A Hybrid approach towards Malware Detection through Machine Learning

MCS

by

Faayed Al Faisal

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

March
2023

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr **Faayed Al Faisal,** Registration No. **00000327055**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature:________________________

Name of Supervisor Prof Dr. Haider Abbas

Date: __________________________

Signature (HoD): ____________________

Date: __________________________

Signature (Dean/Principal): ___________________

Date: _______________________________

Declaration

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere

MS Student

Dedication

“In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to family and teachers who supported me each step of the way.

Acknowledgments

All praises to Allah for the strengths and His blessing in completing this thesis.

I would like to convey my gratitude to my supervisor, Dr. Haider Abbas for his supervision and constant support. His invaluable help of constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research. Also, I would thank my committee members; Dr. Fawad Khan, and Col. Dr. Imran Makhdoom for their support and knowledge regarding this topic.

Abstract

Malware detection and classification is the first step towards understanding the nature of attacks and then deciding a response to future incidents. Due to the level of sophistication, analysis evasion techniques and the ability to achieve stealth, detection and classification of Advance Persistent Threat (APT) malware is especially challenging. Dynamically analysing them is also challenging because APTs may wait for an extended period of time before actually performing their intended malicious tasks. Therefore, most work focuses on Statically analysing APTs, hence ignoring an important aspect of their behavior. In this research, we present a hybrid analysis model to detect APTs. Our APT dataset comprises of 3500+ malware gathered from cyber-research's Github whereas 2800+ benign samples were binaries collected from a standard installation of a Windows 10 (x64). Our hybrid analysis model which combines strings, which are a static feature of APTs, along with the dynamic features of frequency and sequence of API calls, is able to detect APTs with a high degree of accuracy approaching 92.3%, precision of 100%, a recall of 89% and the F1 score of 94%.

Table of Contents

List of Figures & Tables

Acronyms

Advanced Persistent Threats	**APT**
Artificial Intelligence	**AI**
Deep Learning	**DL**
Machine Learning	**ML**
Random Forest Algorithm	**RFA**
Decision Tree	**DT**
K-Nearest Neighbor Algorithm	**KNN**
Support Vector Machine	**SVM**
Comma-Separated Value	**CSV**
True Positive	**TP**
True Negative	**TN**
False Positive	**FP**
False Negative	**FN**
Application programming interface	**API**

1 Introduction

The modern world revolves around information. Data is regarded as “digital gold” and there are plenty of people with malicious intentions that want to steal data and profit or gain leverage from it. Malware is a malicious software used to steal, damage or delete data, for the gain of an attacker. Malware come in many varieties and are also delivered to the victim in even more ways. The common defense against malware are the antivirus software which have traditionally used malware's digital signature to compare it with the suspicious file. The problem with signatures is they are not suitable when the malware can change form thereby defeating the defense mechanisms which rely on existing signatures. Similarly, zero-day malware which exploit previously unknown vulnerabilities, can also not be detected by signature-based defenses. Therefore, there is a growing trend among malware defense solutions to include behavioral analysis in an attempt to better cope with the changing nature of malware.

1.1 Malware

There are several ways malware are altered to become “new” so that antivirus are not able to detect them. Malware designers are known to add meaningless instruction, a technique called junk code injection in order for the malware to have a different signature. There is also code obfuscation where a malware designer intentionally makes their code tougher to understand and interpret **[1]**. This can involve naming variables in an incomprehensible manner to deceive readers, data alteration, comments filling with special characters to cause more distraction and whitespace removal to name a few. Malwares of both metamorphic and polymorphic nature use obfuscation techniques to perform changes to their own code. Polymorphic malware are able to change their code using an encryption key, meanwhile metamorphic malware do not require an encryption key as they enter a system and change their own code.

1.2 Advanced Persistent Threats

Advanced Persistent Threats, commonly known as APTs, are a serious security threat to businesses, government organizations, and individuals. APTs are sophisticated and targeted attacks that are designed to evade traditional security measures, infiltrate systems, and remain undetected for long periods of time. APTs are typically carried out by well-funded and highly-skilled attackers who are motivated by financial gain, espionage, or sabotage.

The goal of APTs is to gain unauthorized access to an organization's network, data, and resources. Once inside the network, attackers can move laterally, gather sensitive information, steal intellectual property, disrupt operations, or deploy malicious payloads. Unlike traditional malware attacks that are one-time events, APTs are ongoing and persistent. Attackers may use a combination of social engineering tactics, spear-phishing emails, and zero-day vulnerabilities to gain access to a target's network. Once inside, they will often deploy custom malware that is designed to evade detection by traditional antivirus software and other security measures.

APTs are notoriously difficult to detect and mitigate. Attackers may spend months or even years inside a target's network, quietly gathering information and exfiltrating data. They may use encryption and other techniques to hide their activities and evade detection. APTs require a different approach to security than traditional malware attacks. Traditional security measures, such as firewalls, antivirus software, and intrusion detection systems, are important but not sufficient to defend against APTs. APTs require a multi-layered approach that includes advanced threat intelligence, behavior-based analytics, and machine learning algorithms.

One of the most effective ways to defend against APTs is to implement a comprehensive security program that includes people, processes, and technology. A strong security program includes regular security awareness training for employees, a robust incident response plan, and advanced security tools that can detect and respond to APTs. Advanced security tools include next-generation firewalls, endpoint detection and response (EDR) solutions, and security information and event management (SIEM) platforms.

1.3 Threat Intelligence

Another important aspect of defending against APTs is threat intelligence. Threat intelligence is the process of collecting, analyzing, and sharing information about security threats. Threat intelligence can help organizations identify and respond to APTs by providing up-to-date information about new threats, attack methods, and vulnerabilities. Threat intelligence can be gathered from a variety of sources, including open-source intelligence, commercial intelligence services, and internal security logs.

Malware analysis is carried out to determine a malware's role in a cyber attack. Understanding a malware's functionality can prevent it from causing damage in the future. There are two types of malware analysis: static and dynamic. Static malware analysis is all the information you can extract from a malware without executing it. This can involve analysing the strings inside the binaries, disassembling the file to see the functions it calls and analyzing its headers. Malware built for the Windows operating systems mostly have the Portable Executable (PE) file format. PE headers provide the metadata regarding the file such as timestamps for its compilation, section names and sizes, and information about the file's raw and virtual sizes. These pieces of information can be used to determine if the file is malicious or not. On the other hand, dynamic analysis is used to extract information from a malware when it is being executed. This can include a malware's interaction with the file system such as tracking disk changes, its interaction with the system Registry such as manipulation of configuration settings, analyzing dynamic call graphs, as well as monitoring malware execution using debuggers and virtual machines.

1.4 Artificial Intelligence

Due to the massive amounts of data extracted from malware analysis coupled with the changing nature and new forms of malware, artificial intelligence (AI) has been deployed often as a way of efficiently and accurately detecting new malware threats. There are two main forms of AI: Deep Learning (DL) and Machine Learning (ML). The authors of **[2]** provide a good introduction on how artificial intelligence is generally used for malware detection and how it is currently being approached by incorporating both static and dynamic analysis for its features. Machine Learning provides the ability to learn by using algorithms that discover patterns from a dataset and provide an output accordingly. Deep Learning works similar to a human brain's neural networks that are used for decision making. It is designed to make sense of patterns, noise and sources of confusion in data. Machine Learning on the other hand, requires structured data that are labelled and categorized while Deep Learning uses neural networks to label and categorize data itself. The study also shows how malware detection research is leaning towards deep learning because of its ability to handle complex data and not needing structured data.

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that can perform tasks that typically require human intelligence. AI can be divided into various subfields, such as robotics, natural language processing, and computer vision, but one of the most significant and promising areas is Machine Learning (ML).

Machine Learning is a subset of AI that enables machines to learn and improve from experience without being explicitly programmed. In other words, it allows computers to learn from data and improve their performance over time. The main goal of Machine Learning is to build models that can accurately predict outcomes or make decisions based on input data.

There are three main types of Machine Learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training a model using labeled data, where the desired output is known in advance. For example, a model can be trained to identify different species of flowers based on labeled images of flowers. Unsupervised learning, on the other hand, involves training a model using unlabeled data, where the desired output is not known. The model is trained to identify patterns and structure in the data. Finally, reinforcement learning involves training a model to make decisions based on feedback from its environment. The model learns to maximize a reward signal and improve its decision-making process over time.

Machine Learning algorithms can be further classified into two categories: parametric and non-parametric. Parametric algorithms make assumptions about the underlying distribution of the data and estimate the parameters of the distribution. Non-parametric algorithms do not make assumptions about the distribution and instead rely on flexible models that can fit a wide range of data.

Some of the most common applications of Machine Learning include natural language processing, image recognition, recommendation systems, and fraud detection. In natural language processing, Machine Learning models are used to understand and process human language, allowing for applications such as speech recognition and machine translation. In image recognition, Machine Learning models are trained to identify and classify images, making it possible for computers to recognize faces, objects, and scenes. In recommendation systems, Machine Learning algorithms are used to suggest products or services to users based on their previous actions or preferences. In fraud detection, Machine Learning models are used to detect anomalies and patterns that indicate fraudulent behavior.

The modern digital landscape is characterized by the ubiquitous presence of data, which is often regarded as the new "digital gold." Unfortunately, this wealth of information also attracts those with malicious intentions, who seek to steal, damage, or delete data for their own gain. Malware is a form of malicious software used to carry out these attacks. It comes in various forms and can be delivered in many ways, making it difficult to detect and defend against.

Traditional antivirus software has relied on the digital signature of malware to detect and defend against attacks. However, this approach has its limitations, particularly when it comes to malware that can change its form or exploit previously unknown vulnerabilities. As a result, there is a growing trend towards incorporating behavioral analysis in malware defense solutions to better cope with the changing nature of attacks.

Malware designers use various techniques to alter their malware to make it undetectable by antivirus software. One such technique is junk code injection, where meaningless instructions are added to the malware to make it appear different from its original form. Another technique is code obfuscation, where the code is intentionally made difficult to understand and interpret by renaming variables in an incomprehensible manner, altering data, and using special characters and whitespace removal. Malware of both metamorphic and polymorphic nature use obfuscation techniques to perform changes to their own code.

1.5 Malware Analysis

Malware analysis is crucial in understanding the functionality of malware and preventing it from causing damage in the future. There are two types of malware analysis: static and dynamic. Static analysis involves analyzing the metadata of a malware file without executing it, while dynamic analysis involves monitoring a malware's behavior when it is being executed.

Malware analysis is a vital process for detecting and analyzing malicious software. There are several approaches to analyzing malware, including static, dynamic, and hybrid analysis techniques.

Static analysis involves examining the code and file structure of malware without executing it. The primary goal of static analysis is to identify malware behavior based on its attributes. The attributes may include file size, file name, date created, and other static characteristics. Static analysis is useful for identifying known malware variants, but it is less effective in detecting new, unknown malware.

Dynamic analysis, on the other hand, involves running the malware in a controlled environment, usually a virtual machine or sandbox, to observe its behavior. Dynamic analysis can help identify the malicious activities of malware such as its communication with a command and control server, registry modifications, file system changes, and system process injection. Dynamic analysis is an effective technique for detecting unknown and polymorphic malware, as it analyzes the behavior of malware in real-time.

Hybrid analysis combines static and dynamic analysis techniques to provide a more comprehensive view of the malware. Hybrid analysis begins with static analysis to extract the code's structural characteristics, followed by dynamic analysis to observe the behavior of the malware in real-time. The primary advantage of hybrid analysis is that it can detect unknown and polymorphic malware while also identifying known malware.

Static analysis is relatively fast and straightforward, while dynamic analysis provides more detailed information about the malware's behavior. However, static and dynamic analysis techniques have their limitations. For example, static analysis cannot detect the malware's behavior at runtime, and dynamic analysis requires a controlled environment for analysis.

Hybrid analysis overcomes the limitations of static and dynamic analysis techniques and provides an effective way to detect known and unknown malware. Furthermore, machine learning techniques can be integrated with static, dynamic, and hybrid analysis to automate the process of malware detection and analysis.

In conclusion, the selection of malware analysis techniques depends on the specific objectives of the analysis. Static analysis is useful for identifying known malware variants, dynamic analysis for detecting unknown and polymorphic malware, and hybrid analysis provides a comprehensive view of malware behavior. The use of machine learning techniques can help automate the malware analysis process, making it more efficient and effective.

Due to the massive amounts of data extracted from malware analysis coupled with the changing nature and new forms of malware, artificial intelligence (AI) has been deployed as a way of efficiently and accurately detecting new malware threats. Machine learning and deep learning are the two main forms of AI that are currently being used for malware detection.

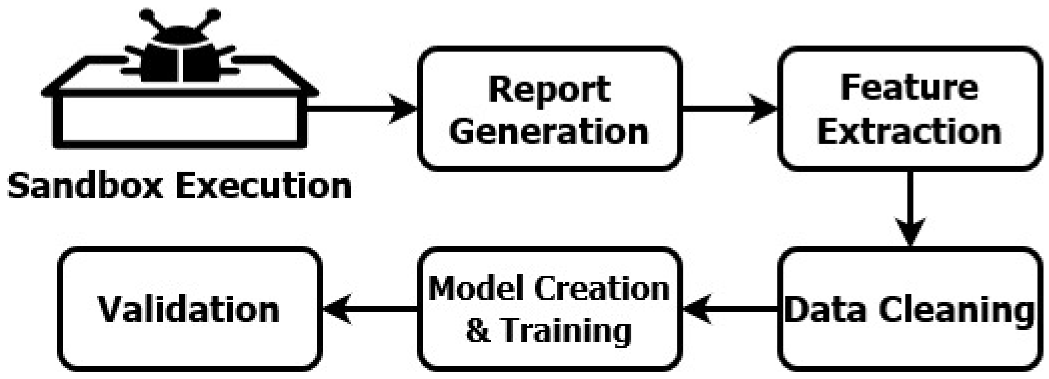


Figure 1: Architecture of Sandbox with Machine Learning module

In this research, we have focused specifically on detecting Advance Persistent Threats (APTs) with the help of machine learning. This is because APTs present a unique challenge for detection owing to their stealth capabilities rendering traditional methods unable to detect them.

1.6 Contributions

Specifically, we have made the following contributions in this research:

- Carried out a survey of contemporary malware detection techniques.
- Proposed a hybrid analysis machine learning model for APT malware detection. Our model combines strings, which are a static feature of APTs and the dynamic features of frequency and sequence of API calls.
- Conducted thorough experimentation to determine the efficacy of the proposed hybrid model.

Rest of this research is organized as follows: Chapter 2 presents a survey of the existing research done on malware detection using machine learning. Our proposed hybrid analysis model and the testing of different Machine Learning classifiers is presented in Chapter 3 whereas a thorough investigation of our model through experimentation along with a discussion on our findings is presented in Chapter 4. Finally, Chapter 5 concludes this research.

2 Related Work

The authors of **[3]** propose various methods to perform machine learning on malware. They have split the extraction of data into three categories: basic structure, low level behavior, and high level behavior. The basic structure refers to the headers of software or in Windows executables case, the PE headers, which contain the information regarding the executable. On the other hand, the low level behaviors are the API calls and the DLLs that are executed to note the tasks that a software performs and the high level behaviors are the malware's network signatures, file system interactions and registry tasks that a malware performs. The research discusses the results of performing analysis based on each of these features on 4250 malware samples and shows the comparative results and also the results when the features are combined. The machine learning classifiers employed are Random Forest, K-Nearest Neighbors and Decision Tree.

The work in **[4]** presents the application of deep learning for the detection of malware in the IoT environment. In the first step the authors apply Eigen graphing and then apply Convolutional Neural Networks on the opcodes extracted from the malware's binaries. The authors have demonstrated that their solution results in more detection accuracy.

The work presented in **[5]** carries out malware detection based on its network activity without applying machine learning. The malware dataset comprises 999 diverse malware samples from Georgia Tech Apiary project **[6]**. More than half of the samples did not show any network activity due to possible inactive IP addresses or the observation period may have been too short. The network activities considered as "High Level Behaviors" include downloading, scanning, reporting, spamming, Command and Control Communication and Propagating. However, the results show that the most performed activities are scanning and propagating. This observation is intuitive because the malware that have network functionalities try to locate and infect all hosts within a network.

A behavior based model is presented in **[7]** which emphasises the importance of preparing data before making it go through multiple classification algorithms. The extraction of the features is carried out by executing the malware in a virtual environment for a specific amount of time and recording the sequence of API calls which determines the behavior. The research uses Decision Tree, Random Forest and Support Vector Machine(SVM) machine learning algorithms.

A comparative study **[8]** of the different types of analysis shows varying results produced by static, dynamic and hybrid analysis using Hidden Markov Models applied on the training datasets. It shows that the dynamic analysis models produce greater accuracy in comparison to the static analysis models.

The image based hybrid deep learning models proposed in **[9]** employ creating images of malware samples to train image based deep learning models on. The models were trained and tested on various sizes of datasets to test its scalability. The Graph based dynamic analysis presented in **[10]** proposes a similar methodology using the graphs created from dynamically extracted instructions from a software sample. This allows mapping to occur to make a prediction model based on similarities between the graphs of different samples to be able to classify them.

Malware detection through static features becomes very challenging when malware authors try to hide its functionality through obfuscation, encryption and/or packing. The problem is further complicated when the type of packers and obfuscators used, is unknown. Such challenges have been demonstrated in Android applications by **[11]**. This necessitates the application of dynamic analysis techniques **[12]** so that the attack code can first be unpacked/deobfuscated before execution thereby enabling its analysis.

The study in **[13]** discusses the issues with deep learning when it comes to malware detection underlines some of the issues that lies with this mechanism as well as show in example of how an attacker can easily bypass that. The author uses the example of detection through PE headers which counts in static analysis, and describes how they can attach a payload to a seemingly benign file without changing the any data in the PE header file. So it's important to have to cover multiple areas of detection and not put all eggs in one basket. This explains part of the reason why the sandbox will have both static and dynamic analysis included.

The paper [3] proposes various methods to extract data from malware, including basic structure, low-level behavior, and high-level behavior. The authors also compare the results obtained from different machine learning classifiers. The work presented in [4] demonstrates the application of deep learning for malware detection in the IoT environment, which provides more accurate results than traditional methods. [5] focuses on detecting malware based on its network activity, while [7] proposes a behavior-based model that uses multiple classification algorithms.

A comparative study of static, dynamic, and hybrid analysis using Hidden Markov Models is presented in [8]. The authors show that dynamic analysis models produce greater accuracy compared to static analysis models. The use of image-based hybrid deep learning models to detect malware is proposed in [9], while graph-based dynamic analysis is presented in [10]. The challenges of static analysis and the importance of dynamic analysis techniques are discussed in [11] and [12], respectively.

The issues with deep learning for malware detection and the possibility of attackers bypassing it are highlighted in [13]. The paper [14] proposes a Federated malware detection architecture for IoT devices that enables individual machines to test and improve the machine learning model. The problem of attackers poisoning the dataset is addressed in [15] by proposing a cross-validation technique.

The Federated malware detection architecture on IoT devices in **[14]** showcases a methodology that can be used once the final hybrid analysis model has been made and deployed on multiple machines. Individual machines are given the trained model to be deployed in a real life environment so that it is able to test the current efficacy of the model on new samples that are being passed through the sandbox instance and the hybrid analysis machine learning model. Once individual machines have tried the model with a sufficient amount of new samples, the logs of the new samples are processed within the machine to create a training dataset that can then be transferred to a main node/server that will append it to the original model to try to produce better results and this process will occur with ever individual machine given the sandbox instance and the machine learning model as to try to create a better model. Federated learning presents a problem regarding attackers poisoning the dataset by injecting false data into the servers main training dataset which in turn can ruin the accuracy of the machine learning model. The researchers in **[15]** proposed a cross validation technique to check the validity of the data by evaluating the logs over other clients local data which should figure out any inconsistencies with training data sent from a machine over to the server side.

What we were able to gather from these papers were the way in which to approach dynamic analysis as well as to test out the accuracy in comparison to the standard static analysis. Data preparation plays a huge role in the results we require as well as balancing the amount of malware to the benigns as to not tip the balance and resulting in our model working in a probabilistic manner. Data formatting must also take place as to not muddle the data as well as being able to handle the junk code attackers insert into their malwares to deter the analysis.

3 Proposed Hybrid Model for Malware Detection

Table 1: The APT Dataset and its Sample Distribution

Country	APT Group	Family	Samples
China	APT 1	N/A	1007
China	APT 10	i.a.PlugX	300
China	APT 19	Derusbi	33
China	APT 21	TravNet	118
Russia	APT 28	Bears	230
Russia	APT 29	Dukes	281
China	APT 30	N/A	164
North Korea	Dark Hotel	DarkHotel	298
Russia	Energetic Bear	Havex	132
USA	Equation Group	Funnyworm	395
Pakistan	Gorgon Group	Different RATs	1085
China	Winnti	N/A	406

To collect malware samples for analysis, various open source threat intelligence reports from multiple vendors were utilized. This approach allowed for a comprehensive view of the threat landscape, as different vendors may have unique perspectives and insights. After collecting numerous threat intelligence reports, all filehashes used as indicators of compromise (IoC) were extracted and compiled into a list. This list served as a starting point for obtaining the actual malware samples. Using VirusTotal, a popular online tool for malware analysis, the filehashes were queried to retrieve the associated samples. As a result, a diverse collection of malware samples was obtained, which enabled thorough analysis of the types of threats present and their characteristics. It is worth noting that this method of collecting malware samples is not without limitations, as not all malware may be identified by the indicators of compromise used in the reports. However, it provides a valuable starting point for analysis and sheds light on the current state of the threat landscape.

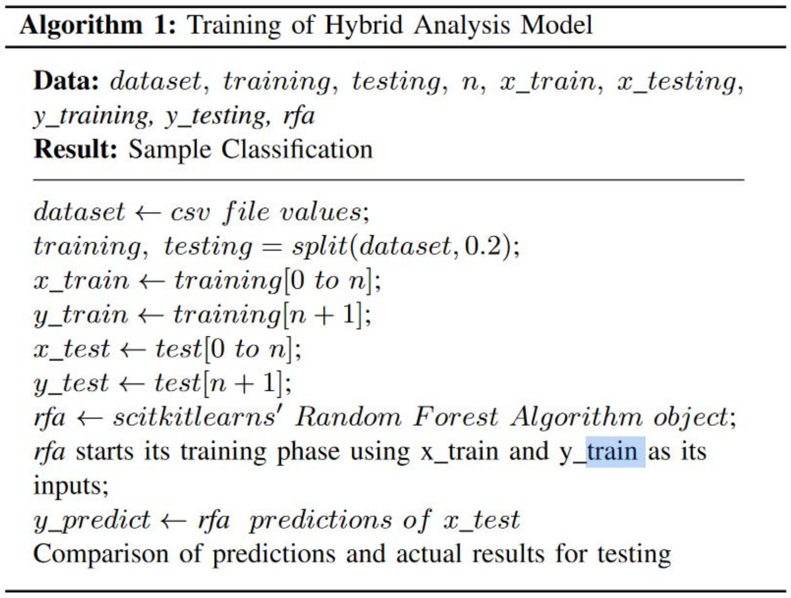
3.1 Preliminaries

The proposed hybrid model uses both static and dynamic features to improve the accuracy of APT malware detection. The static features include strings extracted from the malware's binary, while the dynamic features include the frequency and sequence of API calls made during malware execution. The strings were extracted using YARA, an open-source tool used for pattern matching and rule-based detection, while the API calls were collected using dynamic analysis in the CAPE sandbox environment.

To train and test the machine learning models, we used a dataset of 3500 samples from 12 APT families. The dataset was carefully curated to ensure that it contained a diverse set of APTs, including well-known families such as Dark Hotel, Equation Group, and Energetic Bear. Table 1 provides more details about the dataset, including the number of samples per family.

For dynamic analysis, we used CAPE Sandbox version 2.1, which is a derivative of the popular Cuckoo Sandbox. CAPE Sandbox provides a comprehensive environment for dynamic analysis, allowing us to collect a wide range of information about malware behavior, including file system changes, registry manipulation, and network traffic. For static analysis, we used Python 3.8 along with ScikitLearn, a widely used machine learning library for Python that provides a range of tools for developing classification, regression, and clustering models.

Overall, our experimental setup was designed to provide a comprehensive analysis of APT malware, using both static and dynamic features to train and test machine learning models. We believe that this approach can provide a more accurate and effective way of detecting APTs, which are among the most stealthy and dangerous forms of malware in existence.



Algorithm 1: Training of Hybrid Analysis Model

```
Data: $dataset$, $training$, $testing$, $n$, $x\_train$, $x\_testing$,
$y\_training$, $y\_testing$, $rfa$
Result: Sample Classification
$dataset \leftarrow csv\ file\ values$;
$training,\ testing = split(dataset, 0.2)$;
$x\_train \leftarrow training[0\ to\ n]$;
$y\_train \leftarrow training[n+1]$;
$x\_test \leftarrow test[0\ to\ n]$;
$y\_test \leftarrow test[n+1]$;
$rfa \leftarrow scitkitlearns'\ Random\ Forest\ Algorithm\ object$;
$rfa$ starts its training phase using x_train and y_train as its
inputs;
$y\_predict \leftarrow rfa\ \ predictions\ of\ x\_test$
Comparison of predictions and actual results for testing
```

Figure 2: Hybrid Analysis Training Model

3.2 Machine Learning Classifier Selection and Implementation

Our proposed malware detection approach is built upon a static and a dynamic analysis model, the outcome of which is used to build our hybrid analysis model. The classifier we used for our solution is the Random Forest Algorithm, which is the average of random sampling multiple decision trees **[17]**. We selected random forest algorithm because of its efficiency with classification using string inputs **[18]**. The Random Forest Algorithm applies the training on multiple random decision trees and merges the decisions of those trees to find an average answer between them. Each Decision Tree Classifier employs a supervised learning algorithm and is organized as a hierarchical tree structure consisting of a root node, leaf nodes and internal nodes. The Decision tree starts at the root node whose decision leads into a specific internal node which in turn, may lead into more internal nodes. This can continue until it reaches one of the leaf nodes which represent the possible outcomes. A classification algorithm was chosen because of the categorical nature of our usecase whereas a regression algorithm is used in situations where a continuous stream of outcomes is possible.

The proposed malware detection approach utilizes both static and dynamic analysis models to build a hybrid analysis model. The approach employs the Random Forest Algorithm as the classifier, which is known for its efficiency in classification using string inputs. This algorithm is a combination of multiple decision trees that are built using random sampling. The Random Forest Algorithm applies training on multiple random decision trees and merges the decisions of those trees to find an average answer between them.

Each Decision Tree Classifier in the Random Forest Algorithm employs a supervised learning algorithm and is organized as a hierarchical tree structure consisting of a root node, leaf nodes, and internal nodes. The Decision tree starts at the root node and makes decisions based on the input features. It then moves to a specific internal node which in turn may lead to more internal nodes. This process continues until it reaches one of the leaf nodes, which represents the possible outcomes. In this approach, a classification algorithm was chosen because of the categorical nature of the use case, whereas a regression algorithm is used in situations where a continuous stream of outcomes is possible.

To create the model, the dataset is first loaded from a CSV file onto a dataset dataframe variable. The dataset is then split into training and testing dataframes using an 80:20 ratio. The training data consists of the x_train dataframe, which is a subset of the training dataframe from 0 to n, where n represents the last feature column in the dataset, and the y_train, which is the classification column of the dataset showing whether the sample is malicious or not. The two training values are fit into the random forest algorithm rfa, which is instantiated from the scikit-learn library.

The testing dataframe is also split into x_test and y_test. The rfa object fits the training data and starts creating a model. Once the model has been created and stored into rfa, the testing phase begins, during which rfa creates predictions of x_test and stores them in the y_predict dataframe. The predictions stored in y_predict are then compared with the true classifications of x_test stored in y_test.

The performance of the model is evaluated through numerous comparisons, such as the confusion matrix and classification reports. The evaluation results are presented in Chapter 4 of the research. The proposed approach combines the strengths of both static and dynamic analysis to form a more accurate malware detection mechanism. The use of the Random Forest Algorithm as the classifier allows for efficient classification using string inputs, and the use of the hybrid analysis model enhances the accuracy of malware detection.

The creation of every model in this research is represented through the pseudo-code in Algorithm 1. The dataset is loaded from a *csv file* onto a *dataset* dataframe variable. The *dataset* is split into *training* and *testing* dataframes with a 80:20 ratio. The *x_train* dataframe is a subset of the *training* dataframe from 0 to *n* where *n* represents the last feature column in the dataset. The *y_train* is the classification column of the dataset which shows whether the sample is malicious or not. The two training values are fit into the random forest algorithm *rfa* which is instantiated from the scikitlearn library. The *testing* dataframe is also split into *x_test* and *y_test*. The *rfa* object fits the training data and starts creating a model. Once the model has been created and stored into *rfa*, the testing phase begins, during which *rfa* creates predictions of *x_test* and stores it in the *y_predict* dataframe. The predictions stored in *y_predict* are then compared with the true classifications of *x_test* stored in *y_test*. The performance of our model is evaluated through numerous comparisons such as the confusion matrix and classification reports and are presented in Chapter 4.

Table 2 Accuracy of ML Algorithms for Static Analysis

Algorithm	**Accuracy (%)**		
	Min. 5 char	**Min. 7 char**	**Min. 9 char**
Naïve Bayes	57.10	61.76	64.33
Logistic Regression	97.43	98.44	98.38
Decision Tree	91.33	94.80	96.11
Random Forest	97.83	98.14	98.38
SVM [Linear]	96.65	97.90	97.72
SVM [RBF]	92.29	95.34	94.98
Grad. Descent	95.63	97.41	97.37
Adaboost	94.32	96.53	96.71
Bagging	95.93	96.54	97.43
KNN	85.72	88.53	86.16

3.3 Model Creation with Static Analysis

For the sake of statically analyzing any sample there are multiple options to consider as features such as registry keys, directory and file path names, URLs, IP addresses, information contained in PE files' headers and section names, strings that form various messages for human interactions. However, most of these features are already in the form of strings. Hence strings encompass most of these features and therefore we consider all string-type features as input for our machine learning model. The process of feature extraction is explained as follows: every sample of the dataset was submitted to Microsoft Strings tool with the help of a script, which extracted all strings of a given minimum length and stored them in a file which forms our cleaned dataset.

In the context of static analysis, there are several features that can be taken into account to analyze a sample for malware detection. These features include registry keys, file path and directory names, URLs, IP addresses, and information contained in PE files' headers and section names. Another important feature that is commonly present in malware samples is strings. In fact, strings encompass most of the aforementioned features, as they often appear as human-readable text within a sample's code or data.

Therefore, in this research, all string-type features were considered as input for the machine learning model used for malware detection. The process of feature extraction involved submitting each sample of the dataset to Microsoft Strings tool with the help of a script. This tool extracted all strings of a given minimum length and stored them in a file, which formed the cleaned dataset used for training and testing the machine learning models.

Multiple machine learning algorithms can be used to train and create a model using the strings dataset. In this research, 10 algorithms and 3 different minimum string lengths were selected for the experiments, resulting in varying levels of detection accuracy. The details of the experiments are provided in Table II. It was found that the Random Forest algorithm was the most accurate in classifying the samples as either malicious or benign. This algorithm was selected for the hybrid analysis model used in this research, along with a dynamic analysis model, as described earlier.

```
def get_string_features(path,hasher):
    # extract strings from binary file using regular expressions
    chars = r" -~"
    min_length = 7
    string_regexp = '[%s]{%d,}' % (chars, min_length)
    file_object = open(path, encoding='latin1')
    data = file_object.read()
    pattern = re.compile(string_regexp)
    strings = pattern.findall(data)

    # store string features in dictionary form
    string_features = {}
    for string in strings:
        string_features[string] = 1

    file = open('data.txt', 'w')
    file.write(str(string_features))

    # hash the features using the hashing trick
    hashed_features = hasher.transform([string_features])

    # do some data munging to get the feature array
    hashed_features = hashed_features.todense()
    hashed_features = numpy.asarray(hashed_features)
    hashed_features = hashed_features[0]

    # return hashed string features
    print("Extracted {0} strings from {1}".format(len(string_features),path))
    return hashed_features
```

Figure 3: String feature extraction code snippet

The above code snippet extracts strings from binary files while also filtering out garbage strings that have less than a specified length.

Multiple machine learning algorithms can be used to train and create a model using the strings dataset. We selected 10 algorithms and 3 different minimum string lengths for our experiments, which resulted in varying levels of detection accuracy. The details are provided in Table II. As evident, Random Forest was found to be the most accurate in classifying the samples as either malicious or benign.

Table 3 Accuracy of ML Algorithms for Dynamic Analysis

Algorithm	**Accuracy**	
	5 min run	**5 hr run**
Naïve Bayes	15.1	22.44
Decision Tree	15.2	22.45
Random Forest	77.5	94.73
Logistic Regression	60.1	70.08
SVM(Linear)	62.4	74.55

3.4 Model Creation with Dynamic Analysis

The difference between static and dynamic analysis is that a sample has to be executed and needs to fully perform all of its tasks to completion to capture its behavior in entirety. This is further complicated by the inherent nature of APT's which are persistent and therefore may take much longer time to complete its execution. To cater for this peculiarity, every sample submission to the CAPE sandbox was allowed 5 hours of execution time instead of the default 5 minutes. Due to this, we selected a smaller subset comprising of 100 APT binaries from the original dataset, 8-10 from each APT family.

Static analysis and dynamic analysis are two methods for analyzing malware. The main difference between the two is that static analysis involves examining the characteristics of a sample without actually executing it, while dynamic analysis involves running the sample in a controlled environment to observe its behavior. In dynamic analysis, the sample has to fully perform all of its tasks to completion to capture its behavior in its entirety, and this can take a long time for persistent APTs. To address this issue, the authors allowed every sample submission to the CAPE sandbox to execute for five hours instead of the default five minutes.

To perform dynamic analysis, the authors selected a smaller subset of 100 APT binaries from the original dataset, comprising 8-10 samples from each APT family. Once a sample was submitted to the sandbox, the sandbox executed the binary for five hours and generated an analysis report. The authors then created a script to extract the API calls from the report and stored them in a CSV file to be used as input for their machine learning models. The reason for selecting API calls as the feature for the models is that a program's behavior is represented by the selection, frequency, and sequence of its API calls.

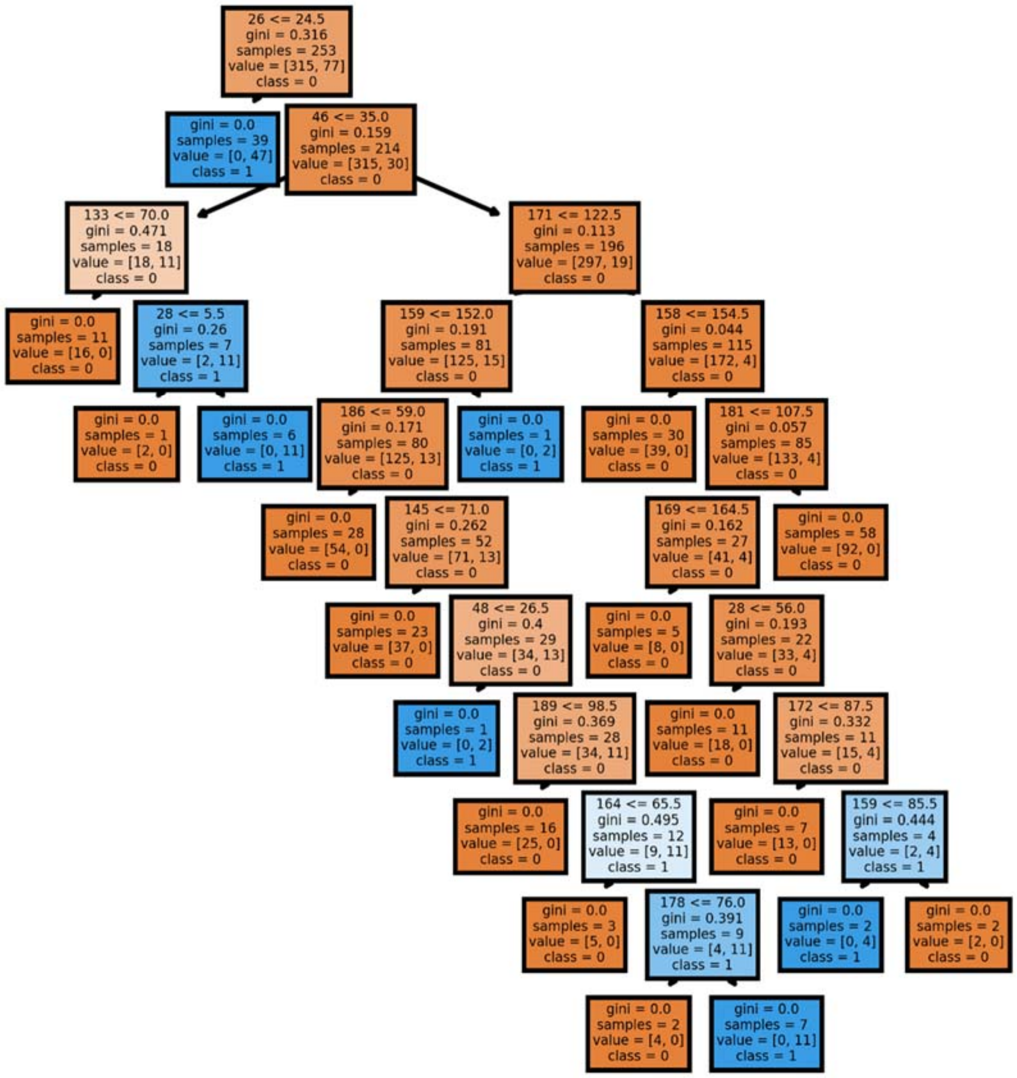
```
for jsons in onlyfiles:
        try:
            with open(jsons) as f_in:
                data = json.load(f_in)
        except:
            print("Wrong")
            continue
        print(jsons)
        print(counter)
        length = len(data['behavior']['processes'])

        #for i in range(len(data['behavior']['processes'])):
        #    length = len(data['behavior']['processes'][0]['calls'])
        api_map = []
        name = "NA"
        for i in range(length):
            length1 = len(data['behavior']['processes'][i]['calls'])
            for j in range(length1):
                api_map.append(data['behavior']['processes'][i]['calls'][j]['api'
])
        api_map = np.array(api_map)
```
Figure 4: json feature extraction code snippet

The above code snippet is what we used to extract the desired features from the json report files. With python, we can iterate through each api call and append it to a final list which is then put into the actual training dataset.

The authors chose the same ten algorithms used in static analysis to compare their performance in dynamic analysis, with three different minimum string lengths. The results of the experiments are presented in Table 3. As with static analysis, Random Forest outperformed all the other algorithms in dynamic analysis, with a detection rate of 94.7%

Once a sample has been submitted, the sandbox executes the binary for 5 hours and then generates its analysis report. A script that we created extracts the API calls from the report and stores them in a CSV file to be used as input for our machine learning models. The reason we selected API calls as the feature for the models is that any programs behavior is represented as the selection, frequency and sequence of its API calls **[19]**.

We chose the same algorithms as we did for static analysis, to compare their performance for dynamic analysis as well, the details of which are presented in Table 3. Just as the case with static analysis, Random Forest outperformed all the other algorithms in dynamic analysis with a detection rate of 94.7%.



26 <= 24.5 gini = 0.316 samples = 253 value = [315, 77] class = 0
gini = 0.0 samples = 39 value = [0, 47] class = 1
46 <= 35.0 gini = 0.159 samples = 214 value = [315, 30] class = 0
133 <= 70.0 gini = 0.471 samples = 18 value = [18, 11] class = 0
171 <= 122.5 gini = 0.113 samples = 196 value = [297, 19] class = 0
gini = 0.0 samples = 11 value = [16, 0] class = 0
28 <= 5.5 gini = 0.26 samples = 7 value = [2, 11] class = 1
159 <= 152.0 gini = 0.191 samples = 81 value = [125, 15] class = 0
158 <= 154.5 gini = 0.044 samples = 115 value = [172, 4] class = 0
gini = 0.0 samples = 1 value = [2, 0] class = 0
gini = 0.0 samples = 6 value = [0, 11] class = 1
186 <= 59.0 gini = 0.171 samples = 80 value = [125, 13] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1
gini = 0.0 samples = 30 value = [39, 0] class = 0
181 <= 107.5 gini = 0.057 samples = 85 value = [133, 4] class = 0
gini = 0.0 samples = 28 value = [54, 0] class = 0
145 <= 71.0 gini = 0.262 samples = 52 value = [71, 13] class = 0
169 <= 164.5 gini = 0.162 samples = 27 value = [41, 4] class = 0
gini = 0.0 samples = 58 value = [92, 0] class = 0
gini = 0.0 samples = 23 value = [37, 0] class = 0
48 <= 26.5 gini = 0.4 samples = 29 value = [34, 13] class = 0
gini = 0.0 samples = 5 value = [8, 0] class = 0
28 <= 56.0 gini = 0.193 samples = 22 value = [33, 4] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1
189 <= 98.5 gini = 0.369 samples = 28 value = [34, 11] class = 0
gini = 0.0 samples = 11 value = [18, 0] class = 0
172 <= 87.5 gini = 0.332 samples = 11 value = [15, 4] class = 0
gini = 0.0 samples = 16 value = [25, 0] class = 0
164 <= 65.5 gini = 0.495 samples = 12 value = [9, 11] class = 1
gini = 0.0 samples = 7 value = [13, 0] class = 0
159 <= 85.5 gini = 0.444 samples = 4 value = [2, 4] class = 1
gini = 0.0 samples = 3 value = [5, 0] class = 0
178 <= 76.0 gini = 0.391 samples = 9 value = [4, 11] class = 1
gini = 0.0 samples = 2 value = [0, 4] class = 1
gini = 0.0 samples = 2 value = [2, 0] class = 0
gini = 0.0 samples = 2 value = [4, 0] class = 0
gini = 0.0 samples = 7 value = [0, 11] class = 1

Figure 5: Sample Decision Tree of Dynamic Analysis Random Forest Model

The figure presented in the statement shows a sample decision tree from the random forest algorithm used for the dynamic analysis model. Decision trees are used to make predictions by following a series of logical steps based on the input features of a given sample. In the context of the random forest algorithm, multiple decision trees are created, and the final prediction is made based on the consensus of all the trees. Random forests are known for their ability to handle noise and overfitting, making them a popular choice for machine learning tasks. The decision tree shown in the figure provides insight into how the random forest algorithm makes predictions for the given dataset. The nodes of the tree represent decision points based on the input features, and the edges represent the logical branches that lead to the final prediction. By analyzing the decision tree, it is possible to gain a deeper understanding of the factors that are most important for predicting the outcome of the given task.

3.5 Static and Dynamic Model Stacking for Hybrid Analysis

As described in the preceding subsections, strings were used as features for training of the static analysis model whereas, the frequency and sequence of the API calls extracted by running the samples through CAPE sandbox were used as features for training the dynamic analysis model. In both of these instances, an 80:20 split ratio was used for training and testing the models **[20]**. For the creation of the hybrid analysis model, the outputs of both static and dynamic analysis models are stored together in a new CSV file which is then used as the training and testing dataset, a technique referred to as model stacking **[21]**. Table 3 shows the format of our cleaned training and testing dataset CSV file which has been created from the collection of the static and dynamic analysis models for the hybrid analysis model. As we demonstrate in the subsequent section, the combination of strings, which is a static feature, and the frequency and sequence of API calls which are the dynamic features of the analyzed APTs, has given us the hybrid model and also turned out to be the best representative of their *behavior*.

The combination of both static and dynamic features in the hybrid analysis model led to better results in detecting APTs. Model stacking was used to create this model by combining the outputs of the static and dynamic analysis models. The training and testing dataset was created by storing the outputs of both models in a CSV file. The split ratio of 80:20 was used for training and testing the hybrid model. Table 3 provides the format of the cleaned dataset CSV file used for training and testing the hybrid model.

The hybrid model was able to capture the behavior of APTs better as it utilized both static and dynamic features. The static features, which included strings, and the dynamic features, which included the frequency and sequence of API calls, were found to be the best representation of the behavior of the analyzed APTs. The use of both static and dynamic features allowed the hybrid model to identify patterns that were not visible in either of the models individually.

Overall, the use of machine learning algorithms and the combination of static and dynamic features in the hybrid model led to a significant improvement in detecting APTs. The results showed that Random Forest was the most accurate algorithm for both static and dynamic analysis, and was also the most accurate in the hybrid analysis model. The hybrid model was able to achieve a detection rate of 97.5%, which outperformed both the static and dynamic analysis models.

4 Performance Evaluation

In this chapter, we will delve into the details of the setup and analysis of our work. We will begin with a description of the datasets used for training and testing our models, along with the pre-processing steps involved in cleaning and preparing the data. We will then move on to a discussion of the findings of our static and dynamic analysis models, including the selection and evaluation of various machine learning algorithms. Finally, we will present our proposed hybrid analysis model and analyze its performance in detecting APTs.

Table 4 Dataset for Hybrid Analysis Model

Sample	Static Analysis	Dynamic Analysis	Classification
Sample 1	0.34	1	Malicious
Sample 2	0.4	0	Benign
Sample 3	0.7	1	Malicious
-	-	-	-
-	-	-	-

4.1 Setup

The simulation environment for this study comprises of a Ubuntu v18.04 distribution installed on a VMWare Workstation 16.0 virtual machine. The virtual machine was provided with 500 Gigabytes of storage, 8 Gigabytes of RAM with 4 processors. CAPE Sandbox v2 is installed in that virtual machine so that the dynamic analysis portion of the study can execute. There are multiple scripts written for the experiment to work alongside CAPE so that the tasks are performed alongside the sandbox. Each file in the dataset used for training the model was password protected/encrypted therefore a script was written to unzip and decrypt all the files which shared the same password "infected". Another script is used to fetch the results from submitted file in CAPE and extract the strings. The strings are then input to the static analysis model which determines if the submitted file is malicious.

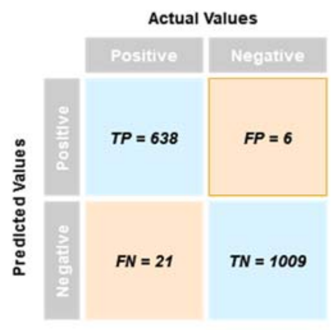
To conduct the study, a simulation environment was set up on a virtual machine running Ubuntu v18.04, with 500 Gigabytes of storage, 8 Gigabytes of RAM, and 4 processors. The dynamic analysis portion of the study was carried out using CAPE Sandbox v2, which was installed in the virtual machine. To work alongside CAPE Sandbox, several scripts were written for the experiment, including a script to unzip and decrypt all the files in the dataset that were used for training the model. The files in the dataset were password protected and encrypted with the password "infected."

Another script was used to fetch the results from the submitted file in CAPE and extract the strings. The strings were then input to the static analysis model to determine if the submitted file was malicious. A different script was used to extract the API calls from the same dynamic analysis run of the sandbox and stored them into the same CSV file that contained the extracted strings. The API calls and strings were input to the machine learning models separately, and the results were stored and input to the hybrid analysis model.

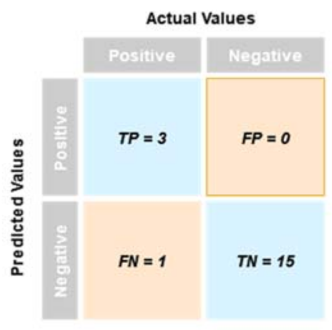
Table VI shows the general architecture for the training dataset of the hybrid analysis model. It displays the classifications of each sample according to both the static and dynamic analysis models alongside their true classification, shown in the rightmost column, to compare the predicted values during the testing phase. With this setup, the proposed hybrid analysis model was trained and tested using the cleaned dataset generated from the static and dynamic analysis models. The details of the evaluation of the hybrid analysis model are presented in the next subsection.

Another script extracts the API calls from the same dynamic analysis run of the sandbox and stores them into the same CSV file which contains the extracted strings. The API calls and the strings are input to the machine learning models separately whose results are stored and input to the hybrid analysis model. Table VI shows the general architecture for the training dataset of the hybrid analysis model showing the classifications of each sample according to both the static and dynamic analysis models alongside their true classification shown in the right most column to compare the predicted values with during the testing phase.

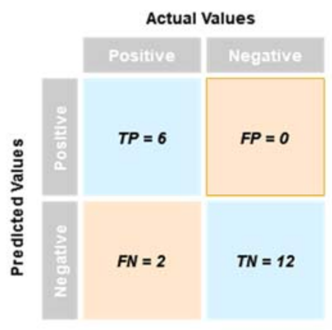
4.2 Analysis and Discussion



(a) Static Analysis with 1674 samples



(b) Dynamic Analysis with 19 samples



(c) Hybrid Analysis with 20 samples

The confusion matrix is a widely used tool in machine learning and statistical analysis that provides a comprehensive evaluation of the performance of a classification model. It is essentially a table that displays the number of correct and incorrect predictions made by the model for each class in the dataset. The matrix is constructed by comparing the actual class labels of the data with the predicted class labels produced by the model. The four possible outcomes of the classification task are true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), which are each represented in the confusion matrix. TP and TN correspond to correct predictions, while FP and FN correspond to incorrect predictions. The confusion matrix is an essential tool for evaluating the performance of a classification model, as it provides a detailed breakdown of the model's strengths and weaknesses.

The confusion matrix can also be used to calculate several important evaluation metrics, such as accuracy, precision, recall, and F1 score. Accuracy is the proportion of correct predictions to the total number of predictions, while precision is the proportion of true positives to the total number of positive predictions. Recall, also known as sensitivity, is the proportion of true positives to the total number of actual positives. F1 score is the harmonic mean of precision and recall and provides a more balanced evaluation of the model's performance. By using the confusion matrix and these evaluation metrics, machine learning practitioners can determine the effectiveness of their models and make necessary adjustments to improve performance. Overall, the confusion matrix is an essential tool for evaluating the performance of classification models and can aid in making data-driven decisions.

Before discussing the results of the study, we need to define the metrics used to assess the models.

4.3 Accuracy

Accuracy is the most common metric when gauging the effectiveness of a model. It is defined as the ratio of correct identifications to the sum of all outcomes as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP is True Positive, TN is True Negative, FP is False Positive and FN is False Negative. All of the individual values are shown in Figure 2 which displays the confusion matrix of each model and showing the testing sample size used for each model. The confusion matrix displays all the positives and negatives produced during the testing phase which are used for the metrics to evaluate the effectiveness of the model. Table II shows the accuracy of 10 different algorithms for the static analysis models. To determine the effects (if any) of the string lengths in the static analysis of the APTs in our dataset, we used 3 different minimum string length values i.e., 5, 7 and 9 characters. The minimum string length is a parameter which is used to exclude shorter strings which may not be relevant in the analysis and may also be detrimental to the accuracy of the models. Some examples of such strings are "eb@", "gy1x" and "cvbn" which may come up because of parsing errors. Garbage / irrelevant strings of bigger size may also get extracted however, their likelihood of occurrence is much smaller. This effect is evident from the increased accuracy of models when the minimum string lengths are increased. This effect can be observed with most of the applied classifiers with the exception of Support Vector Machine.

The paragraph discusses the key metrics used to evaluate the effectiveness of the models in the study. The first metric discussed is accuracy, which is the most common metric used to assess the model's performance. The formula for accuracy is provided, which involves the ratio of correct identifications to the total number of outcomes. The paragraph goes on to explain that the confusion matrix, which displays all the positives and negatives produced during the testing phase, is used to calculate the accuracy of the models. A table is presented which shows the accuracy of 10 different algorithms used for the static analysis models.

The paragraph then discusses the impact of string lengths on the accuracy of the models. The minimum string length is a parameter used to exclude shorter strings that may not be relevant in the analysis and can even be detrimental to the accuracy of the models. Examples of such strings are provided, which may arise due to parsing errors. The paragraph explains that the accuracy of the models increases when the minimum string length is increased. However, it is also noted that the Support Vector Machine is an exception to this trend.

Table 5 Accuracy of Random Forest Classifier for Dynamic Analysis

Criterion	**Result**
Accuracy	94.7%
Precision	100%
Recall	75%
F-1 Score	87%

Table 6 Accuracy of Random Forest Classifier for Static Analysis

Criterion	**Result**
Accuracy	98.4%
Precision	98%
Recall	97%
F-1 Score	98%

Table 7 Accuracy of Random Forest Classifier for Hybrid Analysis

Criterion	**Result**
Accuracy	92.4%
Precision	100%
Recall	89%
F-1 Score	94%

Advanced Persistent Threats are known to go to extreme lengths in order to achieve stealth. This includes extending their execution / dwell times spanning several days, weeks or even months to remain hidden, a technique often referred to as going *low and slow*. To quantify the impact of APTs' execution time, we configured our sandbox to two different settings of 5 minutes and 5 hours of sample execution. As expected, it is evident from the results shown in Table III that detection accuracy of all the algorithms increased by allowing more execution time. This improvement suggests that under ideal circumstances, APTs should be allowed maximum possible time to complete their execution over. However, we have successfully demonstrated the effect of increasing the execution time in the sandbox for analysis. Tables IV, V and VII are showing the results of each metric on the models created. The accuracy of the static, dynamic and hybrid analysis model are 98.4%, 94.7% and 92.3% respectively.

Advanced Persistent Threats (APTs) are notorious for their ability to remain hidden for extended periods of time, sometimes spanning several days, weeks, or even months. This technique is known as "going low and slow," and it enables APTs to achieve stealth and avoid detection. To quantify the impact of APTs' execution time, the authors of this text conducted an experiment in which they configured their sandbox to two different settings: 5 minutes and 5 hours of sample execution. The results showed that the detection accuracy of all the algorithms increased as more execution time was allowed. This suggests that under ideal circumstances, APTs should be given maximum possible time to complete their execution for analysis.

The authors also demonstrated the effect of increasing the execution time in the sandbox for analysis. Tables IV, V, and VII show the results of each metric on the models created. The accuracy of the static, dynamic, and hybrid analysis model are 98.4%, 94.7%, and 92.3%, respectively. Precision, which shows the quality of the positive predictions of the model, is 100% for the dynamic analysis model and 98% for the static analysis model. Although the dynamic analysis model has perfect precision, the precision of the static analysis model is considered more credible since the number of samples used to train and test the static analysis model is much larger than that of the dynamic analysis model.

Recall, which shows the proportion of actual positives identified correctly, achieved by the static analysis model is much greater than that of the dynamic analysis model, i.e., 97% versus 75%. The F1-score, which is the metric of the harmonic mean of precision and recall, shows that the static analysis model provides a greater F1-score of 98% than the dynamic analysis model, which is 87%. Table VII shows an F1-score of 94% for the hybrid analysis model, which stacks the results of the two models as features for the final hybrid model.

Overall, the results show that the dataset available for the dynamic analysis of APTs used in this study is not sufficiently large, and a larger dataset is likely to achieve better results. The authors are currently collecting features from a much larger dataset, and the performance of their models will be presented in future work.

4.4 Precision

Precision is the metric that shows the quality of the positive predictions of the model. It compares the number of true positives with the total number of predicted positives. The equation of precision is as follows:

$$Precision = \frac{TP}{TP + FP}$$

Table IV and V show the precision of our dynamic and static analysis models, respectively. The precision for the dynamic analysis model is 100% whereas the static analysis model has the precision of 98%. Although the dynamic analysis model seems to have perfect precision, however the precision of the static analysis model of 98\% is considered to be more credible. This is because the number of samples used to train and test the static analysis model is much larger than that of the dynamic analysis model i.e., 3000+ samples for static analysis versus just 100 samples for the dynamic analysis model. Table VII shows precision of hybrid analysis model which is at 100% but this may be due to the lack of samples in the dynamic analysis models which are greatly affecting the score.

4.5 Recall

Recall is the metric that shows the proportion of actual positives identified correctly. The dilemma with recall and precision is that one always has to consider the tradeoff between false positives and negatives. The equation of recall is as follows:

$$Recall = \frac{TP}{TP + FN}$$

From tables IV and V, we can see that the recall achieved by the static analysis model is much greater than that of the dynamic analysis model i.e., 97% versus 75%. This difference is again because of the smaller dataset size for dynamic analysis model. The recall of the hybrid analysis model in Table VII presents an 89% score also not being above 90% due to the low recall of the dynamic analysis model.

4.6 F1-Score

The F1-score is the metric of the harmonic mean of precision and recall. There is a tradeoff between precision and recall, the F1-score shows the quality of the model by combining both values in a harmonic mean which means that a model with a higher f1 score is ultimately the better model if we consider false positives and false negatives to be equally undesirable. The equation for F1-score is as follows:

$$F1 - score \ = \frac{TP}{TP \ + \ 0.5(FP + \ FN)}$$

Again, from tables IV and V, we can see that the static analysis model provides a greater F1-score of 98% than the dynamic analysis model which is 87%. Table VII shows an F-1 score of 94% for the hybrid analysis model as a result of stacking the results of the two models as features for the final hybrid model.

As evident from the results, the dataset available for the dynamic analysis of APT's that we have used is not sufficiently large and a larger dataset can reasonably be assumed to achieve better results. Therefore, we have started collecting features from a much larger dataset and the performance of our models will be presented in a future work.

5 Conclusion

In this research we have developed a hybrid analysis model to detect Advanced Persistent Threats (APTs), which are a particularly challenging type of malware to analyse considering their level of sophistication, analysis evasion techniques and the ability to achieve stealth. Our APT dataset comprised of 3500+ malware gathered from cyber-research's Github whereas 2800+ benign samples were collected from the cyber research Github. Our models were able to detect APTs with a high degree of accuracy approaching 92.3%, precision of 100% a recall of 89% and the F1 score of 93%.We also show that our ML models developed with the combination of strings which are a static feature of APTs along with the dynamic features of frequency and sequence of API calls are a good starting point for the detection of APTs.

In conclusion, our study demonstrates the effectiveness of a hybrid analysis model in detecting Advanced Persistent Threats (APTs). Our models achieved high levels of accuracy, precision, recall, and F1 score in detecting APTs. The combination of static features such as strings and dynamic features such as API call frequency and sequence proved to be a good starting point for APT detection. Our research provides a useful contribution to the ongoing efforts in the cybersecurity community to develop effective APT detection mechanisms.

6 Future Work

For our future work, we are working on increasing the number of features to be considered for model creation. These may include locale / language settings, imports and exports information, section names and information from the PE header along with registry keys, network communication artifacts and file system interactions.

Moving forward, there is scope to improve the accuracy of APT detection models by incorporating additional features. These could include locale/language settings, import/export information, section names and information from the PE header, registry keys, network communication artifacts, and file system interactions. The inclusion of these features may help to further refine the accuracy of APT detection models and enable them to identify increasingly sophisticated threats. Our future work will focus on exploring these additional features to further enhance the effectiveness of our hybrid analysis model.

7 References

[1] S. A. Ebad, A. A. Darem and J. H. Abawajy, ”Measuring Software Obfuscation Quality–A Systematic Literature Review,” in IEEE Access, vol. 9, pp. 99024-99038, 2021, doi: 10.1109/ACCESS.2021.3094517.

[2] N. Pachhala, S. Jothilakshmi and B. P. Battula, ”A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques,” 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2021, pp. 1207-1214, doi: 10.1109/ICOSEC51865.2021.9591763.

[3] Weijie Han and Jingfeng Xue, “MalInsight: A systematic profiling based malware detection framework”, Elsevier Journal of Network and Computer Applications 2019.

[4] Amin Azmoodeh, Ali Dehghantanha, “ Robust Malware Detection for Internet of Things Devices Using Deep Eigenspace Learning”, IEEE Transactions on Sustainable Computing, VOL. 4, NO. 1, 2019

[5] Xiyue Deng and Jelena Mirkovic, “Malware Analysis Through High level Behavior”, 11th USENIX Conference on Cyber Security Experimentation and Test 2018, ACM.

[6] Apiary. http://apiary.gtri.gatech.edu/.

[7] Galal, Hisham, “Behavior-based features model for malware detection”. Journal of Computer Virology and Hacking Techniques 2015.

[8] Damodaran, Anusha Di Troia, "A comparison of static, dynamic, and hybrid analysis for malware detection", Springer Journal of Computer Virology and Hacking Techniques 2015.

[9] R, Vinayakumar Alazab, Mamoun Kp, "A hybrid deep learning image-based analysis for effective malware detection", ELSEVIER Journal of Information Security and Applications 47 2019.

[10] Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T., Nov 2011. "Graph-based malware detection using dynamic analysis" J. Comput. Virol.

[11] Bacci, Alessandro Bartoli, Alberto Martinelli, Fabio Medvet, Eric Mercaldo, Francesco Visaggio, Corrado Aaron. (2018). "Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis."

[12] Aghakhani, Hojjat Gritti, Fabio Mecca, Francesco Lindorfer, Martina Ortolani, Stefano Balzarotti, Davide Vigna, Giovanni Kruegel, Christopher. (2020). "When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features".

[13] Demetrio, L., Biggio, B., Lagorio, G.,Roli, F., Armando, A., 2019 "Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries"

[14] R. Taheri, M. Shojafar, M. Alazab and R. Tafazolli, ”Fed-IIoT: A Robust Federated Malware Detection Architecture in Industrial IoT,” in IEEE Transactions on Industrial Informatics, vol. 17, no. 12, pp. 8442 8452, Dec. 2021, doi: 10.1109/TII.2020.3043458.

[15] L. Zhao et al., ”Shielding Collaborative Learning: Mitigating Poisoning Attacks Through Client-Side Detection,” in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 5, pp. 2029-2041, 1 Sept.-Oct. 2021, doi: 10.1109/TDSC.2020.2986205.

[16] APT Malware Datset by Cyber-Research: https://github.com/cyber-research/APTMalware

[17] A. Cuzzocrea, S. Leo Francis, and M. M. Gaber, “An informationtheoretic approach for setting the optimal number of decision trees in random forests,” in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2013, pp. 1013–1019.

[18] S. Buschj ̈ager and K. Morik, ”Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data,” in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 1, pp. 209 222, Jan. 2018, doi: 10.1109/TCSI.2017.2710627.

[19] X. Chen et al., ”CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection,” in IEEE Transactions on Information Forensics and Security, vol. 17, pp. 788-803, 2022, doi: 10.1109/TIFS.2022.3152360.

[20] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon, ”An Empirical Study on Data Distribution Aware Test Selection for Deep

Learning Enhancement", ACM Transactions on Software Engineering. Methodologies 31, 4, Article 78 (October2022), https://doi.org/10.1145/3511598

[21] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer, "A Survey on Distributed Machine Learning" ACM Computing Surveys. 53, 2, (March 2021), 33 pages. https://doi.org/10.1145/3377454

[22] F. Copty, M. Danos, O. Edelstein, C. Eisner, D. Murik, and B. Zeltser, "Accurate malware detection by extreme abstraction," in Proceedings of the 34th Annual Computer Security Applications Conference. ACM, 2018, pp. 101–111.

[23] S. Debray and J. Patel, "Reverse Engineering Self-Modifying Code: Unpacker Extraction," in Proc. of the Working Conference on Reverse Engineering (WCRE), 2010.

[24] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008, pp. 51–62.

[25] Y. Duan, M. Zhang, A. V. Bhaskar, H. Yin, X. Pan, T. Li, X. Wang, and X. Wang, "Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation," in Proc. of theNetwork and Distributed System Security Symposium (NDSS), 2018.

[26] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers, "Malware Target Recognition via Static Heuristics," Computers & Security, vol. 31, no. 1, 2012.

[27] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools," ACM Computing Surveys (CSUR), vol. 44, no. 2, 2012.

[28] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer, “Applying machine learning techniques for detection of malicious code in network traffic,” in Annual Conference on Artificial Intelligence. Springer, 2007, pp. 44–50.

[29] ENDGAME, “Endpoint protection,” https://www.endgame.com, (Accessed: 2018-12-26).

[30] Exeinfo PE, “Signature-based packer detector,” http://exeinfo.atwebpages.com/, (Accessed: 2019-01-07).

[31] C. Feng and D. Michie, “Machine learning of rules and trees,” Machine learning, neural and statistical classification, pp. 50–83, 1994.

[32] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin, “Compatibility Is Not Transparency: VMM Detection Myths and Realities,” in Proc. of the Workshop on Hot Topics in Operating Systems (HotOS), 2007.

[33] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, “Adversarial examples for malware detection,” in European Symposium on Research in Computer Security. Springer, 2017, pp. 6279.

[34] F. Guo, P. Ferrie, and T.-c. Chiueh, “A Study of the Packer Problem and Its Solutions,” in Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID), 2008.

[35] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, “Lemna: Explaining deep learning based security applications,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018, pp. 364–379.

[36] M. Hammad, J. Garcia, and S. Malek, "A Large-Scale Empirical Study on the Effects of Code Obfuscations on Android Apps and Anti-Malware Products," in Proc. of the International Conference on Software Engineering (ICSE), 2018.

[37] S. Han, K. Lee, and S. Lee, "Packed PE File Detection for Malware Forensics," in Proc. of the International Conference on Computer Science and its Applications (CSA), 2009

[38] I. U. Haq, S. Chica, J. Caballero, and S. Jha, "Malware Lineage in the Wild," arXiv preprint 1710.05202, 2017.

[39] O. Henchiri and N. Japkowicz, "A feature selection and evaluation scheme for computer virus detection," in Sixth International Conference on Data Mining (ICDM'06). IEEE, 2006, pp. 891–895.

[40] M. Hurier, K. Allix, T. F. Bissyand ́e, J. Klein, and Y. Le Traon, "On the Lack of Consensus in Anti-Virus Decisions: Metrics and Insights on Building Ground Truths of Android Malware," in Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2016.

[41] I. Incer, M. Theodorides, S. Afroz, and D. Wagner, "Adversarially robust malware detection using monotonic classification," in Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. ACM, 2018, pp. 54–63.

[42] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A Static, Packer-agnostic Filter to Detect Similar Malware Samples," in Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2013.