**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

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.

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

# Contents

[Declaration i](#_Toc5987604)

[Acknowledgements ii](#_Toc5987605)

[Abstract iii](#_Toc5987606)

[Contents iv](#_Toc5987607)

[1 Introduction and Problem Area 1](#_Toc5987608)

[1.1 Introduction 1](#_Toc5987609)

[1.2 Malware Types 1](#_Toc5987610)

[1.2.1 Trojan 2](#_Toc5987611)

[1.2.2 Worm 3](#_Toc5987612)

[1.2.3 Ransomware 4](#_Toc5987613)

[1.2.4 Keylogger 6](#_Toc5987614)

[1.3 Malware Analysis Techniques 7](#_Toc5987615)

[1.3.1 Static Analysis 7](#_Toc5987616)

[1.3.2 Dynamic Analysis 8](#_Toc5987617)

[1.3.3 Comparison 8](#_Toc5987618)

[1.3.4 Cuckoo Environment 9](#_Toc5987619)

[1.3.4.1 Cuckoo API 9](#_Toc5987620)

[1.3.4.2 Cuckoo Web 9](#_Toc5987621)

[1.3.4.3 Cuckoo Scoring System 9](#_Toc5987622)

[1.4 Machine Learning 10](#_Toc5987623)

[1.4.1 Supervised Learning 10](#_Toc5987624)

[1.4.2 Unsupervised Learning 11](#_Toc5987625)

[2 Solution Description and System Requirements 12](#_Toc5987626)

[2.1 Solution Description 12](#_Toc5987627)

[2.2 System Requirements 12](#_Toc5987628)

[3 Design 13](#_Toc5987629)

[3.1 Architectural Design 13](#_Toc5987630)

[3.2 User Interaction 13](#_Toc5987631)

[3.3 Software System Design 15](#_Toc5987632)

[3.3.1 Cuckoo Environment 16](#_Toc5987633)

[3.3.2 Parser 17](#_Toc5987634)

[3.3.3 Machine Learning 17](#_Toc5987635)

[3.4 Key Design Decisions 17](#_Toc5987636)

[3.4.1 Malware Chosen for Analysis 17](#_Toc5987637)

[3.4.1.1 Benign Files 17](#_Toc5987638)

[3.4.1.2 CryptoRansom 17](#_Toc5987639)

[3.4.1.3 InstallCore 17](#_Toc5987640)

[3.4.1.4 Mediyes 18](#_Toc5987641)

[3.4.1.5 Generic WinPE 18](#_Toc5987642)

[3.4.1.6 Zeus / ZBot 18](#_Toc5987643)

[3.4.2 Machine Learning Algorithm 18](#_Toc5987644)

[3.4.2.1 Chosen Method 18](#_Toc5987645)

[3.4.2.2 Cross Validation 18](#_Toc5987646)

[4 Implementation 19](#_Toc5987647)

[4.1 Use of Supporting Tools 19](#_Toc5987648)

[4.1.1 Languages Used 19](#_Toc5987649)

[4.1.2 Development Environment 19](#_Toc5987650)

[4.1.3 Version Control 19](#_Toc5987651)

[4.2 Use of Software Libraries 20](#_Toc5987652)

[4.2.1 Python Libraries 20](#_Toc5987653)

[4.2.2 R Libraries 20](#_Toc5987654)

[4.3 Key Implementation Decisions 20](#_Toc5987655)

[4.4 Important Functions and Algorithms 20](#_Toc5987656)

[4.5 Description of How Each Component Was Implemented 20](#_Toc5987657)

[5 Testing 21](#_Toc5987658)

[5.1 Testing Approach 21](#_Toc5987659)

[5.2 Testing Results 21](#_Toc5987660)

[5.2.1 Test Case 01 21](#_Toc5987661)

[5.2.2 Test Case 02 21](#_Toc5987662)

[5.2.3 Test Case 03 21](#_Toc5987663)

[5.2.4 Test Case 04 21](#_Toc5987664)

[6 System Evaluation and Experimental Results 22](#_Toc5987665)

[6.1 Evaluation of Results 22](#_Toc5987666)

[6.2 Technical Conclusion 22](#_Toc5987667)

[6.3 Personal Conclusion 22](#_Toc5987668)

[6.4 Future Work 22](#_Toc5987669)

[Appendices i](#_Toc5987670)

[User Manual i](#_Toc5987671)

[Meeting Minutes i](#_Toc5987672)

[Table of Figures ii](#_Toc5987673)

[References iii](#_Toc5987674)

# Introduction and Problem Area

## Introduction

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’s 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 doesn’t 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 specimen belongs to.

The goal of this report it to provide an in-depth look into how we could use machine learning in the future to classify malware. The report 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 specimen is. This process will be repeated to determine the optimal category definition.

## 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 descriptive 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’s 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 to 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 vulnerability in the Unix Sendmail program as well as weak passwords on systems as it’s 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 automatically record KeyStrokes, 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 on 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 statically analyse its functionality. 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 it’s 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 sub routine *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 wrote 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 3.3.1, 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 API

#### 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 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 [19].



Figure 7. Cuckoo Web UI

#### Cuckoo Scoring System

The Cuckoo scoring system is an 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).

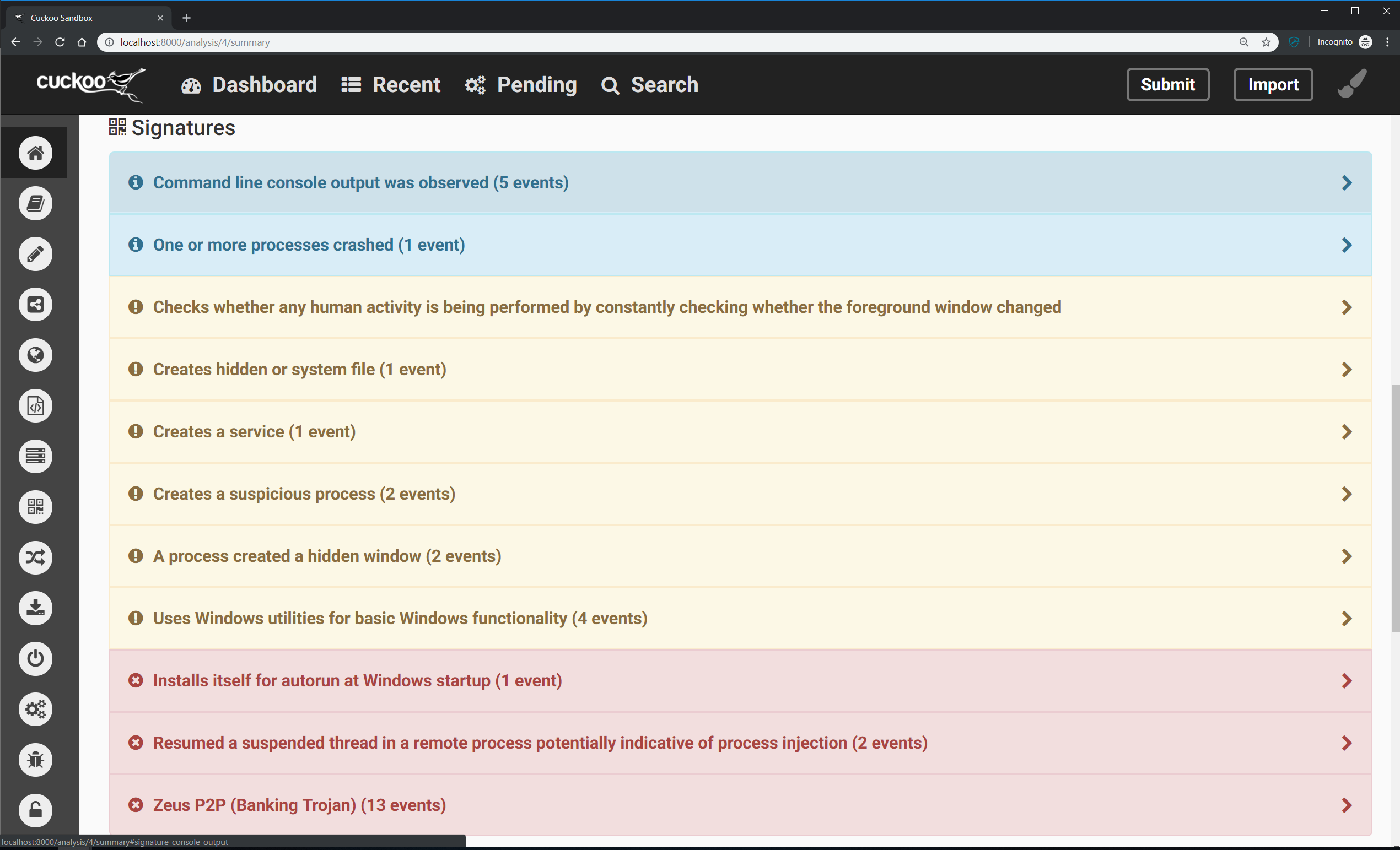


Figure 8 - Example of Cuckoo Signature Alerts

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:

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 finding out what malicious activities the sample is trying to do.

## Machine Learning

Machine Learning will form a fundamental part of this project. It allows us to form decisions based on previous data.

### Supervised Learning

### Unsupervised Learning

# Solution Description and System Requirements

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. To achieve this, a set of key aims, and requirements will have to be agreed, in this section those aims will be discussed.

## Solution Description

## System Requirements

# Design

## Architectural Design

## 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 sub section, 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 8 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 a unseen sample, to creating new Datasets and new Models.

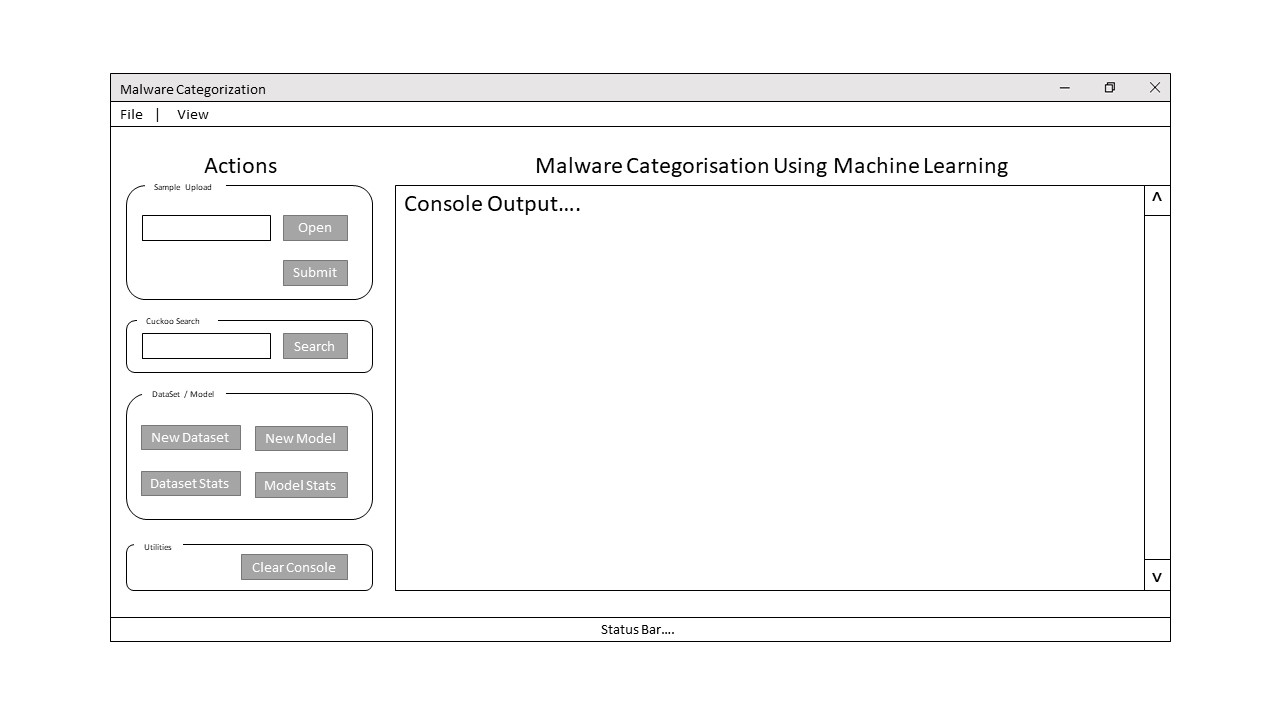


Figure 9 - Main Screen

Figure 9 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 10 - Model Statistics Screen

Figure 10 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 11 - Dataset Statistics Screen

Figure 11 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 12 - Preferences Screen

Figure 12 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.



Figure 13 - New Dataset Options Screen

## Software System Design

There are three main parts to the system that will be developed. 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.

### Cuckoo Environment

The first part is the Cuckoo environment that will perform dynamic analysis on the specimens, this will output a JSON report.



Figure 14. Example of Cuckoo Environment

The environment will consist of a

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.



Figure 15. 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 16. Example of API Call in JSON File

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

From there will we will parse the collection of JSON files into one. From there we will be able to apply a machine learning algorithm.

### Parser

The job of the Parser is to essentially make sense of the JSON files generated by the Cuckoo Environment; it will work in two parts

### Machine Learning

## Key Design Decisions

### Malware Chosen for Analysis

In Section Malware Types is a brief overview of the most prominent malware types that are affecting users today.

#### Benign Files

#### CryptoRansom

#### InstallCore

#### Mediyes



Figure 17. Example of Install Core Installation Screen

djdjdj

#### Generic WinPE

#### Zeus / ZBot

[20] [21]

### Machine Learning Algorithm

The Machine Learning Algorithm is one of the most important design decisions.

#### Chosen Method

#### Cross Validation

# Implementation

## Use of Supporting Tools

For this project, I needed to use plenty of Supporting Tools and Languages to develop this project. 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.

I decided on Python as the scripting language because of its ease of use as well as the fact that it was built with scripting in mind. For the Machine Learning portion of the project, I chose R, this is because of its advantages when manipulating large datasets as well as the inbuilt libraries that are provided with R (Use of Software Libraries).

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 [19]. 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 [20]. 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.

## Use of Software Libraries

### Python Libraries

Tkinter

### R Libraries

Caret [21]

E1071 [22]

Boruta [23]

## Key Implementation Decisions

## Important Functions and Algorithms

## Description of How Each Component Was Implemented

# Testing

## Testing Approach

### Black Box

### White Box

### Model Testing

To fully 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 is checked. In the Figure 18, a table can be seen that provides a Test Reference as well at what is enabled (Green) and Disabled (Red). The results to this testing can be seen in the Section 5.2, will 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* |  |  |

Figure - Testing Reference Table

## Testing Results

For the following tests, a dataset was used with the Families, Quantities, and Average Cuckoo Scores as seen in Figure 18.

|  |  |  |
| --- | --- | --- |
| **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** |

Figure - Dataset Statistics

This gives us a total training sample of 2810 malicious files, spread across Benign files and three major types of Malware with a catch all class in the form of Generic WinPE Viruses.

### Test Case 01

### Test Case 02

### Test Case 03

### Test Case 04

# System Evaluation and Experimental Results

## Evaluation of Results

## Technical Conclusion

## Personal Conclusion

## Future Work

# Appendices

## User Manual

## Meeting Minutes

# Table of Figures

[Figure 1. Example of Flash Player Trojan Webpage 2](#_Toc5964759)

[Figure 2. "Self-Retweeting Tweet" 3](#_Toc5964760)

[*Figure 3. Reveton Malware Screen* 4](#_Toc5964761)

[*Figure 4. Example of Scareware Ransomware* 5](#_Toc5964762)

[*Figure 5. Screenshot of WannaCry virus and background* 5](#_Toc5964763)

[Figure 6. Example KeyLogger Interfaces 6](#_Toc5964764)

[Figure 7. Cuckoo Web UI 9](#_Toc5964765)

[Figure 8 - Main Screen 12](#_Toc5964766)

[Figure 9 - Model Statistics Screen 12](#_Toc5964767)

[Figure 10 - Dataset Statistics Screen 13](#_Toc5964768)

[Figure 11 - Preferences Screen 13](#_Toc5964769)

[Figure 12 - New Dataset Options Screen 13](#_Toc5964770)

[Figure 13. Example of Cuckoo Environment 14](#_Toc5964771)

[Figure 14. An example of the JSON reports collection. 15](#_Toc5964772)

[Figure 15. Example of API Call in JSON File 15](#_Toc5964773)

[Figure 16. Example of Install Core Installation Screen 16](#_Toc5964774)

# 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] | Cuckoo, “Cuckoo API,” Cuckoo, [Online]. Available: https://cuckoo.readthedocs.io/en/2.0.6.2/usage/api/. [Accessed 04 April 2019]. |
| [23] | 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]. |
| [24] | Kaspersky, “Zeus Virus,” Kaspersky, [Online]. Available: https://usa.kaspersky.com/resource-center/threats/zeus-virus. [Accessed 02 April 2019]. |
| [25] | Ubuntu, “Tutorial: Ubuntu on Windows,” Canonical, [Online]. Available: https://tutorials.ubuntu.com/tutorial/tutorial-ubuntu-on-windows#0. [Accessed 02 April 2019]. |
| [26] | GitHub, “Atom,” GitHub, [Online]. Available: https://atom.io. [Accessed 02 April 2019]. |
| [27] | 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]. |
| [28] | 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]. |
| [29] | 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]. |