Improving the Effectiveness and Efficiency of Dynamic Malware Analysis with Machine Learning

Sean Kilgallon
Institute for Financial Services Analytics
University of Delaware
skilgall@udel.edu

Leonardo De La Rosa
Institute for Financial Services Analytics
University of Delaware
dlarosa@udel.edu

John Cavazos
Computer and Information Sciences Department
University of Delaware
cavazos@udel.edu

***Abstract*—As the malware threat landscape is constantly evolving and over one million new malware strains are being generated every day [1], early automatic detection of threats constitutes a top priority of cybersecurity research, and amplifies the need for more advanced detection and classification methods that are effective and efficient. In this paper, we present the application of machine learning algorithms to predict the length of time malware should be executed in a sandbox to reveal its malicious intent. We also introduce a novel hybrid approach to malware classification based on static binary analysis and dynamic analysis of malware. Static analysis extracts information from a binary file without executing it, and dynamic analysis captures the behavior of malware in a sandbox environment. Our experimental results show that by turning the aforementioned problems into machine learning problems, it is possible to get an accuracy of up to 90% on the prediction of the malware analysis run time and up to 92% on the classification of malware families.**

I. Introduction

Malicious software is rapidly evolving and poses a major threat to networks and computer infrastructure. Disruptions, such as denial-of-service attacks and data breaches, are becoming more commonplace and difficult to anticipate. As cyber-attacks are growing in complexity and sophistication, the use of machine learning techniques is indispensable for firms to become more efficient in recognizing patterns that constitute a risk to their information. Nevertheless, current malware analysis systems tend to exhibit high detection rates for previously analyzed malware, for which signatures have been generated, but fail at detecting zero-day exploits for which malware is unavailable [2].

Furthermore, the time required for malicious files to take possession of an organization's data continues to shorten, leaving little margin to effectively respond to a cyber-attack. According to a study published by Barkly [3], an endpoint security company, five ransomware samples (Chimera, Petya, TeslaCrypt 4.0, CTB-Locker, TeslaCrypt 3.0) need less than a minute to execute the encryption process on a user's computer. As a result, with malware being able to execute and do its damage within a matter of seconds, detection of malicious files in a short amount of time corresponds to a critical area in cybersecurity research.

Dynamic malware analysis has emerged as the state-of-the-art detection approach that compensates for the shortcomings of signature-based techniques [4]. In this type of analysis, a malicious file is executed in an isolated environment or sandbox, and its behavior is monitored and reported for further examination. While it is better at identifying malware than traditional signature-based techniques, dynamic analysis still suffers from limitations due to the ability of certain malware to detect the presence of the monitored environment, and stall the execution of any malicious activity [5]. It also takes longer to perform dynamic analysis than signature-based techniques or static analysis. This barrier becomes even more dramatic, as malware authors continue to add logic to detect whether their malware is being executed in virtualized environments, hindering the ability of analysis systems to deem malware as a threat.

Another technique for analyzing malware corresponds to static analysis. Static analysis is a method of reverse engineering that is done by extracting a malware's code without executing the actual program. In contrast to dynamic analysis which can take minutes to hours, static analysis can typically be performed in seconds. Automated tools like portable reverse engineering frameworks can help extract static features from a malware binary in a very accurate and effective manner, providing valuable insights about the intent of the suspicious file. Nevertheless, because of obfuscation techniques, static analysis alone is not enough to detect or classify malicious code [6]. Thus, effective and efficient dynamic analysis of malware is required. However, we use Radare2 (r2) [7], an open-source reverse engineering compiler, to extract static features to supplement our dynamic analysis.

In this paper, we examine the application of machine learning techniques to predict the time required for a file to reveal its malicious intent using dynamic analysis in an open-source sandbox environment called Cuckoo [8]. Cuckoo is a leading dynamic analysis tool used by experts in cybersecurity and therefore was chosen as the platform for this experiment. Any sandbox environment would produce similar results.

We develop a model that can help predict the time required to execute malware in Cuckoo to uncover its malicious intent, based on static information extracted from the suspicious files themselves. We perform dynamic analysis using Cuckoo to determine the number of seconds required for a file to reveal its malicious behavior. The number of seconds to run a malware corresponds to the target labels we want to learn using a machine learning classifier. The classifier is trained on this information and a set of static features extracted from our malware dataset to generate a malware run time prediction model. We explore a large set of machine learning algorithms

978-1-5090-6055-9/17/$31.00 ©2017 IEEE

to evaluate which algorithm gives the best accuracy.

Furthermore, we extract features from dynamic analysis performed in Cuckoo, and combine these features with static analysis features from r2 to form a hybrid feature vector that is used to create a malware family classification model. Our results show that our models are able predict the malware analysis run time and classify malware families with an accuracy of up to 90% and 92%, respectively.

II. Literature Review

The use of dynamic and static analysis for malware detection has been widely studied by researchers in the field of cybersecurity. Schultz [9] applies machine learning in conjunction with static analysis features to classify malware, with additional assistance of byte-sequence n-grams, portable executables (PE) and string features. Broad static analysis of the files on an Android device and Support Vector Machine (SVM), it is possible to detect malware with fairly high accuracy [10]. In the same way, the work of Schmidt et al. [11] [12] demonstrates the high detection rates that can be achieved by incorporating the function calls extracted from static analysis with machine learning techniques to classify malware. On the other hand, static analysis can be deceived by a binary obfuscation scheme and that static analysis alone is not enough to accurately describe a file [6].

Dynamic analysis is used to determine the purpose and functionality of a given malware and is a necessary step in developing detection models. One paper discusses its platform for dynamically analyzing malware [13]. Dynamic analysis features such as system call sequences are modeled for the purpose of malware classification uses Windows native system calls and Windows API functions that the program invokes [14]. Neural networks using dynamic analysis of malware, specifically system call sequences, have been shown to improve upon traditional machine learning methods such as hidden Markov models and SVM [14]. Furthermore, research has been conducted in comparing different machine learning models in conjunction with dynamic analysis with J48 (Decision Trees) being the best classifier [15].

On the other hand, early malware detection corresponds to a crucial aspect within cybersecurity research and various authors have worked on techniques that attempt to minimize the time required to identify a file as malicious. Bayer et al. [16] attempt to reduce the malware analysis run time for dynamic analysis, by concentrating on unique polymorphic files, avoiding the execution of samples that exhibit a similar structure. Comparably, Anderson et al. [17] study the malware classification problem and propose a framework that includes both static and dynamic features to better classify malware that intend to conceal its behavior.

Machine learning techniques have proven to be indispensible tools in the field of cybersecurity. As documented in the research conducted by Kruczkowski and Szynkiewicz [18], as well as Firdausi et al. [15], classifiers such as K-Nearest Neighbors (KNN), Naive Bayes, Decision Tree and SVM can effectively expose the malicious intent of heterogeneous malware datasets, by uncovering the underlying relationships of the features extracted from dynamic and static analysis. Furthermore, Rieck et al. proposes a framework that performs an incremental approach for dynamic analysis that executes and groups thousands of malicious files that display similar behavioral profiles into specific classes then assigns unseen malware with similar patterns to these clusters which reduces the overall malware analysis time [19].

In addition to static and dynamic analysis, there are a few documented cases of research that combines these features into a hybrid approach. One paper uses opcode sequences gathered using static analysis and a large number of dynamic features to train several different machine learning models to classify unseen malware with high accuracy [20]. Another hybrid approach uses function length frequency and string information for static features and API calls for their dynamic features [21]. A more recent paper created a plotform to detect Android malware by characterizing applications with static and dynamic analysis and shows that the integrated (hybrid) model demonstrates significantly more robustness [22]. Another paper states that combining the results of static and dynamic increases the reliability of the inferred class labels [23].

Based on the previous summary, it is evident that there is a deep relationship between dynamic malware analysis and machine learning techniques, as classifiers can aid in reducing the time of malware analysis of an unknown file for which signatures have yet to be recorded. More importantly, by taking advantage of both static and dynamic analysis information, it is possible to construct a model that extracts the underlying structure of the data of a malicious file, and helps researchers make predictions from these results.

III. Methodology

A. Malware Run Time Prediction Model

The basic idea of the first part of this research is to predict malware analysis times, that is, to determine the amount of time required to detect the malicious intent of a file for which no previous signatures exist. For this, we intend to use machine learning to learn the time required for our malware analysis system to identify a sample as malicious, based on the static information that has been previously extracted from the file. To accomplish this goal, we use r2 to extract source code of each malware that is considered for analysis, and generate static features, which are then gathered to form our dataset of static information.

In addition, we define three specific labels: 20 seconds, 60 seconds and 300 seconds, which indicate the time that we will run each sample of our training set for, and that constitute the target classes of our prediction model. As a result, each malware of our training set is executed for each of the previous times and a JSON report is generated by the end of each run, which contains a score indicating the threat-level of the file.

This score ranges from 1 to 10 and based on the reports produced after multiple runs, it is possible to determine that Cuckoo confidently classifies a file as malicious if the final

score is equal or greater than 5.0. Quantifying a malwares maliciousness is a difficult task and is generated by Cuckoo after each analysis is completed. Cuckoo takes an inventory of all the signatures the file exhibits. The signatures generated by Cuckoo are behaviors that could be deemed malicious such as installing itself for autorun during Windows startup or querying for the computer name information. These signatures are given a severity score and the scores are combined into a single 1 to 10 maliciousness score. The threshold of 5.0 will be used to determine if we can reveal the maliciousness of a file for each of our times: if a file shows a score equal or greater than 5.0 when it is run for 20 seconds, it is assigned a label of "20s". Should the malware fail to achieve this score after a 20 second run, it will be resubmitted for analysis for the second predefined time (60 seconds) and its score will be examined at the end of the analysis to establish whether Cuckoo successfully deems the sample as malicious. Finally, if the score fails to get to a minimum of 5.0, the file will be reprocessed for a total of 300 seconds, after which a decision will be made whether to categorize the file as malicious if it achieved a mark of 5.0, or evasive/anti-sandbox in case it does not satisfy this condition. On the other hand, the malware that does not meet our threshold will be further considered for analysis with a different detection system. A summary of this procedure can be seen in Table 1, where the score that determines the label that is assigned to each malware is highlighted in green.

TABLE I
ASSIGNMENT OF LABELS FOR MALWARE DATASET

Malware	Scores for			Label
	20s	60s	300s	
2d1a2e11a90...25dcd362277652	**7.6**	9.4	9.6	20s
0d8e20309f0...802eba6f8b27532	1.0	**6.0**	7.0	60s
2bae0de6cda...18d7f8c5536f142	2.0	2.4	**5.4**	300s

After each malware is labeled as "20s", "60s" or "300s", its corresponding static analysis information is appended, to obtain the input that will be fed to the model that will make the predictions for the malware analysis time. Nevertheless, it is mandatory that our static features undergo a transformation that makes them more suitable for the classifiers. As stated in the SKLEARN Documentation for Data Preprocessing [24], machine learning estimators might yield poor results if the feature vectors do not resemble standard normally distributed data. As a result, the static information is normalized and combined with the label information, yielding the final dataset that will be given to the algorithms as indicated in Table 2.

TABLE II
DATASET SAMPLE FOR MACHINE LEARNING CLASSIFIER

Malware	Static Features				Target Label
	bytes. histogram.0	bytes. histogram.1	⋯	bytes. histogram.255	
1	0.212	0.280	…	0.017	20s
⋮	⋮	⋮	⋱	⋮	⋮
N	0.145	0.306	…	0.096	60s

The complete malware run time prediction model can be seen below in Figure 1.

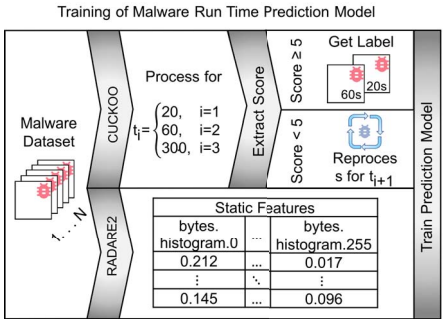

Fig. 1. . This figure describes the process of training the malware run time prediction model. We run dynamic analysis in three phases starting with 20s and continuing for 60s and 300s that do not meet the minimum score. This provides us with a run time score which becomes the label of our training data extracted from static features.

B. Malware Family Classification Model

This section will describe a model trained to predict the malware family of an unknown malware. The malware corpus is collected from the worlds largest malware repository obtained from a company called Reversing Labs, Inc. The ground truth of our model and malware family labels used to assess the accuracy of this model is supplied by ReversingLabs as well.

Static analysis is performed by extracting information from a binary file without executing it. This is different from dynamic analysis in which we execute the malware in a sandbox environment such as Cuckoo and capture its behavior. Our static features which are derived from static analysis are contextual byte features, portable executable (PE) import features, a 2-dimensional histogram of string features, and PE metadata features [25]. Contextual byte features take the form of a byte-entropy histogram, which models the files distribution of bytes. From the PE header, we extract the PE Import and Metadata tables in which we hash each into a size 256 vector. Also, we extract all the strings in the file to be hashed into a vector as well. These features are combined to form a 1024 length vector, which will provide us with static information for that sample. Saxe and Berlin [25] show that this set of static features can be used to build highly accurate machine learning models.

We perform dynamic analysis by observing the behavior of the malware while it is running on a host system. In our dynamic analysis, we run the malware in a Cuckoo sandbox and extract its behaviors in JSON reports. We then convert these reports to dynamic features. The dynamic features that are used to build our machine learning models correspond to API calls from all malware seen in the dataset which are generated from dynamic analysis. In addition, each of the malware's dynamic features is comprised of an API call histogram. The static and dynamic features that are being generated are depicted in Figure 2.

After generating the static and dynamic features, we combine them into a hybrid set of features and use them to train our models. The training of the malware family classification model is shown in Figure 3.

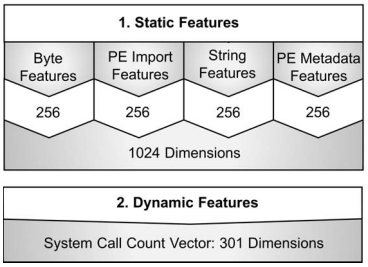

Fig. 2. This figure shows the breakdown of static and dynamic features extracted from the analysis stage. The four static features are hashed into fixed dimension vectors and combined. The dynamic features are created from Windows API sytem calls.

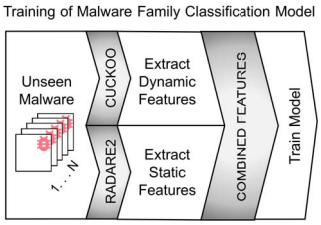

Fig. 3. This figure describes the training of the malware family classification model where static and dynamic features extracted from the analyses stages are combined to train the model. The breakdown of features can be shown in Fig. 2.

C. Deployment of Prediction Models

We perform static and dynamic analysis on our malware test dataset to extract static and dynamic features that are used to examine the accuracies of our prediction models. A vector of static features acts as the input for our malware run time prediction model to estimate the corresponding labels of our test samples. In addition, a hybrid feature vector is generated based on static and dynamic features of our malware, and used to test the accuracy of our malware family classification model. A representation of the workflow used to deploy and test both models is presented in Figure 4.

The ability to reduce the dynamic analysis run time bottleneck provides the ability to classify, and therefore detect, malware in a fast and accurate manner. In a cybersecurity environment that is increasingly reliant on the speed at which we can detect and classify malware, this approach seems to have merit.

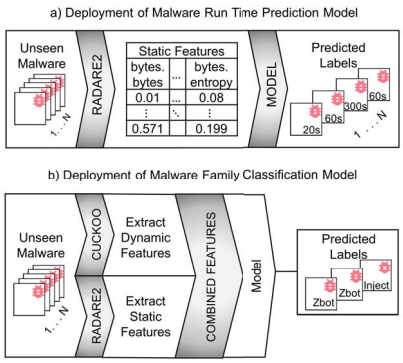

Fig. 4. The top model in the figure above describes the malware run time prediction model which is trained on static features. The figure below is the malware family classification model, which is trained on both dynamic and static features combined into a hybrid feature set. Both models use analyses from the same malware corpus.

D. Machine Learning Models

We use SKLEARN [26], an open source Python Library that implements a wide range of machine learning algorithms for data analysis. Five particular classifiers are selected for our experiments: Decision Tree, Random Forest, Support Vector Machine, Neural Network, and K-nearest Neighbors. Finally, we use 10-fold cross validation to perform the training and testing of our algorithms.

IV. Experiments

In this section, we present and discuss the major results from running and analyzing a dataset of 3320 malware samples using Cuckoo, and the additional implementation of the machine learning classifiers to predict the malware analysis run time, as well as predicting what family a malware belongs to. All of the models discussed in this section are trained using 10-fold cross validation and a separate testing set to gauge their effectiveness. First, we discuss the output of the analysis performed by Cuckoo. As indicated in Table 3, Cuckoo successfully deems a significant number of our files as malicious in only 20 seconds. Particularly, a total of 1766 samples are regarded as malware in this short amount of time, which is indicative of the capabilities of our system to execute a file and detect that it is malware in a rapid manner.

TABLE III
Distribution of Predicted Labels for Malware Execution Time

Predicted Label	Total	Percentage
20 seconds	1766	53.19%
60 seconds	958	28.86%
300 seconds	596	17.95%

Additionally, a notable number of files (958) are also labeled as sixty seconds which, combined with the total tally of files that Cuckoo detects as malicious in 20 seconds (1766), gives us a remarkable indicator that our system does not require a long time to expose malware, which is a tremendous advancement in the field of malware detection, especially when we consider the narrow margin of time that security analysts have when confronted with malware such as ransomware, which is able to execute and encrypt a user's data in a very short amount of time.

On the other hand, a total of five classifiers are used to make the prediction regarding the time required to unveil the maliciousness of the files in our dataset. The target precision and recall values are 99.5

As seen in Table 4, the Random Forest classifier yields the highest accuracy (90.66%), followed by Decision Tree (87.65%), Neural Network (85.24%), K-Nearest Neighbors (83.43%) and Support Vector Machine (82.83%). These results are clearly indicative of the ability of our model to estimate the time that malware should be executed in a sandbox in order to reveal itself. Additionally, we confirm that we can indeed

TABLE IV
RESULTS FOR MALWARE RUN TIME PREDICTION MODEL

Classifier	Accuracy	Precision	Recall
Random Forest	90.66%	90.02%	88.02%
Decision Tree	87.65%	86.28%	85.35%
Neural Network	85.24%	82.30%	83.62%
K-Nearest Neighbors	83.43%	83.11%	79.04%
Support Vector Machine	82.83%	81.03%	79.71%

use features extracted from the suspicious samples (more specifically, static features) to estimate the malware analysis run time, based on the predefined values of 20 seconds, 60 seconds and 300 seconds. Finally these two results provide enough evidence to conclude that it is possible to run a file for a short amount of time and still detect that it is malware.

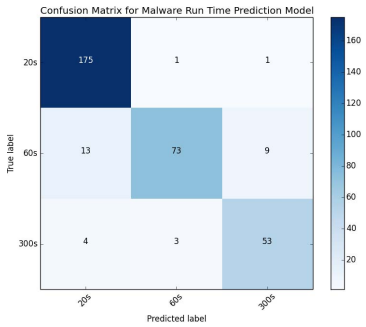

Fig. 5. This figure shows the Confusion Matrix for the Malware Run Time Classification Model Our model exhibits high accuracy regarding the classification of malware as needing 20 seconds or 300 seconds to reveal itself.

In addition, we analyze the time that is saved by using our malware run time prediction model compared to other approaches used to perform dynamic analysis. For the purpose of this study, we use the test dataset of 332 samples for which our model predicted the malware analysis time. Additionally, we define the following three methods to perform dynamic analysis on the test dataset.

1) Using maximum amount of time: The entire malware test dataset is analyzed for the predefined time of 300 seconds.

2) Using prediction model: Our prediction model is used to determine the times that each sample of the test dataset should be executed for in our analysis system. If the threshold of 5.0 is not achieved after running the sample according to the predicted labels, the malware is resubmitted for analysis for the next predefined time until we obtain the desired score.

3) Ignoring prediction model: The test dataset is first submitted for analysis for 20 seconds. The samples that do not yield a score of 5.0 are resubmitted for 60 seconds. Should the files fail to generate a 5.0 mark, they are reprocessed for a total of 300 seconds.

As indicated in Table 5, our malware run time prediction model requires the least amount of time, compared to the other two approaches that dynamically analyze our test dataset. Specifically, our prediction model drastically reduces the average execution time that is required to uncover the malicious intent of our test dataset (by 95.72%), when compared to a method where all the samples are submitted for a total of 300 seconds or 5 minutes, which corresponds to the approach that numerous analysis systems use to examine malware behavior [27].

TABLE V
TOTAL TIME REQUIRED BY THREE DIFFERENT APPROACHES FOR MALWARE DETECTION

Method	Total Time	% Decrease
Using Maximum Time	166.67 hours	-
Using Malware Run Time Prediction Model	7.12 hours	95.72%
Disregarding Prediction Model	8.14 hours	95.11%

Finally we discuss the malware family classification model. To build this model, we use the same malware samples as is used to train the malware run time prediction model. We extract the dynamic and static features from the malware and then train on the features discussed in the methodology section of this paper.

Each model is constructed using 10-fold cross validation, and each model's performance can be seen in Table 6.

TABLE VI
RESULTS FOR MALWARE FAMILY CLASSIFICATION MODEL

Classifier	Accuracy	Precision	Recall
Neural Network	92.18%	89.19%	86.87%
Random Forest	91.70%	89.60%	84.12%
Support Vector Machines	91.58%	88.21%	85.49%
Decision Tree	89.31%	82.53%	81.66%
K-Nearest Neighbors	88.78%	81.67%	78.84%

The Neural Network model outperforms the other models in terms of accuracy achieving over 92%. The precision and recall of this model is very high at 89% and 87%, respectively. This leads to the number of false positives and negatives being very low, with the false negative rate being slightly higher. Not far behind are Random Forest and SVM classifiers which achieve over 91% accuracy. Not far behind are Decision Trees and K-Nearest at 89% accuracy. While these models have lower accuracies than neural networks and SVMs, they have the potential to be human interpretable, which may make these models more advantageous to security analysts trying to understand what makes an executable malicious.

The classifier with the highest precision and recall values were chosen to be the best classifier. However, this dataset is a subsample of a huge corpus of malware with many more malware families than what we have in our experiment. SVM and kNN rely on pairwise similarities over the dataset and training time will increase much more with increases in dataset size than the other classifiers we have tested. Random Forest classifiers have a complexity that depends on the dataset size and could lead to overfitting with an increase in the size of the dataset. For Neural Networks, the duration of each epoch is linear with the dataset size and tends to converge faster with less epochs as the dataset grows. Thus, we choose Neural Networks as being the best classifier for this problem.

ROC curves for each model can be seen in Figure 5. Two models, SVMs and the Neural Network, perform with an Area Under the Curve (AUC) of 0.99 and lie in the far upper left of the figure. They present both high true positive rate and

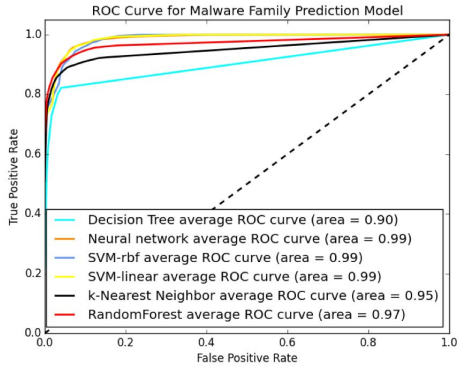

Fig. 6. This figure presents the ROC Curves for the Malware Family Prediction Model. The closer the AUC for a model comes to 1, the better it is. Our results clearly indicate that Neural Network and SVMs yield the best predictive power for the malware family classification problem.

low false positive rate. The Decision Tree model is the worst performing model within the ROC curve with an AUC of 0.90.

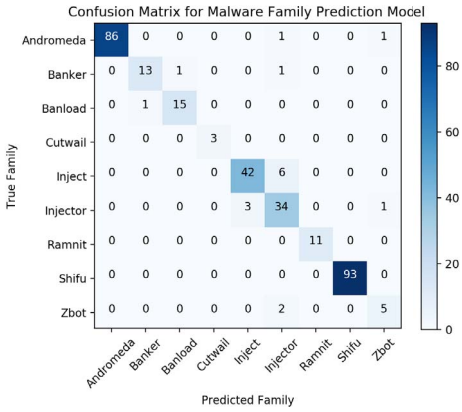

Fig. 7. This figure shows the Confusion Matrix for the Malware Family Classification Model. The high values on the diagonal of the matrix are indicative of the high predictive power of our model.

V. Limitations

For the malware dataset that is analyzed in Cuckoo to generate a training and testing dataset, there is a significant portion of files that do not display any malicious activity, even after 300 seconds. We provide a list of explanations for this kind of behavior based on the observed results.

A. Operating System Constraints

The analyzed malware might be designed to target a specific operating system such as Windows XP. However, the current setup for malware analysis only includes machines which are running Windows 7 Professional. Therefore, a new architecture that incorporates other operating systems such as Windows XP is required to avoid this limitation.

B. Malware can detect monitored environment

The presence of the monitored environment might be detected, prompting the malware to refuse to perform any sort of malicious activity. Additional user interactions with the machines might be required in order to deceive the malware and prompt the execution of its code.

C. Executables in our dataset might require a static import or a DLL not present in our machines

As a result, the behavioral module from Cuckoo is unable to capture any activity when the file is executed and the reports lack of any useful information for our analysis.

VI. Future Work

We plan to explore dimensionality reduction in the future iteration of the models. Currently, there is a large number of features compared to the number of class values. Using Principle Component Analysis (PCA), we can reduce the dimensionality of the data. Hybrid features are a combination of static and dynamic features and therefore dimensionality reduction is even more important our hybrid model's case.

The dataset only contains executable files, but needs to be expanded to handle any kind of malware. As long as the dynamic analysis generates a report for a malware, that instance can be included in our model. Moreover, we aim to increase the current dataset size to the millions of malware.

As discussed in the limitations section, we are only using Windows 7 Professional as it is compatible with bare-metal and cloud-computing environments. Using different operating systems (OS) for the sandbox environment would give us the ability to detect maliciousness in malware that are built for specific OS architectures. Also, some malware has the ability to detect the presence of a monitored environment. We plan to test the current anti-anti-sandboxing environments.

The results of our models depend on reliable output from a dynamic analysis sandbox environment such as Cuckoo. However, the results of our model are not dependent on using Cuckoo as the dynamic analysis tool. The data used for prediction is simple data available in most dynamic analysis sandbox environments. Also, we hope to include more dynamic analysis information in training our malware family classification model. Currently, we are using Windows API call histograms as our dynamic feature set, but there are many other features we can extract from the dynamic analysis reports.

We present two models in this paper: 1) a model that predicts the minimum run time necessary for a suspicious to exhibit its malicious behavior., which we term the "Malware Run Time Prediction Model" and 2) a model that predicts the family a particular piece of malware belongs to, which we call the "Malware Family Prediction Model." These models can be naturally connected by using our first model to predict the time to run a file. The file can then be run for the predicted amount of time and the dynamic analysis results will be used as input for the second model. This natural connection will provide us with an effective family classification of malware, but with the added benefit of being time efficient.

Finally, we consider the application of a time series analysis where we capture the score that Cuckoo generates for a

malicious file over the course of a predefined time interval. This will require the alteration of Cuckoo's main code in order to force it to estimate a score at each time of the selected interval (e.g. 1 second, 2 seconds, 3 seconds) until it observes our threshold and finalizes the analysis of the file in question.

VII. Conclusions

In this paper, we study the application of machine learning techniques to predict the malware analysis run time and classify the malware families of a set of malicious samples. Our novel approach shows that it is possible to determine the the time that malware should be executed for in a sandbox in order to reveal itself and that this execution does not require a significant amount of time, as seen from the fraction of our dataset that is deemed as malware in just 20 seconds. Additionally, we demonstrate that it is possible to use static features extracted from the suspicious samples to estimate the malware analysis run time, based on the predefined values of 20 seconds, 60 seconds and 300 seconds.

Furthermore, we show the capabilities of using a model that incorporates machine learning algorithms and leverages static and dynamic analysis information extracted from the malware to make predictions on the required malware analysis run time and malware family of a dataset of unseen malware.

Being able to classify malware accurately is one of the most important problems in a malware analysis platform. We show that with almost 92% accuracy we can correctly classify a malware family by using both static and dynamic features of the malware.

By using machine learning, it is possible to uncover the underlying relationships of the features extracted from dynamic and static analysis to help researchers create adaptive malware detection systems that better defend and protect the network and information of users and companies.

Acknowledgment

This research was made possible through the generous donation of cloud credits by the AWS Cloud Credits for Research program.

References

[1] "Nearly 1 million new malware threats released every day - apr. 14, 2015," http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/, (Accessed on 04/09/2017).

[2] J. J. Blount, D. R. Tauritz, and S. A. Mulder, "Adaptive rule-based malware detection employing learning classifier systems: a proof of concept," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*. IEEE, 2011, pp. 110–115.

[3] "Startup barkly touts light, fast endpoint protection — network world," http://www.networkworld.com/article/2940139/security0/startup-barkly-touts-light-fast-endpoint-protection.html, (Accessed on 04/09/2017).

[4] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[5] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection." in *USENIX Security*, vol. 2014, 2014, pp. 287–301.

[6] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007, pp. 421–430.

[7] Radare2. [Online]. Available: https://www.radare.org/r/

[8] Cuckoo. [Online]. Available: https://cuckoosandbox.org

[9] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 38–49.

[10] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[11] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," in *Communications, 2009. ICC'09. IEEE International Conference on*. IEEE, 2009, pp. 1–5.

[12] A.-D. Schmidt, J. H. Clausen, A. Camtepe, and S. Albayrak, "Detecting symbian os malware through static function call analysis," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 2009, pp. 15–22.

[13] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006.

[14] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[15] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*. IEEE, 2010, pp. 201–203.

[16] U. Bayer, E. Kirda, and C. Kruegel, "Improving the efficiency of dynamic malware analysis," in *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010, pp. 1871–1878.

[17] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012, pp. 3–14.

[18] M. Kruczkowski and E. N. Szynkiewicz, "Support vector machine for malware analysis and classification," in *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02*. IEEE Computer Society, 2014, pp. 415–420.

[19] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[20] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*. Springer, 2013, pp. 271–280.

[21] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[22] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[23] B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert, "Adaptive semantics-aware malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 419–439.

[24] "4.3. preprocessing data scikit-learn 0.18.1 documentation," http://scikit-learn.org/stable/modules/preprocessing.html, (Accessed on 04/11/2017).

[25] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 2015, pp. 11–20.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[27] P. Vadrevu and R. Perdisci, "Maxs: Scaling malware execution with sequential multi-hypothesis testing," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 771–782.