Machine Learning Classification for Advanced Malware Detection

by
Fabio Di Troia

May 2020

A thesis submitted to Kingston University in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

This introductory document discusses topics related to malware detection via the application of machine learning algorithms. It is intended as a supplement to the published work submitted (a complete list of which can be found in Table 1) and outlines the motivation behind the experiments.

The document begins with the following sections:

- Section 2 presents a preliminary discussion of the research methodology employed.
- Section 3 presents the background analysis of malware detection in general, and the use of machine learning.
- Section 4 provides a brief introduction of the most common machine learning algorithms in current use.

The remaining sections present the main body of the experimental work, which lead to the conclusions in Section 10.

- Section 5 analyzes different initialization strategies for machine learning models, with a view to ensuring that the most effective training and testing strategy is employed. Following this, a purely dynamic approach is proposed, which results in perfect classification of the samples against benign files, and therefore provides a baseline against which the performance of subsequent static approaches can be compared.
- Section 6 introduces the static-based tests, beginning with the challenging problem of *zero-day* detection samples, i.e. malware samples for which not enough data has been gathered yet to train the machine learning models.
- Section 7 describes the testing of several different approaches to static malware detection. During these tests, the effectiveness of these algorithms is analyzed and compared with other means of classification.

- Section 8 proposes and compares techniques to boost the detection accuracy by combining the scores obtained from other detection algorithms, with a view to improving static classification scores and thus reach the perfect detection obtained with dynamic features.
- Section 9 tests the effectiveness of generic malware models by assessing the detection effectiveness of a generic malware model trained on several different families. The experiments are intended to introduce a more realistic scenario where a single, comprehensive, machine learning model is used to detect several families. This Section shows the difficulty to build a single model to detect several malware families.

1.2 Personal contribution

All the research proposed has been the result of a combined effort of several individuals. The first name in each paper is always the master's student that helped in the experiments and the setup of the necessary tools. The remaining nominatives are the creator of the idea at the base of the research. The last name is also the official advisor for the master's student.
In Table 1, the percentage estimation of the author's contribution to every given publication is shown. These percentages take also account of the manual work behind each experiment. An indication of the number of citations per each paper is also added. The data is taken from Google Scholar (Google).

Table 1 - Supporting research work

Title	Personal Contribution	Cited by	Section
Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin & Mark Stamp, "**A comparison of static, dynamic, and hybrid analysis for malware detection**", *J Comput Virol Hack Tech* 13, 2017	30%	155	5.1
• Anusha Damodaran (student) • **Fabio Di Troia** • Visaggio Aaron Corrado	Setting up and iterating the experiments **Generic advising** **Checking experiments and student's work** **Creator of the scripts used** **Disassembling of the malware files** Author's personal adviser		

	• Thomas H. Austin • Mark Stamp	Generic advising Master's student advisor Creator of the idea		
Vemparala Swapna, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., Stamp Mark, "**Malware Detection Using Dynamic Birthmarks**", In Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics		80%	23	5.2
•	• Swapna Vemparala • **Fabio Di Troia** • Visaggio Aaron Corrado • Thomas H. Austin • Mark Stamp	Setting up and iterating the experiments **Joint creator of the idea** **Checking experiments and student's work** **Developing of the machine learning algorithm (PHMM)** **Disassembling of the malware files** My personal advisor Generic advising Master's student advisor Creator of the idea		
Pai Swati, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., Stamp Mark, "**Clustering for malware classification**", *J Comput Virol Hack Tech* 13, 2017		40%	36	6.1
•	• Swathi Pai (student) • **Fabio Di Troia** • Visaggio Aaron Corrado • Thomas H. Austin • Mark Stamp	Setting up and iterating the experiments Creating the tools used **Joint creator of the idea** **Checking experiments and student's work** **Disassembling of the malware files** My personal advisor Generic advising Master's student advisor Creator of the idea		
Narra Usha, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., Stamp Mark **"Clustering versus SVM for malware detection**", *J Comput Virol Hack Tech* 12, 2016		40%	20	6.2
•	• Usha Narra (student)	Setting up and iterating the experiments		

	• **Fabio Di Troia** • Visaggio Aaron Corrado • Thomas H. Austin • Mark Stamp	**Joint creator of the idea** **Checking experiments and student's work** **Disassembling of the malware files** My personal advisor Generic advising Master's student advisor Creator of the idea		
Deshmukh Suchita, Di Troia Fabio, Stamp Mark, "**Vigenère scores for malware detection**", *J Comput Virol Hack Tech* 14, 2018		50%	2	7.1
•	• Suchita Deshmukh (student) • **Fabio Di Troia** • Mark Stamp	Setting up and iterating the experiments **Joint creator of the idea** **Checking experiments and student's work** **Creator of the scripts used** **Handling of the dataset files** Master's student advisor Creator of the idea		
Yajamanam Sravani, Vikash Raja Samuel Selvin, Di Troia Fabio, Stamp Mark, "**Deep Learning versus Gist Descriptors for Image-based Malware Classification**" In 2nd International Workshop on FORmal methods for Security Engineering, 2018		30%	18	7.2
•	• Sravani Yajamanam (student) • Vikash Raja Samuel Selvin (student) • **Fabio Di Troia** • Mark Stamp	Setting up and iterating the experiments Point of contact with Ford Motors Iterate experiments **Joint creator of the idea** **Check experiments and students work** **Handling of the dataset files** Master's student advisor Creator of the idea		
Rajeswaran Deebiga, Di Troia Fabio, Stamp Mark, "**Function Call Graphs Versus Machine Learning for Malware Detection",** Guide to Vulnerability Analysis for Computer Networks and Systems, Computer Communications and Networks, Springer, 2018		40%	3	7.3
•	• Deebiga Rajeswaran (student) • **Fabio Di Troia**	Setting up and iterating the experiments **Joint creator of the idea**		

		Checking experiments and student's work **Creating the scripts used** **Handling of the dataset files**		
	• Mark Stamp	Master's student advisor Creator of the idea		
Huang Wei-Chung, Di Troia Fabio and Stamp Mark, "**Robust Hashing for Image-based Malware Classification**", International Workshop on Behavioral Analysis for System Security, 2018		40%	1	7.4
•	• Wei-Chung Huang (student) • **Fabio Di Troia**	Setting up and iterating the experiments **Joint creator of the idea** **Checking experiments and student's work** **Creating the scripts used** **Handling of the dataset files**		
	• Mark Stamp	Master's student advisor Creator of the idea		
Raghavan Aditya, Di Troia Fabio and Stamp Mark, "**Hidden Markov models with random restarts versus boosting for malware detection**", *J Comput Virol Hack Tech 19*, 2019		60%	2	8.1
•	• Aditya Raghavan (student) • **Fabio Di Troia**	Setting up and iterating the experiments **Joint creator of the idea** **Checking experiments and student's work** **Creating the scripts used** **Handling of the dataset files**		
	• Mark Stamp	Master's student advisor Creator of the idea		
Singh Tanuvir, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin & Mark Stamp, "**Support vector machines and malware detection**", *J Comput Virol Hack Tech 12,* 2016		50%	46	8.2
•	• Tanuvir Singh (student) • **Fabio Di Troia**	Setting up and iterating the experiments **Joint creator of the idea** **Checking experiments and student's work** **Disassembling of the malware files**		
	• Visaggio Aaron Corrado	My personal advisor		
	• Thomas H. Austin	Generic advising		

• Mark Stamp	Master's student advisor Creator of the idea		
Bagga Naman, Di Troia Fabio and Stamp Mark, "**On the Effectiveness of Generic Malware Models**", International Workshop on Behavioral Analysis for System Security, 2018	60%	3	9.1
• Naman Bagga (student) • **Fabio Di Troia** • Mark Stamp	Set up and iterate experiments **Joint creator of the idea** **Checking experiments and student's work** **Creator of the scripts used** **Handling of the dataset files** Master's student advisor Creator of the idea		

1.3 Research Questions

This research work is based on the use of machine learning algorithms in the context of advanced malware detection, that is, detection of highly obfuscated malware families. This approach is analyzed and compared to *ad-hoc* techniques designed to obtain the same goal. In fact, several advanced malware detection techniques have been based on *ad-hoc* solutions. Some examples can be found in (Rajeswaran, et al., 2018), (Xu, et al., 2013) and (Shanmugam, et al., 2013) However, the application of machine learning based techniques has obtained interesting results, even in the case of high percentage of obfuscation. For example, the application of Support Vector Machines in (Singh, et al., 2016), or k-Nearest Neighbors in (Liu L., 2017), and also an algorithm based on Hidden Markov Models in (Wong, et al., 2006). A direct comparison of several *ad-hoc* techniques with machine learning approaches is thus described in this work, with the aim of pushing further the *state-of-the-art* proposing approaches that allow to detect even the most challenging malicious samples.

Several questions have been driven this work. Is it possible to achieve the same level of classification accuracy obtainable taking advantage of dynamic features but still relying on static ones? Which technique would be more effective in case of high level of obfuscation, or when the machine learning model has few samples (or none) to be trained on? To reach the answer to these questions, many complex experiments were performed. For example, testing the accuracy of the classification by constantly increasing the degree of dead code insertion in the infected samples (which is a very effective obfuscation technique against machine learning based detection (Wong, et al., 2006)) and proposing a challenging set of experiments, such as *zero-day* detection (Section 6) and cold start classification (Section 8).

1.4 Novel Contributions

To clarify in which way each of the papers that form the backbone of this research work have introduced novel contributions to the field, a description per each paper is given here.

A comparison of static, dynamic, and hybrid analysis for malware detection.

> A comparison of malware detection techniques based on static, dynamic, and hybrid analysis is proposed. Specifically, this work proposes a new testing scenario where Hidden Markov Models (HMMs) have been trained on both static and dynamic feature sets and the resulting detection rates have been compared.

Malware Detection Using Dynamic Birthmarks

> In this paper, we see a first application of Profile Hidden Markov Models (PHMMs) in detecting dynamically extracted API calls sequences. Furthermore, a comparison is made with a previous work where PHMM was used based on static features.

Clustering for malware classification

> Here, we apply clustering techniques to the malware classification problem. We compute clusters using the well-known *K*-means and Expectation Maximization algorithms, with the underlying scores based on Hidden Markov Models.

Clustering versus SVM for malware detection

> Following up on "Clustering for malware classification" paper, cluster analysis is applied to the challenging problem of classifying previously unknown malware (*zero-day* experiments). In addition, the clustering results are compared to those obtained when a Support Vector Machine (SVM) is trained on the malware family.

Vigenère scores for malware detection

> Previous research has applied classic cryptanalytic techniques to the malware detection problem. In this one, two new malware scoring techniques are proposed, both based on the classic Vigenère cipher. One of them relies only on the index of coincidence (IC), while the other is based on a more complete cryptanalysis of a Vigenère cipher, where the IC calculation is the first step.

Deep Learning versus Gist Descriptors for Image-based Malware Classification

Image features known as “gist descriptors” have been applied to the malware classification problem. This approach is tested against obfuscated malware, and a feature reduction is performed to determine a minimal set of gist features. It is then compared to a deep learning technique working directly on the image files without requiring any further manipulation.

Function Call Graphs Versus Machine Learning for Malware Detection

A function call graph approach is compared against machine learning techniques in detecting highly obfuscated malware samples. This work provides evidence that machine learning is likely to perform better than *ad-hoc* approaches.

Robust Hashing for Image-based Malware Classification

A first time comparison of SVM with Robust Hashing is proposed in this paper. Also, a new Robust Hash approach is proposed, combining distributed coding and wavelet analysis.

Hidden Markov models with random restarts versus boosting for malware detection

A comparison of two different applications of HMM based detection is proposed. That is, implemented for the first time through AdaBoost algorithm, and applying random restarts for the initial matrix distributions. A new set of experiments, called cold start, is also introduced. In these experiments, we train the models starting from a very limited number of samples (to simulate a more realistical and slow discovery of individual samples belonging to the same malware family). The cold start experiments allow to justify the overhead of the AdaBoost based technique in this special challenge.

Support vector machines and malware detection

In this research, three advanced malware scoring techniques, namely, Hidden Markov Models, Simple Substitution Distance, and Opcode Graph based detection, are combined through a Support Vector Machine. We see that combining scores in this way yields results that are significantly more robust than those obtained singularly using any of the individual scores.

On the Effectiveness of Generic Malware Models

A model that results to be too generic to be useful is obtained by extracting common features from extremely general malware families. This issue is analyzed via controlled experiments to determine the tradeoff between generality and accuracy over a variety of machine learning techniques based on n-gram features.

2 Research Methodology

This section describes the methodologies applied throughout these experiments, together with the respective datasets. It also describes the scoring techniques chosen to optimize the information provided by each kind of test.

2.1 The Dataset

The experiments required both malware and benign samples, the aim being to differentiate the former from the latter. However, not all the experiments analysed relied on the same type of feature. It was therefore not possible to run all the experiments on all the samples, though whenever possible files were shared among compatible tests. For example, some experiments are based on the image representation of the binary data, while others require the disassembling of the samples to obtain the list of opcodes. Where, with the term opcodes, we indicate the extracted mnemonic symbols from the disassembled assembly code.

In another set of experiments, *n*-grams are obtained from the binary data. In this context, an *n*-gram is a contiguous sequence of *n* bytes extracted from a given file. A *n*-gram of size 1 is referred to as a "unigram", size 2 as a "bigram" or "digram", size 3 as a "trigram", and so on. For example, if we have a list of bytes, such as “ABCAB”, the corresponding list of digrams would be: AB, BC, CA, AB.

Since the work was limited to malicious software on Microsoft Windows, different malware families and benign file collections were made of Windows executables. For the former, a group of infected files were obtained from the following well-known datasets, infecting Windows 7:

- Malicia (Malicia, 2013),
- Malimg (Nataraj, et al., 2011),
- Microsoft Malware Classification Challenge (Kaggle, 2015).

The choice of those particular datasets was corroborated by the wide number of research works that already relied on them. In this way, a direct comparison with previous work can be applied with ease.

In particular, the family used from the Malicia dataset were Harebot, Security Shield, Smart HDD, Winwebsec, Zbot and ZeroAccess. While from the Microsoft Malware Classification Challenge were Gatak, Kelihos, Lollipop, Obfuscator.ACY and Ramnit. The image-based experiments used the families from the Malimg dataset, that were Adialer, Agent, Allaple, Alueron, Autorun, C2LOP, Dialplatform, Dontovo, Fakerean, Instantaccess, Malex, Obfuscator, Rbot, Skintrim, Swizzor, Swizzor, VB, Wintrim, Yuner.

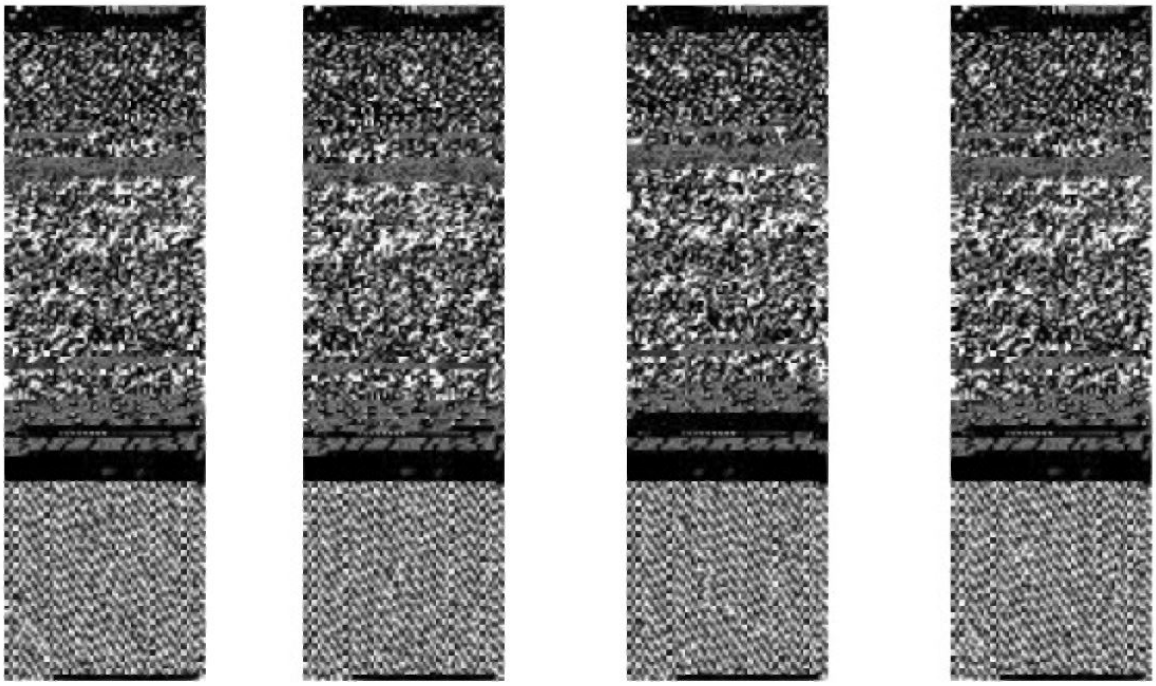


Figure 1 - Variants of the malware family Dialplatform.B

Figure 1 shows four variants of the malware family Dialplatform.B viewed as images. In this case, we clearly see common structure in the images, which indicates that there is significant potential for an image-based malware scoring technique.

However, the majority of the experiments relied on the Malicia dataset, with the exception of our work in (Bagga, et al., 2018) that relied also on the Microsoft Malware Classification Challenge families, and the image-based experiments in our (Yajamanam, et al., 2018) and (Huang, et al., 2018) that relied on the Maligm dataset. Additional samples were generated using the Next Generation Virus Creation Kit (NGVSK) (NGVSK, 2001), but they were used exclusively in our (Deshmukh, et al., 2018) and (Singh, et al., 2016) as a means of comparison with the previous published works.

Benign samples were collected from two primary sources:

- A fresh installation of Microsoft Windows 7, using an official Microsoft DVD-ROM,
- A clean installation in Windows 7 of the command-line Windows interface Cygwin (Cyg)

These samples were tested using the VirusTotal API (VirusTotal) and any obtaining less than 100% positive response was removed.

Another factor was the "balancing" of the samples: ideally each malware family should be represented equally in each sample, but in practice this is not always the case. For this reason, different malware families were combined from various groups, where the samples per family was at least 100. Furthermore, to obtain an equal number of files per each family, it was established that each should contain the same number of files.
A brief description of the families used from the (Kaggle, 2015) dataset can be found in (Bagga, et al., 2018), while a description of the families in the (Malicia, 2013) dataset can be found in (Damodaran, et al., 2017). Finally, the (Nataraj, et al., 2011) dataset is described in (Yajamanam, et al., 2018). In the current document, each experiment is introduced specifying the exact malware families used and the type of extracted features.

2.2 Score techniques

Throughout these experiments, no single technique was used to compare the accuracy of different classification approaches. Instead, the scores were based on Receiver Operator Characteristic (ROC) curve (Fawcett, 2006) and Precision-Recall (Davis, 2006), though other scoring techniques were introduced whenever they proved to be more informative in a given experiment. For example, a modified accuracy (defined as *balanced accuracy*) is introduced in our work (Bagga, et al., 2018) to face the large unbalance between the positive and negative samples. The basic techniques are described below.

2.2.1 Receiver Operator Characteristic

To compute a ROC curve, we begin with a scatterplot containing "positive" and "negative" scores: in these experiments, the "positive" instances ideally represent the malware samples

requiring detection, and the “negative” instances belong to the benign dataset. In practice however, some malware samples score negatively (“false negatives”) and some benign samples score positively (“false positives”). By adjusting the threshold between the two categories, that is, the decision boundary for which a value above it indicates a positive instance and a value below indicates a negative one, we can plot the False Positive Rate FPR (a number between 0 and 1 indicating the portion of samples falsely classified as positive) against the True Positive Rate TPR (again between 0 and 1). Finally we compute the area under this curve (the “AUC”) whose value gives a measure of success for the computed binary classification (Bradley, 1997). AUC is always in the range 0.5 to 1.0, and its two extreme cases are as follows:

- AUC=1.0 indicates that it a threshold level exists for which no classification error occurs.
- AUC=0.5 indicates that the classification is no better results than flipping a coin: there is a 50% chance that any sample may be misclassified.

Figure 1 (left) shows an example of a scattergram, where the solid red circles correspond to malware samples (positives) and hollow blue squares to the benign samples (negatives). The horizontal yellow line represents one possible threshold. Since 7 out of 10 red circles lie above the line, the True Positive Rate TPR=0.7. Similarly, 2 out of the 10 blue squares lie above the line, meaning that the False Positive Rate FPR=0.2. Figure 1 (right) shows the corresponding ROC, where the black dot at (0.2, 0.7) represents this threshold. The rest of the curve corresponds to the variation of the threshold and the resulting rates. The shaded region is the AUC, which is equal to 0.75 in this example.

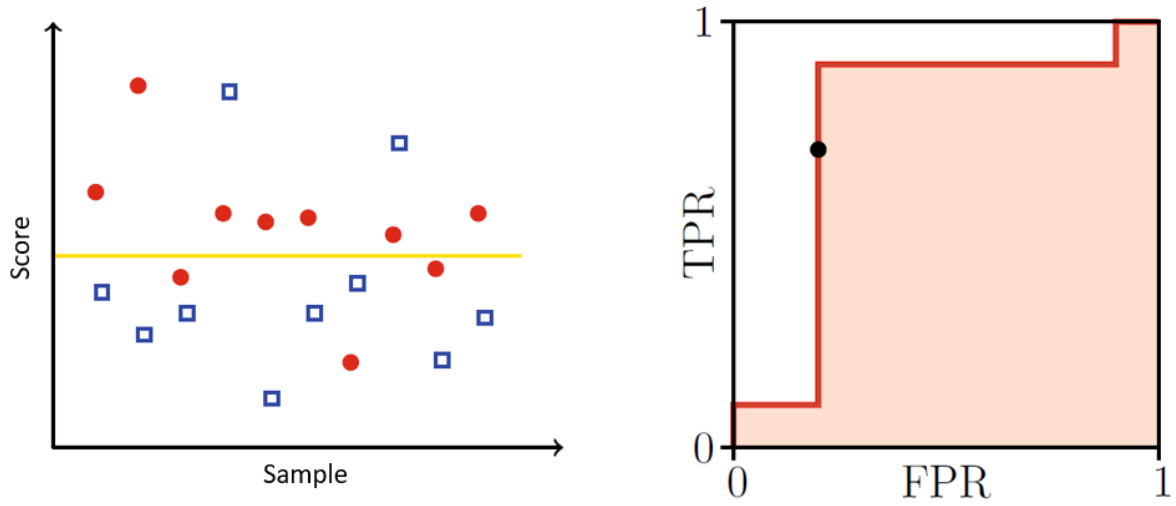


Figure 2 - Scatterplot and corresponding ROC curve

2.2.2 Precision Recall

Another means of quantifying the success of binary classification is Precision and Recall (PR) analysis (Davis, 2006). While similar to ROC in terms of the information gathered, it performs better (for example) when the number of "match cases" (i.e., True Positives) is small relative to the total number of samples. Recall refers to the fraction of elements classified as positive that belong to the positive match set, while precision is the fraction of positive elements classified as positive among the total number of positively classified samples, i.e.

$$Recall = \frac{TP}{TP+FN} \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

where TP is the number of true positives, FP the number of false positives, and FN is the number of false negatives. While recall corresponds to the true positive rate defined for ROC, precision is not the same as the false positive rate used in ROC. Another critical difference is that there is no consideration of true negatives (TN). This is beneficial if our focus is on the positive set, and especially when this is relatively large in comparison to the negative set.

As with ROC, we compute the area under the PR Curve (AUC-PR) as a measure of the success of the classification. This curve is computed following a similar process to the one explained in Section 2.2.1, where per each threshold the ratio is now described by the precision and recall. Figure 3 (left) shows the same data as in Section 2.2.1, while Figure 2 (right) plots the (*recall*, *precision*) pair as the threshold level varies. The threshold shown in the scatterplot

represents $TP = 7$, $FP = 2$, and $FN = 3$, so *recall* $= \frac{7}{7+3} = 0.7$ and *precision* $= \frac{7}{7+2} \approx 0.78$, and this point is represented by the black dot on the PR curve. The AUC-PR of this example is about 0.69.

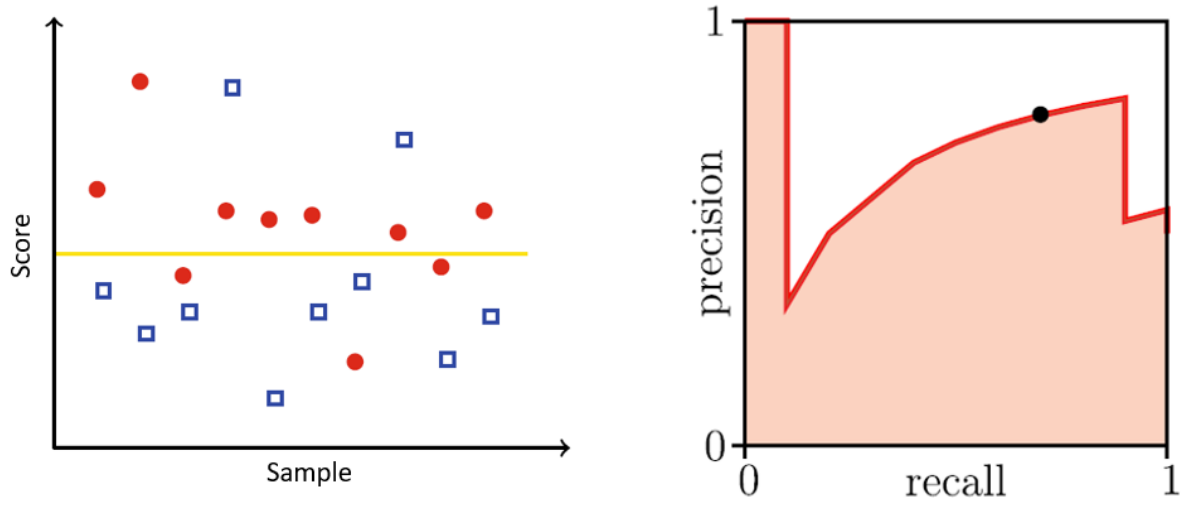


Figure 3 - Scatterplot and corresponding PR curve

2.2.3 Cross-validation

This is used to smooth out bias in experiments. For *n*-fold cross-validation, the malware samples are split into *n* folders; a machine learning model is trained over (*n*−1) of these folders, while the remaining folder is used for testing. These steps are repeated *n* times, always changing the folder used to test. Finally, the average of all the *n* scores is computed and considered as the final score of the experiment. Following this procedure, cross-validation allows to smooth out biases in the dataset (Stamp, 2017).

2.3 Tools

Section 2.1 mentioned that in different experiments, different types of feature were extracted from the dataset. For example, in the majority of the experiments (except (Yajamanam, et al., 2018), (Huang, et al., 2018) and (Bagga, et al., 2018), where the features where extracted from the binary files without disassembling them), executable malware samples were disassembled into a full list of opcodes using IDA Pro 7.0 (IDA), a powerful tool that supports scripting, function tracing, instruction tracing and instruction logging. Scripts were also used to extract the opcodes from the IDA Pro output, which contains a great deal of other irrelevant information. To avoid overfitting and “noise” in the training set, opcodes

were limited to those most commonly represented: for example, for 30 unique opcodes, the 29 most common listed were selected, and a unique symbol was allocated to each. Later, a different symbol was allocated to all the remaining less represented opcodes, for a total of 30 unique symbols. However, the exact number of symbols depends on factors such as the algorithm used or the type of dataset; the value was always picked that gave us the best accuracy in classification.

A very similar approach was used to extract sequences of Application Programming Interface (API) calls, which are provided by Microsoft Windows to facilitate requests for services from the operating system (Ahmed, et al., 2009). Each API call has a distinct name, a set of arguments and a return value. However, only API call names were collected, while arguments and return values were discarded. Other features, such as *n*-grams, were extracted directly from the binary files using Python scripts. However, the training of models on dynamic features always relied on dynamically extracted API calls, though this used IDA Pro not just as a disassembler, but also as a debugger. In fact, IDA Pro has integrated features to deal with anti-debugging techniques introduced by malware creators to avoid detection. Furthermore, due to the Universal PE Unpacker plug-in introduced in IDA Pro since version 4.9, it can also deal with malware code that has been packed (IDA).

In addition to IDA Pro, Buster Sandbox Analyzer (BSA) was also used: this tracks actions executed by a monitored program such as register, file-system and port changes. It runs in a sandbox to protect the system from infection, executing potentially malicious software for a fixed amount of time while logging all API calls that occur. These recorded API calls are then used to form a dynamic API call sequence for each executable file used in the experiments.

Few other external tools are also used, and more details are furnished in the relevant sections and references.

3 Background

This Section gives a brief introduction to the malware detection problem, analyzing different conventional detection techniques prior to introducing the machine learning algorithms that form the central pillar of this research work.
This Section is not intended to be fully complete. However, more details can be found in the references and the papers listed in Table 1.

3.1 Malware detection

Several techniques have been applied to the challenging problem of detecting malicious software, but the most widely used is signature scanning or signature detection (Aycock, 2006). Since it relies on pattern matching, the easiest way to perform this type of detection is to collect a string of bits from a malware sample and store it in a repository. When this string is found as part of a file, we have a match and (even though false positives are not unlikely) we can classify the file as infected. A variety of different pattern-matching algorithms can be used to achieve this goal. Although this technique has several advantages, such as being efficient and easy to implement, it has some crucial drawbacks. Signature scanning is only able to detect a malware file that has previously been analyzed and from which a sequence of bits has been collected. Furthermore, considering that the sequence of bits in an executable file not wholly random, it is subject to false positives when the same sequence is found in a non-infected file. This is due to the generalization problem for which the same signature can indicate a multiple subset of sequences extracted from different files, not necessarily malicious. It is also relatively simple to defeat with even a weak form of code morphing. It is important to note that machine learning based techniques are also affected by false positives. In this case, the generalization is described as generality of the malware model used by the algorithm. This particular problem is described more thoroughly in Section 9.
Another type of detection is anomaly detection, or behavior-based detection (Stamp, 2011). This aims to find any chunks of bit-sequences which are out of the ordinary and classifies them as potential malware. This type of detection is not normally robust enough as a standalone detection system, and it is usually combined with signature detection. Differentiating

between usual and unusual is the fundamental challenge. However, this technique has one significant advantage over the signature-based ones, i.e., the ability to detect new, undiscovered, malware. Analogies with Intrusion Detection Systems are straightforward, especially in terms of potential drawbacks, such as the necessity for the model to change according to new legit behavior, that could be used as an attack vector for a patient attacker. Although the description of these types of attack is outside the scope of this thesis, more details can be found in (Stamp, 2011) and (Corona I., 2013).

Change detection is another technique of interest (Stamp, 2011). For example, a virus would add its payload to an executable file to take advantage of its execution rights in the system. This causes the original file's structure to be modified in a manner that can be detected. As a defence against this, hashes could be generated from all the infectable resources in the system, and before executing or accessing their content, a new hash is computed from the file and compared with the previously stored hash (Stamp, 2011). If the file has been infected, a change in the hash value should appear. Further analyses would then be required by other detection techniques, such as signature detection.

Furthermore, other disadvantages are introduced, predominantly the increased number of false positives since the majority of the files in a system change because of non-malicious behavior. A legit behavior that would be considered suspicious by this technique, hence flooding the system with needless red flags.

3.2 Machine learning for malware detection

The approaches described in the previous Section have crucial drawbacks since they all, in a way or another, rely on signature detection as a final means of detection. However, malware detection can also be based on statistical properties derived from program features allowing a different category of detection techniques based on machine learning. These have the potential ability to detect malware samples even in the presence of relatively intrusive code morphing or other forms of obfuscation. However, while techniques such as instruction permutation (which modifies the order of the instructions without changing the original intended behavior) are not able to hide the pattern left by the morphing engine, some malware obfuscation techniques could introduce substantial problems (Stamp, 2011). For example, one particularly useful obfuscation technique is "dead code insertion", where snippets of code are added to the body of the malware file to skew the statistical properties

of the original list of instructions. When the injected code is taken from benign files, it becomes particularly challenging detecting such modified samples.

There are two main approaches to the building of machine learning models: the first is “static analysis”, which is based on static features extracted directly from the samples without executing them, and the second, “dynamic analysis” is based on dynamic characteristics that are extrapolated during the sample execution. Examples of the information obtainable from the static analysis include opcodes sequences, control flow graphs, and binary n-grams, while patterns of information that can be obtained by dynamic analysis include API calls, system calls, and registry changes. More information on the application of machine learning models in the field of malware detection can be found in (Stamp, 2017).

Some examples of successful machine learning algorithms in the field of malware detection are Hidden Markov Models and Support Vector Machines which are briefly described in Section 4.1 and Section 4.3. Other examples are *k*-nearest neighbors, random forests and convolutional neural networks, introduced in 4.5, 4.6 and 4.7, respectively. All these approaches have been applied in several experiments throughout this work.

3.3 Conclusions

Malware is defined as software that is intended to damage computers and computer systems without the knowledge of the owner. Signature-based malware detection is effective in many cases, but fails to detect advanced threats, such as metamorphic malware. Machine learning techniques have been used to effectively detect metamorphic malware, and other challenging classes of malicious code (Stamp, 2017). Such models are often trained using static features, such as opcode sequences and byte n-grams. Static features are preferable since allow to apply the detection mechanism without executing the files for a less intrusive detection. For this reason, it is more likely to encounter several malware families that concentrate their obfuscation strategies to defeat static based analysis. Therefore, dynamically extracted features are more informative and allow to build more robust machine learning models. In the next Section, a detection technique based on dynamic features is proposed and, subsequently, several static based approaches (based either on machine learning algorithms or *ad-hoc* methods) are introduced with the aim to find the best

technique able to reach the same level of classification accuracy obtained relying on dynamic features.

4 Algorithms

The majority of the experiments described in the subsequent sections mainly focus on Hidden Markov Models (HMMs), which were trained on malware samples and later used to test putative infected files. That is, the scores obtained from HMM (and similarly from PHMM) were used to classify the individual samples based on the given threshold. For example, the model was trained on a specific malware family and later used to classify between samples of such family and non-infected files. A description of the HMM algorithm is given in Section 4.1 and more details can be found in (Stamp, 2018)). Another particularly useful algorithm is Support Vector Machine (Section 4.3) which has proven to help boost classification scores (Section 8.2). In a specific set of experiments (Section 6) called *zero-day* tests, clustering techniques were applied to detect undiscovered malware samples. A description of the clustering algorithms used is given in Section 4.4. All the remaining algorithms used in our experiments are described in the respective research works and relative references.

4.1 Hidden Markov Models

A Hidden Markov Model (HMM) is a statistical model that can be used to build a machine learning technique that relies on a discrete hill climb (Stamp, 2018). That is, a re-estimation process that improves the model by modifying the parameters. It has been widely and successfully used to solve a variety of problems in several fields, such as speech recognition (Rabiner, 1989), artificial intelligence (Ghahramani, 2001), and malware detection (Wong, et al., 2006).

HMM includes a Markov model that cannot itself be directly observed, though, a collection of observations that are probabilistically related to the Markov process can be used to create a set of discrete probability distributions that binds the observable information to the hidden one. The following notation is used for an HMM (Stamp, 2018):

T = length of the observation sequence
N = number of states in the model
M = number of observation symbols
Q = $\{q_0, q_1, \ldots, q_{N-1}\}$ = distinct states of the Markov process
V = $\{0, 1, \ldots, M-1\}$ = set of possible observations
A = state transition probabilities

B = observation probability matrix
π = initial state distribution
$\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1})$ = observation sequence

A generic Hidden Markov Model is illustrated in Figure 4. The part above the dotted line is the state sequence, while below the line is the observation sequence. While we can observe the observation sequence, the state sequence remains hidden. However, through the probability distribution contained in the B matrix, we can associate the observable information to the hidden one.

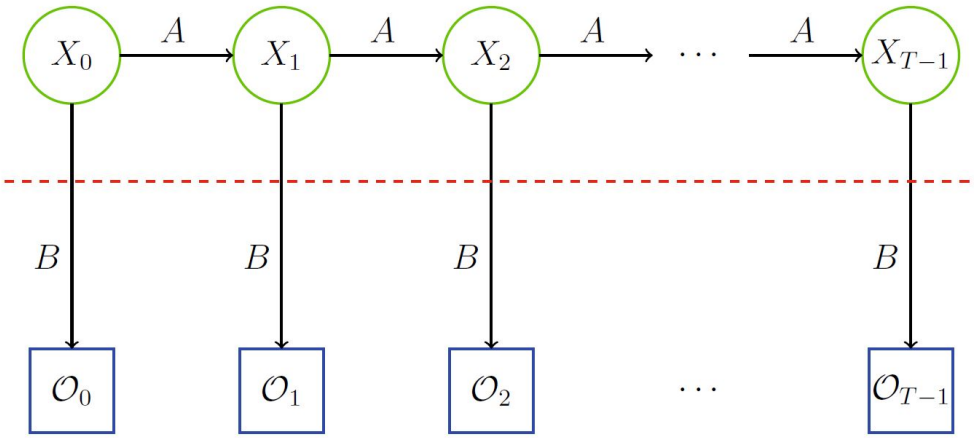


Figure 4 - A generic HMM

This algorithm allows three different problems to be solved:

Problem 1: Given the model $\lambda = (A, B, \pi)$ and an observable sequence $\mathcal{O}$, determine $P(\mathcal{O}|\lambda)$. Given an observation sequence, we can use model λ to score how closely related the two are to each other.

Problem 2: Given the model $\lambda = (A, B, \pi)$ and an observable sequence $\mathcal{O}$, determine the most likely hidden state sequence X. That is, we can uncover the most likely hidden states.

Problem 3: Given an observation sequence $\mathcal{O}$ and the dimensions N and M, determine the model λ such that it maximizes the probability of $\mathcal{O}$. That is, given an observation sequence, we can train a model that best fits with such sequence.

Considering the problem of malware detection, we build a machine learning algorithm applying the solution to Problem 3 to train an HMM to recognize the samples of a given malware family. This training phase is based on a collection of features extracted from malware samples belonging to the same malware family. Then, given an unknown sample, we apply the solution to Problem 1 to score it against the trained model. A set of scores are

thus collected. If the unknown samples belong to the given malware family, we expect to obtain a higher score than the one from a sample that does not belong to that specific family. Following, a description of the algorithms used to solve the three problems. Even further reading, plus a pseudo-code version, can be found in (Stamp, 2018).

4.1.1 Solution to Problem 1

Let $\lambda = (A, B, \pi)$ be a given model and let $O = (O_0, O_1, \ldots, O_{T-1})$ be a series of observations. We want to find the probability of the O given λ, or $P(O|\lambda)$. To find $P(O|\lambda)$, the so-called forward algorithm, or α-pass, is used. For $t = 0,1,\ldots,T-1$ and $i = 0,1,\ldots,N-1$, define

$$\alpha_t(i) = P(O_0, O_1, \ldots, O_t, x_t = q_i|\lambda) \qquad (1)$$

Then $\alpha_t(i)$ is the probability of the partial observation sequence up to time t, where the underlying Markov process is in state q_i at time t. However, $\alpha_t(i)$ can be computed recursively as follows, therefore, the forward algorithm only requires a smaller number of multiplications.

4.1.2 Solution to Problem 2

We want to find the most likely state sequence given the model $\lambda = (A, B, \pi)$ and an observations sequence O. Specifically, we want to find a state sequence so to maximize the expected number of correct states. To obtain this goal, we first define the backward algorithm, or β-pass. This is analogous to the α-pass discussed above, except that it starts at the end and works back toward the beginning. For $t = 0,1,\ldots,T-1$ and $i = 0,1,\ldots,N-1$, define

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_{T-1}|x_t = q_i, \lambda) \qquad (2)$$

Then, for $t = 0,1,\ldots,T-1$ and $i = 0,1,\ldots,N-1$, we define $\gamma_t(i) = P(x_t = q_i|O, \lambda)$ as $\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$. From the definition of $\gamma_t(i)$ it follows that the most likely state at time t is the state q_i for which $\gamma_t(i)$ is maximum, where the maximum is taken over the index i.

4.1.3 Solution to Problem 3

We want to find the elements of A, B and π, subject to the row stochastic condition. For $t = 0,1,\ldots,T-2$ and $i, j \in \{0,1,\ldots,N-1\}$, we define "di-gammas" as

$$\gamma_t(i,j) = P(x_t = q_i, x_{t+1} = q_j | O, \lambda) \quad (3)$$

Thus, $\gamma_t(i,j)$ is the probability of being in state q_i at time t and transiting to state q_j at time $t+1$.

Given the γ and di-gamma we verify that the model $\lambda = (A, B, \pi)$ can be re-estimated as follows.

For $i = 0,1,\ldots,N-1$, let $\pi_i = \gamma_0(i)$. For $i = 0,1,\ldots,N-1$ and $j = 0,1,\ldots,N-1$, we compute

$$a_{ij} = \frac{\sum_{t=0}^{T-2} \gamma t(i,j)}{\sum_{t=0}^{T-2} \gamma t(i)} \quad (4)$$

For $j = 0,1,\ldots,N-1$ and $k = 0,1,\ldots,M-1$, we compute

$$b_j(k) = \frac{\sum_{t\in\{0,1,\ldots,T-1\} O_t = k} \gamma t(j)}{\sum_{t=0}^{T-1} \gamma t(j)} \quad (5)$$

The numerator in (4) can be seen to give the expected number of transitions from state q_i to state q_j, while the denominator is the expected number of transitions from q_i to any state. Thus, the ratio is the probability of transiting from state q_i to state q_j. The numerator in (5) is the expected number of times the model is in state q_j with observation k, while the denominator is the expected number of times the model is in state q_j. Thus, the ratio is the probability of observing symbol k, given that the model is in state q_j.

Re-estimation is an iterative process and in our case it starts initializing the model $\lambda = (A, B, \pi)$ with random values so that $\pi_i \approx \frac{1}{N}$, $a_{ij} \approx \frac{1}{N}$ and $b_j(k) \approx \frac{1}{M}$.

4.2 Profile Hidden Markov Models

A generic view of a PHMM is shown in Figure 5. A significant difference between PHMM and HMM is that while PHMM includes match (M), insert (I), and delete (D) states, a standard HMM does not allow insertions or deletions. Also, a PHMM explicitly accounts for positional information within sequences, whereas an HMM does not. A PHMM can be viewed as a generalization of an HMM that take positional information into account. PHMM scores were obtained by generating pairwise alignments with the sequences of API calls for a given malware family. This step allows the construction of the Multiple Sequence Alignment (MSA). However, some pre-processing is required before we can construct pairwise alignments. The

general approach used by (Attaluri, et al., 2009) were followed, where the API calls are sorted by frequency of occurrence.

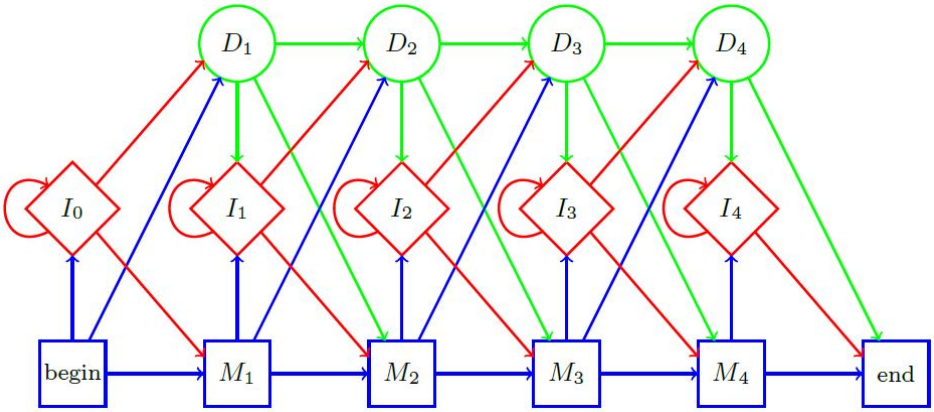


Figure 5 - Profile Hidden Markov Model

Attention was focused on the top 36 API calls, which constitute at least 99.8% of the total API calls for each family tested. The remaining ones were mapped to a single "other" state. The next step involves the creation of a substitution matrix and a gap penalty function. The Feng-Doolittle algorithm (Feng, et al., 1987) was used to create the MSA from the pairwise alignments, from which the PHMM was built. Finally, using the forward algorithm, the sequences were scored against the PHMM. Further details on the forward algorithm can be found in (Durbin, et al., 1988). Whereas, for more information about the specific pairwise alignment and MSA construction strategies, see (Vemparala, 2015).

4.3 Support Vector Machines

Support Vector Machines (SVMs) (Cortes, et al., 1995) are a class of supervised learning technique used for binary classification. Thus, given labeled linearly separable training data, the algorithm separates such data using a hyperplane, where the separation between the two classes of data in the training set is maximized. An SVM also works in higher dimensions: by moving the problem to a higher dimension, the data points tend to be more easily separated, and hence we have a better chance of finding a separating hyperplane. Computing higher dimensions is not a trivial process, and it is implemented using a so-called kernel "trick". This

"trick" is the process by which we transform the data into a higher dimension, in such a manner that they do become linearly separable.

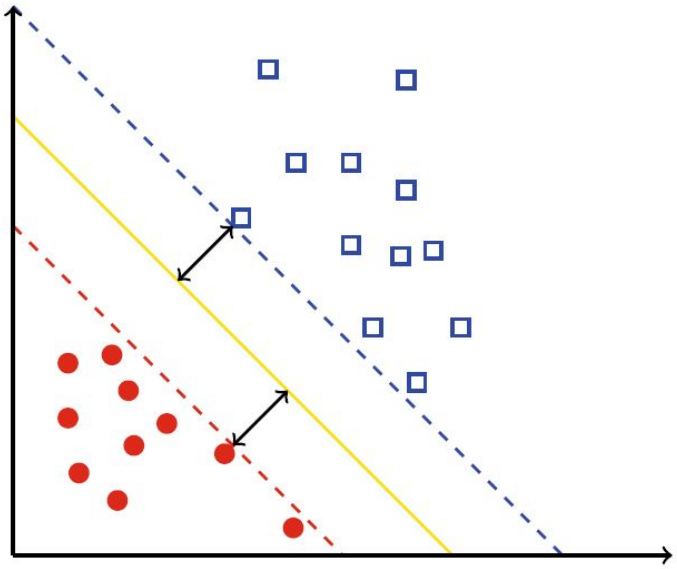


Figure 6 - Maximizing the margin

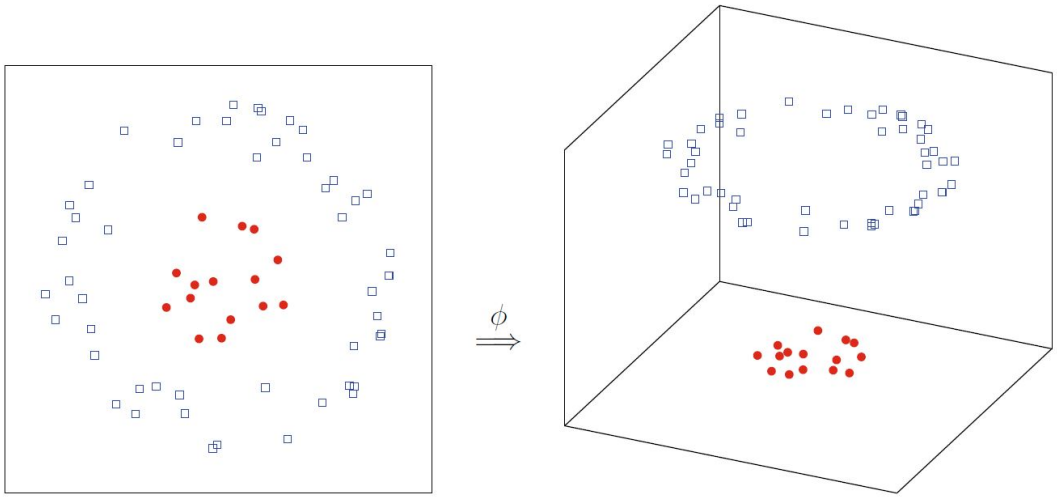


Figure 7 - Transformation from 2 to 3 dimensions

For example, Figure 6, shows a separating hyperplane that maximizes the margin. On the left side of Figure 7, a case where this cannot be done is described. On the right side of the same Figure, instead, a representation is shown of how the kernel "trick" can manipulate such data to easily construct a hyperplane that separates the two data sets, showing the benefit of transforming to a higher dimension. Several alternative tricks are applicable, and tests need to be done to select the most beneficial for the given experiment.

4.3.1 Feature reduction for SVM

It was necessary to collect several features from the dataset but selecting too many of these features proved to be detrimental to the scores. It was therefore decided to apply a feature reduction step before training and testing models. Some features act as noise and it is not uncommon to obtain better results with a reduced feature set.

One technique used to reduce the number of features is based on Support Vector Classification (SVC) (Stamp, 2017), a multiclass version of SVM . The first step is to apply recursive feature elimination (RFE) (Stamp, 2017): compute a linear SVC, eliminate the feature with the smallest weight and then recompute the linear SVC. The process continues until only one single feature remains, where this feature is the strongest in terms of its contribution to the classification.

Another of such techniques is called univariate feature selection (UFS) (Stamp, 2017). Where, the accuracies obtained from several SVCs are used to rank all the features. In this case, one SVC is built for each feature, in order to test them individually.

After testing with these two approaches, the one that gave the best tradeoff between the number of features used and accuracy was selected. Generally, RFE was somewhat more reliable, as it accounts for the interactions between features.

4.4 Clustering techniques

Clustering is an unsupervised classification mechanism that is designed to categorize data. There is a large variety of different algorithms that can be applied and in (Pai, et al., 2017) and (Usha, et al., 2016), the *K*-means and Expectation Maximization (EM) clustering techniques were used.

Expectation-Maximization (EM) is one of the most popular clustering techniques in use today. Given a set of *m* data points, the algorithm partitions the data into a specified number of mutually exclusive clusters. More details about EM can be found in (Do, et al., 2008).

K-means is another example of a clustering technique that differs from EM in some implementation details. K-means is an example of an EM technique where distances are measured directly between data points instead of measuring them based on probability distributions. More details about *K*-means can be found in (Alsabti, et al., 2000).

4.5 *k*-Nearest Neighbor

The *k*-Nearest Neighbor (*k*-NN) algorithm is a supervised learning technique that requires labeled training data (Stamp, 2017). In k-NN we classify a point x based on the k points in the training set that are nearest to it. In *k*-NN, we classify a sample based on a majority vote of the *k* nearest samples in the training set. However, variations exist, such as weighting the "votes" based on the relative frequencies of the training samples, particularly useful when there is an imbalance in the training samples, or weighting them based on their distance from the sample that we are classifying. An important advantage of using k-NN is that it involves no training as the labeled training data is all that the algorithm needs. In fact, all the necessary computations are left to the testing phase only. In Section 9.1, k-NN was implemented using python to build classifiers based on n-gram frequencies as feature.

4.6 Random Forests

Random forests are a generalization of decision trees (Stamp, 2017). Random forests use a process called "bagging" to construct multiple decision trees. In this approach, a random subset of the observations and a random subset of the features is selected to construct multiple decision trees.

For classification, the outputs of all these decision trees are combined using techniques such as the majority vote of each individual outcome from the decision trees to get a single classification. Relying on a random subset of the observations and features, random forests reduce the tendency of decision trees to overfit. In 9.1, scikit-learn Python library (Pedregosa, et al., 2011) was used to implement random forests using *n*-grams frequencies as feature.

4.7 Convolutional Neural Networks

Convolutional Neural Networks (CNN) can be described as a specialized case of the regular neural networks, where, besides implementing a set of learnable weights and biases, the overall architecture is designed to be used for problems involving image recognition. This assumption helps encoding properties that permit a more efficient application of the network, such as simplifying the forward function and reducing the number of parameters in the network. More details about the process involved can be found at (Con20). In 7.2, CNN was implement using Tensorflow For Poets from Google Codelabs (Ten20).

4.8 AdaBoost

Boosting is the process of combining multiple weak classifiers to obtain a stronger classifier. To be combined, a classifier must obtain in classification a result marginally better than flipping a coin. In 8.1, one technique to combine these classifiers was used, that is, “adaptive boost” or AdaBoost (Stamp, 2017). The basic concept is theoretically straightforward: AdaBoost selects the best classifiers at each iteration among those available. Such an algorithm must improve more than the others on the overall accuracy of the new combined classifier. In other words, AdaBoost greedily selects a classifier that does the most to improve on the current iteration of the constructed classifier.

More details about this approach can be found in (Stamp, 2017). However, it is worth noticing that AdaBoost is not a hill climb. That is, at any given iteration, the resulting classifier may be worse than at the previous iteration.

4.9 Conclusions

The algorithms here briefly described are used in the subsequent Sections and compared against *ad-hoc* techniques to select the best static based detection approach. In particular, HMMs are often used as a means of comparison among the several approaches, or are also used to generate scores that are directly integrated in a clustering based set of experiments (Section 6). Particular interest is placed on SVMs, that, after obtaining satisfying results compared to ad-hoc algorithms (Section 7), are used as a meta-classifier to boost up the classification accuracy of our static techniques (Section 8). This, specifically, will allow to reach the perfect detection (based on statically trained machine learning models) that is the backbone of this research work.

5 Preliminary experiments

Before starting the experiments, it was necessary to find the best approach to training and testing for the machine learning models. The very first set of tests compared the effectiveness of four different training and testing strategies based on dynamic and static features, plus combinations in two hybrid approaches. Further experiments were used to analyze the results of a dynamic based machine learning model, i.e. a model trained using dynamic features. Fortunately, this led immediately to a perfect detection against the challenging dataset and it was therefore decided to use these results as a baseline for further experiments using static features. Since static analysis is more efficient, reaching the same level of accuracy as with dynamic analysis could be considered a significant achievement. However, dynamic elements are more accurate and more complicated to obfuscate in comparison with static ones. To prove the effectiveness of static models, Section 6 introduces *zero-day* experiments based on detecting unknown malware samples exclusively using static features. The intent was to find a valid machine learning algorithm able to achieve good accuracy even in this challenging scenario. In Section 7, to understand if machine learning algorithms can indeed offer more than other approaches, they were compared with other four different static methods. Finally, in Section 8, to increase the detection rate, static based experiments were introduced where machine learning scores are combined to obtain better accuracy. It was shown that this particular method reaches the same level of detection as the one obtained by applying dynamic analysis.

The remainder of this Section is organized as follows. Section 5.1 describes the experiments used for training-testing the models and comparing them among themselves to select the most beneficial for the machine learning models. Section 5.2 introduces the purely dynamic approach, where Profile Hidden Markov Models (PHHMs) can achieve perfect detection in all tests.

5.1 A comparison of static, dynamic, and hybrid analysis for malware detection

Static detection involves trying to classify the samples without executing them using only static features, such as opcodes or byte *n*-grams. On the other hand, the dynamic analysis relies on running the samples to collect dynamic features, such as API calls runtime

sequences. While dynamic analysis produces, on average, a more accurate classification, static analysis is much faster.

Both approaches described in our work (Damodaran, et al., 2017) were tested, along with hybrid models which combined static and dynamic analysis. The first of these trained on dynamic features and tested on static ones, while the second trained on static features and tested on dynamic ones.

Purely static analysis (static training and static testing) has proved to be a reliable form of analysis in numerous researches. For example, in (Attaluri, et al., 2009), static features have been used to feed Profile Hidden Markov Models in the difficult task of classifying metamorphic malware samples, while (Deshpande, et al., 2014) and (Jidigam, et al., 2015) produced results using Principal Component Analysis. Other examples are (Singh, et al., 2016), which used Support Vector Machines, and (Pai, et al., 2017), which employed clustering algorithms (see Section 8.2 and Section 6.1).

5.1.1 Related Work

Previous work based on machine learning application on hybrid approach is discussed here. In (Choi, et al., 2012), a framework for classification of malware using both static and dynamic analysis is proposed. The approach used to define features of malware has been called Malware DNA (Mal-DNA). The heart of this technique is a debugging-based behavior monitor and analyser that extracts dynamic characteristics. Another work on the topic is (Eskandari, et al., 2013), where the authors develop and test a tool called HDM-Analyser that uses both static analysis and dynamic analysis in the training phase, but performs static analysis only in the testing phase. Differently from these previous works, the set of experiments described in this Section focuses on directly comparing different hybrid training/testing combinations relying on the same dataset, scoring techniques and features as the previous ones.

5.1.2 Experimental setup

For our experiments, four different combinations of static and dynamic features were analyzed: static/static, dynamic/dynamic, hybrid static/dynamic, and hybrid dynamic/static. To compare them, an HMM was trained on each case in two different sets of experiments: the first based on API calls sequences and the second on opcodes sequences. The tests

focused on six different malware families taken from the Malicia dataset described in Section 2.1: Harebot, Security Shield, Smart HDD, Winwebsec, Zbot, and ZeroAccess.

For the hybrid experiments, to ensure that the features extracted dynamically were compatible with the ones extracted statically (and vice versa), only the common features that were obtained from both approaches were taken in consideration to train the models. For example, in Figure 9 we observe the discrepancy between the unique opcodes extracted dynamically and the ones extracted statically.

5.1.3 Results

The four approaches were compared in terms of AUC (Area under the Receiver Operator Curve, see Section 2) shown in Figure 8, where a purely dynamic approach and a purely static approach consistently yields the best results. It is also clear that the incompatibility between the features used to train and the features extracted in the testing phase causes a degradation of the effectiveness of the hybrid techniques. In fact, the hybrid approach consisting of static training and dynamic scoring produces worse results compared to all the other strategies for almost all the malware family tested.

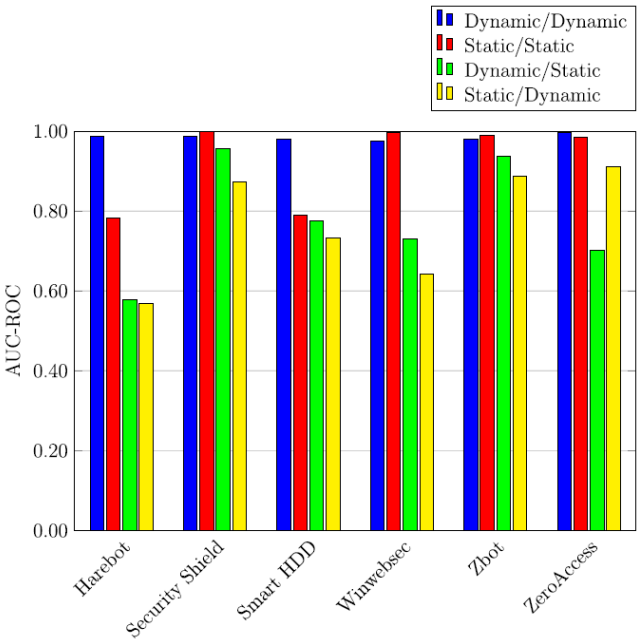


Figure 8 - ROC results for API call sequence

Next, opcode sequences for training and scoring were used. Figure 9 shows the number of unique opcodes in the static and dynamic cases for each family, which leads to the conclusion that scoring in the dynamic/static case is complicated by the many opcodes which appear

scoring that are not part of the training set. To avoid this complication, it was decided that the opcodes not appearing in the training set should be ignored. Figure 10 shows the AUC results. The scores, in this case, are not as strong as the ones obtained using API calls sequences. As in Figure 8, the purely dynamic approach yields excellent results in all cases, while in four cases the purely static approach attains less accuracy than the hybrid approach. However, unlike the API call sequence models, the static/static and the hybrid dynamic/static case yield results that are roughly equivalent.

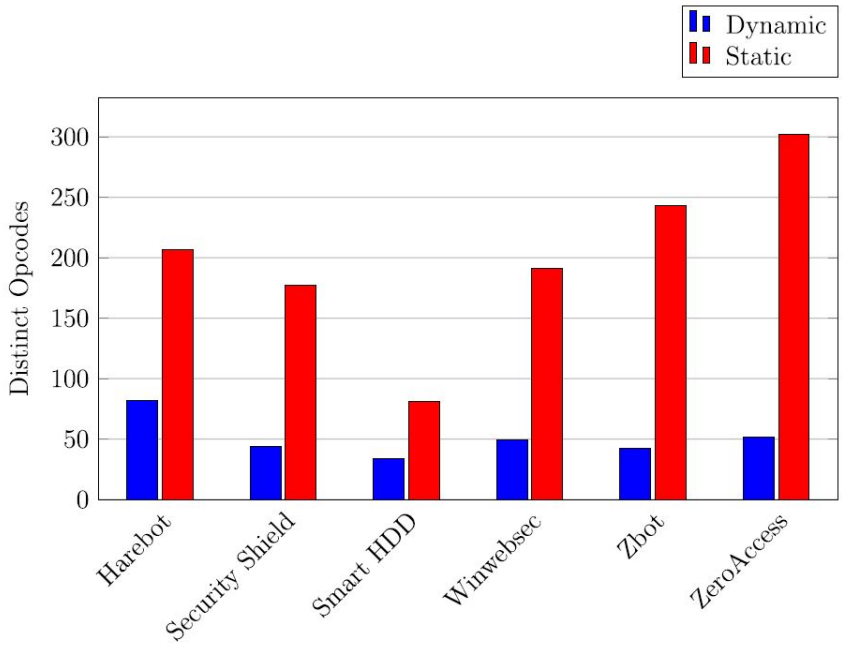


Figure 9 - Distinct opcodes

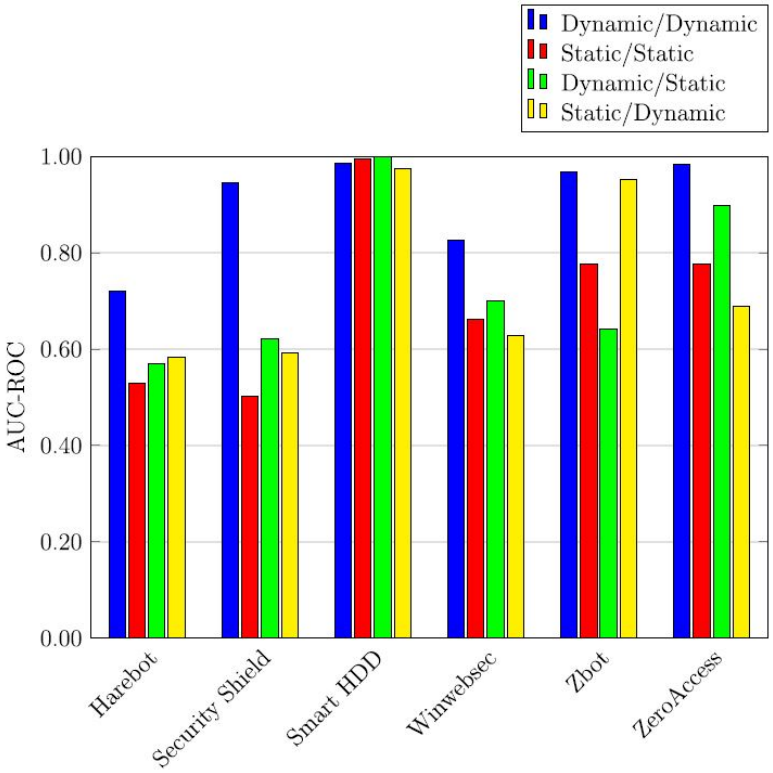


Figure 10 - ROC results for opcode sequences

5.1.4 Conclusions

These experiments have shown that for API calls and opcode sequences, a fully dynamic strategy is generally the most effective. However, it is essential to consider that dynamic analysis is usually costly in comparison to static analysis. While at the training phase, this added cost is not a significant issue, being mostly a one-time work, the scoring phase would likely be impractical, especially where it is necessary to scan a large number of files.

Applying a hybrid dynamic/static approach could potentially overcome this issue, since after dynamic training, we would get all the benefits of static testing. Unfortunately, this strategy has not been particularly successful: for API call sequences, results were consistently worse with the hybrid dynamic/static approach than with a fully static one. For opcode sequences, even though in four cases the results for this hybrid approach have been slightly better than the fully static one, the obtained results were not consistent enough against the whole set of malware families. Consequently, when hybrid approaches are proposed, it would be advisable to test the results against comparable fully dynamic and fully static techniques.

A fully static approach based on API calls was nearly as practical as a purely dynamic approach in most cases. However, as shown in Figure 10, the results have not been impressive for some

families. This outcome is likely due to the nature of the obfuscation techniques employed by malware writers. Current obfuscation techniques are likely to have a significant effect on opcode sequences, but little attention is paid to API calls. Nonetheless, API call sequences could likely be obfuscated with some additional effort, in which case the advantage of relying on API call sequences for detection might diminish significantly (Damodaran, et al., 2017).

In conclusion, our experimental results in this phase indicate that a straightforward hybrid approach is unlikely to be superior to fully dynamic detection. Moreover, even in comparison to fully static detection, the hybrid dynamic/static approach did not offer consistent improvement.

In Section 5.2, more successful purely dynamic experiments are introduced and, as confirmation, a comparison with the purely static approach is repeated. The next Sections, and the remainder of this document focuses on experiments in the purely static domain, abandoning both the hybrid strategies.

5.2 Malware Detection Using Dynamic Birthmarks

These experiments were designed to compare directly the purely dynamic and purely static approaches using two different detection algorithms. In our work (Vemparala, et al., 2016), two different techniques to classify malware samples were applied based on dynamic features: Hidden Markov Models (HMMs) and Profile Hidden Markov Models (PHHMs). Following the promising results of Section 5.1, both models were trained on sequences of API calls.

The term “software birthmarks” was used to describe the set of possible characteristics that can be derived from a specific software file. For the case of an executable file, it describes all the properties that can be extracted without executing it, for example opcode sequences. In contrast, dynamic birthmarks are characteristics that can usually be retrieved during the program execution, for example sequences of API calls. A description of how HMMs works is given in Section 4.1.

5.2.1 Related Work

Although both HMM-based analysis (Wong, et al., 2006) and PHMM-based analysis (Attaluri, et al., 2009) have been previously applied to the malware detection problem, no direct comparison of static and dynamic techniques was tested before. In particular, in (Wong, et

al., 2006) the effectiveness of HMMs for detecting challenging classes of metamorphic malware based on opcode sequences has been demonstrated. While, in (Attaluri, et al., 2009), the use of Profile Hidden Markov models is analyzed in detecting metamorphic malware based on static opcode sequences. This approach works well against certain classes of metamorphic malware but is less effective when the blocks of code are shifted relatively far apart. While PHMMs trained on static data did not prove to be particularly strong for malware detection, in our work we show that PHMMs are an extremely powerful tool for malware detection based on dynamic birthmarks.

5.2.2 Results

These experiments used the same dataset samples as those in Section 5.1 with the addition of a single family, that is, Cridex. In Figure 11 shows all the results per each family using both HMM and PHMM. Note that the HMM results are separated in Static HMM and Dynamic HMM based on the type of features used, respectively, opcode sequences and API calls. HMMs based on static opcode sequences have performed well against one family in particular, namely Smart HDD, for which AUC was 0.996. However, it returned inconsistent results with the other families, the lowest AUC score being 0.596 obtained with Cridex.

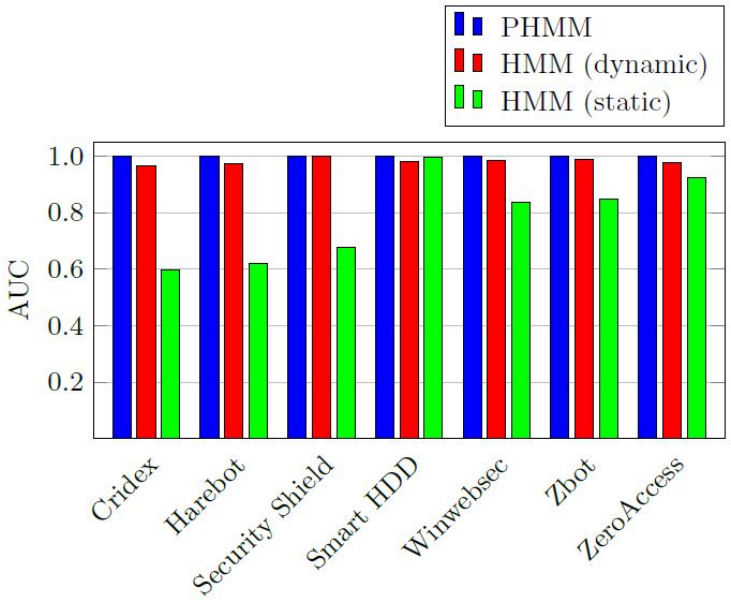


Figure 11 - PHMM versus HMMs

On the other hand, dynamic HMM outperforms the static HMM in almost every case. The average AUC for the dynamic HMM cases was 0.976, whereas for the static HMM cases it was 0.785. This result clearly shows the advantage of dynamic birthmarks. Regarding PHMMs, the

number of sequences used to train was a critical parameter: Table 2 describes the various results obtained through cross-validation, where "group n" means that the PHMM was trained using n sequences. In each case it was possible to achieve an AUC of 1.0, and once such a result was attained there was no need for further experimentation.

Table 2 - PHMM AUC by Group Size

Malware Family	Group Size			
	10	15	20	30
Cridex	1.000	—	—	—
Harebot	0.952	1.000	—	—
Security Shield	0.964	1.000	—	—
SmartHdd	0.905	0.963	1.000	—
Winwebsec	0.995	1.000	—	—
Zbot	0.970	1.000	—	—
Zeroaccess	0.988	0.968	0.975	1.000

5.2.3 Conclusions

These experiments have shown the efficacy of malware classification using HMMs built from API calls. Furthermore, it has been proved that PHMMs, based too on dynamic API call sequences, outperform the HMMs in every case. Thus, the positional information in API call sequences is highly effective, whereas the positional information in opcode sequences is much less so.

These results were used to establish the baseline for further experiments. Dynamic analysis using PHMM reached perfect detection in all cases, while dynamic analysis using HMM reached an average accuracy of 0.976. Both dynamic approaches are impressive, but they introduce the overhead required to extract the API call sequences from each sample. The more challenging problems of applying static detection, such as *zero-day* experiment, are addressed in Section 6. Moreover, to increase the score of our static approaches and, thus, get as close as possible to the results obtained using dynamic detection strategies, different boosting techniques were introduced, and are covered in Section 8.

5.3 Conclusions

After these set of preliminary experiments, we found that hybrid approaches (in particular, dynamic training and static testing) were not particularly effective. Thus, the subsequent experiments are all based on purely dynamic or purely static features. Furthermore, perfect

classification was reached applying PHMM trained on dynamically extracted sequences of API calls, and therefore it will provide a baseline against which the performance of subsequent static approaches can be compared to.

6 Zero-day experiments

This Section introduces the challenging problem of *zero-day* detection, where the malicious samples belong to malware families for which no model has been created yet. In other words, *zero-day* malware is a new undiscovered threat. Clustering techniques were chosen as the main approach in this type of experiments, since these are widely used in data exploration, where information about data itself is poorly understood (Babu, et al., 2012). This characteristic makes clustering a perfect candidate to detect new undiscovered malware files. The following parts of this Section compare two clustering algorithms: Expectation Maximization and *K*-means (Section 6.1), with the goal to find the most profitable tuning between dimensionality (i.e., number of models used for clustering), and the number of clusters, with respect to the accuracy of the clustering in a set of unsupervised experiments. Later, Support Vector Machines (SVMs) are introduced by training their models on compiler code instead of malware code to simulate the case of unknown malicious samples. A comparison between SVM and supervised clustering is also proposed (Section 6.2), proving how effective SVM is in combining scores.

6.1 Clustering for malware classification

In preparation for the experiments on detecting *zero-day* malware, it was necessary to determine the most effective clustering algorithm. For this reason, in our work (Pai, et al., 2017) clusters were computed and compared using the algorithms *K*-means and Expectation Maximization (specifically Gaussian mixture model), where the underlying scores were based on Hidden Markov Models. Specifically, the comparison focused on finding the best combination between the number of models used for clustering (dimensions) and the number of clusters produced.

6.1.1 Related Work

Clustering using the *K*-means algorithm is the focus of the work in (Annachhatre, et al., 2014), where, as in our work, malware samples are scored using multiple HMMs trained on distinct compilers code. Although none of the malware families were used to train the models, the classification results are good, indicating that clustering may be a useful tool for classifying

previously unknown malware. Our work, and specifically the experiments described in this Section, can be viewed as an extension of the work in (Annachhatre, et al., 2014). However, the main goal is to compare *K*-means and EM clustering in the context of malware classification and prepare the base for the experiments in Section 6.2.

6.1.2 Experimental setup

Still relying on the same set of datasets described in Section 2.1, investigations focused on five of the families present in the dataset introduced in Section 2.1: namely Zbot, ZeroAccess, Smart HDD, Winwebsec and NGVSK. The purpose of this experiment was to find the best clustering algorithm to classify each tested sample in the corresponding malware family.

As in the previous static experiments, all training and scoring was based on extracted opcode sequences. The process assigns scores to the families which are used to build the clusters using HMM. For instance, from each sample (regardless of the malware family to which belong) we obtain several scores from the various HMM models (one per each family). These scores represent the features of the sample that is then introduced to the clustering algorithm. That is, an HMM is created for each family and then, using the corresponding HMM scores, clusters were generated using both *K*-means and the EM clustering algorithms. Finally, the files were classified based on the resulting clusters.

Two different classification schemes were used here. One was based on the "silhouette coefficient", the other based solely on the purity of the clusters, a metrics that measures the ratio of each type of sample per each cluster (a purity of 100% indicates that the cluster contains only one type of data) (Stamp, 2017). More details about these schemes can be found in our work (Pai, et al., 2017).

6.1.3 Results

Varying the number of clusters from 2 to 10, and the number of dimensions from 2 to 5, it was found that EM outperforms *K*-means. In particular, as the number of clusters and dimensions increased, the better the EM algorithm performed.

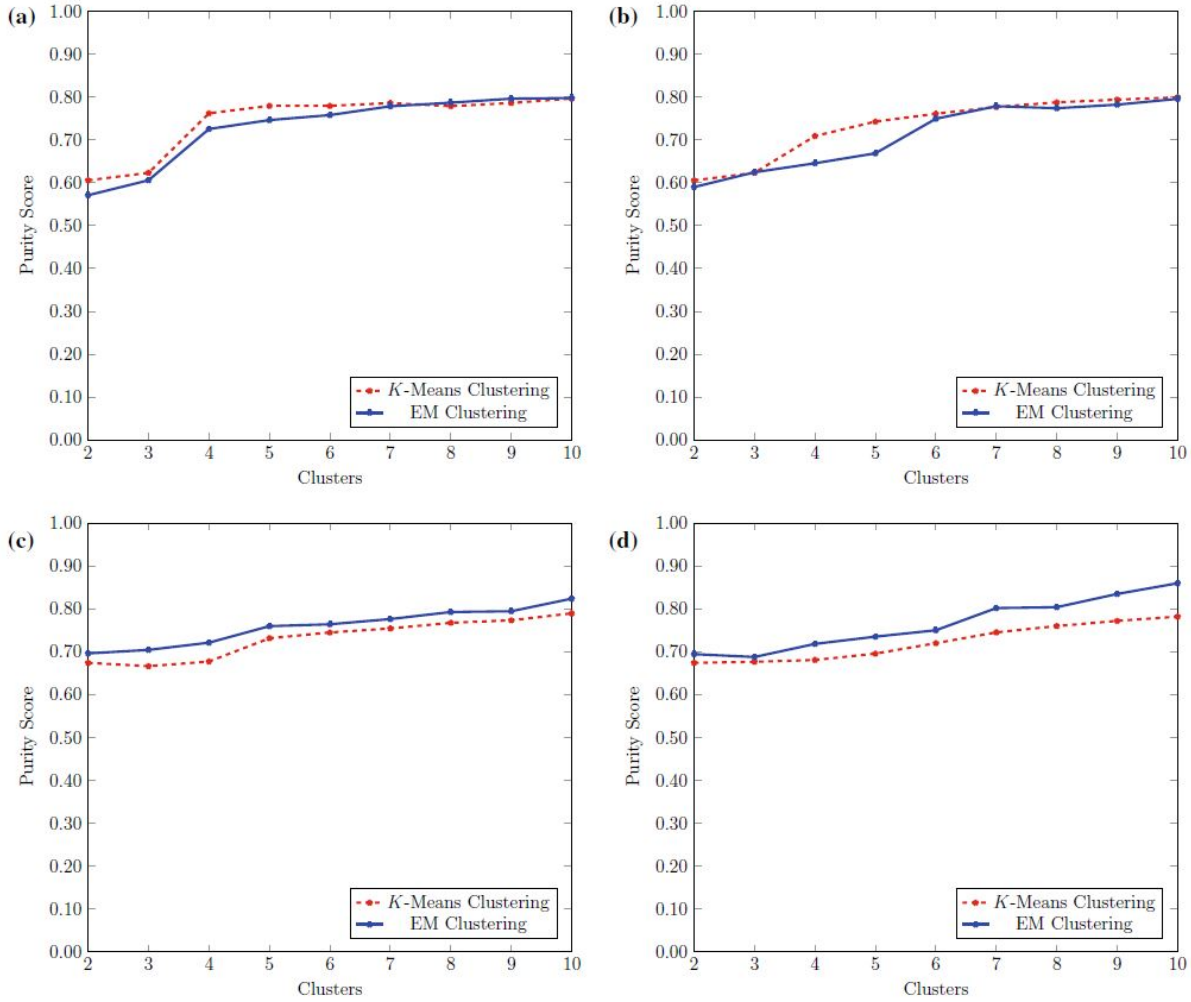


Figure 12 - Purity scores for EM and K-means clustering. (a) 2-dimensional, (b) 3-dimensional, (c) 4-dimensional, (d) 5-dimensional

Figure 12 shows the results based on the purity score, indicating how the distance of the EM scores from the *K*-means scores increases with the number of dimensions. More experiments and results can be found in (Pai, et al., 2017).

Another critical outcome of the experiments is that the overhead introduced by this approach is minimal. This characteristic represents a significant benefit of the clustering approach in relation to other static techniques. In fact, after the training phase, the clusters are known and, hence, to score a sample, HMM score must be computed along with the distance to the centroids (*K*-means) or probabilities (EM). If the number of clusters remains relatively small, this step has no significant impact upon the overall computation required.

After these preliminary set of experiments, in Section 6.2 attention was focused on the detection of the *zero-day* samples.

6.2 Clustering versus SVM for malware detection

After proving that Expectation Maximization (EM) obtains better accuracy in classifying malware samples in their respective families than K-means (Section 6.1), it was decided to test this technique against the more challenging problem of classifying previously unknown malware in our work (Usha, et al., 2016). The term *zero-day* malware is used to describe a type of malware that is still undisclosed and for which there is no existent signature or model applied for detection. To provide a point of reference for our clustering results, Support Vector Machines (SVMs) were used, where the SVM is trained on the malware family to classify.

6.2.1 Related Work

To generate the HMM models, the tools developed in (Annachhatre, et al., 2014) and (Austin, et al., 2013) were used. The overall implementation is followed, but in our case the same HMM scores that we used for clustering were used to train and test the SVM model. Thus, we test SVM for combining multiple HMM scores into a single classifier. Also, these SVM experiments serve as a sort of control, in the sense of providing results that we can compare to our clustering experiments.

6.2.2 Experimental setup

As in Section 6.1, all the underlying scores used for the clustering algorithms and SVMs are based on HMMs. However, while the dataset was identical to the one used previously (Section 6.1), a different approach was applied for the case of the SVM experiments. The HMM model was trained on each of four different compilers, namely, GCC, MinGW, TurboC, and Clang, plus models for handwritten assembly code (TASM) and NGVCK. From these files, seven distinct HMMs were obtained, which were used to score each of the malware and benign sample files. This operation resulted in a 7-tuple of scores for each file in the test set, which were used to generate the clusters. This process gives SVM an understanding of generic programming code without revealing any information on the actual malware families to detect. This process has been inspired by the work in (Annachhatre, et al., 2014), even though the authors relied on clustering experiments only. More details about this procedure can be found in our work (Usha, et al., 2016).

Furthermore, to simulate the *zero-day* experiments, three HMM models were obtained, one for each of the most represented malware families in the dataset, namely, ZeroAccess, Winwebsec, and Zbot. These scores were used to cluster two of the families, which were then used to test how accurate the classification of the third family was. The idea behind this approach is that the third family simulates an undiscovered malware family potentially detectable using a pre-existing model based on different known families.

6.2.3 Results

The purpose of these experiments was to determine how well the SVM can distinguish between families when using 2 to 4 dimensions of HMM scores. For this reason, an SVM model was trained on a subset of Zbot and Winwebsec samples and tested against the remaining Zbot and Winwebsec samples. Figure 13 shows the AUC results for these experiments using various kernel functions, indicating that SVM is very effective in classifying the malware samples.

To compare SVM with K-means and EM, the 7-tuples of compiler scores were used, with the same training and test sets that are used for clustering. However, the training set for the clustering algorithms did not contain benign samples, whereas benign samples must be explicitly included when training the SVM. Figure 14 shows the comparison in classifying the unknown malware family varying the number of dimensions from 2 to 4. There is a clear difference in comparison with the previous set of clustering experiments (Section 6.1). Raising the number of dimensions to 4 improved the effectiveness of EM, while, up to 3, *K*-means is outperformed by EM in classifying unknown malware samples. More importantly, this shows how superior the clustering and SVM scores are, in particular, for the 4-dimensional case. Moreover, considering that SVM had the advantage of being trained directly on the given malware samples to classify, the clustering results become very impressive.

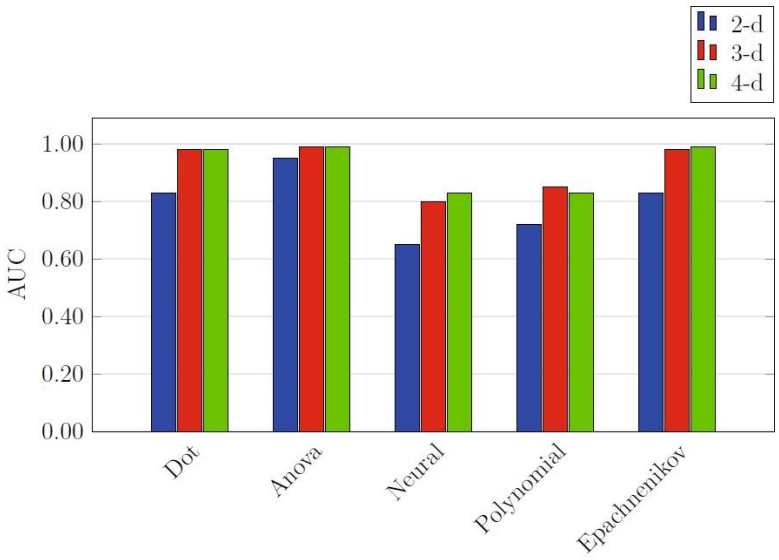


Figure 13 - SVM malware score results

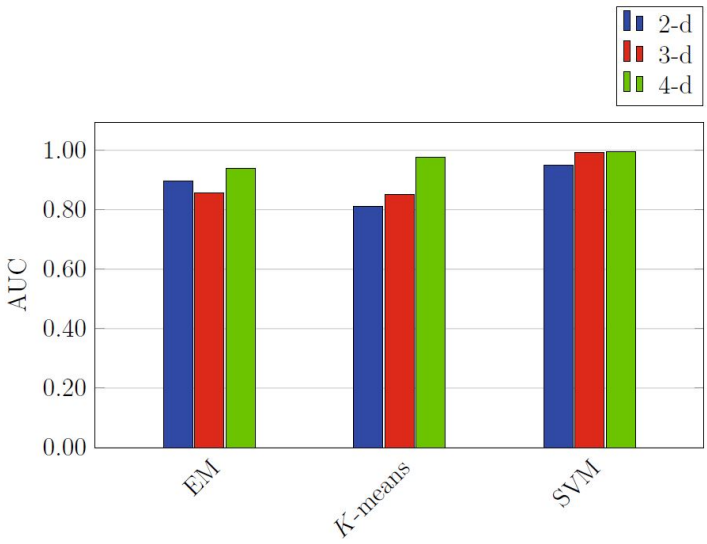


Figure 14 - Clustering and SVM AUC comparison

In the light of these results, the clustering approach can be recommended as a first line of defence against new malware. Also, this technique can be used to automatically filter out new malware samples for further analysis, hence, allowing for a faster process of building new malware models from unknown threats.

6.3 Conclusions

In this Section, we applied machine learning to the problem of detecting previously unknown malware samples, or *zero-day*. A comparison of two clustering algorithms (*K*-means and EM) was performed in a purely unsupervised form. In this case, several clusters were generated

without any prior knowledge of the malware families involved. From these experiments, EM appeared to be a stronger candidate for this type of challenge.

Furthermore, a supervised approach to *zero-day* detection was tested. To simulate this circumstance, we used clustering algorithms to cluster a given number of known families, and then we tried to classify samples from a family still unknown to the algorithm by using these clusters.

As a means of comparison, Support Vector Machines were also used. In particular, an SVM was trained on code from a given number of compilers and specific assembly code. The results were also interesting, especially considering that the SVM model was not aware of the malware families code, thus, proving the potential of SVM as a candidate *meta-classifier* for the experiments in Section 8.

7 Effectiveness of Machine Learning algorithms

After applying machine learning models on opcodes and API calls sequences, attention was shifted to different type of approaches to the malware detection problem. The goal here is to justify the use of machine learning algorithms by comparing them to other unconventional means of detection. One class of approach is applying cryptanalytic techniques to detect malware samples (Deshmukh, et al., 2018) that may be able to disclose underlying structures in the program code treating it as encrypted text. Another converts binary files into images before classifying the malware samples using an artificial neural network (Yajamanam, et al., 2018), taking advantage from its ability to detect known patterns from image files. Then, an *ad-hoc* method based on function call graphs is introduced (Rajeswaran, et al., 2018) to test how analyzing directly the structure of a program compares to machine learning. Finally, robust hashing and its characteristic to ignore modifications to the input data becomes the base of the last proposed technique (Huang, et al., 2018).

7.1 Vigenère scores for malware detection

This approach applies cryptanalytic techniques to detect malware in samples. In our work (Deshmukh, et al., 2018), two malware scoring techniques based on the classic Vigenère cipher were tested. These relied on the index of coincidence (IC), which is used, for example, to determine the length of the keyword in a Vigenère ciphertext.

7.1.1 Related Work

The novelty of this work is to apply the logic of Vigenère cryptoanalysis to the problem of malware detection. This set of experiments is based on the work in (Shanmugam, et al., 2013). The authors show that a distance measure based on simple substitution cryptanalysis can provide a strong score in the context of malware detection. The analysis in (Srinivasan, 2015) extends the simple substitution attack to the case of a cipher that combines simple substitution with column transposition. However, differently that in the previous work, the cryptanalytic techniques are *not* applied to ciphertext in our experiments. Instead, these techniques are applied to obfuscated malware code. The insight is that many common code obfuscation techniques can be viewed as substitutions and permutations at the level of

opcodes, and hence such techniques are analogous to operations performed by many types of classic ciphers. Thus, a score based on a classic cipher can be viewed as providing a distance measure between obfuscated code and some idealized version of the code. For such cipher-based scores, the idealized version of the code plays the role of plaintext, while the obfuscated version plays the role of the ciphertext.

7.1.2 Scoring Techniques

Index of Coincidence (IC) is used in our scoring phase. There are two main ways to compute the IC, namely, the Kasisky's Test and the Friedman's Test, and both were considered in these experiments. Furthermore, a score based on a complete cryptanalysis of a Vigenère cipher was tested. The IC based score in computed as:

$$IC\ score = |k_f - k_s| \qquad (6)$$

where κ_f is the IC value determined from the opcodes of the given family *f* and κ_s is derived from the sample that we want to score. If the sample *s* belongs to the family *f*, we expect the score to be small, as compared to cases where *s* does not belong to *f*.

Following the steps of the Friedman's test, the Vigenère keyword length is estimated (note that the alphabet statistics are computed on the opcode sequences of the malware family samples). That is, we compute:

$$L = \frac{\kappa_f - \kappa_r}{\kappa_s - \kappa_r} \qquad (7)$$

where κ_r is the IC value based on the random case.

Using the value of *L* computed from the formula (2) above, we determine the sequence of shifts $\sigma = (\sigma 1, \sigma 2, \ldots, \sigma_L)$ using the formula:

$$\sigma_\ell = arg\,max_j \sum_{i=0}^{c-1} f_i(\ell) F_{i+j}\ mod\ c \qquad (8)$$

where $f_i(\ell)$, for $i = 0, 1, \ldots, c-1$ and $\ell = 1, 2, \ldots, L$, is the relative frequency of opcode *i* in the ℓth column, F_i is the relative frequency of opcode *i* in the malware family samples, and *c* is the total number of symbols in the alphabet (in this case it would correspond to the number of unique opcodes used in these experiments).

To compute the actual score and be consistent with previous work (Shanmugam, et al., 2013)., we use the recovered offsets to "decrypt", then we compare the digraph distribution matrix of the putative "decrypt" to the expected digraph distribution matrix for the family.

Let $D = \{d_{ij}\}$ be the normalized digraph distribution matrix corresponding to s, and let $E = \{e_{ij}\}$ be the normalized digraph distribution matrix for the family f. Then the Vigenère score is defined as:

$$Vigenère\ score = \sum_{i=1}^{c}\sum_{j=1}^{c}|e_{ij} - d_{ij}| \qquad (9)$$

If the sample s is similar—in the Vigenère sense—to the elements in the family f, then we expect this score to be relatively small. Conversely, in cases where s differs significantly from the family f, we expect the score to be relatively large. Further details about these techniques can be found in our work (Deshmukh, et al., 2018).

7.1.3 Results

The dataset used is still the same defined in Section 2.1, specifically the Malicia and NGVSK samples. The results for both proposed scores are given in Figure 15. As a comparison, SSD scores are included from (Shanmugam, et al., 2013). and SSCT scores from (Srinivasan, 2015), where the SSD scores are based on simple substitution cryptanalysis, while the SSCT scores are based on a simple substitution and column transposition.

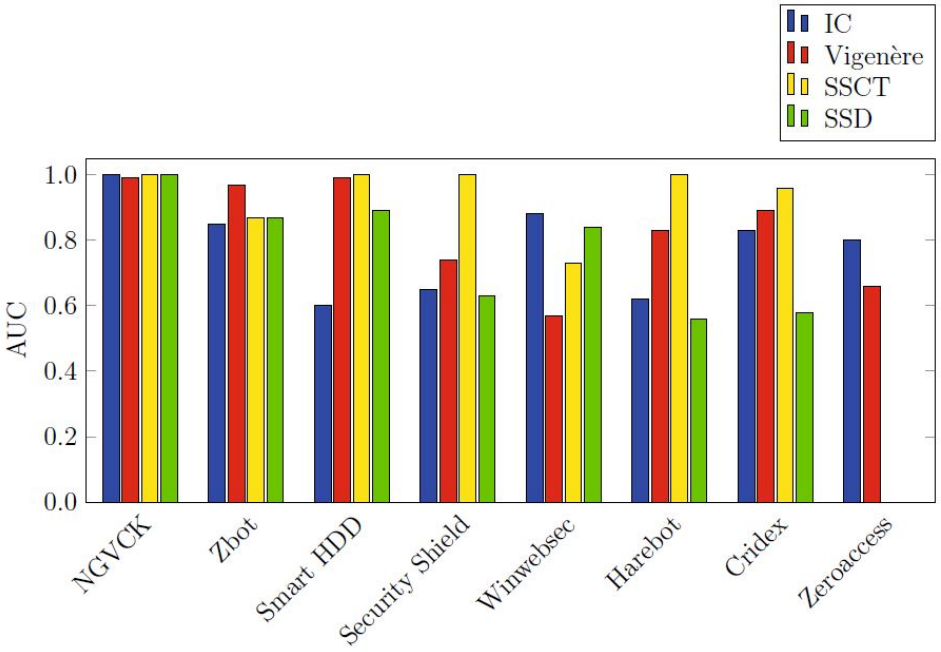


Figure 15 - AUC comparison for the cryptanalytic techniques

Figure 15 shows the results of detecting each individual family against benign samples. In particular, the Vigenère score outperformed the previous techniques in the particular case of detecting the Zbot family, whereas the IC score exceeded the earlier methods in detecting the Winwebsec family. Besides, a complete classification was obtained for the NGVCK family by almost all the proposed techniques, except for the Vigenère score that is anyway very close to the others.

7.1.4 Conclusions

Based on these experiments, none of the proposed techniques has proved to be particularly effective in detecting the malware samples. The average detection rate (which in the best case of SSCT reaches around 0.93) is still far from the perfect detection obtained in Section 5.2 using dynamic features. The cryptanalytic scores, though, are comparable with the static results obtained in previous researches. Unfortunately, this type of approach proved to be inconsistent, with results that vary substantially among the different families used in these tests. This led to the conclusion that the machine learning based approach was the more beneficial.

7.2 Deep Learning versus Gist Descriptors for Image-based Malware Classification

In our work (Yajamanam, et al., 2018) the problem of malware classification was approached by applying a deep neural network where the training was effected by converting malware samples in grayscale images. As a mean of comparison, the work in (Oliva, et al., 2001) was reproduced, where image features known as “gist descriptors” were extrapolated from the samples. In this paper, a method for constructing a “spatial envelope” of an image is introduced in terms of various properties such as “naturalness” and “openness.” The idea is to capture the “gist” of the images based on properties that are shown to be meaningful to humans, providing a connection between visual and semantic information.

7.2.1 Related Work

The motivation for the work presented in this Section is the paper (Nataraj, et al., 2011), where high-level image features known as “gist descriptors” are used to successfully classify malware samples. The strength and robustness of gist-based malware classification is here tested against an image-based deep learning technique that does not employ gist descriptors. In (Oliva, et al., 2001), these descriptors, which are shown to be meaningful properties to

humans, are designed to capture the “gist” or essence of an image by providing a connection between visual and semantic information. The novelty of our work is the direct approach of training the model directly on the obtained images, without extracting any specific feature.

7.2.2 Experimental setup

For this research, the dataset defined in Section 2.1 was used, focusing on three major families: Zbot, WinWebSec, and ZeroAccess. In fact, these families are the only ones in the dataset that have enough samples to train the neural network. However, being our experiments based on image samples, the malicious files taken from such families have been processed to produced grayscale versions. The procedure is relatively straightforward: a single byte of the sample is converted to its grayscale representation, so to obtain a full grayscale image where each pixel corresponds to a given byte in the file. However, to ensure compatibility among files of different size, the dimensions of the obtained images were kept constant. Furthermore, images directly taken from 25 malware families present in Malimg were used (though, in the latter case, no image processing is required, since the samples are already in grayscale image format).

The insight of converting malware samples to their grayscale image representation is to test how effectively CNN is able to ignore the changes due to obfuscation techniques after a model was trained on the given malware family. In fact, the resulting obfuscated samples from the same families look particularly similar to the human eye (an example is shown in Figure 1).

The layers implemented in the CNN are described in Table 3, while the input has a shape of [64 * 64 * 3].

Table 3 - CNN Layers

Layer	Notes
Convolutional	30 filters, (3 * 3) kernel size
Max Pooling	(2 * 2) pool size
DropOut	Dropping 25% of neurons
Flatten	
Dense/Fully Connected	128 neurons, Relu activation function
DropOut	Dropping 50% of neurons
Dense/Fully Connected	50 neurons, Softmax activation function
Dense/Fully Connected	num_class neurons, Softmax activation function

7.2.3 Transfer learning

When working in the deep learning field, a crucial obstacle to overcome is the training of the model. In fact, training a deep neural network from scratch is computationally an extremely intensive task. To speed up this process, an approach known as Transfer Learning was followed (Pan, et al., 2010). This started with a model pre-trained on another problem, which was then retrained on a similar problem. For our experiments, the artificial neural network was pre-trained on the ImageNet Large Visual Recognition Challenge dataset taken from the Google Brain project "TensorFlow for Poets" (Tensorflow, 2017). This procedure was introduced to implement the Transfer Learning feature. Finally, this model was retrained on the raw malware images taken from our dataset.

7.2.4 Results

When attempts were made to classify samples from all the 25 malware families in Malimg against each other, without taking into consideration the unbalancing of such families in terms of samples per family, the obtained accuracy was above 98% (see Figure 16). For these experiments, various splits of the training and test data were tested, specifically from 30-70, where 30% of images were used for training and 70% for testing, to 90-10, where 90% of images were used for training and 10% for testing.

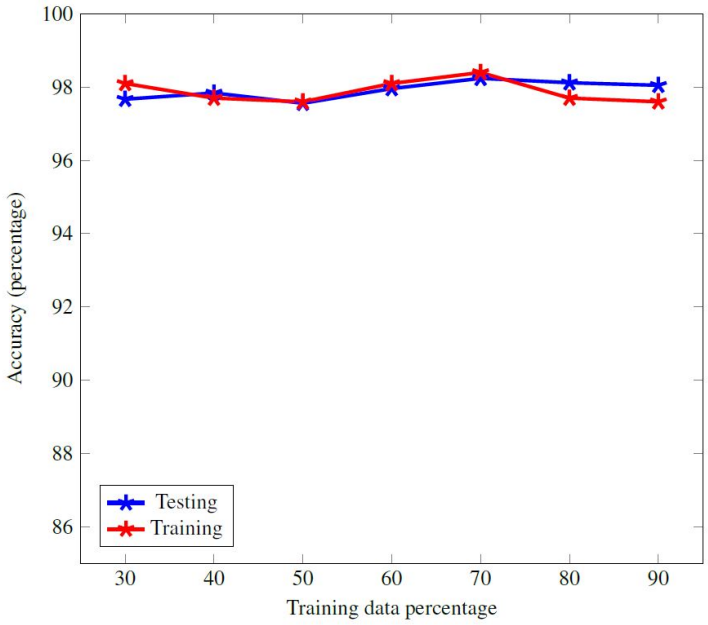


Figure 16 - Accuracy vs training/testing split for TensorFlow experiments with all 25 Malimg families

This over 98% accuracy is as good as that obtained using gist descriptors (Nataraj, et al., 2011). However, since there is no need to extract the gist features beforehand, this technique is likely to offer superior performance, especially when scoring large numbers of samples.

To cope with the unbalancing of the Maligm families, another set of experiments were performed where the dataset was first balanced (see Section 2.1). In fact, the number of samples of the various malware types varies too much among the families, from an overrepresented family with 2949 samples, to an underrepresented family with only 80 samples. Hence, the deep learning experiments were repeated, using 80 samples from each family. The results of these experiments are summarized in Figure 17, which shows that the testing accuracy is significantly lower than in the original unbalanced case. It is likely that the improved results in the previous experiments were due to the ease with which the most represented families were detected in relation to the others. Unfortunately, the tests in (Nataraj, et al., 2011) were based on the unbalanced dataset and no direct comparison can be obtained in this case.

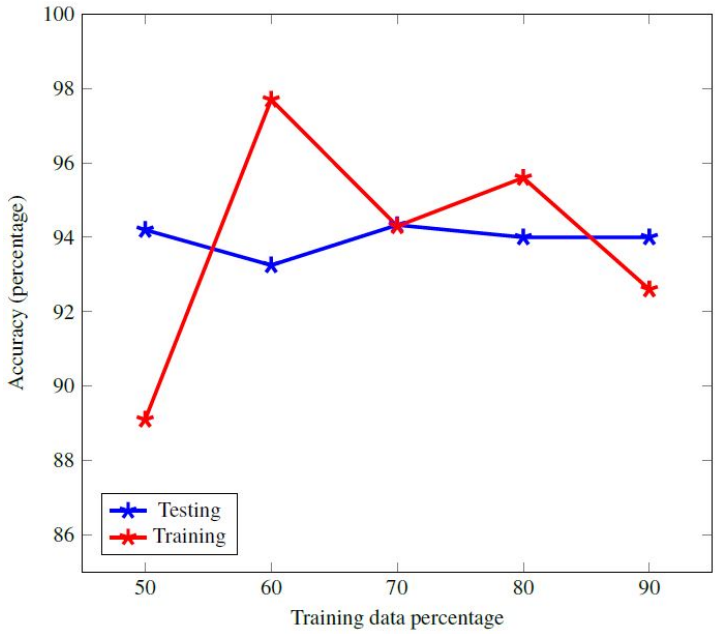


Figure 17 - Accuracy vs training/testing split for TensorFlow experiments with all 25 Malimg families (balanced)

7.2.5 Conclusions

A deep learning approach proved to be effective, even though its results were still far from the perfect detection aimed for, especially for the balanced case. Unfortunately, it also introduces the drawback of training the model, that introduces a non-negligible overhead even if Transfer Learning is applied. Subsequent experiments (especially those introduced in Section 8) do not rely on deep learning, favoring more efficient approaches where no preliminary work on the samples is required.

7.3 Function Call Graphs versus Machine Learning for Malware Detection

An interesting *ad-hoc* approach based on function call graph technique has proved to perform well on some challenging malware detection problems (Shang, et al., 2015). In our work (Rajeswaran, et al., 2018), it was decided to compare this against the machine learning techniques introduced in Section 4, that is, HMM and SVM. In addition, it was decided to add other two methods, namely, Opcode Graph Similarity and Simple Substitution Distance. These experiments were designed to test if machine learning approaches could perform better than *ad-hoc* methods.

7.3.1 Related Work

In (Christodorescu, et al., 2003) we see the first implementation of a static analysis malware detection technique based on a control flow graph. In (Xu, et al., 2013), the authors develop

and analyze a similarity metric based on the function calls in an executable. In (Shang, et al., 2015) a discussion on the time and space complexities involved in control flow graphs in the field of malware detection is proposed.

While these papers prove that a graph-based approach can be beneficial and obtain interesting results, the experimental evidence presented in our work indicates that simpler machine learning based scores are significantly more robust when tested against highly obfuscated malware samples.

7.3.2 Experimental setup

In this research, the dataset described in Section 2.1 focused mainly on the Zbot, ZeroAccess and Harebot malware families. To extrapolate the necessary features for the function call graph approach, tests heavily relied on IDA Pro, which was used to extract the functions from all the executable files used in this research. These functions were categorized as either local or external. Local functions are the ones that begin with "sub_xxxxxx proc near" and end with "sub_xxxxxx endp", while the external functions are system calls and library routines. The function call graph technique considers only the local functions and the opcode sequences within them. A function call graph is of the form $G = (V, E)$, where V is the set of vertices, which in our case are the functions, and E is the set of edges, which are the relationships between functions. More details about the scoring applied can be found in our work (Rajeswaran, et al., 2018).

Another set of experiments were designed to test the robustness of each technique, simulating the effect of code morphing. To achieve this goal, external functions were added as dead code, which are randomly inserted as subroutines in the original structure of the malware.

7.3.3 Results

Table 3 shows the AUC results when function call graph was used for detection against each tested family, showing that this technique is capable of detecting Zbot with reasonable accuracy.

Table 4 - Function Call Graph results

Malware family	***AUC***
Zbot	0.96
ZeroAccess	0.77
Harebot	0.60

However, it is partially ineffective against ZeroAccess and slightly better than a coin toss against Harebot, showing it to be of little or no use as an efficient classifier.

Table 4 shows the results of the robustness experiments on the malware family Zbot. It is clear how much the AUC deteriorates when a greater percentage of dead code functions is added to the samples. The technique reaches a level of detection that is sub-par in comparison to an acceptable detection rate, even adding a relatively small percentage of dead code functions.

Table 5 - Robustness results - Zbot family

Morphing	***AUC***
0%	0.96
5%	0.96
10%	0.93
15%	0.84
20%	0.87
25%	0.84
30%	0.83
35%	0.84
40%	0.83

Figure 18 shows a direct comparison between function call graph and the other techniques, particularly, against the proposed machine learning algorithms. The scores shown for the remaining approaches are taken from our work (Singh, et al., 2016).

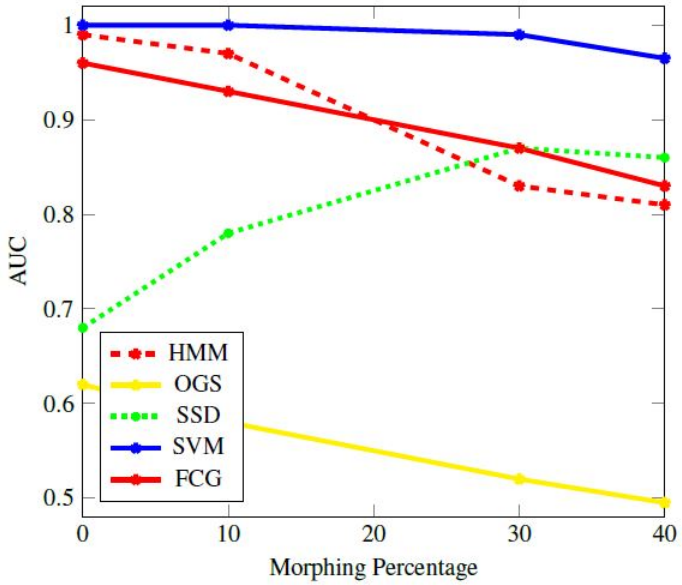


Figure 18 - Comparison Function Call Graph score vs Machine Learning Score for Zbot

It is observable that the machine learning techniques, i.e., HMM and SVM, are generally more robust than the function call graph approach. That is, to defeat the machine learning

algorithms, the malware code must be morphed to a very high degree before observing a degrade in the detection rate. This requirement is particularly true for SVM.

7.3.4 Conclusions

In general, function call graph can perform well against particular families, but it appears to be somewhat "fragile" in comparison to machine learning algorithms. This characteristic is crucial because it indicates that if an attacker can study the *ad-hoc* strategy used for detection, it could be possible to design a morphing strategy that would be able to degrade the detection rate considerably.

7.4 Robust Hashing for Image-Based Malware Classification

Section 7.3 showed that machine learning, and especially SVM, outperformed an *ad-hoc* technique such as function call graph. In our work (Huang, et al., 2018), SVM was compared with another sophisticated approach based on robust hashing which has the peculiarity that when two inputs are similar, the hashes of the two samples look very similar. This property is the exact opposite of the collision resistance characteristic of a cryptographic hash function.

7.4.1 Related Work

The paper (Venkatesan, et al., 2000) proposes robust image hashing to handle the proliferation of digital images. Feature extraction for robust image hashing is considered in (Monga, et al., 2006), where the proposed process consists of feature vector extraction followed by compression, with accuracy close to 83%, while the technique in (Venkatesan, et al., 2000) relies on wavelet decomposition for image feature extraction, with accuracy close to 80%. The novelty of the work described in this Section is to combine both approaches to increase the detection rate in the malware classification problem, obtaining an accuracy superior to 88%.

7.4.2 Experimental setup

Since this is another image-based detection, the same dataset as the one seen in our work (Yajamanam, et al., 2018) in Section 7.2 was used. Two image-based robust hashing techniques were examined, one that relies on wavelet analysis, and one that uses distributed coding. To be able to test SVM against 2D images, an image-based feature was extracted that deals with horizontal edges.

By applying robust hashing to image samples, similar hashes can be obtained from two images differ by a few details. Taking advantage of this property, malware samples that belong to the same family can be detected, even in the presence of some degree of obfuscation. The idea is that the modified code would still maintain enough information on the original structure of the file so as to furnish a similar hash as the one obtained without obfuscation.

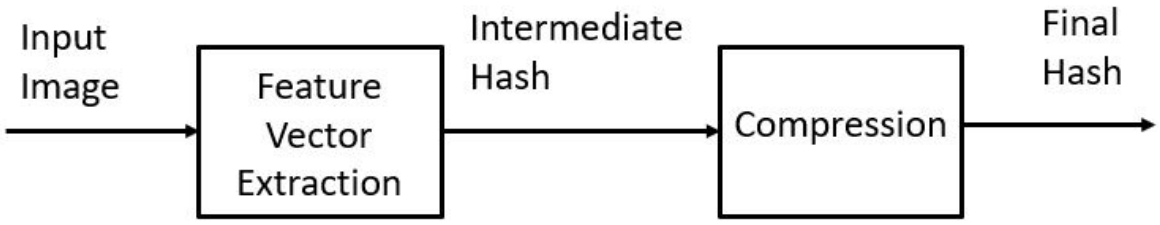


Figure 19 - Process of robust hashing

Figure 19 shows the full process used to apply robust hashing. This technique is the same as that used in (Monga, et al., 2006), where two steps are performed. The first one involves the extraction of the relevant features to generate an intermediate hash value, while in the second negates minor differences and noise through compression step, allowing the final hash value to be generated. Two different compression techniques have been tested in this research: one is based on error-correcting codes, while the other on distributed source coding. For these experiments, several image features were collected from the dataset samples, i.e., Local Binary Pattern, Histogram of Oriented Gradient, Horizontal Edges, Pixel Intensity, Contrast, Median Filter, Frequency Distribution and Wavelet Transform. More details about these image features and the compression techniques can be found in our work (Huang, et al., 2018) and in the referenced papers.

7.4.3 Results

Figure 20 shows the results using SVM per each of the tested families. In this case, the average classification accuracy obtained per family was of around 84%, while per individual sample the average rises to around 93%. It is, in fact, possible to observe in the Figure that some of the least represented families in terms of samples are obtaining an inferior classification accuracy. Furthermore, removing the seven families with the lowest accuracy, an average in excess of 98% was obtained. These results prove how powerful a single feature as the horizontal edge could already be handy for this image-based malware classification problem.

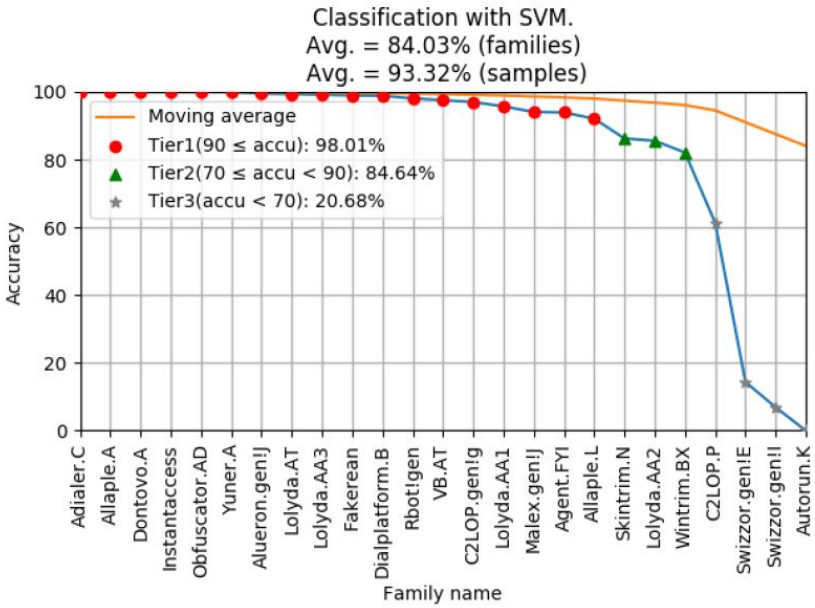


Figure 20 - Classification accuracy using SVM

Further investigations aimed to find the tradeoff between the features used and the accuracy obtained. To achieve this goal, both univariate feature selection (UFS) and recursive feature elimination (RFE) were applied (see Section 4.3.1). The results confirm that the most prominent feature is, indeed, the horizontal edge, which was further analyzed using RFE. Figure 21 shows the outcome of this experiment. Recall that this edge feature is 256 dimensional, and it was observed that by using 169 of the 256 dimensions optimal accuracy was achieved.

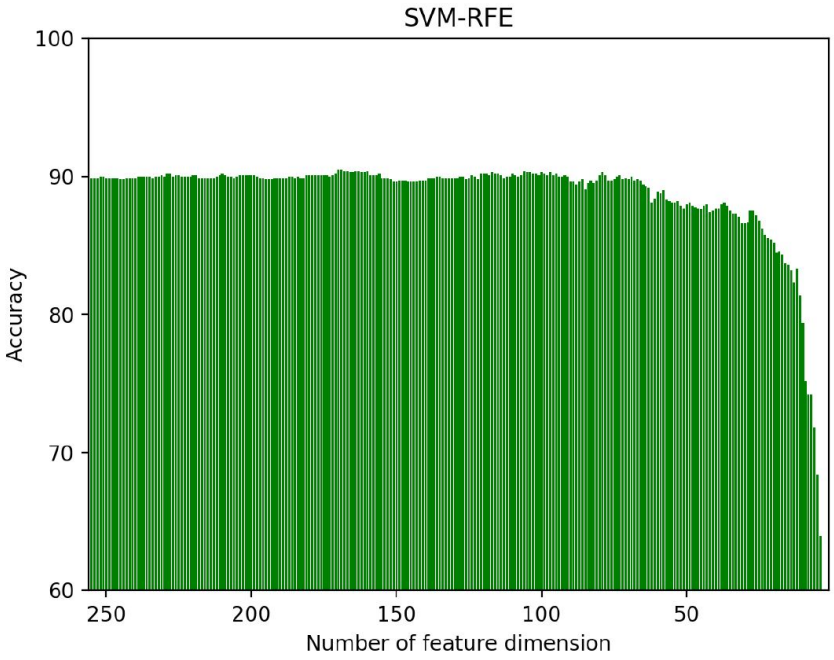


Figure 21 - RFE result on horizontal edge feature

This 169-dimensional set of horizontal edge features was used in the subsequent SVM experiments showing the per family accuracy average increased from around 84% to around 92%.

Next, the robust hashing approaches were tested. The first set of experiments tested an error-correcting algorithm based on a 2-dimensional Haar wavelet transform and a five-level decomposition. The average accuracy, in this case, is about 79%. This result proves that this specific approach is weaker compared to the SVM detection seen earlier. The second set of experiments used distributed coding, from which the average obtained is slightly over 83%. This is an improvement in comparison to the wavelet-based approach, but still not on par with SVM. Finally, a multiphase strategy was implemented and tested to robust hashing, where both wavelet-based and distributed coding based robust hashes were combined. Intuitively, two robust hashing algorithms were combined by scoring with both techniques, and when the classifications disagreed, the one with the higher accuracy for its model was selected. In this combined case, an improved accuracy of above 87% was obtained.

Figure 22 shows the comparison of all the three robust hashing approaches.

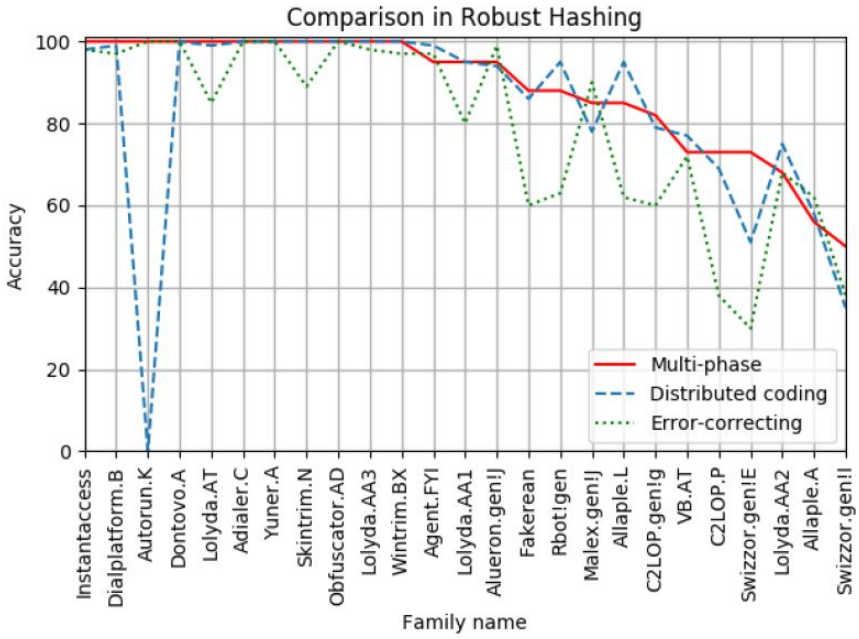


Figure 22 - Comparison of robust hashing approaches

7.4.4 Conclusions

In conclusion, an optimized SVM obtained somewhat superior detection accuracy, but the best robust hashing results are comparable. However, it is worth noting that robust hashing performed better than the SVM on most families, while a few malware families are too complicated to classify. This is due to the amount of obfuscation applied to the samples of these specific families. It seems that the more prevalent obfuscation is in the samples, the more distinct the hashes are, which results in a less accurate detection. In contrast, the SVM yields more consistent results over the 25 families. For this reason, the consistency in accuracy obtained by applying machine learning algorithms was favored. In particular, it was proved how effective SVM could be, even in this case, after being advantageous already in the *zero-day* experiments seen in Section 6. Section 8 introduces techniques to boost the detection rate of different approaches and Section 8.2 uses SVM to combine these scores, proving even further how malleable and effective this algorithm can be in the malware detection problem.

7.5 Conclusions

After this set of experiments, we had a clear understanding of how effective machine learning based classification can be in detecting highly obfuscated samples when compared to *ad-hoc* techniques. In particular, SVMs were tested in several cases and proved to be a feasible

algorithm on top of which building our static classification technique. More importantly, it proved to be particularly effective when used as a *meta-classifier*. HMMs were also tested in several experiments and proved to be reliable and consistent. A combination of these two techniques seemed the right way to obtain improved classification so to reach higher accuracy in classification, as described in Section 8.

8 Improved classification

Section 5.2 applied PHMM using dynamic features to classify malware samples, obtaining AUC of 1.0, that is, perfect detection. The static based experiments have not yet been able to reach this grade of detection. However, this Section proposes approaches to increase further the static analysis detection rate so to achieve the ideal classification. Specifically, the following methods were tested: our work in (Raghavan, et al., 2019) uses a comparison between HMM with random restarts and AdaBoost, while our work in (Singh, et al., 2016) proposes a different application of SVMs where those are used to combine scores from several other techniques.

8.1 Improved Hidden Markov Models for malware detection

Considering that the training for Hidden Markov Models (HMMs) is based on a hill climb, it is possible to improve the model by simply trying different initial values. In our work (Raghavan, et al., 2019), attempts were made to test HMM with random restarts and compare it with another popular technique to boost classifying results. This boosting technique is known as AdaBoost (Section 4.8), which here is used to build a "boosted" version of HMM where the classifiers used in the AdaBoost algorithm are based on different HMMs, each of which initialized with a different random starting model. Apart from the standard classification of the samples, experiments were introduced whereby the samples were "morphed", i.e. a variable percentage of benign opcode sequences were added to the original code. These experiments were meant to simulate morphing strategies by the malware creators. Another more complex testing challenge defined as "cold start" was also proposed: in the "cold start" case, the models rely only on a limited amount of training data, thus limiting the classifying effectiveness of the proposed technique. This approach is meant to reproduce the case where samples of a new malware family have been detected, but they are still too limited in number to obtain sufficient training of the machine learning models.

8.1.1 Related Work

Interestingly, it appears that combining HMM and boosting have not previously received much attention in the information security domain, with the exception of the paper (Chen, et

al., 2009) in the context of anomaly detection. In our work, we proposed a detailed implementation of this technique in detecting obfuscated malware, with in addition a comparison with HMM with random restarts and a classic HMM-based classification. Furthermore, the set of experiments described in this Section includes a so-called "cold start" problem, where limited training data is available. The idea is to simulate the slow discovery of new malware samples belonging to an unknown malicious family while a model for such family is being built.

8.1.2 Results

The dataset used is the one defined in Section 2.1, limiting the experiments to the Cridex, Harebot, Security Shield, Zbot, and ZeroAccess families, while the feature used was opcodes sequences. Figure 23 shows that HMM with random restarts obtains better results than the average HMM, where the "average HMM" is the average model over 1000 HMMs, justifying the additional workload during the training phase. However, the boosted HMM outperforms HMM with random restarts in classifying Cridex and Security Shield, while it has little effect on the results for Harebot, Zbot, and ZeroAccess.

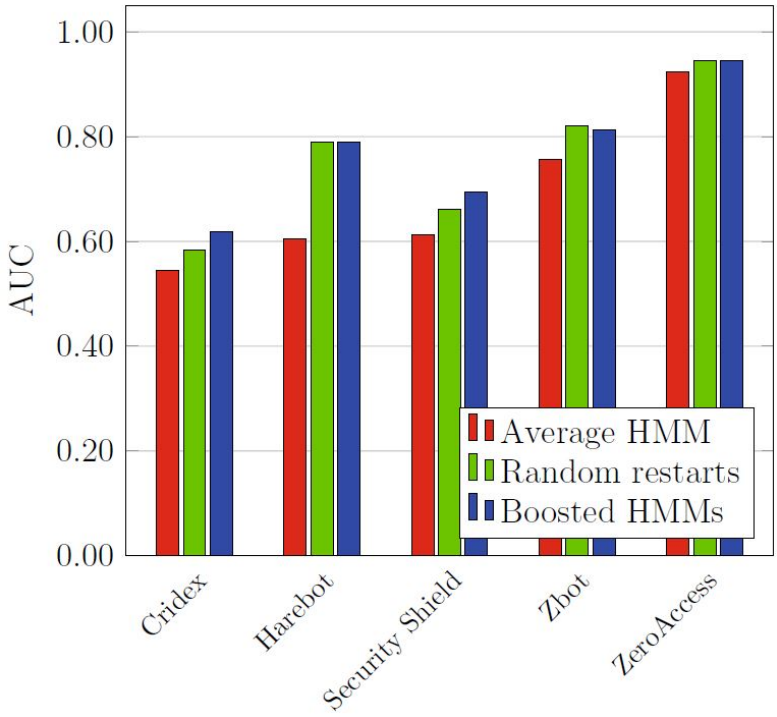


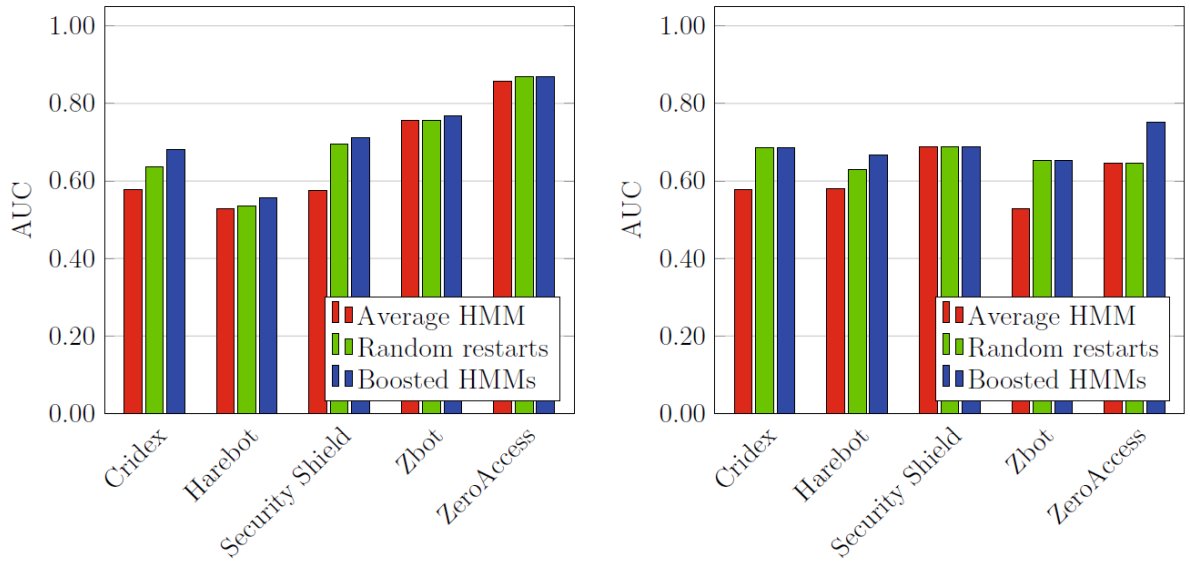
Figure 23 - Initial Experiments

An essential drawback in using AdaBoost is the increased workload during the scoring phase since multiple HMMs are used in the boosted classifier. On the other hand, for random

restarts the best model was selected, and hence the scoring step is no more costly than a single HMM. Keeping this in mind, it was concluded that random restarts were the approach to select. In fact, the advantage given by boosting is marginal in relation to the results obtained with random restarts.

Figure 24 shows the results of the testing experiments after morphing the malware samples, that is, after adding random sequences of opcodes taken from the benign dataset. Note that the obtained samples are not functional, because the sequences have been added directly to the extracted opcodes lists. Three morphing cases were considered. First, benign code equivalent to 10% of the original malware code was inserted, followed by 50% morphing, and finally, 100% morphing. As with the previous experiments, the differences between the average HMM and the best of the random restarts model is generally significant. On the other hand, the differences between the random restarts and the boosted model are not incisive in the majority of the cases. The exceptions are Cridex at 10% morphing and ZeroAccess at 50% morphing, where both show substantial improvement for the boosted models.

Because it is not possible to clearly state that boosting offers better performance, the results still favorited HMM with random restarts over the boosted HMM, since the improvement provided by boosting is insufficient to justify the additional work required when scoring via boosted models.



(a) Morphing at 10%

(b) Morphing at 50%

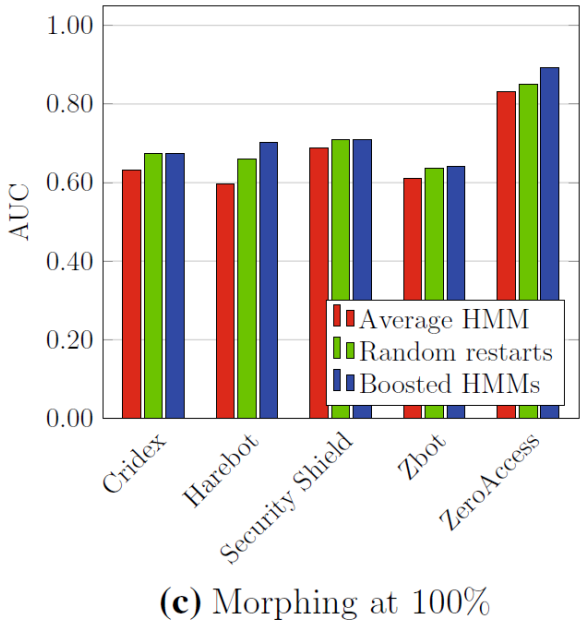


Figure 24 - Morphing experiments

The final experiments introduced the "cold start" cases, a special type of problem that deals with training data that is severely limited. This particular test observes the advantage of the random restarts over the average application of an HMM model. The benefit of boosting in these experiments, though less pronounced, was still significant in some cases, especially those that were the most challenging for the random restarts.

8.1.3 Conclusions

This set of experiments leads to the conclusion that the overhead introduced by boosting is worthwhile only in extremely challenging cases, while HMM with random restarts is the best choice in other cases. However, both approaches proved to be superior to the generic application of HMMs. The scores though, even in the best cases, were still far from the ideal detection obtained in Section 5.2. For this reason, in Section 8.2, a different type of improved classification mechanism is proposed.

8.2 Support vector machines and malware detection

In our work (Singh, et al., 2016), another approach was tested to improve the detection rate of different algorithms. In conjunction with the methods covered in Section 7.3, *ad-hoc* detection algorithms with machine learning using Support Vector Machines (SVMs) were incorporated as a means to combine different scoring techniques. Specifically, feature vectors for SVM were built combining scores from HMM, Opcode Graph Similarity (OGS) and Simple Substitution Distance (SSD). More details about these two algorithms can be found in (Singh,

et al., 2016). As in Section 8.1, this approach was tested by simulating code morphing of the malware samples.

8.2.1 Related Work

Combining various classification techniques has been explored in (Zhang, et al., 2007) where the authors use Dempster–Shafer theory to create combining rules for individual decisions based on probabilistic neural network (PNN) classifiers. The ensemble outperforms the individual PNN classifiers. The paper (Lu, et al., 2010) considers an ensemble method, called SVM-AR, which combines a SVM with association rules. The authors conclude that this algorithm is essentially a single learning algorithm that yields better results than some ensemble techniques. Based on these promising results, our work implements an ensemble technique where the novelty is defined by using SVMs as a *meta-classifier* to combine scores from other algorithms. The idea is to produce a more in-depth classification over a multidimensional space, taking advantage of the so-called kernel trick introduced in the SVM algorithm.

8.2.2 Results

As in the previous research, the dataset used is the one described in Section 8.1, with the addition of NGVSK malware family samples. The feature used was, again, opcodes sequences extracted from the dataset files. NGVSK samples have been classified ideally (AUC = 1.0) by each of the individual techniques. As expected, when SVM was used to combine these scores, the AUC was again 1.0.

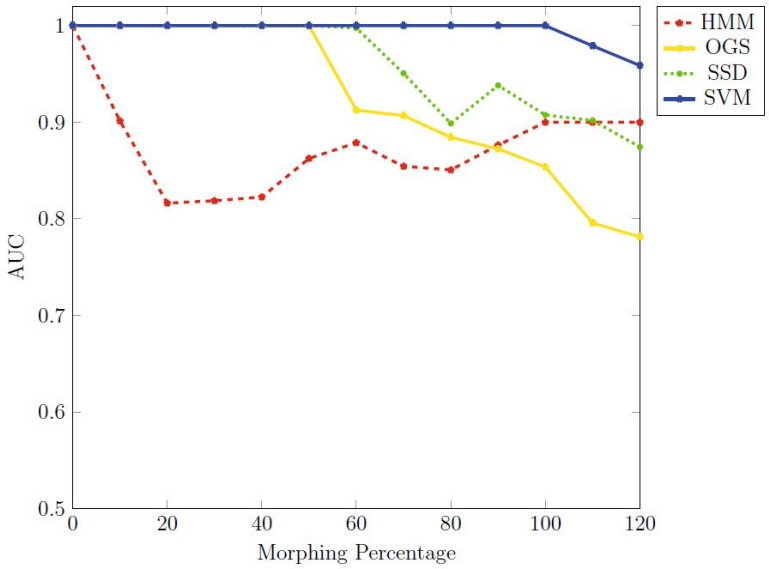


Figure 25 - AUC at various morphing percentages (NGVCK)

Figure 25 shows the results of the morphing experiments, still for the NGVCK family. In this case, opcodes sequences from the benign samples were added directly in the sequence of malware opcodes. It is observable that the HMM score deteriorates significantly even at only 10 % morphing. The OGS score only began to fail at 50 % morphing, while the SSD scores began to fail at 60 % morphing. On the other hand, SVM, combining all the other scores, proved to be particularly robust: in fact, the SVM achieved ideal detection until the morphing rate reached 100%. Moreover, even in the other cases, SVM outperformed the results obtained for all the individual scores. This outcome clearly shows the strength of the SVM as a method for combining scores obtained by other classification algorithms.

After this preliminary test, the results of this approach were compared to those seen in Section 5.2, obtained using PHMM and dynamic analysis. Hence, to allow a fair comparison, this approach was tested against samples taken from the same families, as seen in Section 5.2. All the ROC curves of these experiments can be found in (Singh, et al., 2016). Of the individual scores, the HMM consistently performed well, while the SSD was satisfactory only in some cases. However, the OGS was the weakest of the three algorithms. It was also observed that the SVM achieved ideal separation for all families, a result entirely on par with that seen in Section 5.2.

Moreover, in the morphing experiments the HMM tended again to decline significantly at low morphing rates. However, unlike the NGVCK results, the OGS score generally gave the poorest results. Considering SSD, instead, the results were surprising, with the detection accuracy that improved at lower morphing rates in few cases.

8.2.3 Conclusions

Overall, the use of SVMs to combine individual scores outperformed the direct application of the single techniques, with the one exception of the SSD algorithm, that performed slightly better in a few cases at midrange morphing rates. In almost every case, the SVM combined score technique was better initially and proved to be more robust to code morphing, degrading more slowly than any of the other approaches.

Regarding the dynamic experiments from Section 5.2, combining scores using SVM allowed the same ideal rate of detection to be reached as when using dynamic features. Hence, even

though based on static analysis, when SVM was used to combine scores, a form of improved classification was achieved that is robust and effective.

8.3 Conclusions

The results obtained through AdaBoost were particularly interesting for the "cold start" problem. However, the use of such algorithm (and the overhead introduced in the classification phase) was not beneficial in the other experiments where HMM with random restarts was able to reach comparable results. In both cases, while proving the potential of these techniques, the perfect detection that we were aiming for was not reached yet. However, as intuitively understood by the set of experiments in the previous Sections, the use of SVMs as *meta-classifier* proved to be particularly effective in boosting the outcome of our classification accuracy. In fact, not only this approach reached perfect detection, but it was also able to maintain this detection rate even in case of dead code insertion up to 100% of the original code.

9 Generic Malware Models

This Section analyzes the case where a generic malware model is used for classification instead of a specific model for each family to classify. In the previous Sections, different techniques were tested that proved to be suitable as malware detection tools. However, with the only exception of the *zero-day* experiments, all the models were pre-trained on the specific malware families to detect which created a 1-to-1 relationship between the samples to classify and the model created. However, the approach where a comprehensive model is built from several families sees more realistic since we would be able to detect multiple families without the necessity to run individual models for each family. In a real-life scenario, this would be very complex and inefficient to pursue, because a separate machine learning model would need to be built on each malware family to detect. This Section analyzes how effective a generic malware model could be and how much it degrades as the number of families used to generate it is increased.

9.1 On the Effectiveness of Generic Malware Models

(Bagga, et al., 2018) addressed the problem of generic malware models performing experiments over four machine learning algorithms to determine the tradeoff between generality and accuracy. The first algorithm tested was SVM, which proved to be particularly robust in the experiments seen in Section 8.2. Techniques considered were the chi-squared (χ^2), and two neighborhood based techniques, namely, k-nearest neighbors (k-NN) and random forests. The type of experiments performed can be summarized in analyzing the strength of each of these models when increasing the number of malware families in the training set. In other words, the model used for detection was made constantly more generic.

9.1.1 Related Work

The novelty of our work is to determine the accuracy of n-gram based malware models as a function of the generality of the training data. We build *n*-grams following the steps in (Reddy, et al., 2006), where the authors use a feature selection method that ranks n-grams based on frequency and entropy. Their experiments show that performing a class-wise feature selection improves the efficiency of the models. This process involves extracting the k most

frequent n-grams from the benign and malware sets and using a union of the two sets as features. Considering the interesting outcome of this research, a similar approach is followed in our n-gram analysis described here. On the other hand, in (Raff, et al., 2016), the authors claim that n-grams are not suitable for malware detection obtaining poor results using elastic-net regularized logistic regression. These experiments were conducted using a large dataset consisting of over 200,000 malware samples. Unfortunately, the authors' data was obtained from an undisclosed industry partner and is not available for independent verification of the results.

9.1.2 Experimental Design

This research, used the same dataset as in Section 2.1, though it focused specifically on the Microsoft Malware Classification Challenge samples. This type of experiment needed a large number of samples per family, and not all the families present in the other collections contained enough of them. In total, eight malware families were experimented upon. Furthermore, opcodes sequences were not used since exhaustive experiments had already been tried in past research. This time, the machine learning models were trained based on *n*-grams features, which proved to be more reliable for this type of experiment. Specifically, 2-grams, 3-grams, and 4-grams were used. More details about this procedure and the feature reduction technique used in this case can be found in (Bagga, et al., 2018). To measure the success of such experiments, the concept of balanced accuracy was used, which is computed as

$$balanced\ accuracy = \frac{1}{2}\left[\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right] \qquad (10)$$

where

TP = True positives (malware samples correctly classified as malware)

TN = True negative (benign samples correctly classified as benign)

FP = False positive (benign samples misclassified as malware)

FN = False negative (malware samples misclassified as benign)

This type of scoring technique was chosen because the size of the benign dataset varies depending on how many malware families are combined in each experiment. In fact, balanced accuracy calculation weights all classification errors the same, regardless of any imbalance

that might exist between the sizes of the positive and negative sets. However, for the χ^2 experiments, the area under the ROC curve (AUC) was used as a mean of determining the efficacy of the technique (see Section 2.2.1).

While for k-NNs and random forests there is no need for training, the following approach was used to train SVM and χ^2. First, each of the eight malware families was tested individually. Later, for each family, one experiment using all available samples from the family was performed, and another test was conducted using a selection of 1,000 samples. Next, SVMs were trained for each of the $\binom{8}{2} = 28$ pairs of the malware families in our dataset. Then, SVMs for all $\binom{8}{3} = 56$ combined sets of three families were trained, and afterward, all $\binom{8}{4} = 70$ sets of four families. This continued up to one very large set that includes all 8 families. In each case, the average balanced accuracy over all cases was computed for a given number of combined families.

9.1.3 Results

Each test experimented with 2-gram, 3-gram, and 4-gram features. However, it was found that 2-grams gave the best results. Figure 26 shows the overall results using 2-grams as the number of families in the training set increases. The points are representative of the average accuracy obtained per each number of families in the model. The results show that random forest is the most reliable and most robust technique, with k-NN obtaining close results. On the other hand, SVM (that so well scored in robustness experiments in Section 8.2) proved to be not a suitable algorithm for this type of test. It is also clear that the more generic is the training dataset, the weaker the resulting model.

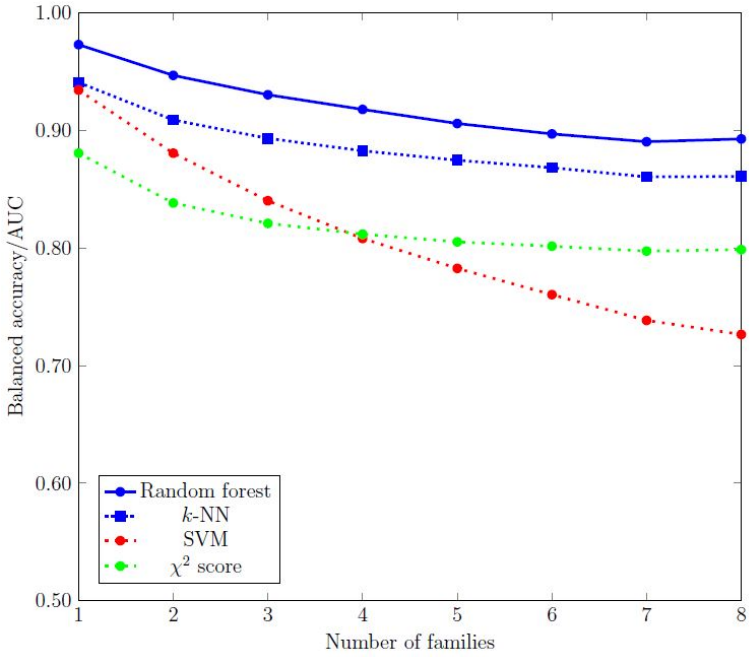


Figure 26 - Comparison of average accuracy (bigrams using all samples)

9.2 Conclusions

In these experiments, we confronted the problem of building a comprehensive malware model from several combined malware families. In particular, we focused on byte *n*-gram based analysis and we found that bigrams gave the best results.

In addition, the neighborhood based techniques (random forest and k-NN) proved to be the more robust. However, we observed that accuracy generally suffers as the models become more generic. Hence, we are likely to suffer a significant penalty with respect to classification accuracy when multiple families are combined into a single model.

10 Conclusions

After demonstrating the limited utility of hybrid approaches to training and testing, this work focused on classic static and dynamic analysis. Experiments with PHMM and dynamic features were able to reach perfect detection in all the tested cases. However, dynamic analysis introduces the critical overhead of the execution of the samples to test. So, to overcome this drawback, experiments on static analysis became the main topic of this work, with the intent to reach perfect detection. With static analysis all the necessary features can be extracted directly from the binary data.

To understand which machine learning algorithms were more effective, their models were challenged against the complex challenge of detecting *zero-day* samples. These previously undiscovered malware files were identified with reasonable accuracy by both clustering algorithms and SVM. Even though the classification was not perfect, the results were still promising, especially considering the difficulty of detecting unknown samples. Also, SVM was introduced as an effective mean of detection.

Another set of experiments were used to justify the use of machine learning against other means of detection, such as *ad-hoc* algorithms. These tested the efficacy of cryptanalytic techniques that obtained decent results, even though it was inconsistent when compared to machine learning approaches. Following this, deep learning with a particular type of image analysis based on so-called gist descriptors was investigated. Although the two methods obtained similar results, the need to convert binary files to images and the slow process of training an artificial neural network provided an incentive to work on different types of information extrapolated from the data. Later, *ad-hoc* methods were tested, such as function call graph; this, while obtaining comparable results to the machine learning algorithms, was not strong enough to resist the robustness experiments. That is, the detection rate deteriorated easily with the increasing of morphing data rate in the malware samples. Finally, robust hashing proved to reach comparable detection rate to SVM, but the results were not as consistent. In all these experiments, machine learning approaches proved to be more productive than the other methods.

The advantage of malware detection based on machine learning models therefore seems to be clear. However, as it became clear that a straightforward static model would not achieve

ideal classification, techniques which combine scores from different detection algorithms were tested. Three different approaches were tested to boost static scores: HMM with random restarts, AdaBoost, and SVM. The first two obtained favorable results, but only SVM was able to reach perfect detection in all cases. Furthermore, the combination technique via SVM was based on particularly efficient algorithms; hence, to reduce the overhead introduced by the need to compute and combine all those scores.

Finally, experiments were introduced to test a generic malware model, that is, a model trained on several malware families. In this case, SVM was not able to obtain promising results, while k-NN and random forests proved to be more effective.

10.1 Countermeasures

Different techniques can be utilized to hamper static detection through SVMs used as *meta-classifier*. SVMs are fed with several scores from various techniques, thus, a way to impede this fundamental step is by reducing the likelihood to get the actual features. A possible example can be by packing the code, or portions of it, in a similar fashion as polymorphic malware. This technique would be sufficient to confuse the statistics of many malware families, but at the same time would make the sample subject to other type of detection techniques based on entropy (an example is (Baysa, et al., 2013)) and still subject to unpacking by advanced disassembler such as IDA Pro. Also, the necessary unpacked code used to expose the packed part of the file may still be subject to detection (Stamp, 2011). An approach that would not require encryption or compression of the information is the addition of dead code. In fact, dead code insertion would influence the statistics of the original code enough to hamper detection for each of the combined techniques used to train the SVM algorithm. However, as shown in this work, the amount of added code needs to be conspicuous (at least more than 100% of the original code). It is also important to highlight that some techniques have been proposed to remove unreached code from executable files (Xi, 1999) and they could be used as preliminary step in detecting these type of samples.

Regarding dynamic detection through API calls sequences, little work has been proposed on obfuscating this feature. However, due to the poor results obtained by PHMM against static (and highly obfuscated) features (Attaluri, et al., 2009), it is possible that an attempt to apply similar obfuscation techniques to the API calls sequences may be sufficient to hamper the classification.

References

Ahmed Faraz [et al.] Using spatio-temporal information in API calls with machine learning algorithms for malware detection [Conference] // Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence. - Chigaco, Illinois, USA : ACM, 2009.

Alsabti Khaled, Ranka Sanjay and Singh Vineet An Efficient K-Means Clustering Algorithm [Conference] // Proc First Workshop High Performance Data Mining. - 2000.

Annachhatre C., Austin T.H. and Stamp M. Hidden Markov model for malware classification [Journal]. - [s.l.] : J. Comput. Virol. Hack. Tech., 2014. - 2 : Vol. 11.

Attaluri Srilatha, McGhee Scott and Stamp Mark Profile hidden Markov models and metamorphic virus detection [Journal]. - [s.l.] : Journal in Computer Virology, 2009. - Vol. 5.

Austin T. [et al.] Exploring hidden Markov models for virus analysis: a semantic approach. [Conference] // 46th Hawaii International Conference on System Sciences. - 2013.

Aycock J. Computer Viruses and Malware [Book]. - [s.l.] : Springer, 2006.

Babu A.R., Markandeyulu M. and Nagarjuna B.V.R.R. Pattern clustering with similarity measures [Journal]. - [s.l.] : Int.J.Computer Technology & Applications, 2012. - 1 : Vol. 3.

Bagga Naman, Di Troia Fabio and Stamp Mark On the Effectiveness of Generic Malware Models [Conference] // International Workshop on Behavioral Analysis for System Security. - 2018.

Baysa D., Low R.M. and Stamp M. Structural entropy and metamorphic malware [Journal]. - [s.l.] : J Comput Virol Hack Tech, 2013. - Vol. 9.

Bradley A.P. The use of the area under the ROC curve in the evaluation ofmachine learning algorithms [Journal]. - [s.l.] : Pattern Recognition, 1997. - 7 : Vol. 30. - pp. 1145-1159.

BSA: Buster Sandbox Analyzer [Online]. - Last Accessed: 9 2020. - https://bsa.isoftware.nl/.

Chen Y. S. and Chen Y. M. Combining incremental hidden Markov model and AdaBoost algorithm for anomaly intrusion detection [Conference] // ACMSIGKDD Workshop on CyberSecurity. - 2009.

Choi Young Han [et al.] Toward extracting malware features for classification using static and dynamic analysis [Conference] // Comput. Netw. Technol. (ICCNT). - 2012.

Christodorescu M. and Jha S. Static analysis of executables to detect malicious patterns [Conference] // Proceedings of the 12th Conference on USENIX Security Symposium, SSYM'03. - 2003.

Convolutional neural networks for visual recognition [Online]. - Last Accessed: 9 13, 2020. - http://cs231n.github.io/convolutional-networks/.

Corona I. Giacinto G., Roli F. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues [Journal] // Information Sciences. - 2013. - Vol. 239. - pp. 201-225.

Cortes C. and Vapnik V. Support-Vector Networks [Book Section] // Machine Learning. - 1995.

Cygwin [Online]. - Last Accessed: 9 2020. - https://www.cygwin.com/.

Damodaran Anusha [et al.] A comparison of static, dynamic, and hybrid analysis [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2017. - 1 : Vol. 13.

Davis J., Goadrich, M. The Relationship Between Precision-Recall and ROC Curves [Conference] // Proceedings of the 23rd International Conference on Machine Learning. - 2006.

Deshmukh Suchita, Di Troia Fabio and Stamp Mark Vigenère scores for malware detection [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2018. - 2 : Vol. 14.

Deshpande P. and Stamp M. Metamorphic detection using function call graph analysis [Journal]. - [s.l.] : MIS Review: An International Journal, 2015. - 1/2 : Vol. 21.

Deshpande S., Park Y. and Stamp M. Eigenvalue analysis for metamorphic detection [Journal] // Journal of Computer Virology and Hacking Techniques. - 2014. - 1 : Vol. 10. - pp. 53–65.

Do Chuong B and Batzoglou Serafim What is the expectation maximization algorithm? [Journal]. - [s.l.] : Nature Biotechnology, 2008. - 8 : Vol. 26.

Durbin R. [et al.] Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids [Journal]. - [s.l.] : Cambridge University Press, 1988.

Eddy S. R. Profile hidden Markov models [Journal]. - [s.l.] : Bioinformatics,, 1998. - 9 : Vol. 14. - pp. 755–763.

Eskandari Mojtaba, Khorshidpour Zeinab and Hashemi Sattar HDM-Analyser: A hybrid analysis approach based on data mining techniques for malware detection [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2013. - 10 : Vol. 9.

Fawcett T. An introduction to ROC analysis [Online]. - 2006. - Last Accessed: 9 2020. - https://people.inf.elte.hu/kiss/13dwhdm/roc.pdf.

Feng D. and Doolittle R. Progressive sequence alignment as a prerequisite to correct phylogenetic trees [Journal]. - [s.l.] : Journal of Molecular Evolution, 1987. - 4 : Vol. 25.

Ghahramani Z. An introduction to hidden Markov models and Bayesian networks [Book]. - River Edge, NJ, USA : World Scientific Publishing Co., Inc., 2001.

Google Google Scholar [Online]. - Last Accessed: 9 2020 - https://scholar.google.com.

Huang Wei-Chung, Di Troia Fabio and Stamp Mark Robust Hashing for Image-based Malware Classification [Conference] // International Workshop on Behavioral Analysis for System Security. - 2018.

IDA Pro [Online]. - Last Accessed: 9 2020. - https://www.hex-rays.com/products/ida/index.shtml.

Jidigam R.K., Austin T.H. and Stamp M. Singular value decomposition and metamorphic detection [Journal] // Journal of Computer Virology and Hacking Techniques. - 2015. - 4 : Vol. 11. - pp. 203-216.

Kaggle Microsoft Malware Classification Challenge [Online]. - 2015. - Last Accessed: 9 2020. - https://www.kaggle.com/c/malware-classification/data.

Krizhevsky A., Sutskever, I., Hinton, G. E. Imagenet classification with deep convolutional neural networks [Journal]. - [s.l.] : In Advances in Neural Information Processing Systems, 2012.

Liu L. Wang B., Yu B., et al. Automatic malware classification and new malware detection using machine learning [Journal]. - [s.l.] : Frontiers Inf Technol Electronic Eng, 2017. - Vol. 18.

Lu Y.B., Din S.C. and Zeng C.F. Using multi-feature and classifier ensembles to improve malware detection [Journal]. - 2010. - 2 : Vol. 32.

Malicia Malicia Project [Online]. - 2013. - Last Accessed: 9 2020. - https://www.malicia-project.com/dataset.html.

Monga V., Banerjee A. and Evans B. L. A clustering based approach to perceptual image hashing [Journal]. - [s.l.] : IEEE Transactions on Information Forensics and Security, 2006. - 1 : Vol. 1.

Nataraj L. [et al.] Malware Images: Visualization and Automatic Classification [Conference] // VizSec '11: Proceedings of the 8th International Symposium on Visualization for Cyber Security. - Pittsburgh, Pennsylvania, USA : ACM, 2011.

NGVSK Next Generation Virus Creation Kit [Online]. - 2001. - Last Accessed: 9 2020. - https://www.informit.com/articles/article.aspx?p=366890&seqNum=7.

Oliva A. and Torralba A. Modeling the shape of the scene: A holistic representation of the spatial envelope [Journal]. - [s.l.] : International Journal of Computer Vision, 2001. - 3 : Vol. 42.

Pai Swathi [et al.] Clustering for malware classification [Journal] // Journal of Computer Virology and Hacking Techniques. - 2017. - 2 : Vol. 13. - pp. 95-107.

Pan S. J. and Yang Q. A Survey on Transfer Learning [Journal]. - [s.l.] : IEEE Transactions on Knowledge and Data Engineering, 2010. - 10 : Vol. 22.

Pedregosa F. [et al.] Scikit-learn: Machine learning in Python [Journal]. - [s.l.] : Journal of Machine Learning Research, 2011. - Vol. 12. - pp. 2825–2830.

Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition [Journal] // Proceedings of the IEEE. - 1989. - 2 : Vol. 77. - pp. 257 - 286.

Raff E. and al. et An investigation of byte n-gram features for malware classification [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2016.

Raghavan Aditya, Di Troia Fabio and Stamp Mark Hidden Markov models with random restarts versus boosting for malware detection [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2019. - 2 : Vol. 15.

Rajeswaran Deebiga [et al.] Function Call Graphs Versus Machine Learning for Malware Detection [Journal]. - [s.l.] : Guide to Vulnerability Analysis for Computer Networks and Systems. Computer Communications and Networks. Springer, Cham, 2018.

Reddy D. K. S. and Pujari A. K. N-gram analysis for computer virus detection [Journal]. - [s.l.] : Journal in Computer Virology, 2006. - 3 : Vol. 2.

Shang S. [et al.] Detecting malware variants via function-call graph similarity [Conference] // In Proceedings of Malicious and Unwanted Software, MALWARE 2015. - 2015.

Shanmugam G., Low R. M. and Stamp M. Simple substitution distance and metamorphic detection [Journal]. - [s.l.] : J. Comput. Virol. Hacking Tech., 2013. - 3 : Vol. 9.

Singh Tanuvir [et al.] Support vector machines and malware detection [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2016. - 4 : Vol. 12.

Srinivasan S SSCT Score for Malware Detection, Master's report, Department of Computer Science, San Jose State University [Online]. - 2015. - Last Accessed: 9 2020 - http://scholarworks.sjsu.edu/etd_projects/444/.

Srivastava Abhinav, Lanzi Andrea and Giffin Jonathon System Call API Obfuscation [Conference] // RAID 2008: Recent Advances in Intrusion Detection. - 2008.

Stamp Mark A revealing introduction to hidden Markov models [Online]. - 10 2018. - Last Accessed: 9 2020. - http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf.

Stamp Mark Boost your knowledge of AdaBoost [Online]. - 2017. - Last Accessed: 9 2020. - https://www.cs.sjsu.edu/~stamp/RUA/ada.pdf.

Stamp Mark Information Security: Principles and Practice, 2nd edition. [Book]. - New Tork : Wiley, 2011.

Stamp Mark Introduction to Machine Learning with Applications in Information Security [Book]. - Boca Raton, FL, U.S.A. : Chapman and Hall/CRC , 2017.

Tensorflow For Poets [Online]. - Last Accessed: 9 13, 2020. - https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/.

Tensorflow Tensorflow for Poets [Online]. - Google Brain, 2017. - Last Accessed: 9 2020. - https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/.

Usha Narra [et al.] Clustering versus SVM for malware detection [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2016. - 4 : Vol. 12.

Vemparala Swapna [et al.] Malware Detection Using Dynamic Birthmarks [Conference] // Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics. - New Orleans, Louisiana, USA : ACM, 2016. - Vol. IWSPA '16.

Vemparala Swapna Malware detection using dynamic analysis [Online] // Master's thesis, San Jose State University. - 2015. - Last Accessed: 9 2020 - https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1403&context=etd_projects.

Venkatesan R. [et al.] Robust image hashing [Conference] // International Conference on Image Processing. - 2000. - Vol. 3.

VirusTotal VirusTotal Developer Hub [Online]. - Last Accessed: 9 2020. - https://developers.virustotal.com/.

Wong Wing and Stamp Mark Hunting for metamorphic engines [Journal]. - [s.l.] : Journal in Computer Virology, 2006. - Vol. 2.

Xi Dead Code Elimination through Dependent Types [Journal]. - [s.l.] : Gupta G. (eds) Practical Aspects of Declarative Languages. PADL 1999. Lecture Notes in Computer Science, 1999. - Vol. 1551.

Xu M. [et al.] A similarity metric method of obfuscated malware using function-call graph [Journal]. - [s.l.] : Journal of Computer Virology and Hacking Techniques, 2013. - 1 : Vol. 9.

Yajamanam Sravani [et al.] Deep Learning versus Gist Descriptors for Image-based Malware Classification [Conference] // 2nd International Workshop on FORmal methods for Security Engineering. - Funchal, Portugal : [s.n.], 2018.

Zhang B. [et al.] Malicious codes detection based on ensemble learning [Conference] // Autonomic and Trusted Computing, 4th International Conference. - 2007.

Appendix

The confirmation of personal contribution in the research work described by the author of the thesis has been furnished by Dr. Mark Stamp - mark.stamp@sjsu.edu - Department of Computer Science , San Jose State University , One Washington Square , San Jose, California , 95192, U.S.A.

Following, the list of publications per each Section. Copies of the papers have been removed from this thesis for copyright reasons

Section 5 - Preliminary experiments

- ***Section 5.1*** *- A comparison of static, dynamic, and hybrid analysis for malware detection. Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, Mark Stamp, J Comput Virol Hack Tech 13, 2017.*
- ***Section 5.2*** *- Malware Detection Using Dynamic Birthmarks. Vemparala Swapna, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., Stamp Mark, In Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, 2016.*

Section 6 - Zero-day experiments

- ***Section 6.1*** *- Clustering for malware classification. Pai Swati, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., and Stamp Mark, J Comput Virol Hack Tech 13, 2017.*
- ***Section 6.2*** *- Clustering versus SVM for malware detection. Narra Usha, Di Troia Fabio, Corrado Visaggio Aaron, Austin Thomas H., Stamp Mark, J Comput Virol Hack Tech 12, 2016.*

Section 7 - Effectiveness of Machine Learning algorithms

- ***Section 7.1*** *- Vigenère scores for malware detection. Deshmukh Suchita, Di Troia Fabio, Stamp Mark, J Comput Virol Hack Tech 14, 2018.*
- ***Section 7.2*** *- Deep Learning versus Gist Descriptors for Image-based Malware Classification. Yajamanam Sravani, Vikash Raja Samuel Selvin, Di Troia Fabio, Stamp Mark, In 2nd International Workshop on FORmal methods for Security Engineering, 2018.*

- ***Section 7.3*** - *Function Call Graphs versus Machine Learning for Malware Detection. Rajeswaran Deebiga, Di Troia Fabio, Stamp Mark, Guide to Vulnerability Analysis for Computer Networks and Systems, Computer Communications and Networks, Springer, 2018.*
- ***Section 7.4*** *- Robust Hashing for Image-Based Malware Classification. Huang Wei-Chung, Di Troia Fabio, Stamp Mark, International Workshop on Behavioral Analysis for System Security, 2018.*

Section 8 - Improved classification

- ***Section 8.1*** *- Hidden Markov models with random restarts versus boosting for malware detection. Raghavan Aditya, Di Troia Fabio, Stamp Mark, J Comput Virol Hack Tech 19, 2019*
- ***Section 8.2*** *- Support vector machines and malware detection. Singh Tanuvir, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, Mark Stamp, "Support vector machines and malware detection", J Comput Virol Hack Tech 12, 2016.*

Section 9 - *Generic Malware Models*

- ***Section 9.1*** *- On the Effectiveness of Generic Malware Models. Bagga Naman, Di Troia Fabio, Stamp Mark, International Workshop on Behavioral Analysis for System Security, 2018.*