each API in the array counting the number of times it was called. This result is then stored in a CSV file detailing the total API calls for each sample.

This entire process outputs a dataset that can be used by the Model Creation Process.

### Model Creation

Figure 27 gives a detailed flow chart of the Model Creation Process after the figure is a detailed overview of each step of the process.

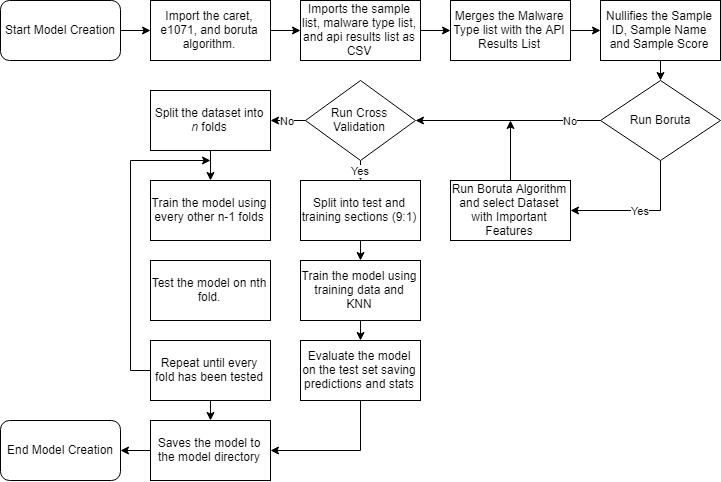


Figure 27. Model Creation Process

The Model Creation process combines the Dataset created in the last process with the *k*NN algorithm to produce a model capable of predicting whether something is malicious and what type of malware family it belongs to.

The first step in this process is to import the libraries and the dataset. After this basic pre-processing is performed to ensure that the dataset is complete and valid, the API Results is merged with the Sample List ensuring that only API reports from the original Sample set are used. Anything not in the sample list is discarded. The new dataset now has the SampleID, SampleName, and SampleScore fields nullified to ensure that the algorithm only bases its decision on the API Calls.

The user preference is then taken into account; if the user has selected not to run the Boruta Algorithm, then it skips this. If the user has selected to run the Boruta algorithm, it then runs it using the Boruta Package. The important features are then selected, and a new dataset is formed using only the features selected.

The next step also allows the user to have an input if the user decides not to run Cross-Validation, the program instead shuffles and splits the dataset into a training set containing 90% of the dataset. The remaining 10% is used as the test set. The training set is then used to train the model using the caret function train. This gives us the model that is later saved. After this the model is then evaluated using the test set, predictions including accuracy and the training set items are saved to the model repository along with the actual model.

If the user selects to run Cross-Validation, the program splits the dataset into *n* folds, *n* being the number of folds specified by the user, the default is set at 10. The program then repeats the steps as without Cross Validation but instead using one fold as the test set and the remaining folds as the training set until each fold has been evaluated. The accuracies are then taken as an average out of all the folds, and then the model is then saved.

This process takes in the dataset and effectively gives us a model that can allow the program to produce informed predictions on further unseen samples.

### Unseen Sample Submission

Figure 28 gives a detailed flow chart of the Unseen Sample Submission Process after the figure is a detailed overview of each step of the process.

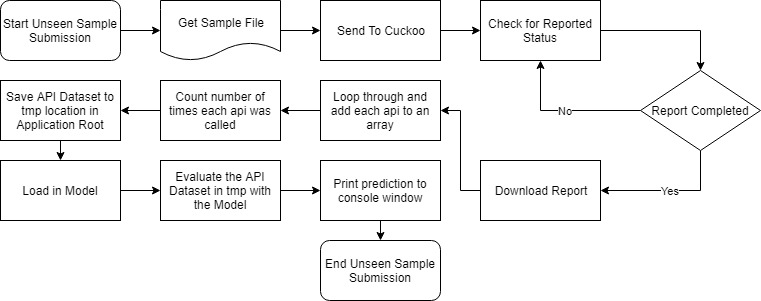


Figure 28. Unseen Sample Submission Process

The Unseen Sample Submission effectively combines the Parser subsystem, Cuckoo Subsystem, and the Machine Learning Subsystem. It is the culmination of every subsystem in the project demonstrating that the end goal is possible.

The process first starts off by getting the sample executable and sending it off to the Cuckoo Server for analysis; it uses the Cuckoo API to submit the sample file. This is then run through Dynamic Analysis, to check up on whether the report is completed, the process uses the API to download the current status; this is repeated until that status *reported* is given.

The Report is then downloaded, the program uses the same process as seen in the Parser process to produce an API dataset, and in this case, only one sample is used.

The program then loads in the model created previously and evaluates the API exported from the last step. It then prints the prediction of malware type to the console window for the user to view. This process ties together the previous steps to truly evaluate how effective the model is.

## Implementation of Components

The Processes shown in Section 4.4 needed the three core subsystems shown in Section 3.2 plus a Graphical User Interface to be implemented, in this section is a discussion detailing the implementation process as well as the challenges that were encountered.

### Cuckoo Subsystem

The Cuckoo subsystem leverages both the Windows 10 portion of the OS as well as the Ubuntu Subsystem provided by Canonical.

The Ubuntu Subsystem has MongoDB and Apache2 installed. The MongoDB database is used to store the results of dynamic analysis, and the Apache2 installation is used for the Cuckoo Web instance.

The Cuckoo subsystem’s part on Windows 10 uses Python 2.7; Cuckoo is installed through PIP (*PIP Installs Packages*). The Windows 10 machine has a firewall exception for both the Cuckoo API (8090) and the Cuckoo Web (8080), this means that the Cuckoo instance can be used from anywhere in the world if proper external firewall rules are set up. In this instance, there are firewall rules set up to port forward from the home network’s IP Address to the Cuckoo Machine’s IP Address. This provides that access.

To perform the dynamic analysis, Cuckoo uses a purpose-built Virtual Machine using Windows 7; this machine has only Python 2.7 installed with Pillow via pip. The Virtual Machine is set up with a Snapshot where only the Cuckoo Agent script is running; this means that no non-native program can interfere with the Dynamic Analysis. The Windows 7 machine is also set up with no Firewall and all programs set to run-as-admin making it the worst case scenario a machine can be. The purpose of this is to allow the sample to take full advantage of the machine exposing its true intentions. The guest machine is hosted in VMWare; this is due to its compatibility with Cuckoo, being able to control the machine automatically means no human interaction is needed to start and stop analysis.

### Parser Subsystem

The Parser subsystem is responsible for taking in the JSON Reports and exporting a dataset for Machine Learning. It is also partly responsible in the unseen sample submission for converting the JSON received from the Cuckoo API into the API Dataset used for the Model Evaluation. This subsystem is implemented using the Python Language.

Due to the handling of data, this subsystem also errors checks at every stage, making sure that the data passed through it is valid. One example of this is in the Unseen Sample Submission; the processing of the sample won’t proceed if there is not a model or dataset. This will then write an error message to the console window in the Graphical User Interface.

The subsystem implements the Parsing of the JSON Reports process in its entirety as well as the submission and API Dataset creation parts of the Unseen Sample Submission Process.

### Machine Learning Subsystem

The Machine Learning subsystem is written entirely in the R Language; it uses the Dataset generated from the Dataset Parser and unseen sample submission parser. It implements the Boruta Algorithm and Cross-Validation with *k*NN, producing a Model. The user can then submit to it an unseen sample which is then evaluated using the model created, to do this it simply imports the Model and uses the predict function from Caret.

This Subsystem implements the Model Creation Process in its entirety as well as the evaluation parts of the Unseen Sample Submission Process.

### Graphical User Interface

The GUI Package was implemented as per the design in Section 3.3; it is created with Tkinter, the Python Library described in Section 4.3.1 and allowed the user to interface with the Subsystems and Processes laid out above.

It was created with a simple design that allows the user to submit samples as well as run commands such as “New Dataset” or “New Model”. As well as this it allows the user to view statistics about the dataset created as well as the model.

Most of the information is displayed on the Console screen, seen in the middle of Figure 29, for example, progress on the commands issued as well as any errors are displayed in this screen. The program performs data validation on any inputs and uses the console window to alert the user to any irregularities.

As the GUI is updated via the main loop, any processor or time intensive functions must be run via threads, this is initiated from this GUI script and is controlled via the script.

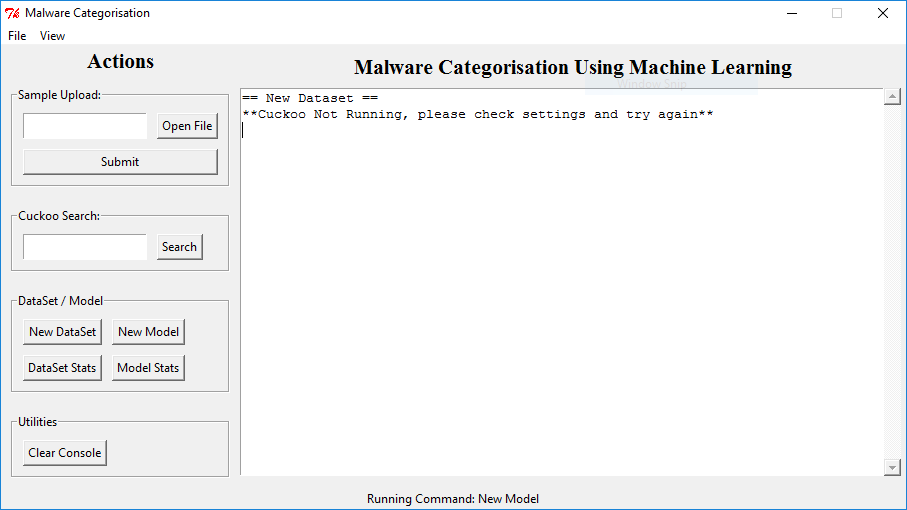


Figure 29. Implementation of GUI

# Testing

Testing is a vital part, ensuring that not only the aims and requirements of the system have been met, but that they are valid and correct. It also provides a way of measuring how successful the project is in meeting those requirements. Over the next section, is an explanation of different testing approaches that are utilised as well as the test results.

A full evaluation of the test results can be seen in Section 6.1.

## Testing Approach

Each section of the system should be tested in one way or another. This is to prove that the system works correctly and handles errors correctly. A mix of testing methodologies should be used to achieve this level of thoroughness.

### Black Box

Black Box testing is to test each and every module to make sure it is performing its function correctly. This can be seen by the fact that each module is developed into a class that can be invoked individually. The modules can also be invoked individually using the GUI Interface, in the case of New Dataset that is a complex process that encompasses three modules, each one can be run individually. This allows for checking of error handling and that the correct output is present.

### White Box

White Box is to test an entire process, an example of this in my program could be seen by initializing the New Dataset via the GUI, this allows me to test that this process is working in its entirety. By having a process run with multiple different inputs, and no errors being thrown, it shows that the process runs correctly and handles errors correctly. This would mean that the test has been passed.

### Model Testing

To thoroughly test the system, a test plan had to be developed that would make sure that the Feature Selection and Cross Validation is not only working but improving the accuracy of the model. The Automatic Feature Selection has been turned on and off along with 10 Fold Cross-Validation to ensure that every eventuality has been checked. Table 1, provides a Test Reference as well as what is enabled (Green) and Disabled (Red). The results of this testing are in Section 5.2, with headers corresponding to the reference numbers provided.

|  |  |  |
| --- | --- | --- |
| **Test Reference** | **Automatic Feature Selection** | **10 Fold Cross-Validation** |
| *Test Case 01* |  |  |
| *Test Case 02* |  |  |
| *Test Case 03* |  |  |
| *Test Case 04* |  |  |

Table 1. Test Reference Breakdown

## Testing Results

The following tests used, a dataset with the Families, Quantities, and Average Cuckoo Scores as seen in Table 1.

The selected testing set gives us a total training sample of 2810 malicious files, spread across Benign files and four families of Malware.

|  |  |  |
| --- | --- | --- |
| **Family** | **Average Cuckoo Score** | **Total Items** |
| Benign | 1.69 | 117 |
| CryptoRansom | 4.06 | 562 |
| InstallCore | 5.53 | 488 |
| Mediyes | 1.05 | 637 |
| Generic WinPE | 2.30 | 441 |
| Zeus | 3.18 | 565 |
|  | **2.97** | **2810** |

Table 2. Dataset Statistics

### Test Case 01

The overall accuracy for this test case was **73.38%**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Specificity | Precision | Recall | F1 |
| Benign | 99.25% | 71.43% | 45.45% | 55.56% |
| CryptoRansom | 90.54% | 61.82% | 60.72% | 61.26% |
| InstallCore | 98.70% | 94% | 97.92% | 95.92% |
| Mediyes | 89.30% | 72.62% | 96.83% | 82.99% |
| Generic WinPE | 98.72% | 85.71% | 40.91% | 55.38% |
| Zeus | 90.09% | 63.93% | 69.64% | 66.67% |

Table 3. Test Case 01 Confusion Matrix

### Test Case 02

The overall accuracy for this test case was **73.02%**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Specificity | Precision | Recall | F1 |
| Benign | 98.88% | 50% | 27.27% | 35.29% |
| CryptoRansom | 94.59% | 72.73% | 57.14% | 64% |
| InstallCore | 98.26% | 92.31% | 100% | 96% |
| Mediyes | 89.77% | 73.81% | 98.41% | 84.35% |
| Generic WinPE | 94.44% | 60.61% | 45.45% | 51.92% |
| Zeus | 90.54% | 64.41% | 67.86% | 66.09% |

Table 4. Test Case 02 Confusion Matrix

### Test Case 03

The overall accuracy for this test case was **81.21%**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Specificity | Precision | Recall | F1 |
| Benign | 99.26% | 60% | 27.27% | 37.50% |
| CryptoRansom | 95.41% | 82.76% | 76.19% | 79.34% |
| InstallCore | 99.12% | 96.30% | 96.30% | 96.30% |
| Mediyes | 91.70% | 73.24% | 100% | 84.55% |
| Generic WinPE | 96.60% | 77.14% | 58.70% | 66.67% |
| Zeus | 94.69% | 79.31% | 83.64% | 81.42% |

Table 5. Test Case 03 Confusion Matrix

### Test Case 04

The overall accuracy for this test case was **80.85%**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Specificity | Precision | Recall | F1 |
| Benign | 99.26% | 50% | 20% | 28.57% |
| CryptoRansom | 95.31% | 83.33% | 73.53% | 78.13% |
| InstallCore | 99.58% | 97.62% | 100% | 98.80% |
| Mediyes | 84.68% | 63.44% | 100% | 77.63% |
| Generic WinPE | 96.57% | 66.67% | 33.33% | 44.44% |
| Zeus | 93.81% | 75.86% | 80% | 77.88% |

Table 6. Test Case 04 Confusion Matrix

# System Evaluation and Experimental Results

Section 6 is an evaluation of the results as well as both Technical Conclusions and Personal Conclusions. Its aim is to provide a look back at whether the original goals and requirements have been met. To finish this section off, a look at future work to expand on this project, as well as future work in the industry, has been provided.

## Evaluation of Results

A first note to touch on is that the error handling on the system worked as expected, halting the program and alerting the user when the input was not valid or present. It would check to see if subsystems had communicated correctly with the main application, an example of this would be the Cuckoo System not responding or being off completely, a user would be alerted to any errors that had happened, or alternatively in a situation where a report wasn’t found, a separate process would be started to correct said error automatically. This passed the first two types of test, the black box and white box tests.

The Model Testing was a fundamental test to get right, it not only provided the validation that the model creation process works, but it also provided a way of verifying how accurate this process had been. The Test Cases One through Four tested the system with all variations of the model process, allowing a complete verification that the process was accurate.

The results showed that the most accurate model creation process was to run both 10-Fold Cross-Validation and the Automatic Feature Selection, Test Case 03. This should have been expected. With the Automatic Feature Selection process, testing multiple variations of the model and then removing features that don’t have a high level of correlation with the output, this provided a smaller more efficient model. Cross-Validation also providing a more accurate estimation of the limited dataset size.

The all features enabled solution is also the most intensive and demanding on the processor. The Automatic Feature Selection algorithm, Boruta, demands a lot of resources, putting a strain on the processor; this is due to the nature of what it does regenerate multiple models and taking features out. In reality, the addition of Automatic Feature Selection only provided an increase in overall accuracy rating of 0.36% over just running Cross Validation, Test Case 04. With this in mind, and given the small size of the dataset, the Test Case 04 might be perceived as the best result. Given it is drastically less intensive on the processor with the omission of the Boruta Algorithm.

Continuing on the Analysis of Test Case 04, the column of interest is the F Score (F1), this is the Harmonic Mean of the Precision and Recall Values. In other words, it portrays the balance between Precision, the measure of a classifiers exactness, and Recall, the measure of a classifiers completeness. It is represented by the formulae:

Equation 3. f Score Equation

There is a wide range of F1 Values shown in the Confusion Matrix for Test Case 04, from 28.57% to 98.80%. There are three scores in interest that may explain the downfalls of this model. The first two being the Benign and Generic WinPE F1 Scores, 28.57% and 44.44% respectively, what seems to be happening in these cases is that because these families contain a wide variety of sample types there doesn’t seem to be one process that is defining the family as a whole. Take the F1 Score for InstallCore; this Malware Family follows a specific process with many of the samples being cosmetic changes of the same Malicious File just perpetrating to install different pieces of software. The result of this would be a predictable API Call structure, that should lead to a high F1 Score with a high level of completeness and exactness, in this case, the section has achieved 98.80% F1 Score which would show that to be true.

## Technical Conclusion

Although the project achieved its original goal in demonstrating that Malware Analysis using Machine Learning is possible, there are definitely some areas in which this project could improve. To start on a positive, the Datasets size, although limited, provided a decent cover to identify the four non-generic families with at least 77% F1 Score. This could obviously be improved to cover even more families, and testing would be re-run to see if there is a decrease or increase to the overall accuracy. With Antivirus companies such as McAfee or Kaspersky building in a behavioural analysis as a last resort, they can capture an immense amount of samples spread across nearly every type of Malware Family.

Another point to touch on in terms of the Technical Conclusion, is although it is possible to use Dynamic Analysis with Machine Learning, it still carries the same downfalls of Dynamic Analysis in that the user would still have to run the file or send it off to an analysis server, possibly slowing down execution of a program that utilises multiple executable files. It could work as a last resort measure that major Antivirus companies are using, as the sample would only be passed to dynamic analysis if the Antivirus program isn’t entirely sure if the executable is malicious or not.

One final note to finish off the technical conclusion is that a conversion from the R Machine Learning Code to Python would be required to better integrate these modules into the main body on the code. It would also allow for more significant error handling on these scripts, as currently the error handling is carried out by the R script and little feedback is provided to the Python Script. This means that only basic error handling can be completed on the Python script, a message displayed on the console window, rather than retrying processes or displaying enhanced error messages.

All in all, I believe that technically this project has been a success, demonstrating that using API Calls it is possible to categorise the behaviour of a program and decide whether or not it is malicious.

## Personal Conclusion

This project has been a steep learning curve in both my technical skills as well as my management skills, having started late on the project I had to de-scope and simplify the project, although it still achieves the end goals it doesn’t include metrics like confidence or risk ratings that would be a welcome addition if time permitted. However, I do believe that this project has enhanced my project management skills as well as teaching me new skills in Machine Learning techniques and the Python Language.

# Appendices

The purpose of this section is to provide any extra documents that may be needed to enhance this dissertation.

## User Manual

# References

|  |  |
| --- | --- |
| [1] | T. J. Pickup, “GitLab Repository,” Queen's University Belfast, 21 September 2018. [Online]. Available: https://gitlab.eeecs.qub.ac.uk/40145342/csc3002-project. |
| [2] | Forbes, “Developing the Conected World of 2018 and Beyond,” Forbes, 16 March 2018. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/03/16/developing-the-connected-world-of-2018-and-beyond/#156cda811e51. [Accessed 02 April 2019]. |
| [3] | Varonis, “60 Must-Know Cybersecurity Statistics for 2019,” Varonis, 28 March 2019. [Online]. Available: https://www.varonis.com/blog/cybersecurity-statistics/. [Accessed 02 April 2019]. |
| [4] | G-Data Software, “Malware trends 2017,” G-Data Software, 10 April 2017. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017. [Accessed 02 April 2019]. |
| [5] | Britannica, “Trojan Horse,” Britannica, 20 July 1998. [Online]. Available: https://www.britannica.com/topic/Trojan-horse. [Accessed 02 April 2019]. |
| [6] | Comodo Security, “Trojan Horse Definition,” Comodo Security, [Online]. Available: https://enterprise.comodo.com/trojan-horse-definition.php. [Accessed 02 April 2019]. |
| [7] | Comodo Security, “What is a Trojan Horse,” Comodo Security, [Online]. Available: https://enterprise.comodo.com/what-is-a-trojan-horse.php. [Accessed 02 April 2019]. |
| [8] | N. Devotta, “Computer Worm Definition and Types. How to Prevent them?,” Comodo Antivirus, 30 January 2019. [Online]. Available: https://antivirus.comodo.com/blog/computer-safety/computer-worm-definition/. [Accessed 02 April 2019]. |
| [9] | I. “What is the Morris Worm?,” Comodo Antivirus, 16 October 2018. [Online]. Available: https://antivirus.comodo.com/blog/comodo-news/morris-worm/. [Accessed 02 April 2019]. |
| [10] | MalwareBytes Labs, “Worm,” MalwareBytes, 09 June 2016. [Online]. Available: https://blog.malwarebytes.com/threats/worm/. [Accessed 02 April 2019]. |
| [11] | T. Scott, “Video: How The Self-Retweeting Tweet Worked: Cross-Site Scripting (XSS) and Twitter,” 11 June 2014. [Online]. Available: https://www.youtube.com/watch?v=zv0kZKC6GAM. [Accessed 02 April 2019]. |
| [12] | MalwareBytes Labs, “Ransom.WannaCrypt,” MalwareBytes, [Online]. Available: https://blog.malwarebytes.com/detections/ransom-wannacrypt/. [Accessed 02 April 2019]. |
| [13] | MalwareBytes Labs, “Ransomware,” MalwareBytes Labs, 13 December 2017. [Online]. Available: https://blog.malwarebytes.com/threats/ransomware/. [Accessed 02 April 2019]. |
| [14] | MalwareBytes, “All About Ransomware,” MalwareBytes, [Online]. Available: https://www.malwarebytes.com/ransomware/. [Accessed 02 April 2019]. |
| [15] | T. Marinho, “Ransomware encryption techniques,” Medium, 30 August 2018. [Online]. Available: https://medium.com/@tarcisiomarinho/ransomware-encryption-techniques-696531d07bb9. [Accessed 02 April 2019]. |
| [16] | Comodo, “What is a Keylogger: A Brief on a Dangerous and Malicious Tool,” Comodo , [Online]. Available: https://enterprise.comodo.com/what-is-a-keylogger.php. [Accessed 03 April 2019]. |
| [17] | Ardamax, “Ardamax Homepage,” Ardamax, [Online]. Available: https://www.ardamax.com/. [Accessed 03 April 2019]. |
| [18] | L. McKenzie, “Hacked From the Inside,” Inside HigherEd, 1 November 2017. [Online]. Available: https://www.insidehighered.com/news/2017/11/01/new-type-hacking-puts-professors-accounts-risk. [Accessed 03 April 2019]. |
| [19] | J. Scott, “Detecting malware through static and dynamic techniques,” NTT Security, 04 September 2017. [Online]. Available: https://technical.nttsecurity.com/post/102efk4/detecting-malware-through-static-and-dynamic-techniques. [Accessed 11 April 2019]. |
| [20] | M. Russinovich, “Strings v2.53,” 04 July 2016. [Online]. Available: https://technical.nttsecurity.com/post/102efk4/detecting-malware-through-static-and-dynamic-techniques. [Accessed 11 April 2019]. |
| [21] | P. OKane, S. Sezer and K. McLaughlin, “Obfuscation: The Hidden Malware,” *IEEE Security and Privacy,* vol. IX, no. 5, pp. 41-47, 2011. |
| [22] | Precision Sec, “Wanna Cry PCap SMB 445,” Precision Sec, 14 May 2017. [Online]. Available: https://precisionsec.com/wannacry-pcap-smb-445/. [Accessed 11 April 2019]. |
| [23] | Cuckoo, “Cuckoo API,” Cuckoo, [Online]. Available: https://cuckoo.readthedocs.io/en/2.0.6.2/usage/api/. [Accessed 04 April 2019]. |
| [24] | Cuckoo, “Cuckoo API Documentation,” [Online]. Available: http://docs.cuckoosandbox.org/en/latest/usage/api/. [Accessed 26 April 2019]. |
| [25] | Trend Micro, “Mediyes - Threat Encylopedia,” Trend Micro, 10 October 2012. [Online]. Available: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/troj\_mediyes.wc. [Accessed 25 April 2019]. |
| [26] | MalwareBytes Labs, “Zbot with legitimate applications on board,” MalwareBytes, 27 January 2017. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2017/01/zbot-with-legitimate-applications-on-board/. [Accessed 02 April 2019]. |
| [27] | Kaspersky, “Zeus Virus,” Kaspersky, [Online]. Available: https://usa.kaspersky.com/resource-center/threats/zeus-virus. [Accessed 02 April 2019]. |
| [28] | M. Kuhn and J. Kjell, “Applied Predictive Modelling,” in *Applied Predictive Modelling*, Springer, 2013, p. 70. |
| [29] | Ubuntu, “Tutorial: Ubuntu on Windows,” Canonical, [Online]. Available: https://tutorials.ubuntu.com/tutorial/tutorial-ubuntu-on-windows#0. [Accessed 02 April 2019]. |
| [30] | GitHub, “Atom,” GitHub, [Online]. Available: https://atom.io. [Accessed 02 April 2019]. |
| [31] | M. B. Kursa and W. R. Rudnicki, “Feature Selection with the Boruta Package,” *Journal of Statistical Software,* vol. 36, no. 11, pp. 1-4, 2010. |
| [32] | D. Bhalla, “Select Important Variables using Boruta Algorithm,” Data Science Central, 01 June 2017. [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/select-important-variables-using-boruta-algorithm. [Accessed 04 April 2019]. |
| [33] | D. Meyer, “e1071 Cran Repository,” R Project, 19 March 2019. [Online]. Available: https://cran.r-project.org/web/packages/e1071/index.html. [Accessed 04 April 2019]. |
| [34] | R Project, “A Short Introduction to the caret Package,” R Project, [Online]. Available: https://cran.r-project.org/web/packages/caret/vignettes/caret.html. [Accessed 04 April 2019]. |

# Table of Figures

[Figure 1. Example of Flash Player Trojan Webpage 2](#_Toc7391663)

[Figure 2. "Self-Retweeting Tweet" 4](#_Toc7391664)

[*Figure 3. Reveton Malware Screen* 5](#_Toc7391665)

[*Figure 4. Example of Scareware Ransomware* 5](#_Toc7391666)

[*Figure 5. Screenshot of WannaCry virus and background* 5](#_Toc7391667)

[Figure 6. Example KeyLogger Interfaces 6](#_Toc7391668)

[Figure 7. Cuckoo Web UI 9](#_Toc7391669)

[Figure 8. Example of Cuckoo Signature Alerts 10](#_Toc7391670)

[Figure 9. UML For the System 14](#_Toc7391671)

[Figure 10. Model Generation Process 15](#_Toc7391672)

[Figure 11. Unseen Sample Submission 15](#_Toc7391673)

[Figure 12. Example of Cuckoo Environment 16](#_Toc7391674)

[Figure 13. An example of the JSON reports collection. 16](#_Toc7391675)

[Figure 14. Example of API Call in JSON File 17](#_Toc7391676)

[Figure 15. Relation Diagram of GUI Screens 18](#_Toc7391677)

[Figure 16. Main Screen 18](#_Toc7391678)

[Figure 17. Model Statistics Screen 19](#_Toc7391679)

[Figure 18. Dataset Statistics Screen 19](#_Toc7391680)

[Figure 19. Preferences Screen 20](#_Toc7391681)

[Figure 20. New Dataset Options Screen 20](#_Toc7391682)

[Figure 21. Euclidean Distance Visualised 24](#_Toc7391683)

[Figure 22. kNN Algorithm Visualised 24](#_Toc7391684)

[Figure 23. Example of Install Core Installation Screen 26](#_Toc7391685)

[Figure 24. API Call Dataset Structure 30](#_Toc7391686)

[Figure 25. Example Sample Repository 31](#_Toc7391687)

[Figure 26. Parsing of JSON Reports Process 34](#_Toc7391688)

[Figure 27. Model Creation Process 35](#_Toc7391689)

[Figure 28. Unseen Sample Submission Process 36](#_Toc7391690)

[Figure 29. Implementation of GUI 39](#_Toc7391691)

# Table of Tables

[Table 1. Test Reference Breakdown 41](#_Toc7391692)

[Table 2. Dataset Statistics 41](#_Toc7391693)

[Table 3. Test Case 01 Confusion Matrix 41](#_Toc7391694)

[Table 4. Test Case 02 Confusion Matrix 41](#_Toc7391695)

[Table 5. Test Case 03 Confusion Matrix 42](#_Toc7391696)

[Table 6. Test Case 04 Confusion Matrix 42](#_Toc7391697)

# Table of Equations

[Equation 1. Equation for Cuckoo Score 10](#_Toc7391698)

[Equation 2. The formula for Euclidean Distance between Two Points 23](#_Toc7391699)

[Equation 3. f Score Equation 44](#_Toc7391700)