Towards an Automated Pipeline for Detecting and Classifying Malware through Machine Learning

Nicola Loi\textsuperscript{1}, Claudio Borile\textsuperscript{2}, and Daniele Ucci\textsuperscript{2}

\textsuperscript{1} Università degli Studi di Torino, via Pietro Giuria 1, 10125 Turin, Italy
\texttt{[name].[surname]@edu.unito.it} \\
\textsuperscript{2} aizoOn Technology Consulting, Strada del Lionetto 6, 10146 Turin, Italy
\texttt{[name].[surname]@aizoongroup.com}

Abstract

The constant growth in the number of malware - software or code fragment potentially harmful for computers and information networks - and the use of sophisticated evasion and obfuscation techniques have seriously hindered classic signature-based approaches. On the other hand, malware detection systems based on machine learning techniques started offering a promising alternative to standard approaches, drastically reducing analysis time and turning out to be more robust against evasion and obfuscation techniques. In this paper, we propose a malware taxonomic classification pipeline able to classify Windows Portable Executable files (PEs). Given an input PE sample, it is first classified as either malicious or benign. If malicious, the pipeline further analyzes it in order to establish its threat type, family, and behavior(s). We tested the proposed pipeline on the open source dataset EMBER, containing approximately 1 million PE samples, analyzed through static analysis. Obtained malware detection results are comparable to other academic works in the current state of art and, in addition, we provide an in-depth classification of malicious samples. Models used in the pipeline provides interpretable results which can help security analysts in better understanding decisions taken by the automated pipeline.

Keywords: automated security analysis, malware pipeline, malware classification, malware detection, static analysis.

1 Introduction

Malware (short for malicious software) is the generic term used to refer to unwanted software developed to infect and interfere with the operations of a single machine or networks of computers \cite{1, 2}. Since the first documented virus appeared in the 1970s, the evolution of computer science has always been accompanied closely by the creation of new, better and more harmful malicious software, in a constant fight between malware developers and security analysts. In recent years, though, the refinement and emergence of new software technologies have allowed an exponential growth in the number of malware in circulation, not only vertically (volume) but also horizontally (types and functionality) \cite{3}. Together with the ever more sophisticated evasion techniques being developed by attackers, security experts and anti-malware vendors struggle to keep up the race by means of “standard” methods, i.e. signature-based and heuristics \cite{4, 5, 6}. In this context, machine learning (ML) seems to be the most promising tool for an automated analysis and prevention of this kind of threats \cite{7}. The strength of ML is its ability to automatically identify hidden patterns and correlations in large volumes of raw data, and exploit these statistical features to, in the case of malware analysis, recognise previously unseen attacks. Generally speaking, classic ML approaches for cyber security purposes focus on a first phase of features extraction through static, dynamic or hybrid analysis. These features are then used to train models that allow to classify malicious and benign files. More recently, the advancements in deep learning methods has inspired a series of studies exploiting other input
formats, like raw binaries [8, 9] and image representations [10, 11, 12] among others. Historically, researchers and security vendors have mostly been more focused on creating models for the detection of malicious and benign files rather than exploring the possibility of using ML for an in-depth analysis of single malware samples. A reason for this might be the difficulty in collecting large and well-annotated datasets, a complex and expensive task, together with the intrinsic difficulty in classifying the characteristics of malwares due to the presence of many variants and the lack of a standard nomenclature [13, 14, 15].

Endgame — now Elastic — released in 2017 the first version of an open-source dataset called EMBER (Elastic Malware Benchmark for Empowering Researchers) containing semi-raw static features from more than 1 million Portable Executable (PE). In 2018, they released a second version of the dataset, using more recent malware, correcting issues in the data collection, and providing also labels for malware classes, using the open source tool AVClass [16]. This tool allows to parse and organise, in a definite taxonomy, the different nomenclatures returned by multiple anti-virus vendors. Exploiting and integrating the last version of the EMBER dataset as described in detail below, we explore the possibility of training a machine learning pipeline for a full, automated analysis of PEs using static features, from malware detection to classification by threat-type, family, and behaviour. At the time of writing, there is no academic paper that has explored the possibility of leveraging the EMBER dataset for a taxonomic malware classification. The paper is organized as follows: Section 2 presents related works, while Section 3 details the EMBER dataset. The proposed classification pipeline is described in Section 4 and experimental evaluation results are reported in 5.

2 Related Work

In the last decade, the number of studies on machine learning detection techniques is constantly increased thanks to: i) the recent growth of new and powerful algorithms and data wrangling methods, ii) the increase in computational capabilities, and iii) the availability of public, annotated malware datasets [17, 18, 19, 20, 21]. However, most of these works only consider the problem of distinguishing malicious software from benign. Furthermore, many papers rely on small or outdated datasets that are unlikely to provide a statistically-significant representation of malware population and labelling procedures create a bias towards easier datasets [22, 23]. The above considerations resulted in a general difficulty in assessing the real performance of these methods “into the wild” and a general incapacity to deliver models that can be effectively deployed, despite notable results summarized in [22]. Nevertheless, many recent works have been very effective in detecting malware, reaching almost perfect performances in accuracy.

In this scenario, the authors of the EMBER dataset provided a benchmark model in 2018, trained on their latest release of the model, obtaining a 86.8% detection rate at 0.1% False Positive Rate (FPR). Results obtained in the previous model release reported a 93% detection rate [24], meaning that the EMBER dataset developers have successfully hardened the process of correctly classifying malicious samples. Despite more recent works on the same dataset provide small improvements in the detection rate [25, 26], they usually rely on deep learning frameworks that make more difficult to interpret model outputs.

To our knowledge, there are no published works that used the 2018 EMBER dataset for a multi-class classification in malware families. Among the studies that tackle the problem of classifying malware families [27, 28, 29, 30], Almadi et al. [10] considered the Microsoft Malware Classification Challenge data, a labeled dataset of about 20,000 malware samples.
representing a mix of 9 different families\(^1\), to build a model for malware families classification. While the dataset is much smaller than EMBER and thus the performances are difficult to compare, they obtained a notable 0.997 accuracy over a 5-fold cross validation, considering a set of features similar to those used in the EMBER model, suggesting their robustness for both malware detection and family classification.

Malware behaviour classification is usually based on dynamic or hybrid analysis \([31, 32, 33]\). This is not surprising, since malware behaviour is expected to manifest itself only upon execution \([22]\). Nevertheless, we believe that it is still interesting to assess behavioral classification on purely static analyses contained in EMBER. We have not find any previous work considering the problem of an integrated, hierarchical malware classification into threat-type, family, and behaviour using features from static analysis.

3 Data description

3.1 The EMBER dataset

In this work we consider the 2018 release of the EMBER (Elastic Malware Benchmark for Empowering Researchers) dataset, an open source benchmark collection of 1 million PEs scanned in or before 2018 \([34]\). The data comes split in two separate sets containing the training and test data. The training set consists of 600,000 labeled samples (benign or malware) and 200,000 unlabeled samples the we did not consider in this study. The test set contains 200,000 labeled samples. The authors claim that the particular splitting between train and test is specifically engineered for having a “harder” dataset with respect to the first release \([24]\). Each sample comes as a JSON object and is uniquely identified via its sha256 and md5 hashes, and provides semi-raw information for static malware analysis parsed using the LIEF open source package \([35]\) and divided in nine major groups: General, header, and section information, imported and exported functions, strings information, raw-byte histogram, and byte-entropy histogram. Finally, additional information is given on the label (0 for benign files, 1 for malicious files, and -1 for unlabeled files), the coarse time stamp of estimate first detection of the malware, the malware class extracted from the VirusTotal \([36]\) report using the open source tool avclass \([16]\). A full description of the dataset can be found in \([34, 24]\). As explained in detail in section 4, we consider all the 600,000 labeled training data for training and validating each stage of the classification pipeline, while the test set will be used only to assess the global performance of the pipeline, mimicking its usage in a real-world scenario.

3.2 Threat-type and behaviour labels collection

The EMBER dataset only provides the malware/benign and a generic “avclass” labels (the output of the AVClass tool \([16]\) for family tagging). In order to classify a specific malware sample by threat-type, family, and behavior, we used the open-source tool AVClass2 \([37]\), not yet available when the EMBER dataset came out, to systematically extract information on the malware threat-type and behavior for each labeled sample in the dataset. Our final dataset consists of the data from the EMBER original dataset plus four labels: malware/not-malware, and if malware its threat-type, family and behavior classes where available. It is important to note that AVClass2 parses the malware label returned from each of the AV vendors and returns a ranking of the returned labels, not necessarily a single result per class. As a consequence, while a powerful tool tailored for each AV vendor and including a complete malware taxonomy, still

\(^1\)https://www.kaggle.com/c/malware-classification
suffers from the lack of a shared nomenclature across different security vendors. Furthermore, the estimated accuracy of the output labels is reported to be around 90% for families (threat-type and behavior not reported) [37], so that we must take into consideration the inherent imperfect labelling of the data.

### 3.3 Classification target

Looking at figure 1 we can see that for each of the classification stages of the pipeline the classes are heavily unbalanced (note the logarithmic scale). In particular, the “Family” group has a very large number of poorly populated classes in the train set. We thus kept the first 20 families and grouped all the remaining in a single class named “other”, that now represents the most populous class. We will discuss the effects of this unbalance in section 5. For the threat-type we can see that, as expected, the most represented classes are greyware, viruses, and downloaders, while dialers are by far the less common with only few tens of samples. Regarding malware behavior, the first two classes - filemodify and execdownload- make half of the total samples. Coherently with the threats, class “dial” is the class with fewest samples.

### 4 Pipeline Architecture Overview

As introduced in Section 1, the main goal of the proposed pipeline is to accurately classify malware using different stages of processing, each one leveraging a properly trained classifier. Figure 4 shows how we implemented the pipeline: the first stage detects whether input samples are malicious or benign; those classified as malicious are propagated to the second stage, which is responsible for labeling them with a known malware category (e.g., virus, backdoor, greyware). It is worth noting that, in this and the next stages, we employ (different) classifiers able to recognize misclassified benign samples, results of errors occurred in the previous stages. After being categorized, malicious files are further classified in families (e.g., xtrat and vtflooder). Analogously to the previous stage, the employed classifier is able to discriminate benign samples — identified as malicious — but also classify malicious samples belonging to less recurrent
families into a specific class, called other. Finally, different malware families are classified according their malicious behaviour. Whenever a sample is classified incoherently through the pipeline, that is, is classified as malware is the detection stage but as a benign in some next stage, we envision another stage that gathers all these suspicious samples, the quarantine, and reclassifies again them to improve the overall malware detection performance.

For training pipeline’s classifiers, we use the same data flow described above: samples used to validate a classifier in a stage are, then, provided in input to the next stage to train the following classifier. During the training phase, at each stage of the pipeline -detection and threat-type, family, and behavior classification-, the output of the preceding stage is split into train and validation sets, and a Gradient Boosting Decison Tree (GBDT) classifier is trained and its hyper-parameters optimized for the specific task. We used the Python API implementation of Microsoft’s LightGBM\(^2\) open source framework, that proved to be the the optimal solution between training time and performance [38].

As described in the original EMBER benchmark model [24], the raw features of each sample are mapped into a fixed-size vector of length 2351. In this work we start from the same feature set but we one slight modification. The original feature vectors rely heavily on the hashing trick in order to contain the variable and potentially very large number (order of millions) of imported functions, while here we chose to keep only the first 151 most common ports in order to keep the balance with the other major groups of features. A comparison between our model

\(^2\)https://github.com/microsoft/LightGBM
and the original EMBER results shows that the model performance is comparable (86.3% TPR @ 0.1% FPR versus 86.8% TPR @ 0.1% FPR of EMBER’s model), but with the advantage of a more interpretable result in our case.

5 Experimental Evaluation

Table 1 summarizes the experimental evaluations carried out on the proposed pipeline. For completeness, it reports both the results obtained during the validation and testing of the classifiers. Obtained results in the first stage (i.e., detection stage) are comparable with those obtained by EMBER developers [34]. It is important to note that our pipeline is able to correct false positives (benign samples detected as malicious), as discussed later in this section.

Samples predicted as malicious are then propagated to the stage that classifies threat types. Results reported in Figure 3 show that this stage has encountered the most troubles in classifying samples: indeed, benign samples are easily confused with grayware, viruses, and downloaders and the separation among different classes is not so clear (for example, as rootkits and grayware). Another aspect to take into account is the fallacy of the ground truth we used to train the threat-type stage classifier: further discussions are outlined in Section 6. Conversely to the previous stage, family classifier is more accurate in classifying specific families. More than 63% of families have been correctly classified with an accuracy greater than 85%, with many families having almost all their test samples identified by our classifier (e.g., xtrat, vtfloder, sivis, and upatre). 84% of misclassified benign samples fall in the ‘other’ class and this can be explained by having too few benign samples to build a classifier that is able to identify them. Further discussions are left in Section 6.

The last stage of the pipeline consists in classifying malicious samples by their behavior. For space constraints, we do not show the confusion matrix associated to the behavior stage. Similar to the threat-type classification stage, the behavior classifier fails to correctly recognize some specific behaviors (such as, autorun and osmodify). In addition to potential similar behaviors, another cause of misclassification is the few number of samples employed to train the behavior classifier: less than 20,000 instances to train the classifier on more than 20 different behaviors.

As described in Section 4, samples classified as benign in stages after the first one are quarantined for further analyses. Of all the 200,000 total samples in the test set, only 1150 (0.57%) end up in the quarantine stage. The classifier is able to recover 221 benign samples that were misclassified in the initial detection stage and, more importantly, recovered 635 malicious samples that were incorrectly labeled as benign in one of the classification stages.

Table 1: Results of the experimental evaluation carried out on the EMBER dataset reporting accuracy, AUC, false positives, and false negatives metrics both for validation and test phases.
Figure 3: Confusion matrix for the threat-type classification stage reporting the performance, in terms of accuracy, on the test set described in Table 1.

Figure 4: Confusion matrix for the family classification stage reporting the performance, in terms of accuracy, on the test set described in Table 1.

5.1 Feature Importance

The LightGBM framework [39], roughly speaking, builds a strong learner upon an ensemble of decision trees as weak learners. Decision trees assign at each learning step a dichotomous split on feature values based on the maximum obtainable information gain, so that it is a highly interpretable ML model.

Looking at table 2 we can see that features having the highest importance are common in almost all the stages, but each stage is characterized by a different ranking in the features’ importance, confirming the differences in the various typology of classes and the necessity to use different models in each stage of the pipeline. Among the most frequent features we note the virtual size, that is known to be a highly discriminative feature in malware classification [40].

It is interesting to note that the most important features for the quarantine stage, that is, the last stage in which we try to recover some wrong classifications of the previous stages, belong to the major groups of the byte entropy and printable strings. Since it has been observed that entropy and the presence or not of readable strings are correlated to the presence of packed or encrypted code, this could be suggestive of the fact that the “hardest” samples to correctly classify might be packed or encrypted, a known evasion techniques and a clear limitation to an approach based solely on static analysis.
Table 2: Top feature groups for the detection, threat-type, family, behaviour, and quarantine classification stages.

| Family        | Behavior                                  |
|---------------|-------------------------------------------|
| Header:dll characteristics Hashed | General Info:Vsize                      |
| Data Directories:RESOURCE TABLE:size | Header:dll characteristics Hashed         |
| General Info:Vsize | Header:timestamp                         |
| Section:Entropy Hashed | Data Directories:RESOURCE TABLE:virtual address |
| Strings:printabledist | General Info: n° imports                  |

5.2 Model Interpretability

In order to better understand the distribution of our data in the high-dimensional original feature space and, hence, be able to identify regions in which a certain classification stage might be less reliable, we apply UMAP [41, 42], a non-linear and unsupervised dimensionality reduction technique, to the full feature vector of size 1252 keeping the first 3 components. In turn, this allows also to suggest potential data mislabelling that could need an analyst further investigation.

Figure 5: 3-d representation of 5 different malware families. On the left the train set (fitting phase), on the right the test set. The frequency of the classes in the train and test set are highly uneven.
Figure 6: Prediction probability obtained by providing in input previously unseen benign samples to a family classifier, trained only on malware samples.

In figure 5, we show the resulting embedding for the first five most frequent families. We can see that all different families are generally well separated, often forming isolated sub-clusters that can be linked to specific types or variants of the main malware class. Other regions show instead a mixing of various classes. That is where the classification is more difficult and the model is more likely to fail. There are various reasons for the appearance of this regions: first of all, as explained in section 3.2, the labels are not uniquely defined and thus it could simply be an effect of an imperfect labelling. Moreover, as pointed out in Section 5.1, many classification errors are likely to be linked to the presence of packing or encryption, that is a well known limitation to static analysis. Nevertheless, an analyst using our model could easily assess the confidence in the classification based on the position of the considered sample in this low-dimensional space, providing an useful tool for interpreting the results.

6 Discussion

As discussed in 5, some issues arise in the different stages of the pipeline due to: (i) mislabeling of malware families reported in the EMBER dataset (refer to Section 3), (ii) overlaps among different threat types and behavior, and (iii) few samples used to train classifiers in the last stages of the pipeline.

Regarding the second issue, we have observed that malware behaviors are not completely independent one from another: as an example, osmodify is often confused with filemodify and execdownload behaviors, because in AVClass it is associated with rootkit malware that, in general, establishes communications with command-and-control servers to receive new commands and malicious payloads. Finally, Table 1 reports how the number of input samples, useful to train stage classifiers, is halved at each stage. As mentioned in Section 5, few samples explain poorer performance in accuracy for family and behavior classifications.
A final consideration regards, instead, the choice of using classifiers able to discriminate between malware and benign samples to refine the overall classification process. The idea was born out of trained classifiers’ capacity of accurately distinguishing between malicious and benign samples. As an example, Figure 6 shows the confidence that a different family classifier, trained only on malware samples, has in assigning previously unseen benign samples to other class. As already discussed in Section 5, it includes more than 2,800 malware families that have too few samples to build a classifier able to identify them, as already mentioned in Section 3.3. It is worth noting that no benign samples has been classified as belonging to one of the most-frequent malware families of the EMBER dataset, listed on the x-axis of Figure 6. The plot has been computed on 100,000 benign samples, extracted from the test set used for the family classification stage.

This supports our hypothesis that the poor performance in correctly detecting benign samples in the classification stages of the pipeline is due to the very small number of benign samples during model training ($\approx 1\%$ of all the benign samples).

7 Conclusion and Future Work

In this work we proposed a first implementation of a pipeline for a complete classification of Windows PE files using a machine learning approach on static features. The pipeline is able to separate benign and malicious samples, and for those samples classified as malicious it provides a exhaustive classification in terms of threat-type, malware family, and behaviour. Classification results, although suffering from known limitations such as the size of the training data, the imperfect labelling of the ground truth, and the semantic gap of models based on static features only, are comparable to the current state of the art for similar works while providing much more detailed information on malware characteristics. Finally, the extracted feature vector characterising the raw PE and the specific ML model implemented provide an interpretable result, and the pipeline is scalable to much larger datasets. Therefore, we consider this work as a first step towards a useful tool that can help security analysts to manage novel threats, reducing time and costs of the analysis. For the near future, we plan to improve the described pipeline by: (i) considering a larger dataset for training (ii) fixing the ground truth and considering the possibility of a multi-label classification scheme, and (iii) further explore the properties of the embedded features’ space described briefly in section 5.2. It could be interesting to include also a detector for packed/encrypted samples in the early stages of the pipeline, and move to a hybrid approach, combining static and dynamic features for a better characterization of the sample files.

References

[1] Michael Sikorski and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. No Starch Press, 2012.
[2] Nikola Milošević. History of malware. arXiv preprint arXiv:1302.5392, 2013.
[3] AV-TEST malware statistics. https://www.av-test.org/en/statistics/malware/. Accessed: 2020-03-11.
[4] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. International Journal on Advanced Science, Engineering and Information Technology, 8(4-2):1662, 2018.
[5] Ahmad Ridha Jawad, Khaironi Yatim Sharif, and Ammar Khalel Abdulsada. N/a and signature analysis for malwares detection and removal. *Indian Journal of Science and Technology*, 12:25, 2019.

[6] KA Monnappa. *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*. Packt Publishing Ltd, 2018.

[7] Alireza Souri and Rahil Hosseini. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, 8(1):1–22, 2018.

[8] Muhammad Furqan Rafique, Muhammad Ali, Aqsa Saeed Qureshi, Asifullah Khan, and Anwar Majid Mirza. Malware classification using deep learning based feature extraction and wrapper based feature selection technique. *arXiv preprint arXiv:1910.10958*, 2019.

[9] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. Malware detection by eating a whole exe. *stat*, 1050:25, 2017.

[10] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[11] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[12] Lahouari Ghouti. Malware classification using compact image features and multiclass support vector machines. *Iet information security (01 2020)*, 2019.

[13] Federico Maggi, Andrea Bellini, Guido Salvaneschi, and Stefano Zanero. Finding non-trivial malware naming inconsistencies. In *International Conference on Information Systems Security*, pages 144–159. Springer, 2011.

[14] Peter Szor. *The Art of Computer Virus Research and Defense*. Pearson Education, 2005.

[15] Felipe N Ducau, Ethan M Rudd, Ted M Heppner, Alex Long, and Konstantin Berlin. Automatic malware description via attribute tagging and similarity embedding. *arXiv preprint arXiv:1905.06262*, 2019.

[16] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International symposium on research in attacks, intrusions, and defenses*, pages 230–253. Springer, 2016.

[17] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[18] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[19] Rahul, Priyansh Kedia, Subrat Sarangi, and Monika. Analysis of machine learning models for malware detection. *Journal of Discrete Mathematical Sciences and Cryptography*, 23(2):395–407, 2020.

[20] Jagsir Singh and Jaswinder Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, page 101861, 2020.

[21] Edward Raff and Charles Nicholas. A survey of machine learning methods and challenges for windows malware classification. *arXiv preprint arXiv:2006.09271*, 2020.

[22] Michael R Smith, Nicholas T Johnson, Joe B Ingram, Armida J Carbajal, Ramyaa Ramyaa, Evelyn Domschot, Christopher C Lamb, Stephen J Verzi, and W Philip Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and information security. *arXiv preprint arXiv:2005.01800*, 2020.

[23] Zhang Fuyong and Zhao Tiezhu. Malware detection and classification based on n-grams attribute similarity. In *2017 IEEE International Conference on Computational Science and Engineering*. 11
(CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), volume 1, pages 793–796. IEEE, 2017.

[24] P. Roth. Ember Improvements. Conference on Applied Machine Learning for Information Security (CAMLIS19), 2019.

[25] Yoshihiro Oyama, Takumi Miyashita, and Hirotaka Kokubo. Identifying useful features for malware detection in the ember dataset. In 2019 seventh international symposium on computing and networking workshops (CANDARW), pages 360–366. IEEE, 2019.

[26] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. Static pe malware detection using gradient boosting decision trees algorithm. In International Conference on Future Data and Security Engineering, pages 228–236. Springer, 2018.

[27] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 108–125. Springer, 2008.

[28] Mohamed Nassar and Haidar Safa. Throttling malware families in 2d. arXiv preprint arXiv:1901.10590, 2019.

[29] Xin Hu, Kang G Shin, Sandeep Bhatkar, and Kent Griffin. Mutantx-s: Scalable malware clustering based on static features. In 2013 {USENIX} Annual Technical Conference ({USENIX} {ATC} 13), pages 187–198, 2013.

[30] Bowen Sun, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. Malware family classification method based on static feature extraction. In 2017 3rd IEEE International Conference on Computer and Communications (ICCC), pages 507–513. IEEE, 2017.

[31] Bo Yu, Ying Fang, Qiang Yang, Yong Tang, and Liu Liu. A survey of malware behavior description and analysis. Frontiers of Information Technology & Electronic Engineering, 19(5):583–603, 2018.

[32] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware. In Botnet Detection, pages 65–88. Springer, 2008.

[33] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. Unknown malware detection using network traffic classification. In 2015 IEEE Conference on Communications and Network Security (CNS), pages 134–142. IEEE, 2015.

[34] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. ArXiv e-prints, April 2018.

[35] Romain Thomas. Lief - library to instrument executable formats. https://lief.quarkslab.com/, April 2017.

[36] Virus Total. https://www.virustotal.com/gui/.

[37] Silvia Sebastián and Juan Caballero. Avaclass2: Massive malware tag extraction from av labels. In Annual Computer Security Applications Conference, pages 42–53, 2020.

[38] C. Galen and R. Steele. Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset. In 2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS), pages 1–7, 2020.

[39] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30:3146–3154, 2017.

[40] J. Yonts. White paper: Attributes of malicious files. Technical report, SANS Institute, June 2012.

[41] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. ArXiv e-prints, February 2018.

[42] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. The Journal of Open Source Software, 3(29):861, 2018.