Taxonomy-based Intelligent Malware Detection Framework

Qublai K. Ali Mirza∗, Fatima Hussain‡, Irfan Awan†, Muhammad Younas‡, Salah Sharieh§,
∗ University of Gloucestershire, UK. (Email: qalimirza@glos.ac.uk)
† University of Bradford, UK. (Email: I.U.Awan@bradford.ac.uk)
‡ Oxford Brookes University, UK. (Email: m.younas@brooks.ac.uk)
§ Royal Bank of Canada, Toronto, Canada.
Email:(fatima.hussain, salah.sharieh)@rbc.com

Abstract—Timely detection of a malicious piece of code accurately, in an enterprise network or in an individual device, before it propagates and mutate itself, is one of the most challenging tasks in the domain of cyber security. Millions of variants of each latest malware are released every day and each of these variants have a unique static signature. Conventional anti-malware tools use signatures and static heuristics of malware to segregate them from legitimate files, which is not an effective technique because of the number of malware variants released every passing day. To overcome the fundamental flaw of operational techniques, we propose a framework that generalizes the static and dynamic malware features that are used to train multiple machine learning algorithms. The generalization of clean and malicious features enables the framework to accurately differentiate between clean and malicious files.

Index Terms—Malware, ML and Malware Detection, Malware Analysis, Machine Learning

I. INTRODUCTION

The evolution of malware over the past decade and their lethal proliferation has conspicuously challenged the effectiveness of anti-malware packages and other security mechanisms. Millions of malware variants are released every day that are modified versions of older malware and a vast majority of these variants are able to bypass the security mechanisms quite easily. Malware authors release their code, along with variant engines, which not only enable anyone, even with less programming knowledge, to generate their own version of an older malware. This approach is the reason behind exponential rise in malware propagation across the internet, it also makes it significantly harder for the current security mechanisms to eliminate or even mitigate the damage caused by such attacks.

The lack of novelty, improvisation, and excess amount of resource consumption are the foundational problems of current anti-malware packages. The general defence mechanisms faced by modern malware, which have the capability to dynamically mutate while propagating, is somewhat limited and predictable. The defence mechanism conventionally used is based around signature and heuristics detection, raising flags against defined rules, and traffic monitoring. All of the characteristics of defence mechanisms, commonly implemented in combination, are quite predictable and malware with polymorphic and metamorphic capabilities are evolving at such a pace that it is nearly impossible detect them with such predictable techniques.

Polymorphic malware continuously change their appearance while keeping the primary functionality intact. Most of these malware use encryption to pack the core functionality, which avoids any reverse engineering or heuristics-based analysis to be performed on the malicious code. Polymorphic malware carry a number of mutation engines (MtE), which are not malware themselves and do not possess any malicious code that raise any red flags by security mechanisms. The sole purpose of these MtEs is to change or mutate the encrypted stub of malware that contains the core functionality. When the stub is evolved by an MtE the old stub is deleted and the new one takes charge, this process is performed quite frequently, which gives very little time to the security mechanisms to identify and stop the malicious code before it is changed.

This paper presents a dynamic characteristic’ building framework supported by a unique combination of machine learning algorithms that are trained and tested against 2 million malicious files and more than hundred thousand clean files. This enables the framework to not only identify the variants of analyzed malware, but also enables the framework to dynamically form characteristics in real-time to differentiate between clean and malicious files effectively, without consuming a noticeable amount of system or network resources.

The rest of the paper is structured as follows; section II presents the critical evaluation of the related research and their effectiveness while section III discuss the anatomy of behavioural characteristics generation. In section IV, we present our continuous learning framework, which learns from customized combination of behavioural characteristics. In section V, we discuss the experiment setup, training of algorithms (through customized behavioural characteristics), along with the critical evaluation of results. Finally paper is concluded in section VI.

II. EVALUATION OF EXISTING TECHNIQUES

Combination of dynamic malware analysis techniques and machine learning tools proves to be a power full duo for malware classification and identification. However, most machine learning based malware identification methods are excessively feature dependent. It is very challenging to identify effective
A. Framework Architecture

The architecture of the proposed framework is divided into multiple layers as shown in Figure 1. Each layer is dedicated to produce specific results that are used by the following layer.
Each layer deals with the data from both clean and malicious files.

- **Layer 1 – Analysis Layer**: The Analysis Layer performs a thorough analysis of clean and malicious files and store the analysis results in separate repositories, both type of files are analyzed in the same manner, which means that the reports generated are in the same format and follow the same conventions

- **Layer 2 – Category Layer**: The Category Layer divides the feature-sets in meaningful categories that are present in both the file types, along with associating the related occurrences. The associations are then sent to the third layer

- **Layer 3 – Existence Layer**: The Existence Layer separates the instances in which each feature occurred in the behavioural analysis, along with the related sets. Intersection of both malicious and clean features is taken, along with the complement of both M & C. It is then sent to layer 4

- **Layer 4 – Machine Learning Layer**: The machine learning layer retrieves the customized dataset and applies ten-fold cross-validation to remove any bias. It then uses Support Vector Machine, Decision Tree, and Boosting on Decision Tree to train and test the framework

1) **Categorization Strategy**: The illustration of the layered framework presents a functionality breakdown of each layer and what they produce, along with their significance. As highlighted earlier, the first layer, which is the analysis layer, possess the analysis datasets of clean and malicious files. As discussed in the previous section, this dataset is generated as a result of thorough static and behavioural analysis of clean and malicious files and stored in a standardized JSON file format. The functionality of second layer is quite pivotal in this framework, which is the category layer. Each category layer stores the feature-set in five different features, namely; Network, CPU Usage, Dropper Files, API; Noti API, Certi API, Processes; Processes Generated, as shown in Figure 2. Additionally, API and Processes categories have subcategories; API Name & Frequency and Processes Deleted and Names, respectively. All these categories store the respective extracted features from both clean and malicious files. The primary reason behind categorizing the features and combining the extracted features is to generalize different behavioural patterns in clean and malicious files. Along with categorizing specific behavioural features, this layer links these generalized categories with two subcategories that are derived from Cat4 API and Cat5 Processes. The subcategories derived from Cat4 and Cat5 are also connected with first three categories; Network, CPU Usage, Dropper Files. The primary reason behind connecting the subcategories of Cat4 & 5 with other categories is the relevance of APIs and Processes with Network, CPU Usage, Dropper Files is quite significant and have a noticeable impact on these three categories. Category 4.1 holds the information about a specific API and its frequency. When Category 4.1 is linked with Cat1, it identifies the frequency of occurrence of a specific API with respect network traffic. It is then linked with Cat2 and the frequency of occurrence of a specific API and its impact on CPU consumption is monitored. Finally, it is linked with Cat3 and the association of dropping new files in a networked environment or in a single computer with the occurrence of a specific API. Moreover, Category 5.1, which is Processes Deleted and Name, is also associated with first three categories. The association of Cat1 with Cat5.1 monitors the name of processes deleted in a networked environment by a specific process. The association of Cat2 with Cat5.1 monitors the name of processes deleted in a single computer with the occurrence of a specific API. The impact on CPU consumption and whether it replaced a specific process and their previous and current CPU consumption. Additionally, the association of Cat3 with Cat5.1 monitor the files dropped or deleted by a specific process, along with their locations.

2) **Existence Identification**: Layer 3 is the existence layer, which integrates the categories, as illustrated in the category layer and takes intersection and complement to identify and differentiate the instance of each category for both clean and malicious feature-set. As mentioned in the previous section, the association of Cat1 with Cat4.1 and 5.1, for both clean and malicious feature-set, identify the existence with respect to Cat1. The later phase takes the intersection of existence between clean and malicious features from customized categories, along with a complement of malicious features in clean feature-set and complement of clean features malicious feature-set. The contents of final categories 1-5 are based on the intersection and complement of the aforementioned feature-set, as illustrated in figure. It is pivotal for the operational efficacy of the framework to include common and unique behavioural features of clean and malicious files, which will enable the framework to identify polymorphic and metamorphic malware. As discussed earlier, these types of malware dynamically evolve and attach themselves with legitimate files, as they propagate. Having a generalized combination of behavioural feature of clean and malicious files not only enables the framework to distinguish between clean and malicious files, it specifically empowers the framework against legitimate files that have malicious code embedded in it.
IV. IMPLEMENTATION

In this section we present the implementation details of our proposed framework. This framework is specifically designed for Portable Executable (PE) files, also known as .exe files, which are specific to windows environment. Therefore, all the files used in this research are windows-based executable, both; clean and malicious. The operations of every system or its implementation is based on a set of rules, which define the conditions of its functionalities and the parameters involved in their operations. The pseudocode presented in this section elaborates the condition that make this framework unique and efficient as compared to the ones discussed earlier.

As presented in the pseudocode, RepoC and RepoM hold the static and dynamic features of clean and malicious files respectively, acquired through static and dynamic analysis of both type of PE files. This leads to the existence module, which has three different phases; Categorization, Existence, and Combination. In the categorization phase, the union of clean and malicious features comprised of; network behaviour, CPU usage, dropped files, APIs, processes generated are stored in Cat1, Cat2, Cat3, Cat4, and Cat5 respectively. The final version of each category after this step stores the union of both clean and malicious features of each category mentioned earlier.

In the existence phase of this module, the virtually connected features, as shown earlier in the figure, are combined together. The union of Cat4 and Cat5 is combined with separately with the set of each category; Cat1, 2, 3, 4, 5. This combines the individual features of each category with the hierarchical features of Cat4 and 5, which involves both clean and malicious files. The combination phase of the existence module is the final phase and the information retrieved from this phase is used to train and test the machine learning algorithms in the next phase. In this phase, common features from both; clean and malicious feature-set are identified and stored in a separate set. The whole idea behind this step is to identify the differences a polymorphic malware makes to a legitimate file. The identification of common features and features exclusive to clean and malicious files enable

Algorithm 1 Malware Analysis

1: Input: C, M
2: Output: Cat 1, Cat 2, ..., Cat n
3: C ← Set of Clean PE Files
4: M ← Set of Malicious PE Files
5: while c ∈ C do
6:   while m ∈ M do
7:     S ← StaticAnalysis(c, m)
8:     D ← DynamicAnalysis(c, m)
9:     Repo C ← Repo C ∪ {C_s ∪ C_D}
10:    Repo M ← Repo M ∪ {M_s ∪ M_D}
11:  Existence Module - Categorization
12:  Cat 1 ← Cat 1 ∪ {RepoC_Net ∪ RepoM_Net}
13:  Cat 2 ← Cat 2 ∪ {RepoC_CPU ∪ RepoM_CPU}
14:  Cat 3 ← Cat 3 ∪ {RepoC_files ∪ RepoM_files}
15:  Cat 4 ← Cat 4 ∪ {RepoC_API ∪ RepoM_API}
16:  Cat 5 ← Cat 5 ∪ {RepoC_procs ∪ RepoM_procs}
17:  Existence Module - Existence
18:  Cat 1 ∈ Cat 1 ∪ {Cat4 ∪ Cat5}
19:  Cat 2 ∈ Cat 2 ∪ {Cat4 ∪ Cat5}
20:  Cat 3 ∈ Cat 3 ∪ {Cat4 ∪ Cat5}
21:  Cat 4 ∈ Cat 4 ∪ {Cat4 ∪ Cat5}
22:  Cat 5 ∈ Cat 5 ∪ {Cat4 ∪ Cat5}
23:  Existence Module - Combination
24:  Malfeat ← Set of Malicious Features
25:  Cleanfeat ← Set of Clean Features
26:  while Malfeat ∈ (Cat 1, 2, 3, 4, 5) do
27:    while Cleanfeat ∈ (Cat 1, 2, 3, 4, 5) do
28:      Cat 1.1 ← Cat1.1 ∪ {Cat1Malfeat ∩ Cat1Cleanfeat}
29:      Cat 2.1 ← Cat2.1 ∪ {Cat2Malfeat ∩ Cat2Cleanfeat}
30:    end while
31:  end while
32: end while
33: end while
the algorithm to differentiate even the slightest embedding of malicious code in a legitimate file. In the proliferation process in a single machine or in a network, multiple executables are used as hosts. In such a scenario, every executable even with a slightest of mutation needs to be identified before it can pass the mutation to the next executable. This technique will enable the algorithm to differentiate between clean and malicious files with significantly high accuracy.

V. RESULTS AND DISCUSSION

As the experiments are windows-based malware, Ubuntu is chosen as the host environment. The modules containing; clean and malicious files repositories, customized static analysis tool, Cuckoo sandboxed environment, repository layer comprised of clean and malicious files’ analysis reports. The existence layer is programmed in Python and connected with the previous two layers. The Python program is comprised of two modules; the existence layer and the machine learning layer. The existence layer connects with the file system, which holds the analysis reports and after performing the operations as discussed in the implementation section. It passes the finalized parameters to the next module that holds multiple machine learning algorithms that are trained and tested using these parameters.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>121523</td>
</tr>
<tr>
<td>Malicious</td>
<td>2068796</td>
</tr>
</tbody>
</table>

TABLE I: DISTRIBUTION OF BENIGN AND MALICIOUS FILES IN DATASET

Table I presents the distribution of clean and malicious files that are used for static and dynamic analysis, along with populating the feature-set in the existence layer that is later used for training and testing of algorithms. The large number of malware used in the experiment ensures that the framework is highly accurate and eliminates the existence of false-positives in the results. Also, the majority of the malware present in the dataset are variants of highly advanced polymorphic malware that have the ability to forge their presence as clean files, while propagating. Additionally, the dataset is comprised of different types of malware families, as shown in Table II. This also enables the framework to be more thorough when it comes to detection of different types of malware, along with the malware that share features from other malware families.

A. Results

In order to identify the effectiveness of the proposed framework, the initial experiment was performed on the same dataset but without the customization and categorization of the dynamic feature-set of clean and malicious files. In the first experiment, the same dataset of clean and malicious files was used. The features were generated using the same static and dynamic analysis tools and combined together to form a rich set of features. The primary idea behind the initial experiment was to identify the effectiveness of the combination of machine learning algorithms without using the taxonomy-based detection technique.

The results illustrated in Figure 3, present the comparison of different machine learning algorithms used in the framework that are trained and tested against the rich feature-set acquired from clean and malicious files without applying the categorization techniques. As illustrated in figure, the trained algorithms were able to detect malware with a decent rate but the rate still provide malware with an opportunity to escape detection, especially in real-time situations. SVM and decision tree were able to provide an accuracy of 0.79 and 0.60 respectively. However, when boosting was applied on the results of decision tree, the accuracy went up to 0.91, which is significantly higher than the actual accuracy of decision tree but still not an optimum solution for polymorphic malware. The same dataset was then used on the proposed framework, which has the same set of tools for static and dynamic analysis for clean and malicious files. However, when the rich feature-set from the analysis layer is generated, it is then reduced to a taxonomy discussed in earlier sections and then combined with relevant features. The experiment performed using the hypothesis proposed at the start and implementation discussed in the earlier section, produced the results presented in figure.

The results illustrated in Figure 4, present the outcome of the evaluation of the proposed framework. Once the aforementioned algorithms were trained against the customized characteristics, the detection accuracy significantly improved...
acquired after the categorization process was thoroughly rich. This means the final feature-set of a significantly large number of features that populated means that any result acquired went through a detailed analysis. This evaluation was based on more than 2.1 million files, which performed on the proposed framework. The dataset used in quite important to identify the importance of this evaluation, which makes the absolute accuracy even more significant. It is based on the malware executed in a live environment. The detection was not just based on static file system, it is the real-time detection based on the dynamic behaviour.

VI. CONCLUSION AND FUTURE WORK

To cater the problem of rapid malware evolution in modern malware, which are equipped with dynamic evasion techniques that enable the malware to continuously keep changing its appearance, code sequence, and even logic in some scenarios, there is a significant requirement of detection techniques that have the similar dynamics embedded in them. The framework proposed in this paper, not only has the capability to detect malware, it can has the capability to accurately differentiate between clean and malicious files. The proposed framework is primarily designed for polymorphic malware that have self-mutation properties and can attach themselves dynamically to multiple legitimate files, which means that the characteristics of the framework discussed in this paper can accurately differentiate between the legitimate and malware host files. The machine learning algorithms used are trained and tested on specially customised feature-set acquired from a quite large dataset of clean and malicious files. The end result illustrate the absolute accuracy of the framework against large set of polymorphic malware. Even though, the proposed framework demonstrated accuracy in malware identification, it is required to scale it to a level where it can cater the needs of a network. The modern polymorphic malware specifically target enterprise network, which means that any solution proposed should cover the domain of an enterprise network. In order to enhance the effectiveness of this framework, it is crucial to scale it to the level of an enterprise network and evaluate its effectiveness.

REFERENCES