Behavioral Analysis
of Malware
using Machine Learning

Anushtha Kalia, Arjun Sharma

11506, 11507

Cluster Innovation Center

University of Delhi

This report is submitted as the fulfillment of the internship paper for

semester V of BTech (IT & Mathematical Innovations)

at Cluster Innovation Center, University of Delhi

Abstract

With the recent increase in malicious attacks via ransomware and the losses incurred by various segments of the society, both in terms of data and money, the need of the hour is to find novel techniques to improve detection rates and performance. Current antivirus techniques rely on hash or signature comparisons via static analysis, which makes zero-day detection impossible. In order to cope with this many antivirus companies are now incorporating behavioral approaches.

In this project we have worked on how machine learning can be combined with behavioral analysis in order to cluster the malware samples into distinct similar-behavior families which can further facilitate a paradigm shift in detection techniques. Alongside proposing a behavioral profile based malware detection, we have also used machine learning to reveal inconsistencies associated with antivirus labels of malware.

Acknowledgement

The internship opportunity we had with Department of Computer Science, University of Delhi was a great chance for learning and professional development. Therefore, we consider ourselves very lucky as we were provided with an opportunity to be a part of it.

We also express our deepest gratitude and special thanks to the Prof. S.K. Mutto of DUCS who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep us on the correct path and allowing us to carry out our project at his esteemed organization.

Finally, we would also like to thank Cluster Innovation Centre for providing us with the facilities and the necessary resources to complete our work.

Table of Contents

List of Figures

List of Tables

1. Introduction

The Internet is an essential part of our life, but its security is seriously threatened by malware. The technological revolution during the last decade has introduced a significant increase in individuals that rely on computers and the Internet in order to communicate, store information and conduct business. But an increase in the users has been accompanied by an increase in malicious attacks. Millions of computers have been compromised by various malware types and families which are used to launch all kinds of attacks and illicit activities such as remote encryption, spam, phishing and DDoS attacks. In order to cope with this, antivirus (AV) vendors are focussing all their efforts in making use of state-of-the-art methods to detect these malicious softwares with limited number of false positives.

Current AV detection methodologies rely, among other methods, on reverse-engineering the malware executable in order to extract code-block patterns that will point to various malware types and families, which is known as signature-based analysis. Hashes are then generated for the each malicious executable and added to a vast database that is bundled in AV software for detection on the client machine and has the purpose of fast-detecting the threat. As the number of malware types increase with new potential gains, their implementation evolves and adapts in order to take advantage of new vulnerabilities. This high number results often represents a significant increase of data that needs to be provided by the AV vendor and affects the performance and accuracy of the AV solutions. Since it is necessary that the hash is present in the database in order to detect the malware, zero-day detection becomes impossible.

An alternate to this is behavioral analysis, where a file is classified as malicious based on the features it depicts during its execution. When a new file is to be checked, it is executed in an isolated environment, known as a sandbox, and its executional behavior is then monitored. If the recorded behavior resembles that of some known malware, then the given file is classified as malicious.

We worked on how features generated via behavioral analysis can be used for clustering malware types via Self Organizing Maps (SOM). We also emphasize on the fact that state-of-the-art methods often rely on the correctness of the labels and presume that a link exists between AV generated detection labels and the exhibited behavior of malware samples, which we show can be better associated to the cluster labels extracted from the trained SOM. We next compare the malware type-based classification results achieved via AV labels with those achieved via cluster labels in order to validate our assertion.

2. Literature Survey

Most traditional antivirus vendors use signature based malware detection. Shanklin S. D. et al (2002) [1] have patented an intrusion detection system that works on malware signatures stored in a high level syntax for future reference ease. Schmidt A.D. et al (2009) [2] proposed a two-fold model that extracted features from static analysis and then used those to classify android malware. Recently, Chua Z.L. et al (2017) [3] have proposed a new system called EKLAVYA in which they have trained an RNN to recover function type signatures from disassembled binary code. However, static analysis have its own disadvantages such as code obfuscation and polymorphism. Moser et al (2007) [4] have discussed these in detail.

Therefore, in recent years, research efforts have gone towards shifting from signature analysis to behavioral analysis, where the execution of a malicious sample is conducted and its behavior recorded in an isolated sandbox environment. Rieck K. et al (2011) [5] have developed an automated behavioral model capable of analysing and classifying more than 3,000 samples a day. Tian et al (2010) [6] attempted to use behavioral features to differentiate between malware and cleanware. Pirscoveanu et al (2016) [27] came up with a behavioral clustering model to cope with antivirus label inconsistencies. Chen S. et al (2016) [7] used a combination of behavioral features such as API calls and permissions to develop a machine learning based android malware detection framework. Bayer et al (2010) [8] proposed to improve the efficiency of dynamic analysis techniques by searching for modified API calls instead of running the whole analysis again for polymorphic viruses.

Nowadays, many antivirus vendors are using both static and dynamic analysis combined with machine learning for efficient detection of malware. Wang J. et al (2015) [9] proposed a model that combines static and dynamic analysis in order to detect javascript files. Graziano M. et al (2015) [10] have proposed a novel methodology which makes use of both static and dynamic analysis to automatically identify malware development cases.

But using machine learning for malware detection has certain drawbacks too. One major problem is the aging of classifiers. Jordaney R. et al (2017) [11] have addressed this issue by showing how a concept drift in malware can render these machine learning frameworks useless. Another drawback is that, malware creators often manipulate malwares to detect virtual machine environment to evade detection. Miramirkhani N. et al (2017) [12] have shown that features related to wear and tear can help in effectively detecting a sandbox environment which can help malware to easily evade detection. This takes us to the most critical drawback: the issue of evasion of the classifiers by adversaries. Xu W. et al (2016) [13] have proposed a method to

evaluate robustness of classifiers under attack. Their method involves stochastically manipulating a malicious sample (pdf malware) to find a variant that preserves the malicious behavior but is classified as benign by the classifier. Dhilung K. and Giovanni V. (2015) [14] have proposed a novel technique called MalGene which automates the extraction of evasive signatures after performing evasive malware analysis. While Smutz C. and Stavrou A. (2016) [15] have proposed a method, ensemble classifier mutual agreement analysis, that makes use of ensemble classifiers is detect many forms of classifier evasion without additional external ground truth.

Due to the existing drawbacks of malware detection techniques, Meng X. and Taesoo K. (2017) [16] have proposed a new perspective for detection of malicious documents called platform diversity. Making use of this concept, they came up with PlatPal which can hook to Adobe Reader to detect malicious documents. Some other techniques include malicious server identification through fingerprint generation as discussed by Xu Z. et al (2014) [17] and malicious network traffic identification to detect new or unseen variants of malware as discussed by Bartos K. et al [18].

3. Problem Identification and Discussion

The recent surge in malware attacks all around the world has made the issue of efficient malware detection a hot topic of discussion. Our project deals with how analysing malware on behavior based features and not only on features extracted from static analysis. In this section, we'll discuss what is malware and what are its types. We will also discuss the various labelling techniques used antivirus vendors and finally carry out a detailed discussion about the problem we have tried to solve through our project.

3.1. Malware and its types

Malware is an abbreviated term meaning "malicious software". This is software that is specifically designed to gain access or damage a computer without the knowledge of the owner. There are various types of malware including spyware, keyloggers, true viruses, worms, or any type of malicious code that infiltrates a computer. [Norton antivirus]

Generally, software is considered malware based on the intent of the creator rather than its actual features. Malware creation is on the rise due to the sheer volume of new types created daily and the lure of money that can be made through organized Internet crime. Malware was originally created as experiments and pranks, but eventually led to vandalism and destruction of targeted machines. Today, much of malware is created for profit through forced advertising (adware), stealing sensitive information (spyware), spreading email spam or child pornography (zombie computers), or to extort money (ransomware). Various types of malware exist in the wild. Below we have discussed about some types and families found in antivirus labels.

- **Adware** represents one of the least dangerous types as its only purpose is to display ads to the user. In order to provide the infected machine with ads that the user might be interested in, it logs information like browser history, search engines history or history of installed programs. Depending of the severity of the logging, Adware may be labeled by AV vendors as Spyware.
- **Spyware** represents a type of malware which installs itself without the permission of the user. Used to collect browsing history and tracking information it usually bundles with free software (Symantec, 2009) [19]. AV vendors also name this type, PUP, just because of the bundling with freeware.
- **Worm** represents a similar type as a Virus, being able to do the same amount of damage to an infected machine. The main difference is represented by its independence from

other software as it does not require a host program to attach itself to. A worm usually infects its target via exploits or vulnerabilities and it uses different transport protocols to spread and infect other machines (Cisco, 2015) [20].

- **Bot** represents a malware type that grants access of the infected machine to its master. This type can spread using Backdoors opened on the target by a Virus or a Worm and it is mostly known for using Internet Relay Chat (IRC) to communicate with its master. With multiple bots, Distributed Denial of Service (DDoS) attacks can be initiated that could block the services of the target by overwhelming it with requests.
- **Ransomware** represents a more sparse type of malware that takes control of the graphical interface and blocks the user from accessing its machine until a certain amount of money is paid. Most commonly, these types infect their targets via Trojan Horse.
- **Trojan Horse** is presented as software that the user might find useful, just like any other legitimate program. By opening the package, this malware releases other types of malware that will infect the machine, including key-loggers, account stealers etc. Compared with Viruses and Worms, Trojans do not replicate on their own but instead they require user interaction to do so. For this reason, this type is one of the most dangerous out there as it is usually detected when it has already infected the machine (Cisco, 2015) [20].
- **Virus** represents a malware type that can exhibit actions ranging from just showing random errors to taking the system in a Denial of Service (DoS) state. The main difference between a Trojan and a Virus represents the ability to self-replicate by becoming part of other legitimate software. These types are commonly spread by sharing files, disks or e-mails to which the virus has attached on (Cisco, 2015) [20].

Some companies follow the CARO convention which presents a standard format for malware labelling. In the next section we will be discussing the various techniques adopted by antivirus companies to come up with these labels.

3.2. Labelling techniques adopted by AV vendors

Malicious software has advanced in both complexity of coding and methods of obfuscation and polymorphism the detection more and more in order to cover all possible use cases. As technology evolved, the detection methods that AV vendors used have changed as well, improving their accuracy for new-released malware. Methods currently used are:

- **Signature Based** - Most commonly used by AV vendors to fast detect known malicious software. There exist two different signature-based methods:

Hash Signatures are used by AV vendors to compare the hash of the file with known malware hashed signatures. This provides a fast detection rate only if the malware exists in the AV database or if the malware has not changed. AV vendors commonly use MD5 message-digest algorithm that provides a very accurate result. For example if a single character is changed in a file-name, the MD5 will return a totally different hash [21].

Byte Signatures represent a sequence of bytes that are present in executable files or data streams. This method can accurately detect malware families by analysing sequences in data streams of an executable file [22].

- **Heuristic Based** - The method attempts to detect malware by simulating the run-time of the suspected executable and determine its intent. This approach is useful when, for example, an updated version of a known malware is released and it is mostly used alongside signature based detection to form a completed real-time protection in terms of known and new malware threats.

- **Behavioral Based** - A more advanced method of detection that requires the malware to run and infect a machine in order to record its actions. Behavioral based detection is done in a confined environment to prevent the spread of the malware. Information collected is usually used for classification or clustering of malware behavior [23].

Modern AV programs use all of the above methods to provide a more complete solution to the customer for a higher and more accurate detection of malware. AV vendors do not only detect malware but also label them according to patterns seen when reverse engineering the executable. We will now discuss the problem that we will be addressing in our project.

3.3. Problem Formulation

As discussed in the previous sections, many antivirus companies use signature based analysis to identify malware while some adopt behavioral analysis using a sandbox based environment. There are also researchers who use machine learning to automate the process of behavioral analysis and use antivirus labels to facilitate these sorts of models. In this project we'll be showing the inconsistencies associated with antivirus labels by performing and analyzing various experiments and will further propose a behavior based malware detection model automated using machine learning without using antivirus labels.

4. Data set generation

To start with malware analysis, we'll need a dataset consisting of various malware samples and features associated with those samples. This section deals with generation of dataset which will consist of behavior based features and we will further use this dataset to perform our analysis.

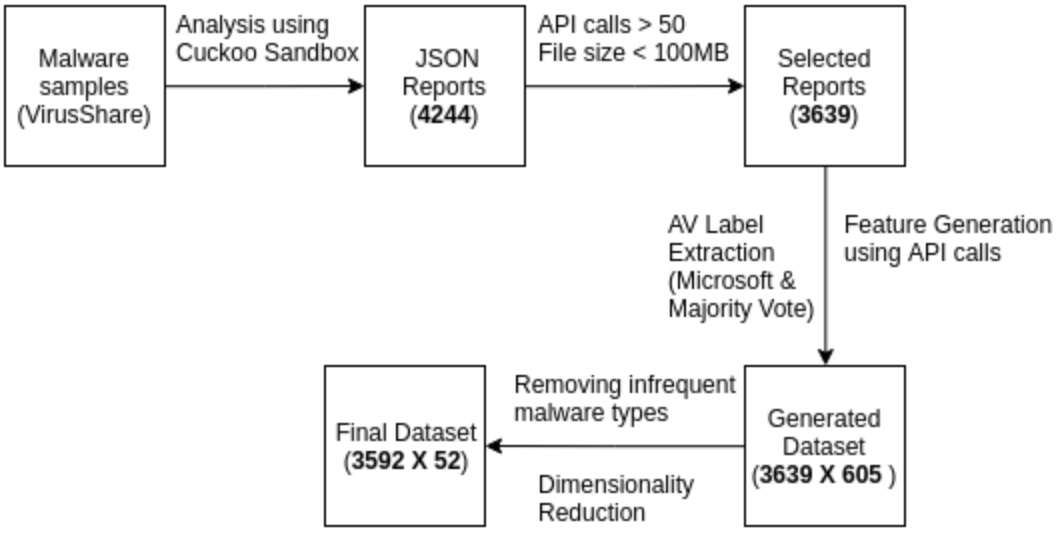


Figure 4.1: The workflow of the Data Generation module

4.1. Analyzing malware samples

For performing any machine learning task, it's important to have a dataset. For this project we made our own dataset using malware samples from VirusShare. The samples primarily consisted of executables, html documents, javascript files and binaries. VirusShare is a repository of malware samples to provide security researchers, incident responders, forensic analysts, and the morbidly curious access to samples of live malicious code. Note that we needed a dataset consisting of behavioral features of various malwares, hence, acquiring only malware samples wouldn't have sufficed.

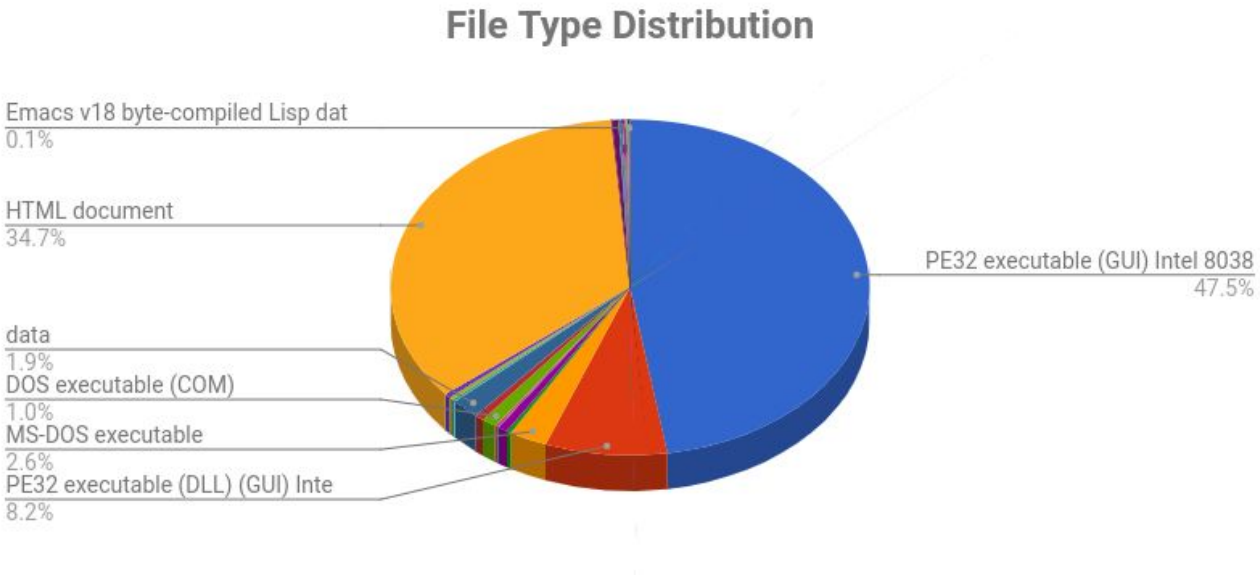


Figure 4.2: Pie chart representing file types found in the dataset

For analyzing the malware sample, we used an open source automated malware analysis sandbox called Cuckoo. By definition, a sandbox is often used to execute untested or untrusted programs or code, possibly from unverified or untrusted third parties, suppliers, users or websites, without risking harm to the host machine or operating system. It is most frequently used for testing. In this sense, a sandbox is really nothing more than a test or staging server. The goal is to give enough access to allow proper testing while not exposing critical systems to potentially flawed code. In practice, this can take the form of a separate server or mirrored production environment. Cuckoo sandbox can automatically run and analyze files and collect comprehensive analysis results that outline what the malware does while running inside an isolated operating system. It can retrieve the following type of results:

- Traces of calls performed by all processes spawned by the malware.
- Files being created, deleted and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.
- Screenshots taken during the execution of the malware.
- Full memory dumps of the machines.

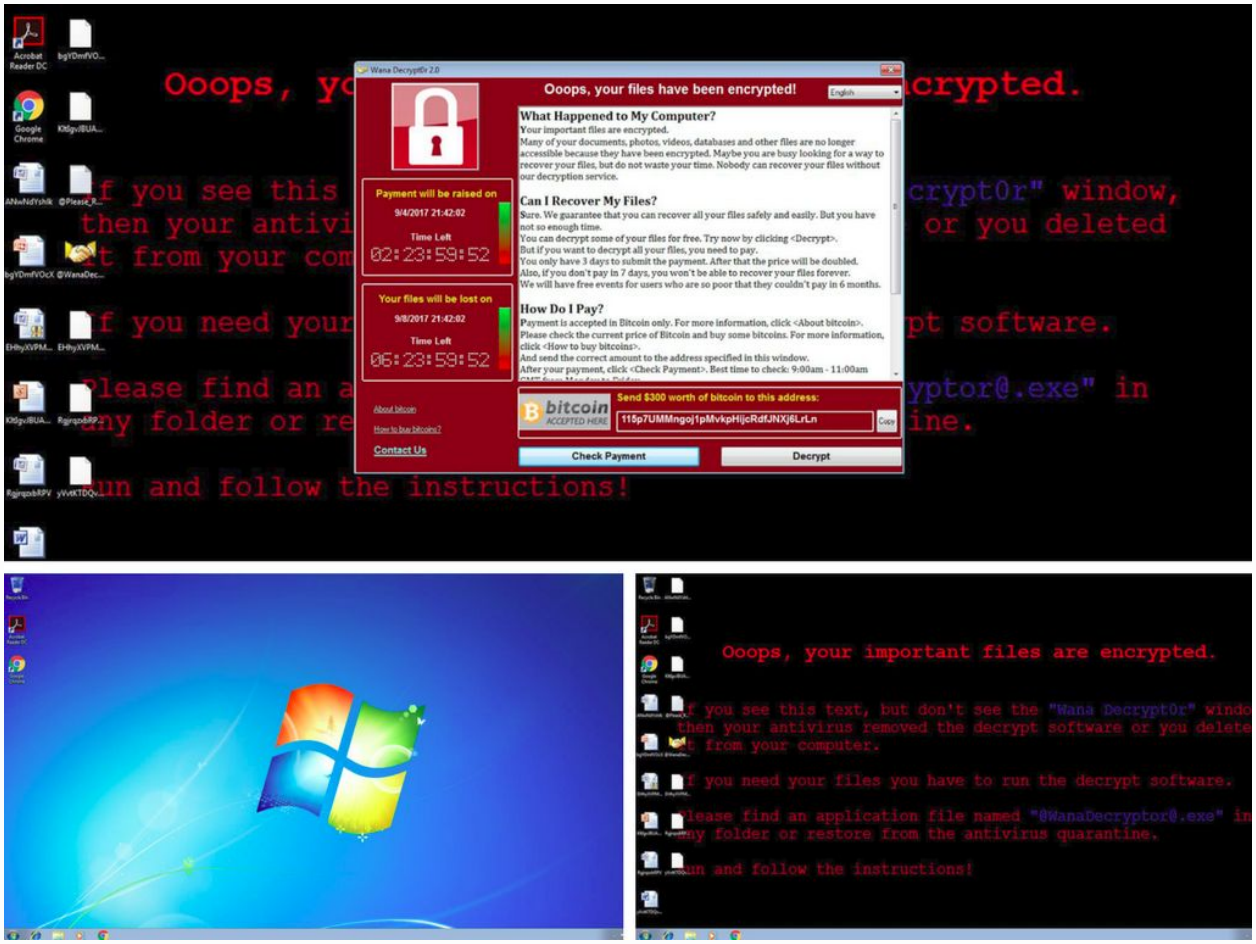


Fig 4.3: Screenshots depicting execution of WannaCry inside Cuckoo

We had submitted around 8350 samples to cuckoo but due to various hindrances such as system crashes, only 4513 malware analysis reports were generated. The analysis was run on two physical machines (Dell Optiplex 9010) with the following specifications:

Table 4.1: System specifications

Distribution	Ubuntu 16.04 LTS
Memory	7.7 GiB
Processor	Intel® Core™ i7-3770S CPU @ 3.10GHz × 8
Graphics	Intel® Ivybridge Desktop
OS-type	64 bit
Disk	1 TB

For storing the reports, we had synced our systems with one.ducic account which had unlimited storage space. A typical report generated by cuckoo has a size of about 100MB. Our json report had the following format:

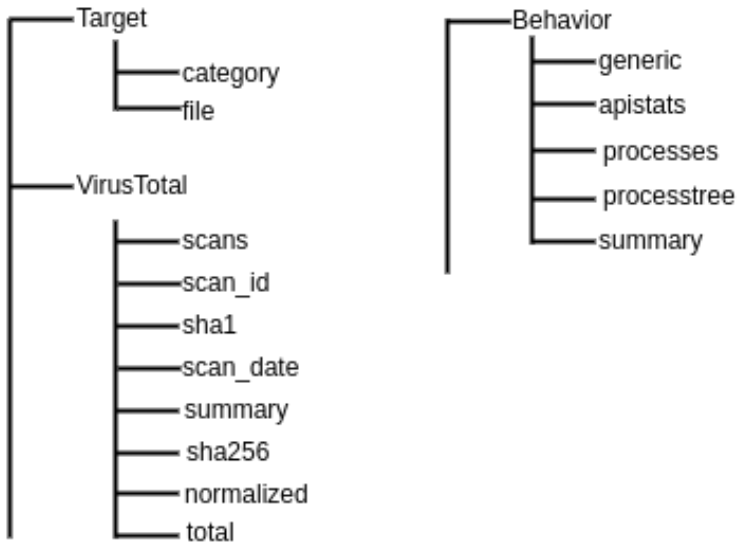


Figure 4.4: Format of the generated JSON report

4.2. Feature Representation

The json reports we obtained through automated analysis of the cuckoo sandbox contained behavior based information in form of API calls. Let's have a look what API calls are and what role do they play in behavioral analysis.

4.2.1. API calls

Microsoft Operating System (OS) provides several ways to interact with the system in order to make interaction and exchanging information a simple task for programs and applications. Even though this makes the task of connecting and using features of the OS easy, it also provides easy access to hackers to do so as well by using malicious software to take control or gain sensitive information from the infected machine.

As APIs define the interaction between software and OS, each call must provide additional information about the task it wants to do. This information is passed to the OS via arguments. If, for example, the software does not have enough privileges to run a certain command or task, the API call will fail and will provide the software with a coded response explaining the failed call.
Going a level of detail deeper into the problem, in this project, the status of the API call along with the return code provided by the Operating System if the call had failed, were collected and analyzed carefully. By using the full list of error codes provided by Microsoft, a correlation

between a failed API call and the reason of failure were extracted to construct the behavior profile of the malicious software.

4.2.2. Removing Junk Reports

Before parsing the json file to obtain API calls describing the behavior of a malware, it was important to filter out the junk reports i.e the reports without behavioral features. Junk reports were produced when a particular malware sample failed to execute inside the cuckoo sandbox. Samples that have less than 50 API calls, denoted by the sum of failed with passed APIs, were excluded from the representation. Assuming that no malicious action can be done with less than **50 API calls**, the excluded samples could contain no information that could help the ML algorithm to better discriminate the types. Due to memory and processing limitations, all JSON files with size greater or equal to **100MB** were removed. We finally obtained 3639 json files after performing the filtering process.

4.2.3. Generating the dataset

For building the behavioral profile of each malware, we needed the passed api, failed apis and the corresponding error code frequencies. The representation of the object which contained information regarding the API calls is presented below:

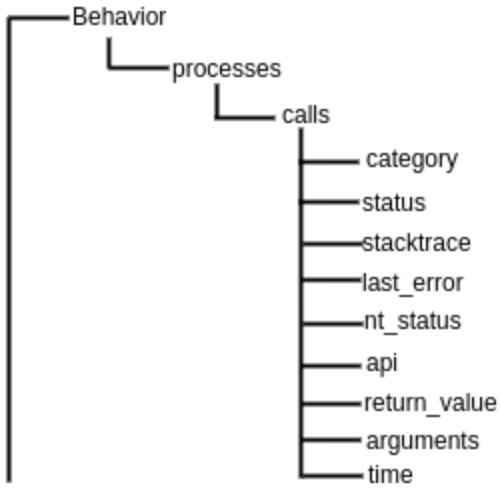
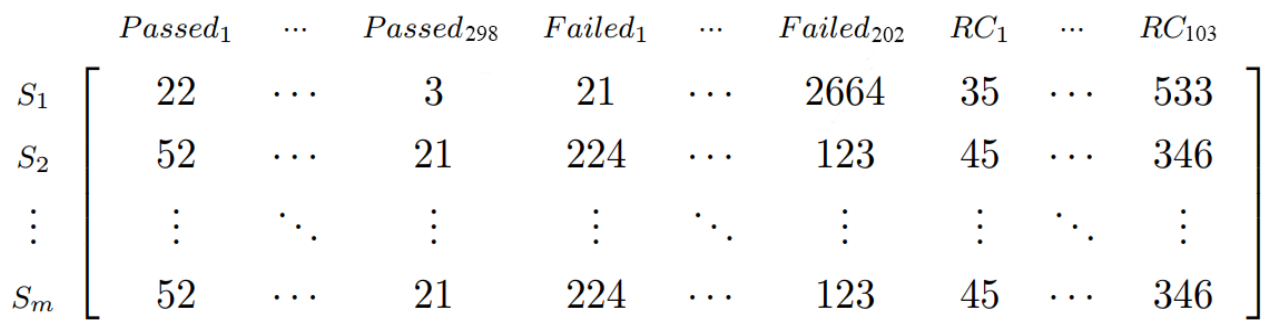


Figure 4.5: JSON object where information regarding API calls is available

The passed api calls had *status* equal to one while the failed api calls had *status* equal to zero. If an API call failed to execute, an error code was returned which we accessed via *nt_status*. The representation of the behavioral profile is given below:



	$Passed_1$	$\cdots$	$Passed_{298}$	$Failed_1$	$\cdots$	$Failed_{202}$	RC_1	$\cdots$	RC_{103}
S_1	22	$\cdots$	3	21	$\cdots$	2664	35	$\cdots$	533
S_2	52	$\cdots$	21	224	$\cdots$	123	45	$\cdots$	346
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
S_m	52	$\cdots$	21	224	$\cdots$	123	45	$\cdots$	346

Figure 4.6: Frequency matrix consisting of API calls and return codes

After parsing the json reports, we obtained 298 passed apis, 202 failed apis and 103 error codes. The generated dataset represents behavioral features corresponding to malware sample. Below is the table consisting of API calls with large number of frequencies:

Table 4.2: Some frequent API calls

	Name	Frequency
Passed APIs	NtClose	2670490
	NtWriteFile	1634460
	LdrGetProcedureAddress	1214009
	GetSystemMetrics	1160397
	NtCreateFile	825545
Failed APIs	RegQueryValueExW	601219
	FindWindowA	354284
	NtCreateFile	317321
	NtReadFile	216416
	RegOpenKeyExW	216393

We will now discuss the procedure we followed to extract antivirus labels for each malware sample.

4.3. Label extraction

Cuckoo provided us with detection results from 57 Anti-Virus programs along with their appropriate labels for each malware sample which was fed to it. This information was available via the *scans* object of the json file. We will now extract the labels corresponding to each malware sample so that we can highlight the inconsistencies of antivirus labels and find out which labels should be used in order to achieve the best predictive performance when using the data in an ML algorithm.
Below is a table showing labels assigned by different antivirus vendors for a particular malware:

Table 4.3: Inconsistencies amongst AV vendors

Anti-Virus Software	**Label**
MicroWorld-eScan	Gen:Variant.Kazy.18883
nProtect	Trojan-Downloader/W32.DNSKrab.133120
CMC	Trojan-Downloader.Win32.DNSKrab!O
CAT-QuickHeal	Trojan.Vundo.Gen
McAfee	Vundo.gen.dp
Zillya	Downloader.DNSKrab.Win32.210
SUPERAntiSpyware	Trojan.Agent/Gen-Kundo
TheHacker	Trojan/Downloader.DNSKrab.abz
K7GW	Trojan-Downloader (0028535d1)
K7AntiVirus	Trojan-Downloader (0028535d1)
Baidu	Win32.Trojan.WisdomEyes.151026.9950.9972
Cyren	W32/Virtumonde.BY.gen!Eldorado
Symantec	Trojan.Vundo!gen8
Avast	Win32:MalOb-DQ [Cryp]
Kaspersky	HEUR:Trojan.Win32.Generic
F-Secure	Gen:Variant.Kazy.18883
McAfee-GW-Edition	Vundo.gen.dp
Microsoft	TrojanDropper:Win32/Vundo.J
Fortinet	W32/Kryptik.IHN!tr
AVG	Cryptic.DQP
Yandex	Trojan.Kryptik!VftEADljv6U

Caro naming convention:

```
<malware_type >://<platform >/<family_name >.<group_name >
```

Microsoft has adopted the standard CARO naming convention while other anti-viruses use other type of naming conventions. This causes inconsistency and makes it tedious to extract the malware family name or malware type. For this project, we extracted the Microsoft anti-virus label corresponding to each malware sample. We chose Microsoft because it is the one of the best antivirus based on performance (according to AVast) and extracting the malware family name from its labels is relatively easy.

To have another alternative, we extracted labels corresponding to the majority vote of all antivirus softwares for a particular sample. For this, we used a tool called AVclass which used a vocabulary to find out the appropriate malware family name corresponding to each malware sample.

We obtained a data set of 3639 rows consisting of behavioral features and antivirus labels (majority vote and Microsoft). Removing samples with label frequency less than 5 reduced the number of rows in the dataset to 3592.

4.4. Dimensionality Reduction

Dimensionality reduction represents the extraction process of a subset from the original feature list, either via raw feature selection or by combining a set of them (feature extraction), that can better discriminate the data. This implies that the irrelevant and redundant features will be prone to removal as they hold no relevant information that can improve the model's predictive performance.

Reduction of dimensionality in unsupervised learning represents an important problem due to the absence of labels. We have used the algorithm Principal Component Analysis to reduce the dimensionality of our data set. The detailed description of the algorithm will be given in the upcoming section, we will now discuss the dataset we obtained after using this algorithm.

Our dataset had 603 features and therefore, it was really difficult to do any processing or perform computations on that data. Hence, PCA helped us by retaining 99% of the variance of the

original dataset while minimising the number of features to be taken into consideration. For retaining 99% variance, we chose 50 principal components which can be seen from the figure given below:

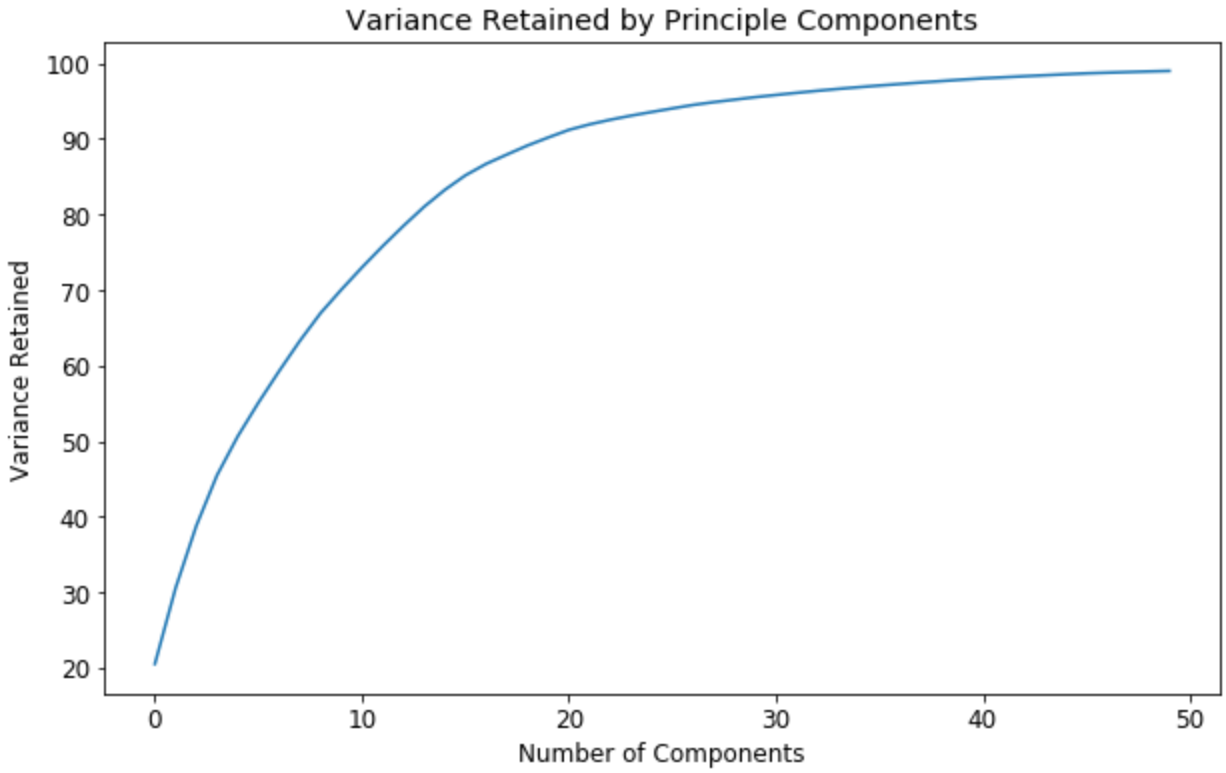


Figure 4.7: Cumulative variance retained by different number of principal components

Therefore, the dimensions of the final dataset we obtained after performing all the preprocessing is **3592 X 50.**

5. Algorithms Used

In this section we will discuss the various algorithms that we have used in this in project. We will majorly discuss the theory behind each algorithm which were used to develop our model.

5.1. Principal Component Analysis

In the last section, we used principal component analysis in the preprocessing step of our data set generation. Now, we will carry out a detailed discussion about this algorithm. Principal Component Analysis (PCA) has the main purpose of reducing the dimensionality of the data by combining features and creating new features called principal components based on variance or covariance. It is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences. As it keeps the subset of features that are accountable for the highest variance it can also be used for feature selection, which we opted for as well. PCA simply uses linear algebra to compute and determine the components with the highest variance. The steps below are followed:

1. The whole d-dimensional data-set is taken. If classes exist, they are ignored.
2. The mean of each column representing the features is computed.
3. The covariance matrix of the data set is computed.
4. The eigenvectors and corresponding eigenvalues are determined using the means and covariance matrix.
5. The eigenvectors are decreasingly sorted by their eigenvalues.
6. The n largest eigenvectors, representing the principal components, are selected.

A representation of the 2-dimensional visualization of our dataset, and its corresponding first and second principal components can be seen in figure 5.1.

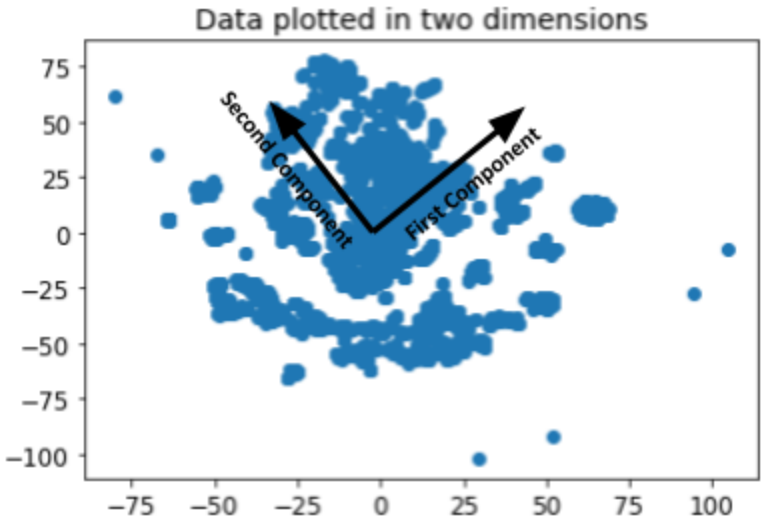


Figure 5.1: Two dimensional representation of behavioral data

5.2. Cluster Selection: Elbow and Gap Statistics

The decision of the number of clusters to be used on a data-set is the most frequent problem in clustering. This problem is also referred as finding k, corresponding to algorithms like k-means.

There are multiple methods that try to find the best amount of clusters required by the data-set, however, only two have been chosen for review as they provide good performance for large number of samples. They are:

- Elbow Method
- Gap Statistics

5.2.1. Elbow Method

The Elbow method represents one of the simplest ways of trying to achieve the best number of clusters by examining the variance as a function of the number of clusters. As many other methods, it requires the use of a clustering algorithm, like k-means, to perform clustering and return the amount of variance each cluster produces. It is expected that after the correct number of clusters reaches the marginal gain, or the inertia, will increase or will decrease at a much smaller rate than with less clusters.

In order to calculate the variance, Euclidean Distance is used to determine the distance between two points. The resulting calculation will represent a 2-dimensional vector containing information about inter and intra cluster similarities. In other words, it will contain information about a point and the similarities with points inside the same cluster along with similarities with points belonging to different clusters. The equation can be given as:

$$D_k = \sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2 = \sum_{x_i \in C_k} \sum_{x_j \in C_k} ((x_j - \mu) - (x_i - \mu))^2 = 2n \sum_{x_i \in C_k} (x_i - \mu)^2$$

where,

- C_k represents the cluster k
- x represents some point in cluster C_k
- D_k represents the intra-cluster sum of cluster k
- n denotes the total number of samples in cluster k

The above equation has determined that the intra-cluster difference between points is denoted by the squared difference between each point of the cluster and the mean. The final *D*, representing the distance measure will contain a matrix of the summed distances between the points and the center of cluster. However as the clusters may contain different number of samples, the distance matrix must be normalized. This will reflect a fair calculation for each sample and it is done using the following equation:

$$E_K = \sum_{k=1}^{K} \frac{1}{2n} D_k$$

where,

- E_K - Explained variance using K number of clusters
- D_k - Euclidian distance matrix for cluster k

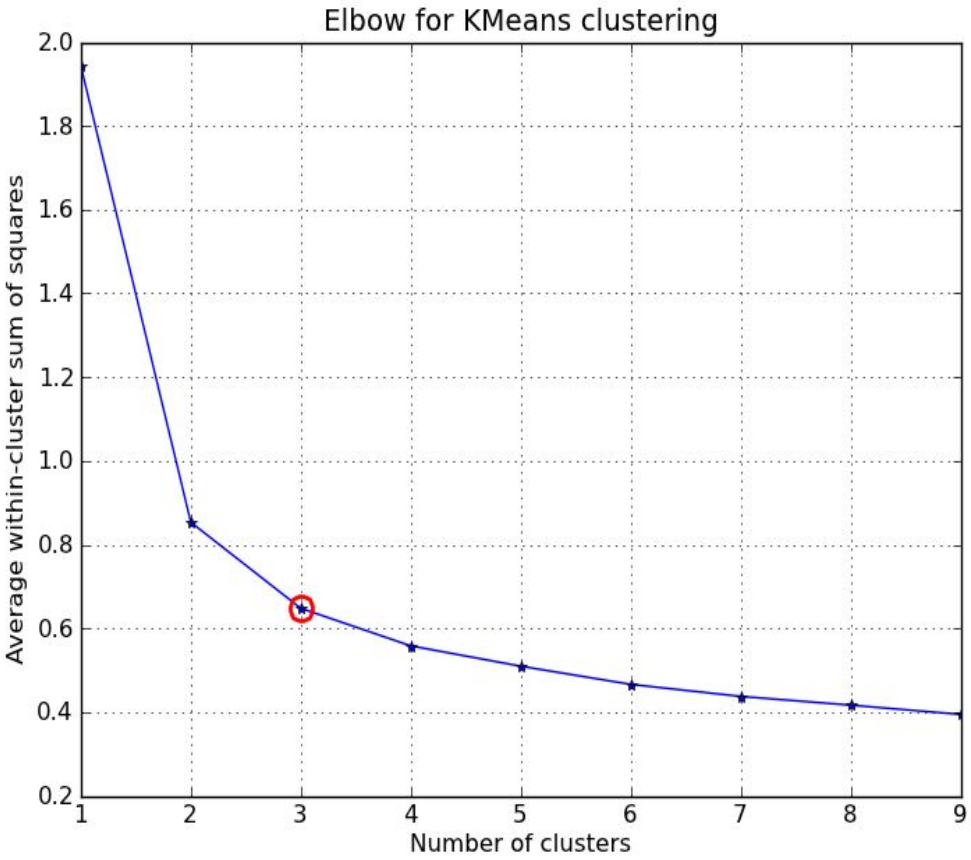


Figure 5.2: Sample inertia vs number of clusters graph

Explained variance variable E_k now contains a single value which denotes the metric of evaluation using K number of clusters. The Elbow method should iterate through a range of clusters in order to determine the optimal solution. It is expected that the explained variance should drop at a fast pace until it reaches the optimal solution, after which the values should decrease slower than before.

5.2.2. Gap Statistics

Gap statistics represents a continuation of the Elbow Method, presented by Tibshirani et al. (2001) [24]. The idea behind this approach is to find a standardized comparison method between the log-likelihood of the data and a null reference. The null reference can be described as a random distribution with no obvious number of clusters. The estimation of the number of clusters is represented by the lowest $log\, E_k$ in comparison with the null reference curve. The gap statistic is calculated using the following equation:

$$Gap_n(k) = E_n\{log\, E_k\} - log\, E_k$$

where,

- E_K represents the explained variance of K clusters
- E_n represents the reference distribution

From this equation the optimal number of clusters can be extracted, defined as being the smallest k where the gap statistic of the previous value is larger than the gap statistic of the current value. This can be understood via the following equation:

$$K = argmin \forall_k Gap(k) \geq Gap(k+1) - s_{k+1}$$

where,

- K represents the number of clusters
- k represents the current number of clusters for which gap statistic is calculated
- s_k represents the simulation error of the reference distribution

5.3. Unsupervised Learning

This section will describe the approach used in this project for applying unsupervised Machine Learning (ML) on the data. The goal of this approach is to find hidden structures or groups of data. It does so by using measures of similarity that are usually calculated using Euclidian distance. There exist a large variety of clustering algorithms that aim at generating groups from the input data-set as accurately as possible. Different types of clustering algorithms can be distinguished by the methods used and can be classified as follows:

- Based on similarity or distance measure between input points. This method represents one of the traditional approaches and can be seen in k-means.
- Probabilistic where the sample set is assumed to be generated by some distribution. The Gaussian Mixture Model (GMM) is an example of such probabilistic method which is based on the EM algorithm.
- Hierarchical. This method can be further split into Agglomerative and Divisive.

In this project we have considered two algorithms namely K Means and Self Organizing Map. We will go over both the algorithms and analyze which one should be chosen for clustering.

5.3.1. K-means

The kmean algorithm is directly based on an article published by Hartigan and Wong(1979) [25]. The k-means clustering method aims at minimizing the average squared distance between points that are part of the same cluster. Given k, denoting the number of clusters, X denoting the data points and V denoting the set of centers, k-means algorithm does the following:

- Randomly select k cluster centers.
- Calculate the distance between each data point X_i and cluster center V .
- Assign the data point to the cluster center whose distance from the center is minimum of all other cluster centers.
- Re-calculate new cluster centers V .
- Re-calculate the distance between each data point and the newly obtained centers from step 4.
- If algorithm has converged stop, otherwise repeat from step 3.

The algorithm is easy to understand and performs clustering at a faster rate in comparison with other complex clustering algorithms. However, it does require to have a data set with well separated points to return an accurate grouping of the data.
As there is no indication of such separation in the malware data, k-means is not guaranteed to be the best possible solution in clustering malware types or families. It also has to be mentioned that the results can be different with each run of the algorithm as the initial step of the k-means algorithm depends on selecting a random location for each center. This problem can be resolved by controlling the randomness using a pseudo-random seed by selecting the centers from a known sequence of numbers. The k-means implementation that uses seeds to select the centers is called kmeans++ and can lower the complexity of the algorithm, see Arthur and Vassilvitskii (2017) [26].

5.3.2. Self Organizing Map

The Self-Organizing Map (SOM) algorithm is based on the book published by Professor Teuvo Kohonen, see [Kohonen, 1989]. SOM was developed as a competitive learning algorithm, part of Neural Networks that contains a hidden layer. The algorithm has two phases: *Learning phase* and *Prediction phase*. The prediction phase works just like a classification algorithm where new data points are assigned to clusters generated in the Learning phase of the algorithm. The steps are:

1. Create the map using a predefined grid size of n by m. There will be $n * m$ number of nodes. Dimensionality of the data is non-linearly reduced to fit the grid size.
2. Initialize the weight of each node.
3. Choose a random input vector and present it to the map.
4. Find Best Matching Unit (BMU). The distance between the input vector and the weight of each node is calculated to determine the BMU.
5. Calculate the radius of the neighbors around the found BMU. With each iteration, the neighborhood size should decrease.

6. Modify the node weight such that its properties are closer to the BMU. The nodes further away from the BMU are slightly changed in comparison with the nodes closer to the BMU.
7. Exit if algorithm has converged, otherwise repeat from step 3.

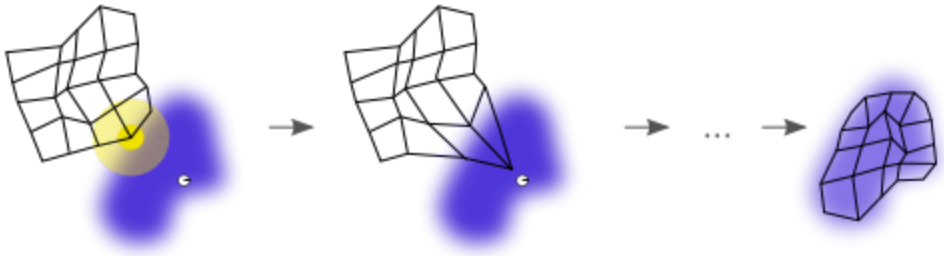


Figure 5.3: Self Organizing Map adapts to the topography of the dataset

6. Methodology

The flowchart corresponding to our behavioral analysis model is given in figure 6.1. With the reduced dataset in our hands, we'll now proceed towards our proposed model. First, we'll be looking at how the ideal number of clusters for the problem will be chosen using both, the elbow method and gap statistics. Next, we'll be comparing the 2 clustering methods that we previously discussed: K-means and SOM, by evaluating their respective absolute errors and convergences associated with Microsoft and Majority Vote labels. Then, we'll assess the ability of a Random Forest Classifier to classify unseen malware samples across Microsoft, Majority Vote and cluster labels when trained on those respective labels. Finally we'll discuss about an automated malware behavioral type classifier which will work on the extracted cluster labels, independent of the AV-assigned labels.

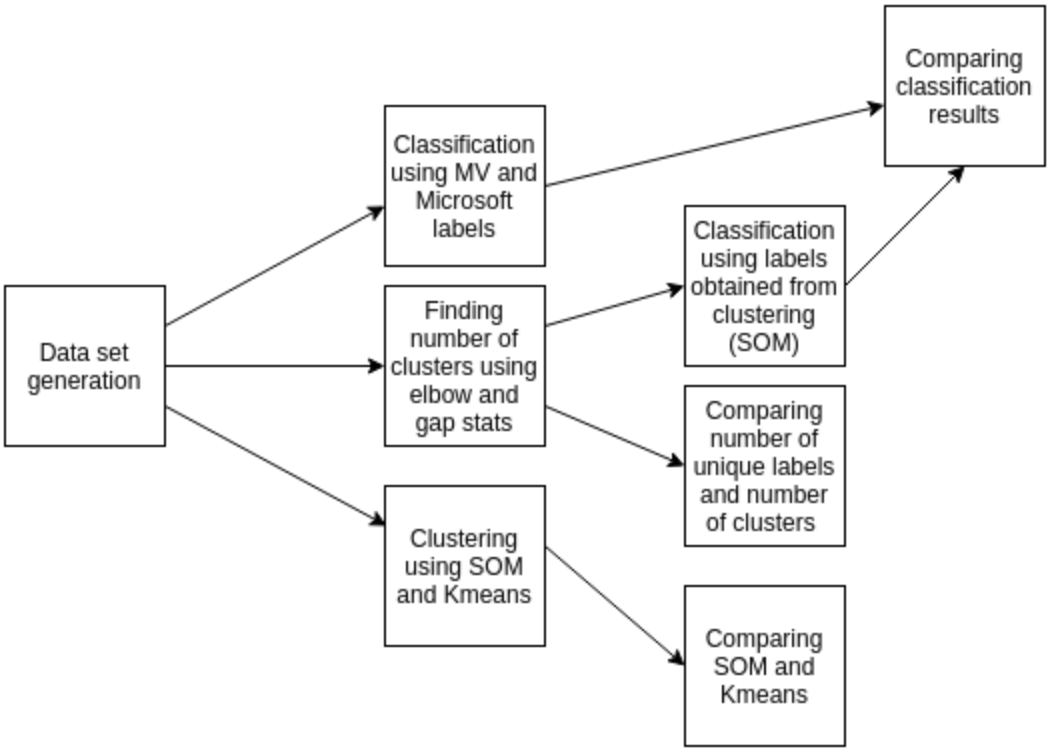


Figure 6.1: Workflow of the behavioral analysis module

7. Results

7.1. Number of Clusters

For finding the most suitable number of clusters for our data, we'll be considering both the elbow method and gap statistics.

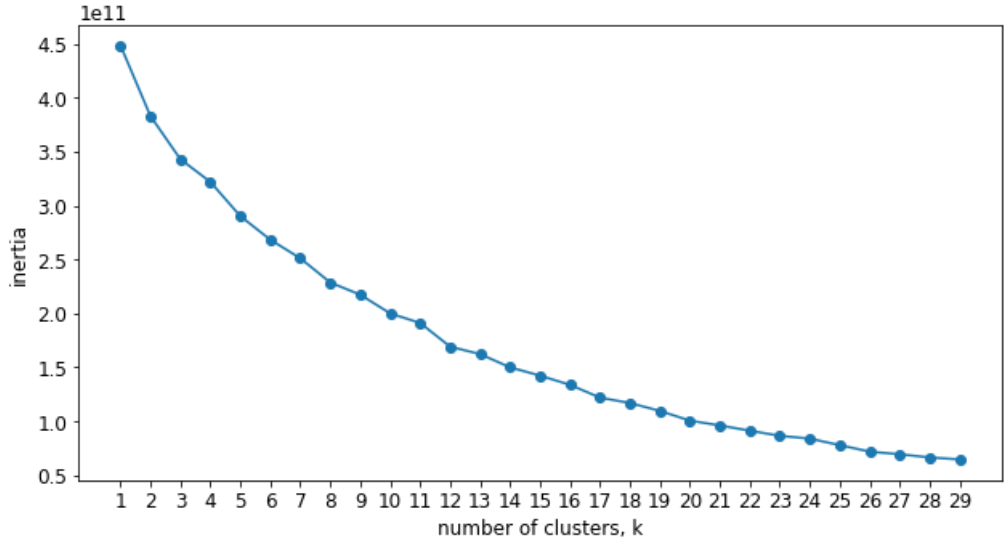


Figure 7.1: Inertia vs number of clusters for the behavioral dataset

In our case, we iterated the elbow method for range 1 to 30 of K values, as can be seen from figure 7.1. A slight elbow was observed at 23 and a saturation was observed at 24, but the result wasn't convincing enough. Hence, we opted for gap statistics in order to verify it.

The gap statistic curve that we obtained for our data, on iterating on k values ranging from 1 to 20, is given in the figure 7.2.

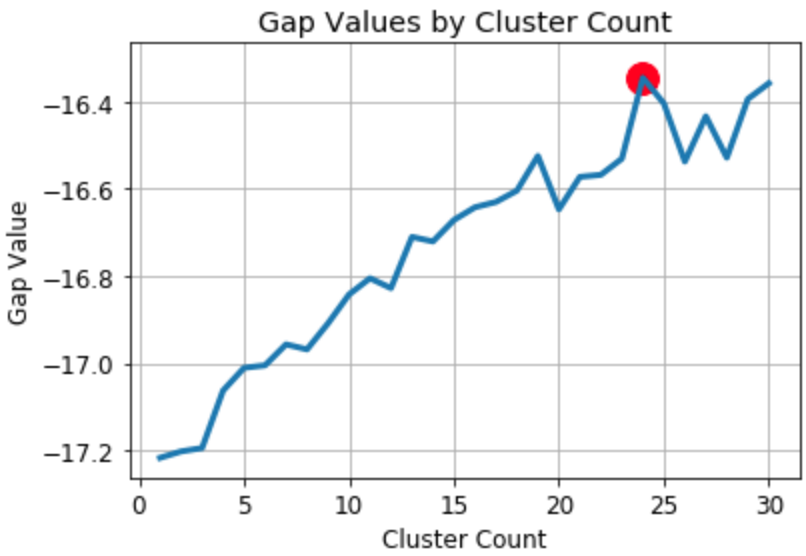


Figure 7.2: Gap value vs cluster count for the behavioral dataset

This curve also shows that the ideal number of clusters is 24, as the curve almost saturates at that value. This means that the data consists of 24 distinct behavioral classes while according to Microsoft labels there are 21 unique malware classes and according to Majority Vote labels there are 17 unique malware classes. This shows that there may be some inconsistency associated with identification of distinct malware classes by the anti virus vendors. In the upcoming sections, we will evaluate the results of other experiments and try to find concrete proof that inconsistencies exist in labelling techniques adopted by antivirus vendors.

Table 7.1: Microsoft label count

Type	Count
Clean	1218
Trojan	873
Virus	240
Backdoor	222
TrojanDownloader	217
PWS	148
VirTool	147
Worm	113
Exploit	89
TrojanSpy	57
TrojanClicker	52
TrojanDropper	51
BrowserModifier	35
Rogue	29
SoftwareBundler	28
Adware	25
TrojanProxy	12
Ransom	12
HackTool	10
Flooder	8
DoS	6

Table 7.2: Majority Vote label count

Type	Count
Trojan	2454
Clean	238
Adware	231
Backdoor	216
Virus	206
Worm	92
TrojanDownloader	36
Dropper	22
Toolbar	22
Flooder	17
Banker	15
Hacktool	13
Nsis	7
Bundler	7
Virtool	6
Rootkit	5
Trojanclicker	5

7.2. Clustering Compared

In order to select an unsupervised learning technique, we performed clustering using both the techniques on the reduced dataset, assigned clusters labels according to the majority samples present in them on the basis of both Majority Vote and Microsoft labels and finally summed the wrongly clustered samples. This sum was then averaged in order to get our error metric. Multiple cluster combinations were tested for each algorithm so as to allow the study of error convergence via absolute errors corresponding to each algorithm-cluster combination. The results can be seen in table 7.3.

Table 7.3: Comparison of clustering techniques

Algorithm	Mapsize	Number of Clusters	Majority Label	Microsoft
Kmeans	N/A	4	0.3135	0.6598
		8	0.3149	0.6592
		16	0.3135	0.6531
		24	0.3135	0.6495
SOM	20 X 20	4	0.3168	0.6604
	40 X 20	8	0.3121	0.6590
	40 X 40	16	0.3109	0.6490
	40 X 60	24	0.3104	0.6339

For the K-means-Majority combination, the error doesn't change much when the number of clusters are increased while for the K-means-Microsoft combination, the error convergence has a very slow rate.

On the other hand, for Self Organizing Maps, the error reduces, though slowly, after every increase in the number of clusters for Majority Vote labels while the error convergence is relatively much faster for Microsoft labels. The ability of SOM to capture the topographic properties of the data allows to better capture the intrinsic details of the dataset with an increased

number of nodes. This is the reason behind the better performance of SOMs and is hence the right choice of algorithm for the underlying problem.

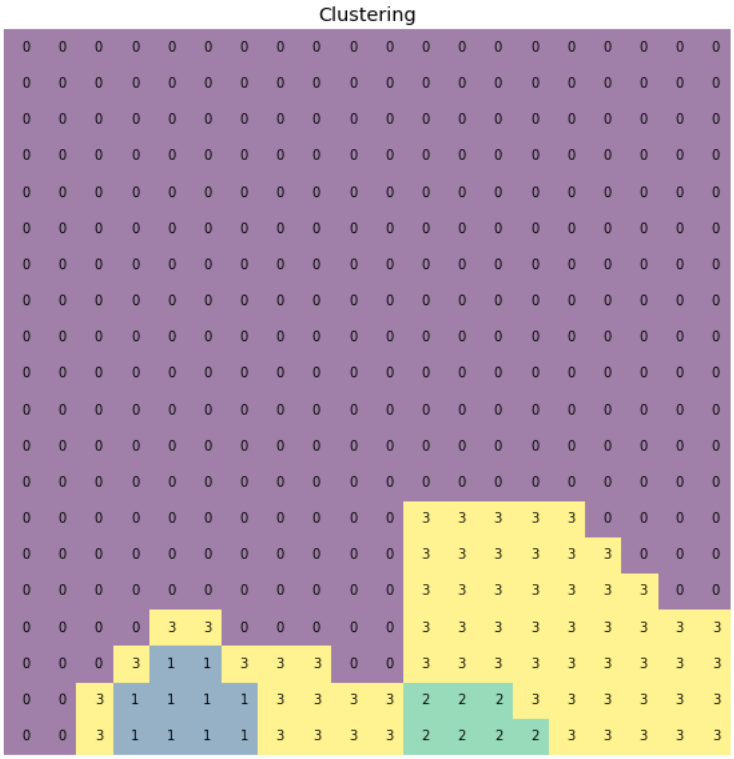


Figure 7.3: SOM clusters on 20 X 20 map

The corresponding 20x20 and 40x60 maps can be seen in figure 7.3 and 7.4 respectively. In the 20x20 map, the largest cluster spans much larger proportion of the map as compared to the largest cluster in the 40x60 map due to the large number of Trojan samples. In the 20x20 map, the lesser number of nodes are swamped by the large trojan population, which thereby renders it inefficient in differentiating between the samples. The larger number of nodes in the 40x60 map are able to better cope with this problem and therefore have a smaller trojan associated cluster. The larger number of nodes allow the 20x20 clusters to be split into multiple behavioral groups that discriminate the malicious samples better.

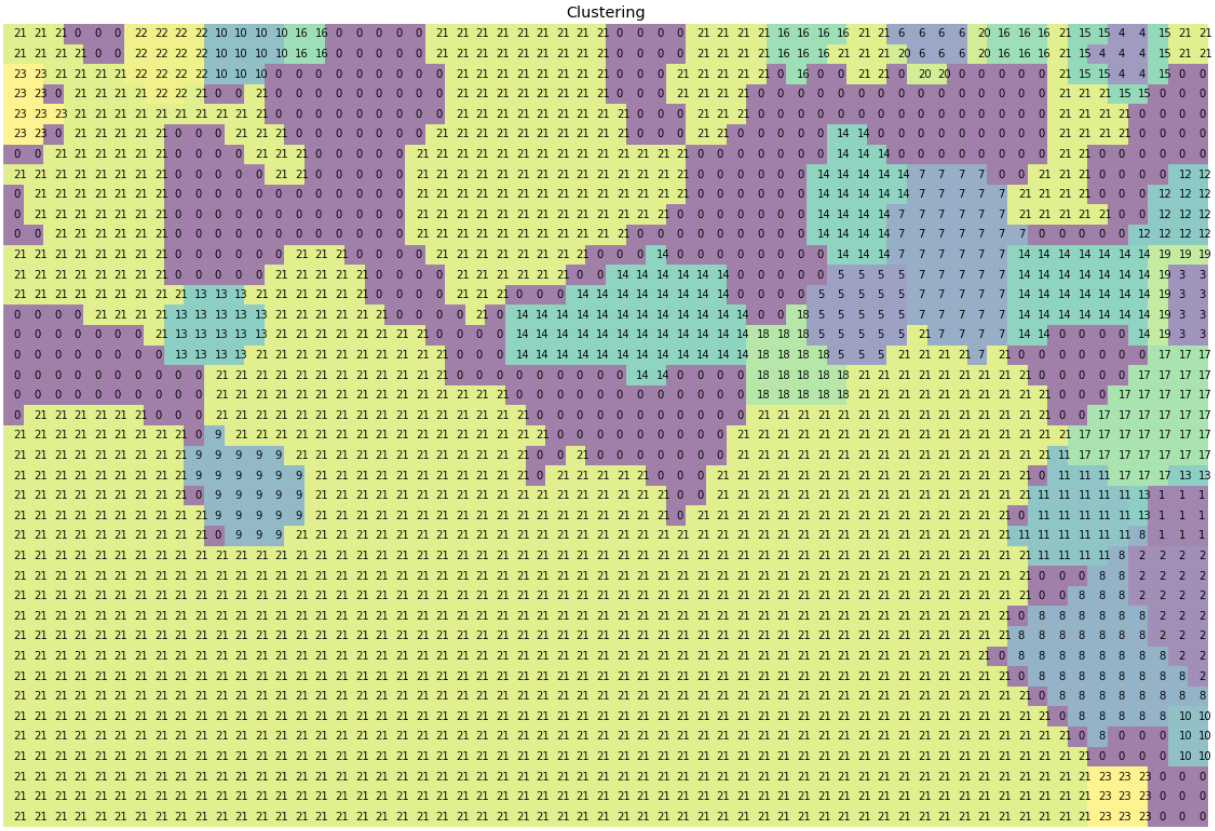


Figure 7.4: SOM clusters on a 40 X 60 map

Due to the better suitability of SOM to the data, we henceforth use it as our choice of clustering technique.

7.3. Classification

In order to assess the behavioral considerations of labels, we next analyze whether a classifier is able to efficiently allocate Microsoft, Majority Vote and Cluster labels to new unseen samples when trained on behavioral features and the respective labels.

We make use of a Random Forest classifier as it is known to perform well even when the samples are not evenly distributed across classes. The to-be-assessed labels were first appended to the behavioral features dataset, which was then split into separate training and testing sets. Finally, the classifier was trained on the training set with the appended labels and was asked to predict those labels for the testing set samples. The achieved results can be seen in table 7.4.

Table 7.4: Classification results using different label types

Label type	Microsoft	Majority Vote	Cluster labels
Accuracy	55.84%	74.86%	92.12%

From the results, it is evident that the AV vendors do not give much importance to the behavioral aspects of malware as both Microsoft and Majority Vote give poor results. The better performance via Majority Vote labels is due to the pooling of labels leading to generalization. However, the results via the clustering labels show that still much more can be achieved if a more behavioral suited approach is taken up.

8. Conclusion

Through various experiments we can safely conclude that a paradigm shift needs to be brought in the current malware analysis and labelling techniques to promote both automated detection and further research in the field. The results have shown significant differences between the groups created via Self Organizing Maps and those via Majority Vote and Microsoft labels, highlighting the fact that the behavioral aspect of malware is not given due consideration by the AV vendors. This was further validated by the classification accuracies secured by the random forest classifier on these labels. Adequate knowledge about the behavioral properties of a certain malware type can help in development of a generic protocol to neutralize it, a technique whose current application is inefficient with inconsistent and behavioral-profile-independent AV labels.

As an elementary solution to this, a cluster-based classification model can be adopted which will not take AV labels under consideration. The approach would consist of two main phases: the clustering phase, where various samples would be clustered on the basis of their behavioral profile and the classification phase, where new unseen samples will be allocated an ideal cluster. This would help with the current labelling inconsistencies and would also act as a catalyst for further research on the subject in order to bring out better solutions.

9. Future Work

The above solution we proposed may work at a basic level but cannot be incorporated in this ever-evolving industry due to numerous factors:

1. Researchers till now have incorporated API calls as behavior based features and not features related to system profile modifications.
2. We also need to formulate an algorithm for efficient selection of behavior based features.
3. Behavior based analysis is not widely accepted because researchers believe that some malware detect the presence of a virtual execution environment and hence, do not reveal their behavioral profile and evade detection.
4. Recent research has also shown that aging classifiers i.e infrequently trained classifiers are more prone to zero day attacks.

Keeping this in mind we plan to work on a holistic model that will incorporate the following components:

1. While extracting features to build the behavioral profile corresponding to a sample, we'll include system specific features in addition to features corresponding to API calls.
2. We plan to use autoencoder, an unsupervised deep learning algorithm for feature selection.
3. Use of bare metal environment can prevent malware from evading detection. Bare metal is a computer system or network in which a virtual machine (or Sandbox) is installed directly on hardware rather than within the host operating system (OS). The term "bare metal" refers to a hard disk, the usual medium on which a computer's OS is installed.
4. Develop a model to identify optimal training cycles that a classifier should undergo to prevent aging.

References

1. Shanklin, S. D., Bernhard, T. E., & Lathem, G. S. (2002). U.S. Patent No. 6,487,666. Washington, DC: U.S. Patent and Trademark Office.
2. Schmidt, A. D., Bye, R., Schmidt, H. G., Clausen, J., Kiraz, O., Yuksel, K. A., ... & Albayrak, S. (2009, June). Static analysis of executables for collaborative malware detection on android. In Communications, 2009. ICC'09. IEEE International Conference on (pp. 1-5). IEEE.
3. Chua, Z., Shen, S., Saxena, P. (2017). Neural Nets Can Learn Function Type Signatures From Binaries.
4. Moser, A., Kruegel, C., & Kirda, E. (2007, December). Limits of static analysis for malware detection. In Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual (pp. 421-430). IEEE.
5. Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 19(4), 639-668.
6. Tian, R., Islam, R., Batten, L., & Versteeg, S. (2010, October). Differentiating malware from cleanware using behavioural analysis. In Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on (pp. 23-30). IEEE.
7. Chen, S., Xue, M., Tang, Z., Xu, L., & Zhu, H. (2016, May). Stormdroid: A streaminglized machine learning-based system for detecting android malware. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (pp. 377-388). ACM.
8. Bayer, U., Kirda, E., & Kruegel, C. (2010, March). Improving the efficiency of dynamic malware analysis. In Proceedings of the 2010 ACM Symposium on Applied Computing (pp. 1871-1878). ACM.
9. Wang, J., Xue, Y., Liu, Y., & Tan, T. H. (2015, April). JSDC: A hybrid approach for JavaScript malware detection and classification. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (pp. 109-120). ACM.
10. Graziano, M., Canali, D., Bilge, L., Lanzi, A., & Balzarotti, D. (2015). Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In USENIX Security Symposium (pp. 1057-1072). USENIX Association.
11. Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., & Cavallaro, L. (2017). Transcend: Detecting Concept Drift in Malware Classification Models.
12. Miramirkhani, N., Appini, M. P., Nikiforakis, N., & Polychronakis, M. (2017, May). Spotless Sandboxes: Evading Malware Analysis Systems using Wear-and-Tear Artifacts. In Security and Privacy (SP), 2017 IEEE Symposium on (pp. 1009-1024). IEEE.

13. Xu, W., Qi, Y., & Evans, D. (2016). Automatically evading classifiers. In Proceedings of the 2016 Network and Distributed Systems Symposium.
14. Kirat, D., & Vigna, G. (2015, October). MalGene: Automatic extraction of malware analysis evasion signature. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (pp. 769-780). ACM.
15. Smutz, C., & Stavrou, A. (2016). When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. In NDSS.
16. Xu, M., & Kim, T. (2017). PlatPal: Detecting Malicious Documents with Platform Diversity.
17. Xu, Z., Nappa, A., Baykov, R., Yang, G., Caballero, J., & Gu, G. (2014, November). Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In Proceedings of the 2014 ACM SIGSAC Conference (pp. 179-190).
18. Bartos, K., Sofka, M., & Franc, V. (2016). Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants.
19. Fossi, M., Turner, D., Johnson, E., Mack, T., Adams, T., Blackbird, J., ... & Wueest, C. (2009). Symantec global internet security threat report. White Paper, Symantec Enterprise Security, 1.
20. Forecast and Methodology, 2014–2019 White Paper, Cisco, 2015
21. Griffin, K., Schneider, S., Hu, X., & Chiueh, T. C. (2009, September). Automatic Generation of String Signatures for Malware Detection. In RAID (Vol. 5758, pp. 101-120).
22. Kephart, J. O. (1994). Automatic extraction of computer virus signatures. In Proc. 4th Virus Bulletin International Conference, Abingdon, England, 1994 (pp. 178-184).
23. Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009, February). Scalable, Behavior-Based Malware Clustering. In NDSS (Vol. 9, pp. 8-11).
24. Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(2), 411-423.
25. Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *28*(1), 100-108.
26. Arthur, D., & Vassilvitskii, S. (2007, January). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics.
27. Pirscoveanu, R. S., Stevanovic, M., & Pedersen, J. M. (2016, June). Clustering analysis of malware behavior using Self Organizing Map. In Cyber Situational Awareness, Data Analytics And Assessment (CyberSA), 2016 International Conference On (pp. 1-6). IEEE.