Final Year Project Report

MSci full unit – Final Report

Author Attribution of Binaries with Machine Learning

Moller Rodrigues

A report submitted in part fulfilment of the degree of

MSci Computer Science (Information Security)

Supervisor: Peter Komisarczuk

Department of Computer Science

Royal Holloway, University of London

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11260

Student Name: Moller Rodrigues

Date of Submission: 01/04/2022

Signature: Rodrigues

Table of Contents

Abstract

The focus of author attribution in Information technology is to attribute malicious code (malware) to the individual(s) that wrote it although this is just a subset of binary author attribution as it can also apply to other tasks such as plagiarism and intellectual property rights [1]. However, due to my interests in cyber security I will be focusing on malware author attribution (MAA). According to AV-Test Institute since 2013 malware has been spreading exponentially and there are now more than 1 billion malware programs ‘in the wild’ with their statistics showing that 560,000 new samples of malware are detected every day [2]. The effects of malware can be severe from denial of services to real financial loss for individuals.

Looking at these statistics it is clear, this is a concerning problem and the hopes are that one of the ways in which we can strengthen our defence against malware is through using author attribution. Knowing the author of a piece of malware using author attribution can help with the following, figuring out the functionality of the malware, Identifying the author of the malware to take legal action, Reversing the effects of the malware, classification of new malware samples, Aids in malware detection [3].

Introduction

Aim

The aim of this project is to be able to extract stylistic features from a dataset of malicious binaries using dynamic analysis techniques and using these extracted features along with an appropriate machine learning classifier to identify the authors of each binary to a good level of accuracy; with the goal of this project being to bel able to use this author attribution system as a tool against malware attacks.

Background

Although author attribution in the field of information technology and security is a relatively new concept, that is not quite the case in the field of literature.

Author attribution has long been around in the field of literature and is more commonly referred to as Stylometry, which is the evaluation of an author's style through the application of statistical analysis to a body of their work [4]. You may wonder what the purpose and benefits of author attribution are, well in the case of literature it aids in legal scenarios as to finding out who the owner of a piece of work is but also academic purposes; a famous example being the question of the authorship of Shakespeare's works.

So how does author attribution relate in the field of information technology and security?

Well, the focus of author attribution in Information technology is to attribute malicious code (malware) to the individual(s) that wrote it although this is just a subset of binary author attribution as it can also apply to other tasks such as plagiarism and intellectual property rights [2]. However, due to my interest's cyber security I will be focusing on malicious binary author attribution.

According to AV-Test Institute since 2013 malware has been spreading exponentially and there are now more than 1 billion malware programs 'in the wild' with their statistics showing that 560,000 new samples of malware are detected every day [5]. The effects of malware can be severe from denial of services to real financial loss for individuals.

Looking at these statistics it is clear, this is a concerning problem and the hopes are that one of the ways in which we can strengthen our defence against malware is through using author attribution. I'd like to echo a quote from the Art of War by Sun Tzu which I heard from a lecture regarding malware author attribution. The quote goes as follows, 'If you know the enemy and know yourself, you need not fear the result of a hundred battles'; from this quote we can ask ourselves the question, how well do we really know the enemy in the case of malware? And how does knowing the author help us? [6]

Well, knowing the author of a piece of malware ('the enemy') can help with the following:

1. Figuring out the functionality of the malware

   If we were able to identify the authors of malware samples and create a database of signatures, then if we came across a malware sample that we attributed to author x we can then look at our database and analyse previous malware written by author x which would greatly aid in identifying the functionality of the new malware sample.

2. Identifying the author of the malware in order to take legal action

   Malware attacks are quite different to traditional criminal attacks in the sense they are remote, digital and we cannot see a physical convict. As a result, it is often the case perpetrators of malware attacks get away 'Scott free' due to lack of evidence and traces to the origins of the malware.

3. Reversing the effects of the malware

   Knowing who wrote a piece of malware can help us understand what their signatures are and by profiling them when we come across a new malware sample which is attributed to that author we can use our previous knowledge to carry out actions to reverse the effects of the malware and neutralise it.

4. Classification of new malware samples

   It is often the case new malware samples are just old previously discovered malware repackaged and so if we knew the author of the new malware sample we can just look back to that authors previous malware samples and compare them.

5. Aids in malware detection

   Most intrusion detection systems are signature based and so author attribution of malware provides us with signatures to use for malware detection.

   [6]

However, when it comes to malware author attribution far more challenges arise compared to literature author attribution. Malware can take many forms but often we do not have access to the source code and are left with compiled binaries. Even if we did have access to the source code programming languages have a much more restricted vocabulary and syntax compared to a traditional language in the case of literature and this results in authors who write code/malware to have far less opportunity to express their unique individual style.

Dealing with binaries causes a new heap of challenges to arise, some of which are:

1. Compilation process

   The compilation process is the stage where a compiler reads source code and translates into a lower-level machine code which we call binaries. During the compilation process the compiler may replace many author specific attributes from the code such as comments, spacing, lines and its often the case that compilers optimize the source code therefore changing author specific implementations of certain instructions.

2. Obfuscation

   Obfuscation is the process of the concealment of data and is a technique widely used by malware authors to mask the functionality and details of their malware. Obfuscation is a problem when it comes to author attribution as it disguises the appearance of code through manipulation of [6]:

- Layout

  Adding/removing spaces, comments, lines.

- Control flow

  Adding layers of needless function calls to distort the control flow of the program making it more difficult for a researcher to analyse what is going on.

- Protection measures

  Malware authors can also add checks to their malware where if a researcher tries to dynamically analyse the malware in a tool like IDA then it causes the tools to crash therefore not allowing analysis of the malware.

Objectives and Structure of project

In this section I will describe my approach to implement a solution to malware author attribution (MAA) and outline the structure of this project document.

I will first start off with researching the problem area and in order to that I need to clearly state my research questions. Below are the research questions I want to find answers for whilst conducting my research in order to implement a solution to the problem of MAA.

Research Questions

- **What Is Author attribution and why is it beneficial?**
- **What are the current solutions for MAA?**
- **What datasets are used for MAA?**
- **What feature extraction tools are used for MAA?**
- **What machine learning classifiers are used for MAA?**

Once I have created reports for each of the topic areas surrounding these research questions, I will then write a chapter about the methodologies and software engineering principles I will use; the decisions to which will be informed by my prior research.

After deciding my design methodologies, I will begin analysing and design a solution for a MAA system. In this section I will provide content such as class diagram, flow charts, pseudocode, testing strategies etc.

Once I have completed the design stage, I will begin the implementation of my MAA system. In this section I will provide a log of the development of the system using screenshots of annotated code and outputs.

I will finally provide an evaluation of my project and describe the professional issues I came across throughout the process.

Summary of outline

- Chapter 1: Literature Review
- Chapter 2: Methodologies and software engineering
- Chapter 3: Design
- Chapter 4: Implementation
- Chapter 5: Professional issues
- Chapter 6: Evaluation

Chapter 1: **Literature Review**

1.1 Introduction

In this chapter I will be conducting a literature review on Author attribution of binaries with machine learning to gain a comprehensive knowledge of the problem area and inform my own implementation of a solution.

Before I set out to conduct a literature review, I thought it would be best to first formulate a set criteria to help me understand what questions I am looking to be answered while doing my research as then this criteria could easily be applied uniformly to all the papers I read to efficiently generate relevant notes.

These are the main topic areas of research I came up with:

1. Malware Analysis
2. Current solutions to Author Attribution using machine learning
3. Dataset for MAA (malware author attribution)
4. Feature extraction tools for MAA
5. Machine learning algorithms for MAA

1.2 Malware Analysis

1.2.1 Introduction

In this section I will be researching about malware analysis, but more in depth what malware is, what are malicious binaries, what malware analysis is and the challenges of malware analysis.

1.2.2 What is malware?

Malware can be formally defined as a malicious program which consists of a set of instructions that violates a site's security policy. Malware can take many forms such as worms, viruses, Trojan horses, ransomware, etc. The goals for a malware attack can vary from denial of service, financial loss or even in extreme cases harm to life. For more on malware and its taxonomy refer to source [7].

1.2.3 What are malicious binaries?

Binaries are files in binary format and are usually compiled to be platform specific. A malicious binary is a binary file containing a set of executable instructions which will violate a site's security policy.

Of course it is often the case a user unintentionally or without even knowing executes a malicious binary as there are many infection vectors by which a malicious binary can get onto a system and execute itself and attain persistence. Some examples of infection vectors are Exploiting vulnerable services over the network, Drive-by Downloads, Social Engineering, etc. For more on Infection vectors refer to source [8].

1.2.4 What is malware analysis?

Malware analysis is the study of malicious software using tool and techniques to understand the characteristics and behaviour of a sample of malware. There are two main types of malware analysis techniques, static analysis, and dynamic analysis.

1.2.5 Static Analysis

Static analysis is the analysis of malware without executing it. The goal of static analysis is to find technical identifiers such as file names, hashes, string such as IP addresses, domains and file header data to determine whether a file is malicious [9].

Below are some methods of static analysis:

- Detecting the file type

  Knowing the file type of a suspected binary and can provide with useful information such as how to read the file (the format of the file), details about the malware's target operating system (Windows/Linux) and architecture (32bit/64bit) [10].

  Example using Python's python-magic library [11]

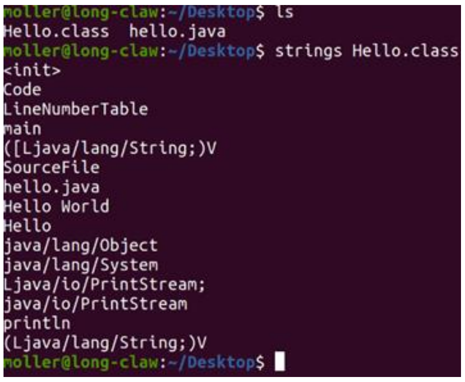
  ```
  # Import python-magic library
  import magic

  # Get file type
  magic.from_file("PATH_TO_FILE")
  ```

- Fingerprinting

  The process of fingerprinting a malware returns a unique identifier for that specific malware it is typically carried out by computing the hash of the file usually the MD5/SHA-1/SHA-256 hash [10]. Using this hash value, we can search online malware research databases to see if that specific malware has been found before and if there is any information about it. And of course, this can be used in Intrusion detection systems (IDS) to prevent malware from exploiting a system.

- Inspecting strings

  Strings are ASCII and unicode-printable sequences of characters included in a file. Extracting (meaningful) strings from a malware can provide hints about the program functionality and indicators associated with a suspect binary, or malware's authors themselves. For example, we can extract references to file names, URLs, domains, IPs, registry keys, etc [10]



```
moller@long-claw:~/Desktop$ ls
Hello.class  hello.java
moller@long-claw:~/Desktop$ strings Hello.class
<init>
Code
LineNumberTable
main
([Ljava/lang/String;)V
SourceFile
hello.java
Hello World
Hello
java/lang/Object
java/lang/System
Ljava/io/PrintStream;
java/io/PrintStream
println
(Ljava/lang/String;)V
moller@long-claw:~/Desktop$ 
```

Figure 1. Example of using strings Unix program on Linux OS produced by myself

1.2.6 Dynamic Analysis

Dynamic analysis is the analysis of malware during execution, it is carried out by executing malware in a safe environment ('sandbox') where it allows researchers to step through the execution and watch for changes and effects caused by the malware to the memory and state of the host system [9].

Below is a list of dynamic analysis techniques which I found through research whilst reading source [8]

- Function Call Monitoring

  Monitoring what functions are called by a program can help give us an overview of the behaviour of the program. To monitor what functions are being called by a program we can intercept these calls, and this is known as hooking.

  Hooking involves manipulating the program by adding in addition to the intended function a hooking function. When the intended function is called by the program it executes as expected but it also triggers the hooking function which performs the required analysis functionality such as logging the invocation of the intended function to a log file or analyse the input parameters of the intended function.

- Function Parameter Analysis

  Function parameter analysis involved tracking the actual values passed as parameters when a function is invoked by a program. Tracking the actual inputs and return values of functions can give us a detailed insight into the program's behaviour.

- Information Flow Tracking

  This technique involves tracking the propagation of targeted data throughout the execution of a program It essentially works by marking the data that needs to be monitored (tainted) with a corresponding value. Whenever the data is processed by the program its taint-label is propagated.

- Instruction Trace

  This involves extracting the sequence of machine instructions executed by a malicious program. This trace may provide important information not represented in a higher-level abstraction.

- Autostart Extensibility Points

  Autostart extensibility points (ASEPs) are mechanisms in the system that allow programs to automatically execute upon the operating system's boot or some other trigger. It is quite common for malware to make use of ASEPs to achieve persistence and therefore it is important to monitor ASEPs when analysing a malicious program.

For a more in-depth explanation of dynamic analysis and its techniques please refer to [8].

1.2.7 Challenges to malware analysis

Due to the nature of malware a series of complications arise whilst trying to analyse them. This is because one of the main characteristics of malware is to maintain a low presence that is to be undetectable and untraceable. Given this characteristic of malware requiring having a low presence attackers employ a range of countermeasures such as compiled binaries and obfuscation methods for both static and dynamic analysis. This section provides an overview on these countermeasures.

Compiled binaries

Most authorship attribution systems available are developed to extract stylistic features from source code, however in this case of malware it most often the case that we do not have access to the source code but instead are left with compiled binaries. The main problem with compiled binaries is that many of the techniques employed by extracting stylistic features from source code such as the developer’s unique style of programming, naming conventions, etc, are lost during the compilation stage. Therefore, we can apply techniques for binary analysis.

Static analysis obfuscation techniques

The main goal of obfuscation techniques for static analysis is to make it as hard as possible for analysts to read and understand the malicious binary. Here some of the main techniques used as detailed by [12].

- Encoding transformations
  - XOR

    This involves an attacker encoding their binaries by iterating through every byte of the file and applying the XOR operation to that byte and a randomly selected one byte key value.

  - Base64 encoding

    This involves an attacker encoding their binaries with a base64 encoder.

- Dead code insertion

  This involves adding obsolete and unnecessary code to the original source code in order to make the programs control flow more complex and difficult for analysts to understand.

- Instruction changes

  This is where attackers alter instruction codes for example by replacing the load operation with a set of semantically equivalent instructions that are more difficult to analyse statically [13].

- Packers

  Packers involve compressing the original source code therefore reducing the size of code. When a packed binary executable is executed, it unpacks itself.

I found most of these techniques whilst reading an article which can be found by referring to [12].

Dynamic analysis obfuscation techniques

As well as deploying static analysis techniques to make binaries more difficult to understand dynamic analysis also deploys techniques which enable a malicious program to detect whether it is being analysed by analysis tools such as an anti-virus software, debugger, reverse-engineering tool, etc. If the malicious binary detects that it is being analysed, then it would have some functionality in place to either crash the analysis tools to not allow further analysis or even perform clean-up and remove itself leaving no traces.

This section provides some of the main dynamic analysis evasion techniques employed by attackers.

- Sandbox Evasion

  When dynamically analysing a malicious program, we do so by executing it in a sandbox environment to avoid infecting our physical host machine. Sandboxes can be implemented as virtual machines where the malicious program can be executed in and the analyst can then easily revert the virtual machine to a snapshot before the malware was executed after they have conducted their analysis.

  The goal of sandbox evasion techniques is detecting whether the malicious program is being executed in such a sandbox. One example of a method would be to check if any hypervisor software is running on the system e.g. VMware, VirtualBox, etc.

- Debugging Evasion

  One method used by dynamic analysis is debugging. Debugging allows analysts to step through the execution of a malicious program to assess and track the state of values in memory and affects on the exploited system.

  The goal of debugging evasion is to detect whether a debugging program is running, and this can be done for example by querying the active running processes to see if there are any commonly known debuggers running.

1.3 Current solutions to Author Attribution using machine learning

1.3.1 Introduction

A more modern solution to author attribution of binaries is the use of machine learning. This solution works by extracting features from the binaries and using them as input to machine learning classifier which will attempt to identify the author of a sample of malware.

My research has concluded there are two main ways about implementing a solution using machine learning the first being extracting and using binary features from the malware samples (using static analysis techniques) and the second being using a sandbox tool to generate a report on each malware sample (using dynamic analysis techniques).

In this section I will provide an overview of these two possible approaches.

1.3.2 Binary features

This solution involves extracting binary features using static analysis techniques. Papers [1], [3], [14], [15], [16] focus on this method although only papers [1] and [3] focus on author attribution whilst the others are more on malware detection and classification.

Both papers [1] and [3] provide a good description of the binaries features that can be extracted for author attribution.

A brief overview of these features is provided below, please refer to [1] and [3] for more details.

Idioms

Idioms are short sequences of instructions. These are extracted as they can provide an author’s unique programming style.

Graphlets

Graphlets are 3-node subgraphs of the control flow graph that reflect the local structure of the program [1].

Supergraphlets

Supergraphlets are obtained by collapsing and merging neighbour nodes of the CFG [3].

Call graphlets

Call graphlets are graphlets containing only nodes with a call instruction [3].

N-grams

N-grams are a set of byte sequences or instruction level sequences. N-grams are short sequences of bytes of length N [3].

1.3.3 Sandbox reports

This approach involves using an automated sandbox tool to analyse the malware samples dynamically to generate a report on the malwares features and behaviour. This method is both more comprehensive in analysing malware but also safe as it does this in a sandbox environment.

[17] uses this sandbox report approach. Below is an overview of their approach.

[17] **approach**

[17] uses Cuckoo sandbox [18] to generate a report consisting of information about the malware using both static and dynamic analysis. More information on Cuckoo sandbox will be provided in a later section regarding feature extraction tools. They then used the reports as raw input for their machine learning classifier. For their classifier they used a DNN classifier.

1.4 Dataset for MAA

1.4.1 Introduction

A key ‘ingredient’ to implementing my author attribution system is a dataset of malware to use for feature extraction and to test and train a machine learning classifier. From my research I have concluded that there are two main types of datasets that are currently being used are malware datasets and for lack of a better word, non-malicious binaries.

In this section I describe the two main types of datasets and give an overview of what datasets are being used by current solutions.

1.4.2 Non-malicious datasets

Non-malicious datasets are datasets containing non-malicious binaries with labelled authors. Some solution uses these datasets due to the lack of labelled malware datasets. These non-malicious binaries are sourced from places such as student code competitions like and Google Code Jam (GCJ) [19] and code repositories like GitHub [20].

Google Code Jam (GCJ) [19]

What is GCJ?

GCJ is an international student competition run annually by Google the results and solutions of which are published publicly for anyone to download.

Why is it being used for binary author attribution (BAA)?

It is useful for BAA as all participants code similar programs and so this allows researchers to focus on author style rather than program functionality. The dataset also consists of diverse authors from all over the world.

What are the drawbacks?

The drawbacks to using this dataset for MAA is that the dataset does not contain malicious binaries and so the results may vary if a malware dataset was used instead.

GitHub [20]

What is GitHub?

GitHub is code repository site where developers can host and distribute their code.

Why is it being used for binary author attribution (BAA)?

GitHub contains a very large number of code repositories with many of them having open-source licenses and they also have labelled authors.

What are the drawbacks?

The drawbacks to using this dataset for MAA is that most of the code samples may not be malware/malicous.

1.4.3 Malware datasets

Malware datasets are datasets containing author labelled malicious binaries. In particular, I am interested in APT malware as APT groups create a large number of unique and sophisticated malware.

It was quite difficult for me to find an open-source dataset of APT labelled malware but I did manage to find an APT malware dataset created by "cyber-research" on GitHub [21].

APT Malware Dataset by "cyber-research" [21]

The dataset created by "cyber-research" on GitHub [21] contains 3594 malware samples which are attributed to 12 APT groups. The malware sample were collected using open-source threat intelligence report from multiple vendors and hashes that were collected from the threat intelligence reports were used to obtain the malware samples from VirusTotal.

Table 1: Dataset Characteristics Provided by "cyber-research" [21]

Country	APT Group	Family	Requested	Downloaded
China	APT 1		1007	405
China	APT 10	i.a. PlugX	300	244
China	APT 19	Derusbi	33	32
China	APT 21	TravNet	118	106
Russia	APT 28	*Bears*	230	214
Russia	APT 29	*Dukes*	281	281
China	APT 30		164	164
North-Korea	DarkHotel	DarkHotel	298	273
Russia	Energetic Bear	Havex	132	132
USA	Equation Group	Fannyworm	395	395
Pakistan	Gorgon Group	Different RATs	1085	961
China	Winnti		406	387
Total			**4449**	**3594**

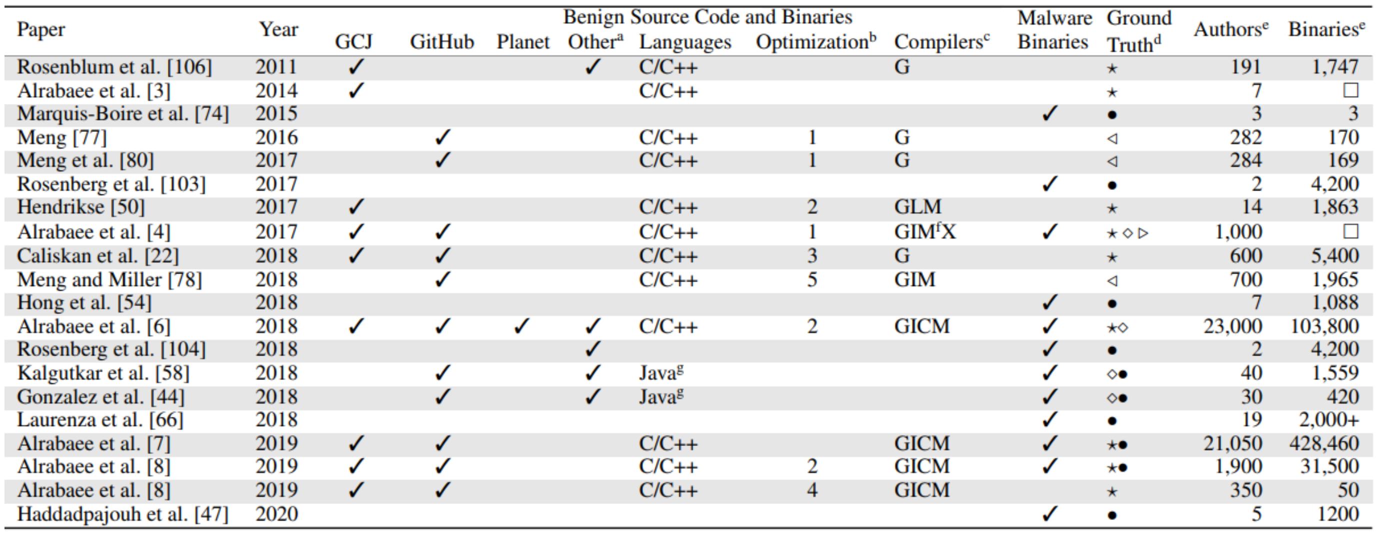
An issue however, about this dataset raised by [2] is that "cyber-research" do not state the method they used to label the malware hashes from the 29 sources and so researchers have no assurances on the validity of the label, for more details refer to [2].

1.4.4 Datasets used by current solutions

Whilst researching what datasets current solutions are using, I came across an extremely useful paper in regards to data and data modelling techniques for identifying authorship style in malicious binaries by [2].

[2] provides a good summary of the which datasets are being used in the form of a table which I refer to below. Note. The refs in the table relate to those found in [2].

Figure 2. A summary of datasets used by current solutions, credit to [2]



Paper	Year	Benign Source Code and Binaries							Malware	Ground	Authors[e]	Binaries[e]
		GCJ	GitHub	Planet	Other[a]	Languages	Optimization[b]	Compilers[c]	Binaries	Truth[d]		
Rosenblum et al. [106]	2011	✓			✓	C/C++		G		⋆	191	1,747
Alrabaee et al. [3]	2014	✓				C/C++				⋆	7	☐
Marquis-Boire et al. [74]	2015								✓	●	3	3
Meng [77]	2016		✓			C/C++	1	G		◁	282	170
Meng et al. [80]	2017		✓			C/C++	1	G		◁	284	169
Rosenberg et al. [103]	2017								✓	●	2	4,200
Hendrikse [50]	2017	✓				C/C++	2	GLM		⋆	14	1,863
Alrabaee et al. [4]	2017	✓	✓			C/C++	1	GIM[f]X	✓	⋆ ◇ ▷	1,000	☐
Caliskan et al. [22]	2018	✓	✓			C/C++	3	G		⋆	600	5,400
Meng and Miller [78]	2018		✓			C/C++	5	GIM		◁	700	1,965
Hong et al. [54]	2018								✓	●	7	1,088
Alrabaee et al. [6]	2018	✓	✓	✓	✓	C/C++	2	GICM	✓	⋆◇	23,000	103,800
Rosenberg et al. [104]	2018				✓				✓	●	2	4,200
Kalgutkar et al. [58]	2018		✓		✓	Java[g]			✓	◇●	40	1,559
Gonzalez et al. [44]	2018		✓		✓	Java[g]			✓	◇●	30	420
Laurenza et al. [66]	2018								✓	●	19	2,000+
Alrabaee et al. [7]	2019	✓	✓			C/C++		GICM	✓	⋆●	21,050	428,460
Alrabaee et al. [8]	2019	✓	✓			C/C++	2	GICM	✓	⋆●	1,900	31,500
Alrabaee et al. [8]	2019	✓	✓			C/C++	4	GICM		⋆	350	50
Haddadpajouh et al. [47]	2020								✓	●	5	1200

Summary of the stats

With regards to the finding in figure 2 I am especially interested in the papers conducted in 2018 and after. Given these more recent stats it is clear the majority of solutions use malware datasets over non-malicious datasets and the majority use datasets with authors between 10-40 with a sample size between 1000-4000.

1.5 Feature extraction tools for MAA

1.5.1 Introduction

In this section I will be reviewing the most popular tools used for feature extraction by current papers. Once again paper [2] provides a useful feature extraction tool comparison table which I will reference in this section as well.

1.5.2 Sandbox tools

Cuckoo Sandbox [18]

Cuckoo sandbox is a popular open-source automated malware analysis system which allows analysts to submit malware samples to the system and the system will execute the malware in a sandbox and analyse its characteristics and behaviour and output them in a report which is in JSON format.

1.5.3 Disassemblers

IDA Pro/Hex-Rays [22]

IDA Pro is a powerful disassembler and debugger. Some of its key features are reverse engineering binaries to source code, stepping through execution and its interactive features to help understand decompiled binaries such as control flow structures.

The downside to IDA Pro is that it is not free and requires a license to use, although there is a free version available but with limited capabilities.

Dyninst [23]

Dyninst is similar to IDA Pro as it is a disassembler which provides tools for binary instrumentation, analysis and modification. However, it appears to be used more in the problem of MAA and this is most likely because it is open-source.

1.5.4 Automated malware analysis tools

VirusTotal [24]

VirusTotal is an anti-virus solution which provides an API to submit malicious files to analyse generating a report on the results.

1.5.5 Feature extraction tools used by current solutions

Once again it is worth look at a table generated by [2] which compares what feature extraction tools are used by current systems. Please refer to [2] to view the original table. Note. The references in the table are relative to the references found in [2].

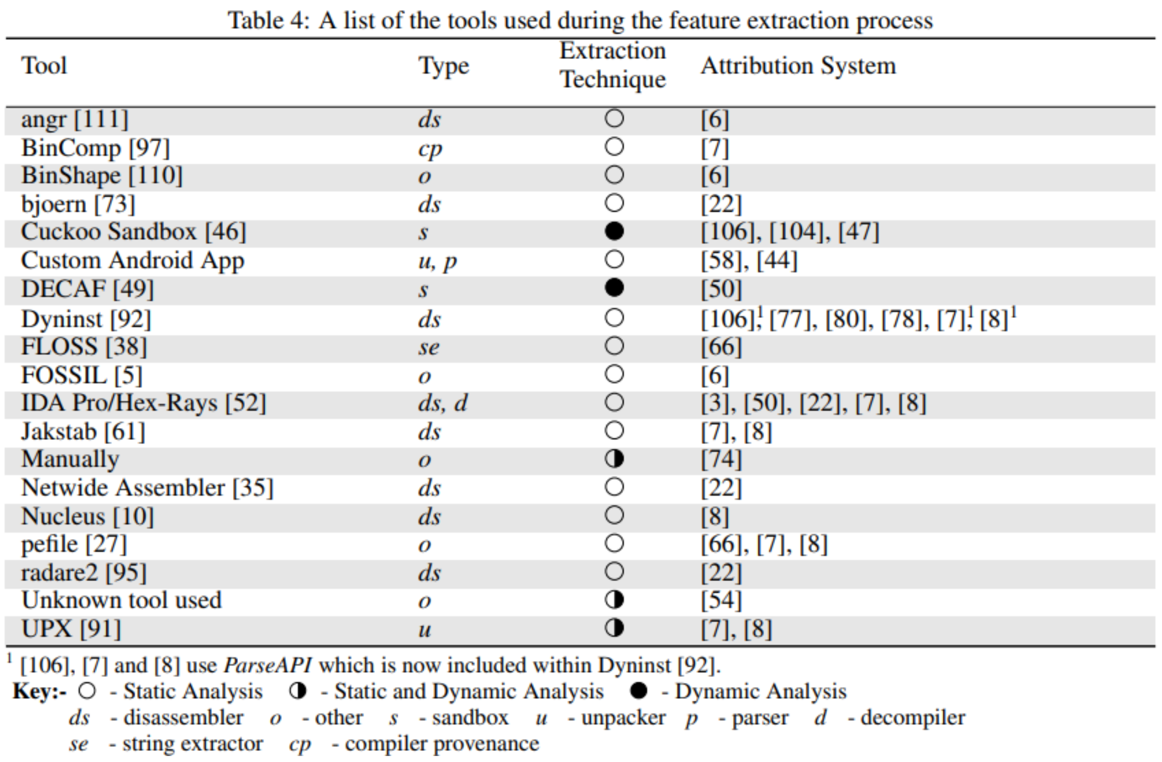


Table 4: A list of the tools used during the feature extraction process

Tool	Type	Extraction Technique	Attribution System
angr [111]	*ds*	○	[6]
BinComp [97]	*cp*	○	[7]
BinShape [110]	*o*	○	[6]
bjoern [73]	*ds*	○	[22]
Cuckoo Sandbox [46]	*s*	●	[106], [104], [47]
Custom Android App	*u, p*	○	[58], [44]
DECAF [49]	*s*	●	[50]
Dyninst [92]	*ds*	○	[106][1], [77], [80], [78], [7][1], [8][1]
FLOSS [38]	*se*	○	[66]
FOSSIL [5]	*o*	○	[6]
IDA Pro/Hex-Rays [52]	*ds, d*	○	[3], [50], [22], [7], [8]
Jakstab [61]	*ds*	○	[7], [8]
Manually	*o*	◑	[74]
Netwide Assembler [35]	*ds*	○	[22]
Nucleus [10]	*ds*	○	[8]
pefile [27]	*o*	○	[66], [7], [8]
radare2 [95]	*ds*	○	[22]
Unknown tool used	*o*	◑	[54]
UPX [91]	*u*	◑	[7], [8]

[1] [106], [7] and [8] use *ParseAPI* which is now included within Dyninst [92].

Key:- ○ - Static Analysis ◑ - Static and Dynamic Analysis ● - Dynamic Analysis
ds - disassembler *o* - other *s* - sandbox *u* - unpacker *p* - parser *d* - decompiler
se - string extractor *cp* - compiler provenance

Figure 3. Table comparing tools used by current systems, credit to [2]

From the table we can conclude that the most popular sandbox tool is Cuckoo Sandbox [18] and the most popular disassembler is Dyninst [23] closely followed by IDA Pro [22].

1.6 Machine learning algorithms for MAA

1.6.1 Introduction

The problem of identifying an author of a sample of malware is a classification one. In this section will provide an explanation of classification algorithms and how they can be used for my MAA system.

My core research for this topic stems from an online article which you can refer to [25] to view

1.6.2 Classification algorithms

What is classification in machine learning?

Classification is the process of categorizing a set of data into classes [25]. Classification is a form of supervised learning. It works by using a classifier which is an algorithm that maps the input data to a specific class. The process works by training the classifier with the labelled data samples and then testing the classifier to predict the class of the unlabelled test samples.

Some of the use cases of classification algorithms are Pattern recognition, image recognition, video recognition, Disease predictions, word classification, etc.

Classification Algorithms

- Logistic regression [25]

  What it is?

  A classification algorithm that uses one or more independent variable to determine an outcome. It will only have two possible outcomes. The goal of logistic regression is the find the best-fitting relationship between the dependent variable and a set of independent variables.

  What are the pros and cons?

  Pros are that it is useful to understand how a set of independent variables affect the outcome of the dependent variable.

  Cons are that it only works when the predicted variable is binary.

- Naïve Bayes Classifier [25]

  What it is?

  It is a classification algorithm based on Bayes's theorem which gives an assumption of independence among predictors.

  What are the pros and cons?

  Pros are that it requires a small amount of training data to estimate the necessary parameters.

  Cons are that they are known to be a bad estimator.

- K-Nearest Neighbour [25]

  What it is?

KNN stores all sample in training set in n-dimensional space where their classification is computed from a majority vote of the k nearest neighbour to each point (sample).

What are the pros and cons?

Pros are that it is simple to implement and quite efficient with large datasets.

Cons are that it uses more compute than other algorithms.

- Decision Tree [25]

  What it is?

  The decision tree algorithm builds the classification model in the form of a tree structure. It utilizes the if-then rules which are equally exhaustive and mutually exclusive in classification.

  What are the pros and cons?

  Pros are that decision trees make it easier to understand and visualise.

  Cons are that it can form complex trees which may not categorize efficiently.

- Random Forest [25]

  What it is?

  Random decision trees or random forest are an ensemble learning method for classification, regression, etc. It operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes or classification or mean prediction(regression) of the individual trees.

  What are the pros and cons?

  Pros are that it is more accurate than decision tress.

  Cons are that it is quite complex to implement.

- Artificial Neural Nets [25]

  What it is?

  A neural network consists of neurons that are arranged in layers, they take some input vector and convert it into an output. The process involves each neuron taking input and applying a function which is often a non-linear function to it and then passes the output to the next layer.

  What are the pros and cons?

  Pros are that it has a high tolerance to noise data.

  Cons are that it is hard to interpret

.

- Support Vector Machine [25]

  What it is?

The support vector machine is a classifier that represents the training data as points in space separated into categories by a gap as wide as possible. New points are then added to space by predicting which category they fall into and which space they will belong to.

What are the pros and cons?

Pros are that it is memory efficient.

Cons are that it does not directly provide probability estimates.

Classifier evaluation

In this section I will provide an overview of methods that can be used to evaluate classifiers.

- Holdout method

  This involves splitting the dataset into training set and test set in 80/20% split. The training set is then used to train the classifier and the test set is used to test the classifiers predictions.

- Cross-Validation

  Cros-validation is used to check for over-fitting it works by randomly partitioning the data set into k mutually exclusive subsets of the same size. One of these subsets is used for testing whilst the others are used to train the model. This process is applied for all k folds.

- Classification report

  Report produced by SVM classifiers.

  - Accuracy

    Ratio of correctly predicted samples.

  - F1-Score

    The weighted average of precision and recall.

  - Precision and recall

    Precision is the fraction of relevant instances among the retrieved instances.

    Recall is the fraction of relevant instances retrieved over the toal number of instances.

    These are used as a measure of relevance.

Chapter 2: **Methodologies and Software Engineering**

2.1 Introduction

In this chapter I will describe the methodologies and software engineering principles I have decided to use. Below is a breakdown of what I mean by methodologies and software engineering principles:

- What Design methodologies will I use?
- What testing strategies will I use?
- What development tools will I use?
- What Dataset will I use?
- What feature extraction tool will I use?
- What machine learning classifier will I use?

Throughout the rest of this chapter, I intend to answer these questions.

2.2 Methodologies

2.2.1 Introduction

In this section I will describe the design methodologies and software engineering prinicpples I will use to design and implement my solution.

2.2.2 Top-down modular design

I will be using a Top-Down strategy which is explained in [26]. It works by breaking down a system from its highest-level module and moves down towards its lowest-level module. The benefits of using such a strategy are that it breaks down the system into smaller and more manageable tasks.

2.2.3 Conventions

I will try to adhere to the latest Python conventions such as naming conventions, indentation, case as best as I can. Following the conventions of the programming language I am using will make it easier for other to read my code.

2.2.4 Comments

I will also be annotating my code with clear comments. This is a good princlple to follow as it will not only allow myself to remember what certain code does but it will also help others understand my code and thought process.

2.2.5 Test cases

The first testing strategy I will use is test cases. Test cases will allow me to conduct black box testing. Black box testing is where we are not concerned with the actual code but instead, we are concerned with the inputs and outputs of a process.

A test case will typically have the format of the test to be carried out, test steps, test data, expected results, actual results, and status of test.

2.2.6 Test driven Development (TDD)

Test driven development (TDD) is something in my second year from my Software Engineering module. TDD involves writing a unit test first running it and watching it fail as that feature has not been implemented yet. You then write code to add just enough functionality to pass the unit test. You then repeat this for each unit that is to be tested.

I am choosing to use TDD as it is known to greatly reduce the number of errors and bugs in a program.

2.3 Development tools

2.3.1 Introduction

In this section I will describe the development tools I will use to implement my MAA.

2.3.2 Programming language

I will be using Python [28] to program my solution. Python is a powerful lightweight cross platform scripting language.

I chose to use Python as I am quite familiar with it and It is quite popular so the majority of readers in this space will be able to understand my code. Furthermore, Python has an extensive set of libraries.

2.3.3 Development Environment

Firstly, I will be developing the system in an Ubuntu Server Virtual machine as this will act as the host VM for Cuckoo Sandbox. I have particularly chosen the Server version over the Desktop version because it is more lightweight and requires less resources as I am restricted with hardware resources.

Secondly, for my IDE I will be using Visual Studio Code [29]. Visual Studio Code is a powerful IDE with many features and tools and is widely used in the industry. One particular reason I am using Visual Studio Code over my personal favourite text editor Sublime Text is that it has a useful SSH tool which will allow me to SSH into my Ubuntu Server VM and develop from my physical host machine.

In order to run my code, I will be using the Python compiler via the command line.

2.3.4 Version control

As for version control, I will be using GIT to locally develop my system. This will allow me to keep track of changes I have made allowing me to go back or make releases and organise my code during development.

2.4 Dataset

2.4.1 Introduction

In this section I will be describing the dataset I have chosen to use for my MAA system.

2.4.2 APT Malware Dataset by “cyber-research” [21]

I have decided to use the APT malware dataset generated by “cyber-research” [21] for the following reasons:

- Open Database License
- Has a good sample size (3594)
- Samples are attributed to 12 different APT groups
- Easy to get a hold of (available on GitHub)

2.5 Feature Extraction tool

2.5.1 Introduction

In this section I will go over what feature extraction toll I have chosen and why.

2.5.2 Cuckoo Sandbox

I have chosen to use Cuckoo Sandbox for the following reasons:

- Used by multiple current solutions (proven to work)
- Automated sandbox environment so it is safe to analyse malware
- Generates detailed report in JSON format which can be used as input for machine learning classifier.
- Open source

2.6 Machine learning classifier

2.6.1 Introduction

In this section I will go over what machine learning classifier I have chose to use and why.

2.6.2 Random Forest Classifier (RFC)

The main classifier I have chosen to use is RFC, this is because:

- It is quick at training and predicting using large datasets.
- It is robust to outliers and non-linear data.
- It has methods to handle unbalanced data.
- It is a low bias, moderate variance model

2.6.3 Further remarks

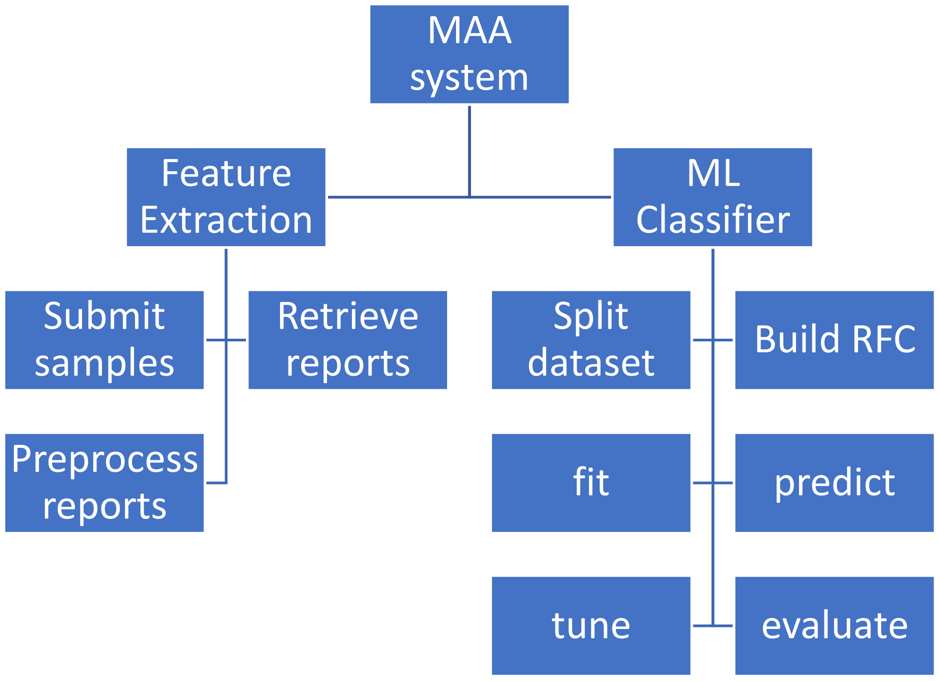
If I have time I would also like to implement additional classifiers like logistic regression and Naïve Bayes Classifier.

Chapter 3: **Design**

3.1 Introduction

Having a good design methodology makes easier to implement software and saves a lot of time in the long term in regards to refactoring and debugging. In this section I will design a solution for the MAA system.

3.2 Top-down modular design



I have broken my MAA system into two main submodules Feature Extraction and ML Classifier. Below is a breakdown of each module.

- Feature Extraction

  This module is responsible for extracting features from the malware samples.

  My initial plan to do this is by iterating through every sample in the dataset and submitting it to cuckoo sandbox to generate a report. The implementation of this will require a function to submit reports, retrieve reports and pre-process the reports to use as inputs for the machine learning classifier.

- ML Classifier

  This module is responsible for implementing the Random Forest Classifier (RFC) which will be used to predict the author of a malware sample.

The implementation of this module will involve a function to split the dataset into a training set and test set, Initialising the RFC, fit the data to the model, use the model to make predictions, tune parameters accordingly and finally evaluate the model.

Chapter 4: Implementation

4.1 Introduction

In this section I will log each step of the development of my MAA system with the aid of screenshots of commented code and output.

4.2 Setting up a virtual machine

I will implement the solution in an Ubuntu Server 18.04.6 VM. This is because a host VM is needed to Install Cuckoo Sandbox and it will provide a clean and secure environment to develop my solution.

Below are the steps I took to set up my host VM.

1. Installed VMware Workstation 16 non-commercial from [30] to act as my hypervisor.
2. Downloaded an Ubuntu Server 18.04.6 ISO from [31].
3. Created a new VM in VMware workstation with appropriate hardware specs to support Cuckoo Sandbox (8GB RAM, 4 Core CPU).
4. Installed the Ubuntu Server 18.04.6 ISO on the VM which I have named Cuckoo-Host.
5. Patched and installed required development tools.
6. SSH into Cuckoo-Host VM via Visual Studio Code.

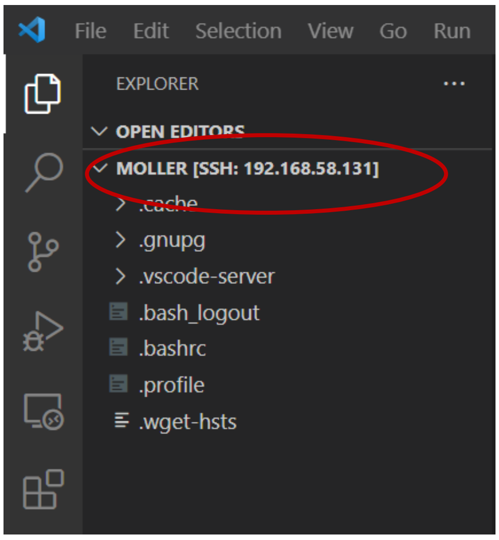


Figure 4. Visual Studio Code SSH session with Cuckoo-Host VM

Now the Cuckoo-Host VM is ready for installation of Cuckoo Sandbox and development of my MAA system.

4.3 Installing Cuckoo Sandbox

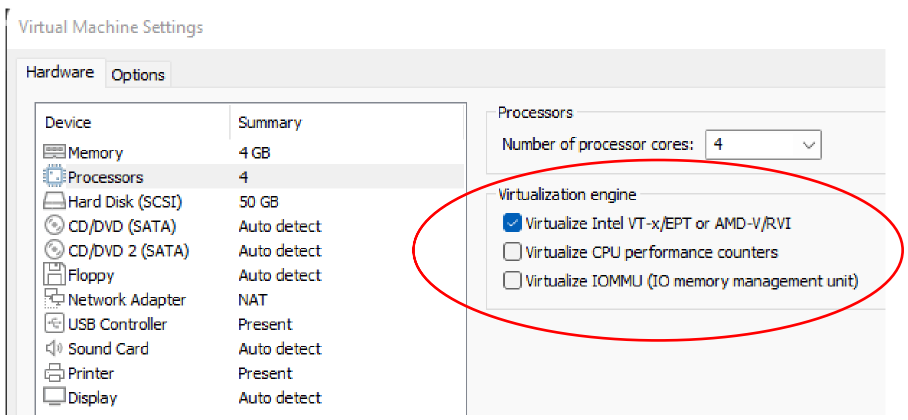
4.3.1 Introduction

In this section I will be following the official documentation given by Cuckoo to install Cuckoo Sandbox. The documentation can be found at [32].

Below are the steps I took to Install Cuckoo Sandbox.

4.3.2 Enable Virtualization for host VM

The Cuckoo-Host VM requires virtualization to be enabled as we will be installing a Guest Windows VM within the host VM in order to act as the sandbox to execute and analyse the malware.



Device	Summary
Memory	4 GB
Processors	4
Hard Disk (SCSI)	50 GB
CD/DVD (SATA)	Auto detect
CD/DVD 2 (SATA)	Auto detect
Floppy	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Processors

Number of processor cores: 4

Virtualization engine

- [✓] Virtualize Intel VT-x/EPT or AMD-V/RVI
- [ ] Virtualize CPU performance counters
- [ ] Virtualize IOMMU (IO memory management unit)

Figure 5. Enabling Virtualization for Ubuntu Host VM

4.3.3 Installing Dependencies

I will now install the dependencies required for Cuckoo Sandbox as described by the documentation [32].

- Installing Python dependencies (Cuckoo Host components are written completely in Python):

  ```
  $ sudo apt update
  $ sudo apt upgrade
  $ sudo apt-get install python python-pip python-dev libffi-dev
  libssl-dev -y
  $ sudo apt-get install python-virtualenv python-setuptools -y
  $ sudo apt-get install libjpeg-dev zlib1g-dev swig -y
  ```

- Installing MongoDB (required to use web interface)

  ```
  $ sudo apt-get install mongodb -y
  ```

- Installing PostgreSQL as database recommended by Cuckoo Sanndbox:

  ```
  $ sudo apt-get install postgresql libpq-dev -y
  ```

- Installing VirtualBox (For Guest VM):

  ```
  $ sudo apt install virtualbox -y
  ```

- Installing tcpdump (dump network activity performed by malware during execution):

```
$ sudo apt-get install tcpdump apparmor-utils -y
```

- Creating a cuckoo user which will run Cuckoo.

```
$ sudo adduser --disabled-password --gecos "" cuckoo
```

- Giving cuckoo user the required privileges needed to use tcpdump

```
$ sudo groupadd pcap
$ sudo usermod -a -G pcap cuckoo
$ sudo chgrp pcap /usr/sbin/tcpdump
$ sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
```

- Installing M2Crypto required by Cuckoo Sandbox

```
$ sudo pip install m2crypto
```

- Add cuckoo user to “vboxusers” group so They can run VirtualBox

```
$ sudo usermod -a -G vboxusers cuckoo
```

- Switch to cuckoo user

```
$ sudo su cuckoo
```

- Cuckoo recommends installing Cuckoo in a virtualenv (creating and activating venv)

```
$ sudo apt-get update && sudo apt-get -y install virtualenv

$ sudo apt-get -y install virtualenvwrapper

$ echo "source /usr/share/virtualenvwrapper/virtualenvwrapper.sh" >>
~/.bashrc

$ sudo apt-get -y install python3-pip

$ pip3 completion --bash >> ~/.bashrc

$ pip3 install --user virtualenvwrapper

$  echo  "export  VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3"  >>
~/.bashrc

$ echo "source ~/.local/bin/virtualenvwrapper.sh" >> ~/.bashrc

$ export WORKON_HOME=~/.virtualenvs

$ echo "export WORKON_HOME=~/.virtualenvs" >> ~/.bashrc

$ echo "export PIP_VIRTUALENV_BASE=~/.virtualenvs" >> ~/.bashrc

$ mkvirtualenv -p python2.7 cuckoo-venv
```

I have now installed virtualenv and created and activated a virtualenv called cuckoo-venv.

```
(cuckoo-venv) cuckoo@moller-dev:/opt$ 
```

Figure 6. Screenshot showing active cuckoo-venv virtual environment

- Upgrade setuptools in cuckoo-venv

```
$ pip install -U pip setuptools
```

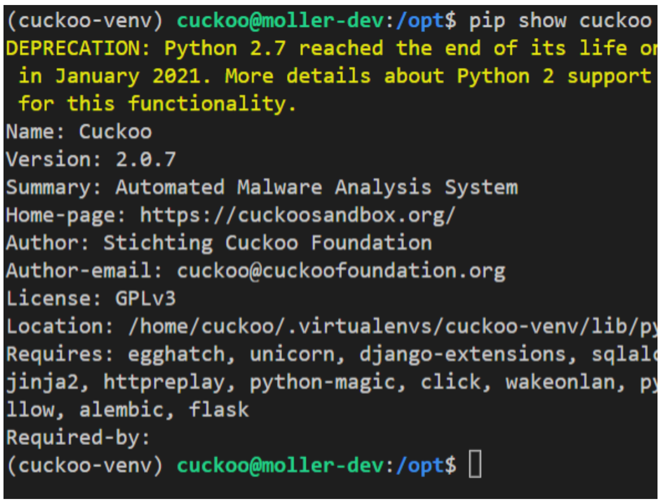
4.3.4 Installing Cuckoo

Now that all the dependencies are installed and the virtual environment is set up we can install Cuckoo Sandbox via pip within the cucoo-venv.

- Installing Cuckoo in cuckoo-venv

```
$ pip install -U cuckoo
```

Cuckoo Sandbox has now successfully been installed in my Python virtual environment called "cuckoo-venv"



```
(cuckoo-venv) cuckoo@moller-dev:/opt$ pip show cuckoo
DEPRECATION: Python 2.7 reached the end of its life o
 in January 2021. More details about Python 2 support
 for this functionality.
Name: Cuckoo
Version: 2.0.7
Summary: Automated Malware Analysis System
Home-page: https://cuckoosandbox.org/
Author: Stichting Cuckoo Foundation
Author-email: cuckoo@cuckoofoundation.org
License: GPLv3
Location: /home/cuckoo/.virtualenvs/cuckoo-venv/lib/p
Requires: egghatch, unicorn, django-extensions, sqlal
jinja2, httpreplay, python-magic, click, wakeonlan, p
llow, alembic, flask
Required-by:
(cuckoo-venv) cuckoo@moller-dev:/opt$ 
```

Figure 7. Screenshot Showing the successful Installation of Cuckoo Sandbox

4.4 Setting up the Guest VM

4.4.1 Introduction

Cuckoo Sandbox requires an additional guest VM running within the Host VM on which Cuckoo is running. This is because that guest VM will be used to automate the process of executing and analysing the malware samples in a secure manner. After a sample of malware is executed and analysed the guest VM is reverted to an earlier stable snapshot allowing for analysis of the next malware sample. I will be using Windows 7 as the OS for the guest VM as this is recommended by cuckoo and yields the best results.

4.4.2 Steps taken to setup Guest VM

- Downloading Windows 7 ISO image provided by Cuckoo [32]

```
$ sudo wget https://cuckoo.sh/win7ultimate.iso
```

- Mounting downloaded ISO

```
$ sudo mkdir /mnt/win7
$ sudo chown cuckoo:cuckoo /mnt/win7/
$ sudo mount -o ro,loop win7ultimate.iso /mnt/win7
```

- Installing vmcloak

  Vmcloak allows us to take and load snapshots rapidly

  Steps to install vmcloak:

```
$ sudo apt-get -y install build-essential libssl-dev libffi-
dev python-dev genisoimage
$ sudo apt-get -y install zlib1g-dev libjpeg-dev
$ sudo apt-get -y install python-pip python-virtualenv python-
setuptools swig
$ pip install -U vmcloak
```

- Creating VM using vmcloak, VirtualBox and the Windows 7 Image

```
$ vmcloak-vboxnet0
$ vmcloak init --verbose --win7x64 win7x64base --cpus 2 --
ramsize 2048
$ vmcloak clone win7x64base win7x64cuckoo
$ vmcloak list deps
$ vmcloak install win7x64cuckoo ie11
$ vmcloak snapshot --count 1 win7x64cuckoo 192.168.56.101
$ vmcloak list vms
```

- I then configured the iptables rules cuckoo conf

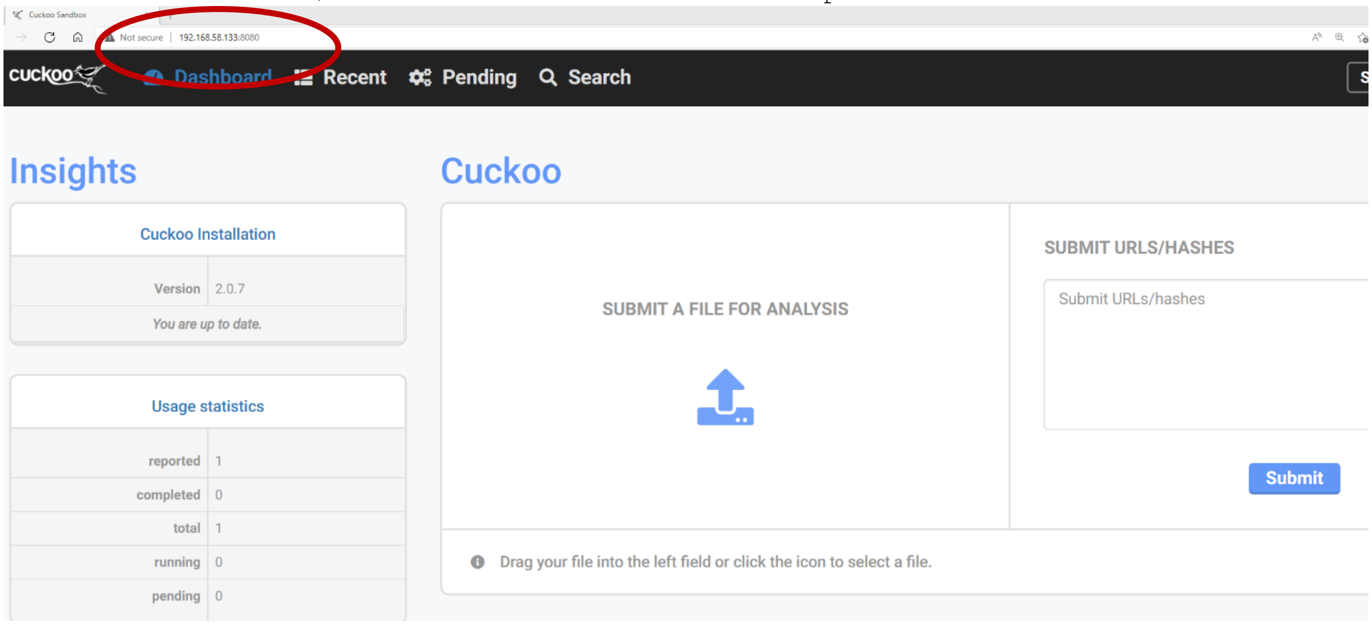
```
$ sudo sysctl -w net.ipv4.conf.vboxnet0.forwarding=1
$ sudo sysctl -w net.ipv4.conf.*your interface name*.forwarding=1

$ sudo iptables -t nat -A POSTROUTING -o *your interface name* -s
192.168.56.0/24 -j MASQUERADE
$ sudo iptables -P FORWARD DROP
$ sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j
ACCEPT
$ sudo iptables -A FORWARD -s 192.168.56.0/24 -j ACCEPT
```

The Guest VM is now ready to run Cuckoo

4.4.3 Initialising Cuckoo

Cuckoo is now ready to be launched here are the commands to do so.

```
$ cuckoo init
$ cuckoo community
      $ cuckoo rooter --sudo --group cuckoo

      $ cuckoo web --host 127.0.0.1 --port 8090
```



Version	2.0.7

You are up to date.

reported	1
completed	0
total	1
running	0
pending	0

Drag your file into the left field or click the icon to select a file.

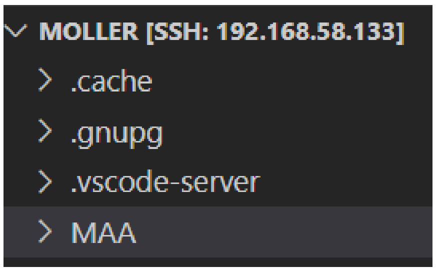
Figure 8. Screenshot of Cuckoo Sandbox up and running on host VM

4.5 Setting up project directory

4.5.1 Introduction

Now that host VM and guest VM are all set up and Cuckoo is up and running I can begin development of the MAA system. The first step to doing so is creating a project structure and directory.

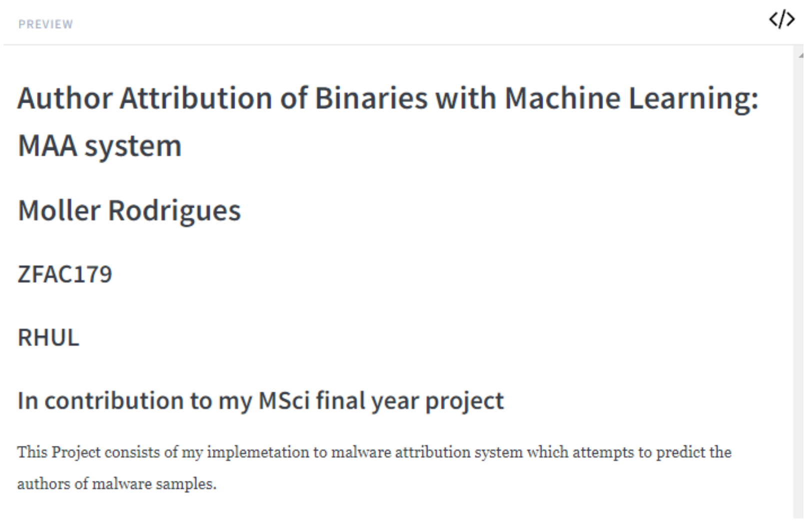
4.5.2 Creating Project directory



```
MOLLER [SSH: 192.168.58.133]
  > .cache
  > .gnupg
  > .vscode-server
  > MAA
```

Figure 9. Creating directory to develop project in

4.5.3 Creating project readme


PREVIEW

Author Attribution of Binaries with Machine Learning: MAA system

Moller Rodrigues

ZFAC179

RHUL

In contribution to my MSci final year project

This Project consists of my implemetation to malware attribution system which attempts to predict the authors of malware samples.

Figure 10. Preview of initial project readme.md

4.6 Submitting malware samples and retrieving reports

4.6.1 Introduction

I have already downloaded the APT malware dataset from [21] and so I am now ready to submit the samples to my Cuckoo Sandbox and retrieve results.

4.6.2 Submitting samples

“cyber-research” [21]the same people that created the APT malware dataset I am using also have created scripts to unzip and submit malware samples to Cuckoo. This is done using the API provided by Cuckoo and implemented in Python. The code to which can be found in “submit_samples.py” which can be found at their GitHub repository [33].

- I had to alter the code to use my Cuckoo API (line 15)

```
BASE_URL = 'http://192.168.58.133:8090'
```

4.6.3 Retrieving samples

Again I will be using the script provided by “cyber-research” to retrieve the reports from my Cuckoo Sandbox. This script is called “retrieve_reports.py” and can be found at their GitHub repo [33].

- I had to alter the code to use my Cuckoo API (line 9)

```
BASE_URL = 'http://192.168.58.133:8090'
```

4.6.4 Running the code

As expected when I ran the code it the script unzipped the zip file containing the malware samples and iteratively submitted them to my Cuckoo Sandbox to be analysed.

4.6.5 Pre-processing the reports

In order to use the report with a machine learning classifier I need to do some pre-processing to the reports. The reports come in JSON format so I created a Python script to read the JSON file and convert it a bag of words so that it can be used as input for a machine learning classifier.

The code to read and pre-process the JSON reports can be found in the preprocess_report.py file.

4.6.6 Problem with this approach

Whilst I was waiting for the malware samples to be analysed I realised that it was just taking to long. I even tried doing it again with a reduced sized dataset, but the analyses and retrieval of report was too time consuming, and I did not have the time to do this for a substantial sized dataset nor the hardware resources.

4.7 An alternative solution for the dataset

4.7.1 Introduction

As mentioned in the previous section this current approach to submitting samples to Cuckoo and retrieving reports is taking too long. Because of this I started searching for another malware dataset and preferably one which has already extracted the features from the malware samples. This is because I had already seen the process of feature extraction through the use of Cuckoo but now I just need a large enough dataset of the extracted features so I can implement a machine learning classifier.

4.7.2 A dataset of extracted features from the APT malware dataset samples

Fortunately, I managed to find a pre-processed dataset of extracted features from malware samples on GitHub with an MIT Licence.

The project can be found here [34]. It was carried out by a user called “PritomDas” using the same malware dataset produced by “cyber-research” [21]

In order to generate the final pre-processed dataset [34] submitted the hashes of each sample from the APT malware dataset provided by “cyber-research” [21] to VirusTotal [24] using their web API to generate reports. They then extracted these following attributes from each report:

- Pe-resource-langs
- Imports
- Pe-entry-points

The consolidated all these extracted attributes from all the malware samples in one single csv file.

They then performed some pre-processing on this dataset by removing brackets, commas, changed all the characters to the same case (UPPER CASE) to avoid differentiating between samples due to this feature.

4.7.3 Why the approach from [34] is a good alternative

The approach from [34] is a good alternative for the main reason that it is much quicker process to extract features from the malware samples. Instead of executing each malware sample and dynamically analysing them this approach simply takes the hash values for each malware sample which are populated in a single csv file provided by “cyber-research” [21] and submits them to VirusTotal [24] which simply check against several AV solutions to give a result.

This approach also saves time in the regards that you do not have to setup a sandbox like Cuckoo or any VM but instead just make API requests to a web server.

The drawback however to using this dataset are that only static analysis techniques are used to extract features.

4.7.4 Summary

So in summary I have decided to use the new dataset provided by "PritomDas" which can be found at and downloaded at [34].

4.8 Implementing the MAA system

4.8.1 Introduction

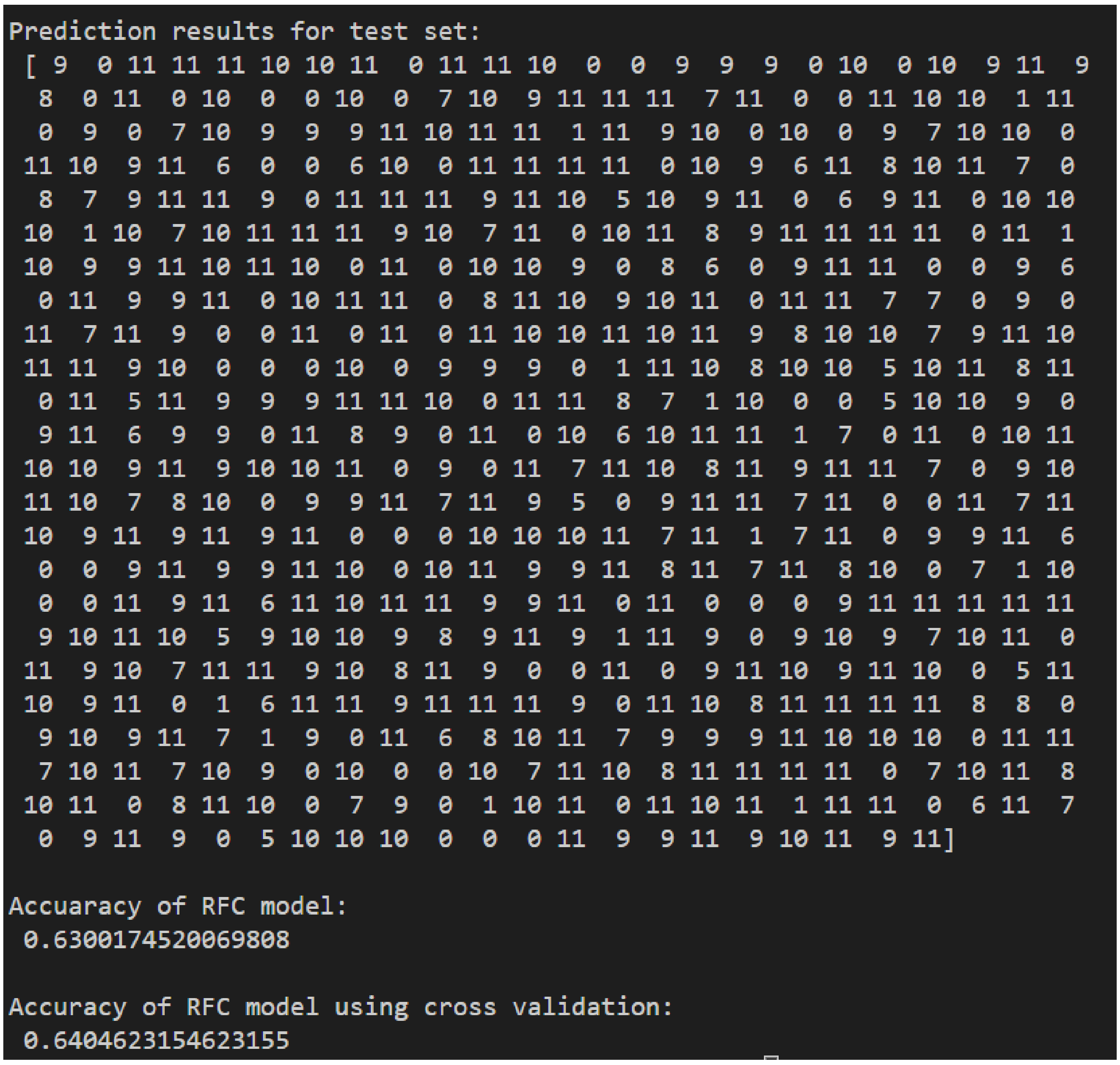
Now that I have a dataset of extracted features, credit to [34]. I can begin implementing my MAA system. Also now that I do not have to wait for the lengthy process of extracting, submitting, analysing and retrieving reports from my previous approach I have decided to implement additional classfiers along with RFC and compare them. I used sources [35] and [34] to gain some understanding on how to implement RFC in Python.

Below I outline the steps I took to implement the MAA system.

4.8.2 Implementation steps

1. Created main.py
2. Imported necessary libraries
3. Loaded dataset (Appendix A. final_preprocessed_dataset.csv) using pandas dataframe
4. Removed non-numerical data from the dataset using pandas drop method
5. Split the dataset into training and testing using Sklearn's train_test_split method
6. Built RFC classifier
7. Fit the training sets to the model
8. Used model to predict labels for test set
9. Evaluated the models performance

4.8.3 RFC model results



```
Prediction results for test set:
 [ 9  0 11 11 11 10 10 11  0 11 11 10  0  0  9  9  9  0 10  0 10  9 11  9
   8  0 11  0 10  0  0 10  0  7 10  9 11 11 11  7 11  0  0 11 10 10  1 11
   0  9  0  7 10  9  9  9 11 10 11 11  1 11  9 10  0 10  0  9  7 10 10  0
  11 10  9 11  6  0  0  6 10  0 11 11 11 11  0 10  9  6 11  8 10 11  7  0
   8  7  9 11 11  9  0 11 11 11  9 11 10  5 10  9 11  0  6  9 11  0 10 10
  10  1 10  7 10 11 11 11  9 10  7 11  0 10 11  8  9 11 11 11 11  0 11  1
  10  9  9 11 10 11 10  0 11  0 10 10  9  0  8  6  0  9 11 11  0  0  9  6
   0 11  9  9 11  0 10 11 11  0  8 11 10  9 10 11  0 11 11  7  7  0  9  0
  11  7 11  9  0  0 11  0 11  0 11 10 10 11 10 11  9  8 10 10  7  9 11 10
  11 11  9 10  0  0  0 10  0  9  9  9  0  1 11 10  8 10 10  5 10 11  8 11
   0 11  5 11  9  9  9 11 11 10  0 11 11  8  7  1 10  0  0  5 10 10  9  0
   9 11  6  9  9  0 11  8  9  0 11  0 10  6 10 11 11  1  7  0 11  0 10 11
  10 10  9 11  9 10 10 11  0  9  0 11  7 11 10  8 11  9 11 11  7  0  9 10
  11 10  7  8 10  0  9  9 11  7 11  9  5  0  9 11 11  7 11  0  0 11  7 11
  10  9 11  9 11  9 11  0  0  0 10 10 10 11  7 11  1  7 11  0  9  9 11  6
   0  0  9 11  9  9 11 10  0 10 11  9  9 11  8 11  7 11  8 10  0  7  1 10
   0  0 11  9 11  6 11 10 11 11  9  9 11  0 11  0  0  0  9 11 11 11 11 11
   9 10 11 10  5  9 10 10  9  8  9 11  9  1 11  9  0  9 10  9  7 10 11  0
  11  9 10  7 11 11  9 10  8 11  9  0  0 11  0  9 11 10  9 11 10  0  5 11
  10  9 11  0  1  6 11 11  9 11 11 11  9  0 11 10  8 11 11 11 11  8  8  0
   9 10  9 11  7  1  9  0 11  6  8 10 11  7  9  9  9 11 10 10 10  0 11 11
   7 10 11  7 10  9  0 10  0  0 10  7 11 10  8 11 11 11 11  0  7 10 11  8
  10 11  0  8 11 10  0  7  9  0  1 10 11  0 11 10 11  1 11 11  0  6 11  7
   0  9 11  9  0  5 10 10 10  0  0  0 11  9  9 11  9 10 11  9 11]

Accuaracy of RFC model:
 0.6300174520069808

Accuracy of RFC model using cross validation:
 0.6404623154623155
```

Figure 11. Output of MAA program displaying the results of the RFC model

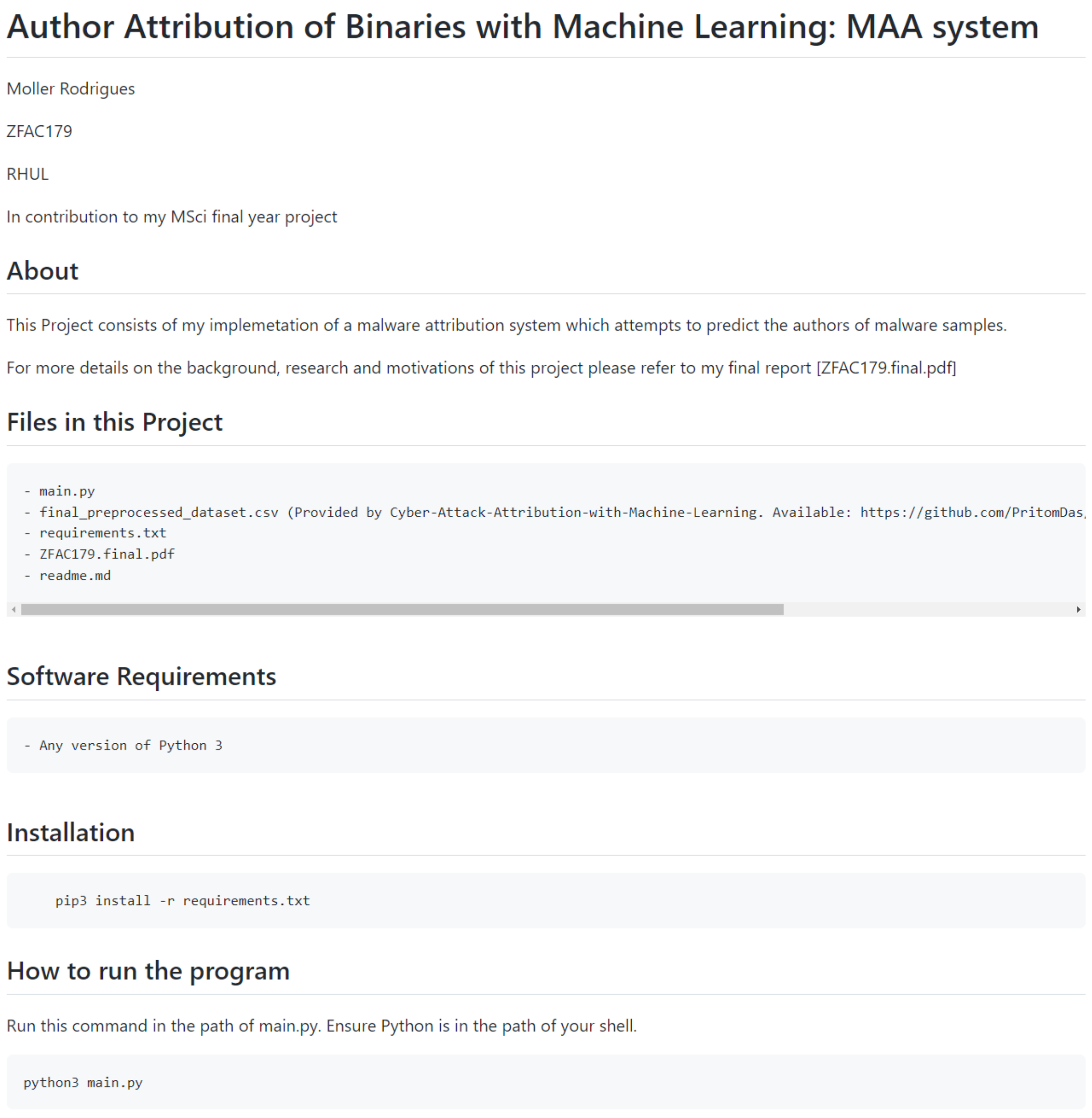
4.9 Packaging the program

4.9.1 Introduction

Now that I have finished implemting the MAA system I will begin cleaning up and packaging the solutions ready for submission. In this section I will give an overview of the steps I took to do so.

4.9.2 Project Structure

- final_preprocessed_dataset.csv (Appendix A.)
- main.py (Appendix B.)
- requirement.txt (Appendix C)
- readme.md (Appendix D)



Author Attribution of Binaries with Machine Learning: MAA system

Moller Rodrigues

ZFAC179

RHUL

In contribution to my MSci final year project

About

This Project consists of my implemetation of a malware attribution system which attempts to predict the authors of malware samples.

For more details on the background, research and motivations of this project please refer to my final report [ZFAC179.final.pdf]

Files in this Project

```
- main.py
- final_preprocessed_dataset.csv (Provided by Cyber-Attack-Attribution-with-Machine-Learning. Available: https://github.com/PritomDas,
- requirements.txt
- ZFAC179.final.pdf
- readme.md
```

Software Requirements

```
- Any version of Python 3
```

Installation

```
    pip3 install -r requirements.txt
```

How to run the program

Run this command in the path of main.py. Ensure Python is in the path of your shell.

```
python3 main.py
```

Figure 12. Preview of project's readme file

Chapter 5: **Professional Issues**

5.1 Introduction

Proffesionalism and ethics is vital in conducting the field of Computer Science and Information Security. In this section I will be detailing the professional issues such as usability, plagiarism, licensing, safety and management that I came across while carry out my project.

5.2 Usability

With regards to usability my main concerns whilst researching the problem area was the power of the tools avaible such as Cuckoo Sandbox and how such tools could potentially mean reducing the number of security analysts required.

With such powerful tools such as Cuckoo Sandbox which can automate the process of analysing a large number of malware samples and generate in depth reports the demand for security analysts could potentially fall. I say this because if a information security researching company employed tools such as Cuckoo Sandbox they could easily reduce their number of staff and have a few skilled operators instead as this would greatly reduce their costs, especially if the tools are open-sorce like Cuckoo.

5.3 Plagarism and Licensing

Plagiarism is a topic that always comes up during asssignments and coursework. During my project I was conscious on referencing and giving credit to any content that I was using from other sources. With regards to code and datasets I ensured I checked the licenses first to make sure I was allowed to re use and modfy them freely. In cases where I did use code or datasets provided by others I was sure to give credit to them and reference them.

5.4 Risk management and safety

Another topic I was concerned about was the security and safety of my personal Desktop and network. This was ofcourse a particular concern I was dealing with maware samples. So to reduce the risk I was through is following the correct measures such as using sandboxed enrironments, virtual machine and having network and security policies in place.

5.5 Management

Management turned out to be critical issue during the course of my project. This is because over the course of my project I encountered many obsatackes from personal circumstances to technical circumstances. These obstacles resulted in me having to revise my deadlines and timelines and also refactor design approaches and solutions. Another problem I faced was with regards to scheduling and attending meeting with my supervisor. Due to personal circumstances I was occupied with other affairs for a long period of time resulting in me in not being proactive about scheduling meetings and missing one meeting aswell.

5.6 Conclcusion

Beware aware of these professional and etic issues and reflecting on how they have effected me during my project has definitely been a good and insightful experience. I definitely think I will carry these principles and be more aware about them in my professional career.

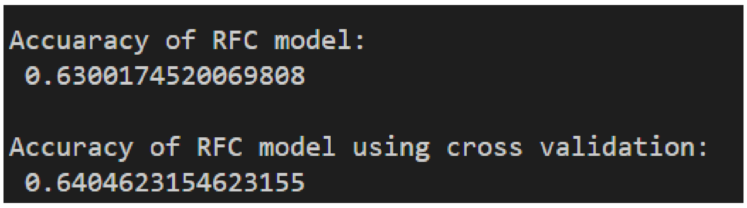
Chapter 6: **Evaluation**

6.1 Introduction

I have to got say this project has gone far from smoothly. I have met many unforeseen personal circumstances which have consumed a lot of my time, but all things considered I am happy with they way I dealt with these obstacles to still continue and attempt to finish the project with some kind working solution.

In this section I will evaluate the RFC model I implemented aswell as a general evaluation of my process of completing this project

6.2 RFC model



```
Accuaracy of RFC model:
 0.6300174520069808

Accuracy of RFC model using cross validation:
 0.6404623154623155
```

Figure 13. Remider of RFC model results

The accuracy of my RFC model came out to be 63% without cross validation and 64% with cross calidation. In my opinion 64% accuracy is satisfactionary however I believe it is possible to increase the accuracy of this model if I tuned the parameters for the RFC model.

One method which I remembered after development from my Machine Learning module that I could have used to tune the parameters for the RFC model was the grid search method.

Unfortunately due to time constraints I was not able to implement this. But as a raw model with not much fine tuning I am happy with 64% accuracy.

6.3 Conclusion

Personal matters aside I would like to focus on the technical methods that went wrong during my project. The main technical problems I face during this project was feature extraction. My initial approach of using Cuckoo Sandbox was just not ideal for my limited time and rsources. Firstly, it took quite some time to setup Cuckoo Sandbox and then after modifying and resuing scripts to extract, submit and retrieve reports for the malware sample I realised it would just take too long to generate reports for a decent sample size.

Despite this I continued to make a proof of concept with a much smaller sample size of 5 APT groups with 3 samples each. I managed to create a proof of concept which extracted features from these samples, generate reports, preprocess the json report, vectorize them into a bag of words, use them as input for classifiers such Logisitic Regression and MultinomialNB.

I demoed this proof of concept to my supervisor however a major problem occurred a days after. The VM which I developed the proof of concept was corrupted and I did not have a snapshot. This was a major setback for me but despite this I carried on and found an alternative solution.

In the end though I am pleased with the speed at which I managed to catch up was good and my adaptablty in problematic scenarios.

6.4 Future plans

My future plans regarding this project is hopefully to conduct a more thorough and detailed critical literature review and also to perhaps write a feature extraction tool myself which would make use of static analysis methods. As mentioned in previous sections I would also like to optimise my current RFC model and make it more accurate.

Bibliography

References

[1] N. Rosenblum, X. Zhu and B. P. Miller, "Who wrote this code? identifying the authors of program binaries," in *Computer Security – ESORICS 2011*Anonymous Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 172-189.

[2] J. Gray, D. Sgandurra and L. Cavallaro, "Identifying Authorship Style in Malicious Binaries: Techniques, Challenges & Datasets," 2021. Available: https://arxiv.org/abs/2101.06124.

[3] S. Alrabaee *et al*, "OBA2: An Onion approach to Binary code Authorship Attribution," *Digital Investigation*, vol. 11, pp. S94-S103, 2014. Available: https://dx.doi.org/10.1016/j.diin.2014.03.012. DOI: 10.1016/j.diin.2014.03.012.

[4] T. Neal *et al*, "Surveying Stylometry Techniques and Applications," *ACM Computing Surveys*, vol. 50, *(6)*, pp. 1-36, 2018. Available: http://dl.acm.org/citation.cfm?id=3132039. DOI: 10.1145/3132039.

[5] (Mar 20,). *A not-So-Common Cold: Malware Statistics in 2021* . Available: https://dataprot.net/statistics/malware-statistics/.

[6] Natalia Stakhanova, "Know your Enemy: Malware Authorship Attribution ," 2018. Available: https://www.youtube.com/watch?v=5cp8-HI2SLQ.

[7] Daniele Sgandurra, "Introduction to Malicious Software," 2020.

[8] M. Egele *et al*, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, *(2)*, pp. 1-42, 2012. Available: http://dl.acm.org/citation.cfm?id=2089126. DOI: 10.1145/2089125.2089126.

[9] (Mar 26,). *Malware Analysis* . Available: https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/.

[10] F. S. Ortega, "Basic Malware Analysis," *Iy3840*, Jan 16, 2020.

[11] Adam Hupp, "python-magic software on GitHub," .

[12] (Jul 23,). *Simple malware obfuscation techniques*. Available: https://libraryguides.vu.edu.au/ieeereferencing/webbaseddocument.

[13] A. Moser, C. Kruegel and E. Kirda, "Limits of static analysis for malware detection," in Dec 2007, pp. 421-430.

[14] W. Hardy *et al*, "DL4MD: A Deep Learning Framework for Intelligent Malware Detection," *Proceedings of the International Conference on Data Mining (DMIN)*, pp. 61, 2016. Available: https://search.proquest.com/docview/1835643953.

[15] W. Huang and J. W. Stokes, "MtNet: A multi-task neural network for dynamic malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*Anonymous Cham: Springer International Publishing, 2016, pp. 399-418.

[16] K. Rieck *et al*, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, *(4),* pp. 639-668, 2011. . DOI: 10.3233/JCS-2010-0410.

[17] I. Rosenberg, G. Sicard and E. David, "End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware," *Entropy (Basel, Switzerland),* vol. 20, *(5),* pp. 390, 2018. Available: https://search.proquest.com/docview/2056808707. DOI: 10.3390/e20050390.

[18] (Cuckoo). *Automated Malware Analysis Tool*. Available: https://cuckoosandbox.org/.

[19] (Google). *Google Code Jam*. Available: https://codingcompetitions.withgoogle.com/codejam/.

[20] (GitHub). *GitHub repositories*. Available: https://github.com.

[21] (cyber-research). *APTMalware*. Available: https://github.com/cyber-research/APTMalware.

[22] (Hex-Rays). *IDA Pro*. Available: https://www.hex-rays.com/products/ida/.

[23] (Paradyn). *Dyninst*. Available: http://www.dyninst.org.

[24] (VirusTotal). *VirusTotal*. Available: https://www.virustotal.com/gui/home/upload.

[25] (Mar 28,). *How to Implement Classification in Machine Learning?* . Available: https://www.edureka.co/blog/classification-in-machine-learning/.

[26] (). *Design Strategies*. Available: https://www.tutorialspoint.com/system_analysis_and_design/system_analysis_and_design_strategies.

[27] (). *What is Class Diagram*. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/.

[28] (Python). *Python Programming Language*. Available: https://www.python.org/.

[29] (Microsoft). *Visual Studio Code*. Available: : https://code.visualstudio.com/.

[30] (VMware). *VMware Workstation 16 Player Non commercial*. Available: https://www.vmware.com/uk/products/workstation-player/workstation-player-evaluation.html.

[31] (Ubuntu). *Ubuntu Server 18.04.6 LTS*. Available: https://releases.ubuntu.com/bionic/.

[32] (Cuckoo). *Cuckoo Sandbox*. Available: https://cuckoo.sh/docs/installation/index.html.

[33] (cyber-research). *APTAttribution*. Available: https://github.com/cyber-research/APTAttribution.

[34] (pritomdas). *Cyber-Attack-Attribution-with-Machine-Learning*. Available: https://github.com/PritomDas/Cyber-Attack-Attribution-with-Machine-Learning.

[35] (Oct 1,). *Predicting Authors of Bible Passages with Machine Learning(Author Attribution)*. Available: https://www.youtube.com/watch?v=bMbjDi-JVkY.

Appendix

Appendix A: final_preprocessed_dataset.csv

Appendix B: main.py

Appendix C: requirement.txt

Appendix D: readme.md