Collaboration of Intelligent Systems to Improve Information Security

Lili Diao[1] and Honglan Xu[2(✉)]

[1] Nanjing Normal University of Special Education, Nanjing, China
[2] ZTE Corporation, Nanjing, China
xu.honglan@zte.com.cn

Abstract. In more and more popular computer systems, industry protects the network on top of them via scanning malware (malicious software or applications) through some generic properties. It is useful but not accurate enough – even 0.01% of accuracy gain can cause millions of malicious software or applications over internet to steal privacy or break down computer systems. To address this problem, the paper proposes building independent intelligent systems to predict possibilities of malware through different angles: Generic properties, Import table properties, Opcode properties etc. Each single intelligent system does not have highest prediction accuracy; whereas, collaboration of independent intelligent systems can bring accuracy improvements over single ones in experiments, which brought tremendous value on helping improving computer information security.

Keywords: Information Security · Malware · Intelligent System · Model Collaboration

1 Introduction

Information security is always crucial to protect computer networks. Nowadays there are more and more software or applications very useful for helping people in daily life. Unfortunately, malicious software which aims to steal secretes or privacies and break down computer or network systems, for example bank accounts, credit cards, etc., also grows up rapidly. In this sense, how to identify malicious software efficiently becomes significant to guarantee our digital world running correctly. The traditional way is storing virus' static patterns (known blocks of binaries) and comparing when scanning. However, it is far from being efficient since such patterns are increasing crazily. Artificial Intelligence or Machine Learning is recently applied to comprehensively calculate possibilities of being malicious based on general properties or features of software or applications. As an instance, researchers organize information about file size, segment size, text size, segment offset and so on to form a big table on top of malicious or benign software samples. Thus a machine learning model recording the knowledge of discriminating good or bad is worked out, and is able to be applied into real network environment to predict if any unrecorded software or application is benign or malicious.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
B. Wang et al. (Eds.): ICMTEL 2023, LNICST 535, pp. 64–75, 2024.
https://doi.org/10.1007/978-3-031-50580-5_6

Now the problem is, single machine learning model with generic properties of software or applications cannot reach the highest accuracy level (a combination of precision and recall). How to further improve the accuracies is still a very tough problem to predict malicious software or applications. This paper introduces a mechanism of collaboration among different intelligent systems to further improve the accuracies in predicting malicious software or applications.

2 Related Works

Malicious software or applications (Malware) is a well-known name for the software or applications which do not intent to do good things on computer and/or network systems. In information security field, there are multiple research teams using intelligent systems to predict Malware. Some of them are using different kinds of features to build a hybrid system to detect them. A standard way to detect malware is signature based methods, which is encountering more and more problems [1]. Many modern malicious applications were designed to have multiple polymorphic layers to hide themselves [2]. A set of intelligent systems to detect malwares were introduced in [3]. Boosting algorithms employing n-grams were observed outperforming Bayesian classifiers and Support vector machines in [4]. [5] used association rules found from Windows API execution sequences to identify good or bad. Hidden Markov Models, neural networks, as well as Self-Organizing Maps were used to detect variants of Windows executables in [6, 7] and [8]. [9] proposed a way to cope with ransomware. Active Learning was used to obtain 91.5% accuracy rate on iOS applications as in [10]. [11] obtained 91.43% accuracy. In [12], static and dynamic features were utilized on Android applications. [13] presented a hybrid technique using permissions & traffic features. KNN and K-Medoids algorithms as a whole produced 91.98% accuracy. Mac OS X applications also have multiple approaches to detect malware as in [14] and [15].

Some research work got good results. However, there are still problems of relying on single types of features to build intelligent systems, or the ways to mixture multiple types of features are naïve. In the sense, we need more specifically designed mechanism to apply malware detection in real network environments.

3 Intelligent Systems

3.1 Windows PE Properties and Intelligent Systems

Windows applications including malware and good ware are basically using PE (Portable Executable) file structure. PE file format is a data structure that tells the Windows OS loader what information is required to manage the wrapped executable code. This includes dynamic library references for linking, API export, import tables, resource management data, and TLS data [16]. The architecture of PE is illustrated by Fig. 1:

The PE format starts from a MS-DOS header accompanied by executable code. The header has 64 bytes. The PE header indicates information about the entire file. The basic header contains the machine type or architecture, a time stamp, a pointer to symbols, if the application handle addresses above 2 GB, if the file needs to be copied to the swap

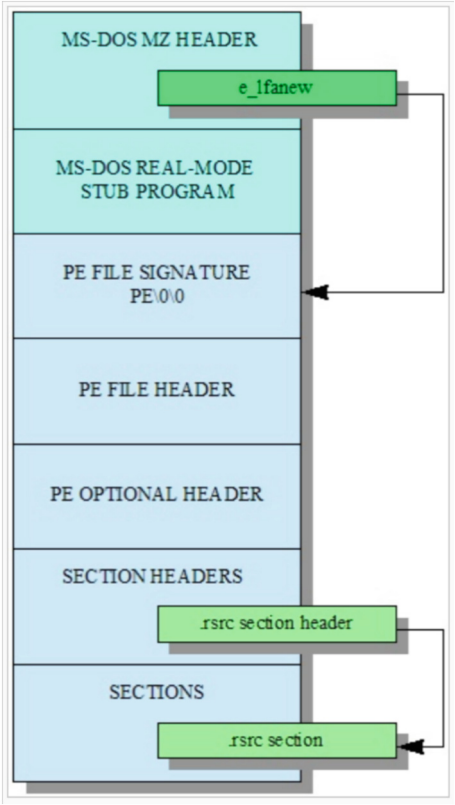

Fig. 1. An Overview of PE Format

file, and so on [17]. More detailed information is stored on different sections of the PE file and needs to be discovered based on domain knowledge.

Based on PE structure and contained information, we can design various types of features to describe the executables. There are many static properties of PE such as file length, file entropy, and file segments etc. which are unchanged characteristics after the PE file formed. We call such type feature as generic features. Another type of feature is the export/import table of PE file, which lists all connected function calls. They are IAT features for convenience. Opcode is the machine code in PE which is to be performed when run it. Because Opcode directly connects to executing, we believe it will have great value when make it as a type of feature. The last type of feature is the readable text in all PE sections. Sometimes such feature make contain some information of what the software is aiming for. We call this type of feature "String" feature.

The first intelligent system to detect malware is based on the generic properties. We selected nearly 600 features on top of the properties of PE files to build up machine models to predict malicious. They are the base information of each software or application, and may have (partially) indicators of good or benign. We then use various analysis to

rank importance of the features and compress the intelligent system into a much smaller feature set. We call the intelligent system Generic Model or intelligent system.

The second intelligent system is based on export/import table of PE. It is all the API calls of the software or application. This set of properties has value of disclosing what functions the software or applications wants to quote. We mapped them into another feature space to build up an easy to control intelligent system to discriminate good or bad. We call this intelligent system IAT Model or intelligent system.

The third intelligent system is based on Assemblies of PE operational codes. They are the machine code of each software manipulating registers, jumps, moves, etc. This set of properties has value of disclosing what actions the software or application really wants to do. The original features are also mapped into another feature space through mapping functions before building up intelligent system to detect malware. We call this intelligent system Opcode Model or intelligent system.

The last independent intelligent system is based on readable characters within PE files. This set of properties has value of disclosing what message the software or application wants to deliver to its direct user. Also, for ease of control, we mapped the properties into another feature space to build up intelligent system to identify malwares. We call this intelligent system String Model or intelligent system.

3.2 Intelligent Algorithms

Given feature space, we then can choose appropriate machine learning algorithms as the core of intelligent systems, for instance SVM, deep learning, bagging and boosting, etc. To obtain accuracies as high as possible, we choose XGBoost, a specific algorithm in boosting family.

XGBoost (Extreme Gradient Boosting) is a popular machine learning algorithm that has gained significant attention in recent years due to its high accuracy and efficiency in solving a wide range of supervised learning problems, such as classification and regression. XGBoost is an ensemble learning method that combines multiple weak models to create a strong model. The algorithm uses decision trees as the base learners. The underlying principles of XGBoost are based on the gradient boosting framework, which involves iteratively adding new weak models (trees) to the ensemble. The loss function is defined as the difference between the predicted values and the actual values, and the goal is to minimize this difference [18]. The final prediction is obtained by summing the predictions of all the trees.

Here we listed some key features of XGBoost algorithm.

Regularization. XGBoost uses L1 and L2 regularization to prevent overfitting and improve the generalization of the model. By adding penalty terms to the loss function, XGBoost encourages the model to use only the most important features and to avoid overfitting.

Tree Pruning. XGBoost uses a technique called tree pruning to remove unnecessary branches from the decision trees, which helps to reduce overfitting and improve the accuracy of the model. Tree pruning involves removing branches that do not contribute significantly to the reduction in the loss function. XGBoost uses a greedy algorithm to determine which branches to prune, starting from the leaves and working its way up to the root.

Parallel Processing. XGBoost is designed to be highly scalable and can take advantage of parallel processing to speed up the training process. The algorithm can be run on a single machine or on a distributed cluster of machines. XGBoost uses a technique called approximate computing to reduce the computational cost of the algorithm, by approximating the gradients and Hessians of the loss function.

Handling Missing Values. XGBoost can handle missing values in the input data by assigning them to the most appropriate node during the tree construction process. XGBoost uses a technique called sparsity-aware split finding to handle missing values, which involves splitting the data into two groups: one group with missing values and one group without missing values. The algorithm then determines the best split for each group separately.

Feature Importance. XGBoost provides a measure of feature importance, which can be used to identify the most important features in the input data. Feature importance is calculated by summing the number of times each feature is used in the decision trees, weighted by the gain of each split. The gain of a split is the reduction in the loss function that is achieved by the split.

XGBoost has been used successfully in a wide range of applications, including image classification, natural language processing, and financial forecasting. In image classification, XGBoost has been used to classify images based on their content, such as identifying objects in a scene. In natural language processing, XGBoost has been used to classify text based on its content, such as identifying the sentiment of a tweet. In financial forecasting, XGBoost has been used to predict stock prices and other financial variables.

3.3 Working Model of Single Intelligent Systems

As Fig. 2 shows, PE files with feature generation/mapping can be fed into different independent intelligent systems to decide to block a malware. Multiple types of features can build multiple independent intelligent systems.

Unfortunately, single independent cannot have highest accuracies in detecting malware in computer systems. Each intelligent system has its strong part and weak part. Cooperation of various intelligent systems may bring hope for further improving via mutually covering weak points. In the sense we designed cooperation mechanism of the independent intelligent systems. Make the independent intelligent systems collaborating with each other may be a novel way to improve the accuracy of malware detection.

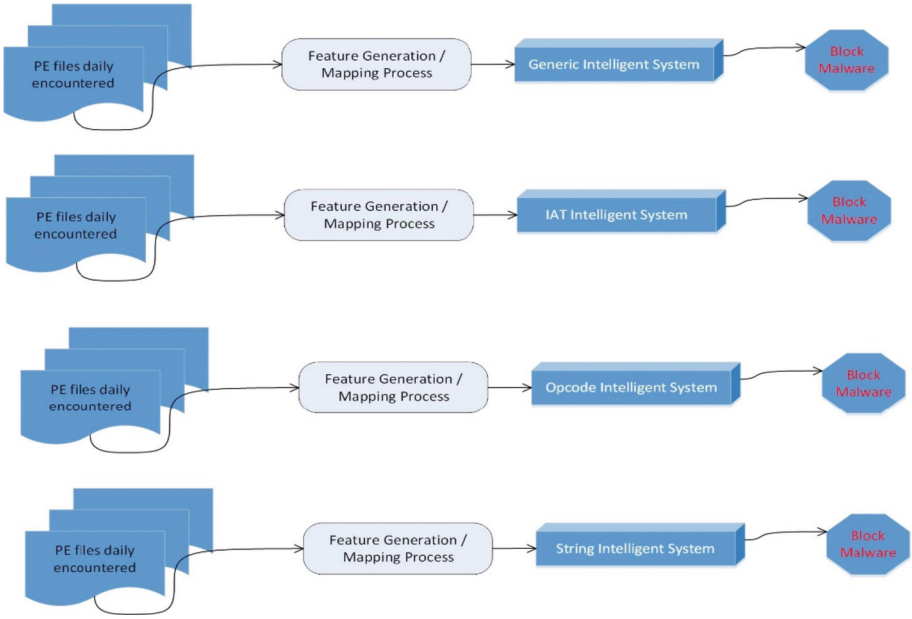

Fig. 2. The Connections of Intelligent Systems and PE files

4 Collaborating of Intelligent Systems

Each single intelligent system can have its ability to detect malware or bypass good software quickly. It is very difficult to further improve the accuracies when advanced machine learning algorithms have been given. Collaboration is the only way of further improvement other than improve each individual intelligent system. The advantages of model fusion compared with a single model detection include (but not limited to):

1. Easier to optimize small models and search for best parameters.
2. Logically, different types of features will not impact each other before final scoring – thus can concentrate on specific knowledge mining.
3. Most importantly, model fusion can no longer request for the same dataset to train; instead, we can combine various models from various data/various parameters/various algorithms
4. …

In the design for collaboration, the concept can be represented by Fig. 3.

Based on a training dataset, use approaches of resampling (bootstrapping and so on) to build various sample set. Various sample set combined with various intelligent systems can generate multiple classifiers or predicting models. They are the first tier classifiers. The first tier classifiers have their own (different) predictions on same training data. The predictions can be regarded as input features into training the second tier classifier – the so-called "Meta" classifier. The Meta classifier, as a second level classifier, may bring extra improvements than single intelligent systems.

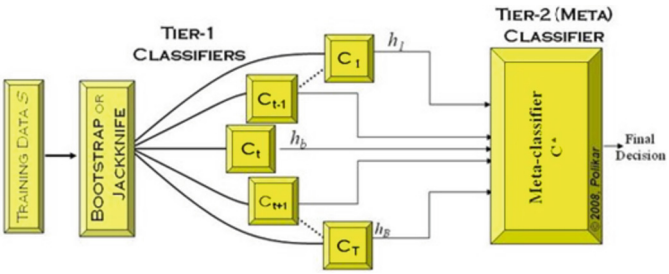

Fig. 3. Concept of Collaboration among Independent Intelligent Systems

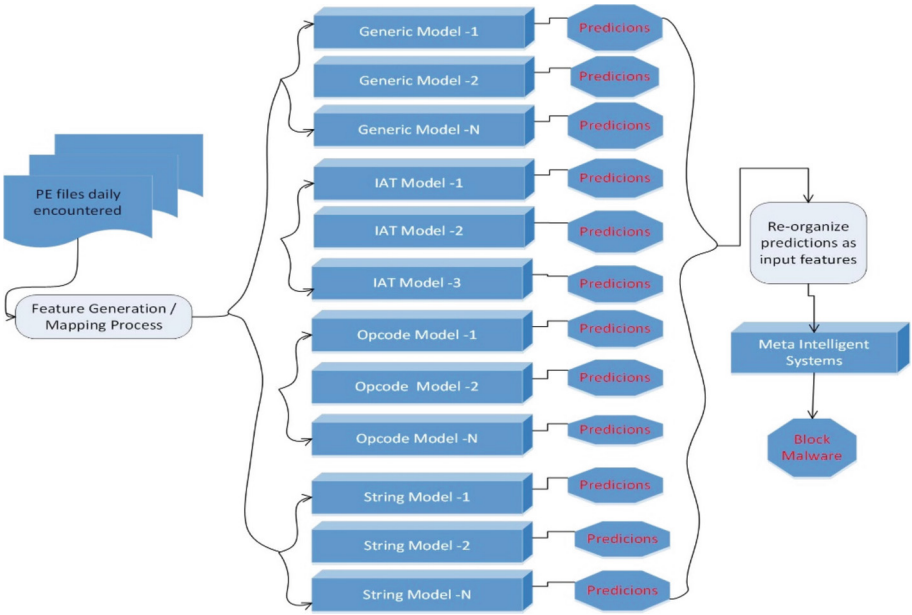

Fig. 4. Architecture of the Collaborated Intelligent Systems to Detect Malware

The architecture of the collaborated intelligent systems is shown in Fig. 4.

For input PE files, system prepares features and do feature mapping operations for what intelligent systems need. Each independent intelligent system (Generic, IAT, Opcode, String) in training process has generated multiple (1, 2, …, N) machine learning models with selected machine learning approaches. Each of the models (intelligent systems) can have its own predictions on whether a PE file is good or bad. The collaboration system collected all the predictions and quote the "Meta" classifier to make final decision on good or bad.

5 Experiments

5.1 Feature Analysis

With the datasets of PE malware and good ware, we get 6000+ malware samples, 2 million+ good samples, as well as 2 million+ pending (unknown good or bad) software applications. Based on them, we can partition the data into training sets and tests sets with appropriate sizes to build up (train) intelligent systems. The huge pending dataset will be used to estimate generalization errors empirically.

With Removing logically and mathematically invalid features as well as miss valued ones, we then have 397 features remained. We rank the 397 features through different ways:

Gap of Mean and Variance;
Chi2 Test and Mutual Info Selection;
Model Ranking.

Figure 5 shows comprehensive ranking results by sorting the features through the height of blue bars of features. On top of that we built an intelligent system with 52 generic features.

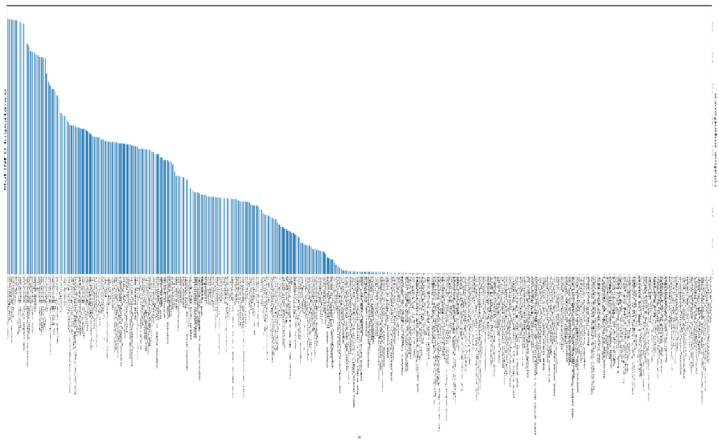

Fig. 5. Feature Ranking Results

5.2 Building Intelligent Systems

We compared XGBoost model prediction errors (or accuracies) with different number of selected features. Feature number around 50 can generate almost best model with high precision and score along with lower costs (feature size). The test results show that 52-feature-models can have simple form with best level accuracies. As key indicators for accuracy, the precision and recall are calculated in known good (benign) and bad (malicious) software/application sets. In information security context, Precision stands

for how many predicted malicious are really malicious, while Recall stands for how many malicious are predicted malicious by intelligent systems. FPR (false positive rate) stands for how many good samples are incorrectly predicted or classified as malicious.

In mathematics, the indicators are calculated by following equations:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \tag{1}$$

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \tag{2}$$

$$\text{F1} = 2\text{PR}/(\text{P} + \text{R}) \tag{3}$$

$$\text{TPR} = \text{TP}/(\text{TP} + \text{FN}) \tag{4}$$

$$\text{FPR} = \text{FP}/(\text{TN} + \text{FP}) \tag{5}$$

Besides precision and recall, we also need to estimate the generalization errors of machine learning models in order to apply them in real production environments – where all samples are new and unknown (good or bad). With the 52-feature-models, we can have three kinds of machine learning models and the corresponding evaluation results (precision, recall, FPR, TP (true positives), TN (true negatives), FP (false positives), FN (False negatives), Pending-Test (detected positive rate in Unknown datasets), etc.

Machine learning models built from the 52 generic features can contribute fundamental precision and recall of generic intelligent system. As described above, we also built other intelligent systems with XGBoost algorithm using IAT (import table information), Opcode (ASM machine code) and String (readable characters in PE) features respectively. Table 1 shows their performance indicators.

Table 1. Performance Indicators of various Independent Intelligent Systems.

Intelligent System	Precision	Recall	FPR	Pending-Test	TP	TN	FP	FN
Generic	99.72%	98.04%	8.9E−6	7.5E−4	6427	2017169	18	128
IAT	95.72%	97.43%	1.4E−4	2.0E−3	6387	2016902	285	168
Opcode	98.14%	97.77%	6.0E−5	6.4E−4	6409	2017066	121	146
String	99.18%	97.74%	2.6E−5	4.3E−4	6407	2017134	53	148

Comparing the performance indications of different intelligent systems, we found each intelligent system has its own pros and cons. Collaboration of different intelligent systems may have the effect of $1 + 1 > 2$ as discussed in Sect. 4. With a specifically designed strategy of collaboration, we got promising experimental results.

5.3 Collaboration Results

We conducted experiments with combining multiple intelligent systems via approaches described in Sect. 4. Their single models have performance indicators listed in Table 1. Here Table 2 shows that with more intelligent systems combined, the final (Meta) system can have improved precision, recall, and generalization errors (in pending data sets) compared with single intelligent systems as what Table 1 recorded.

Table 2. Performance Indicators of Multiple Intelligent Systems Combined

Intelligent System	Precision	Recall	FPR	Pending-Test	TP	TN	FP	FN
Generic + IAT	99.95%	97.71%	1.5E−6	4.4E−4	6405	2017184	3	150
Generic + String + IAT	99.95%	97.67%	1.5E−6	4.4E−4	6402	2017184	3	153
Generic + String + IAT + Opcode	99.97%	98.08%	9.9E−7	3.8E−4	6429	2017189	2	126

From the first line of Table 1, we can find significant improvements of FP errors (15, from 18 to 3) even only combined two intelligent systems (Generic and IAT). With all intelligent systems joined, TP and TN got improvements by 2 and 20 respectively. The most important thing is, FP errors were improved by 16. That means, multiple intelligent systems collaboration can mostly decrease the crucial error of false positives, which is key of malware detection products.

In other parts, precisions and recalls are also improved to some extent through collaboration. Though their numeric improvements are not that much, the absolute number of correct detections of malware is big because malware detecting is coping with 10 million+ new software in internet to protect users.

In Pending Test, collaboration of multiple intelligent systems also implied its lower potential generalization false alarm detections. It is a precious merit among all information security products.

6 Conclusion

In detecting malicious executables for PE, building intelligent systems to make prediction automatically is useful and necessary to handle millions of newly appeared software in internet every single day. However, single intelligent systems are far from enough in making a precise enough detection result with reasonable generalization errors. We analyzed various properties of PE files and built different intelligent systems based on them. With a specific collaboration strategy, combination of different intelligent system can output the best accuracy (precision and recall etc.).

With the power of collaborated intelligent systems, industry can block more malware with higher accuracy. Thus information exchange in Internet can be secured. Further

research may include exploring more strategy to build intelligent systems such like CNN or Transformer, as well as exploring more efficient approaches of collaboration for even higher performance.

Another important thing of malware detection is to collaborate between static and dynamic features of software. The four intelligent systems we designed are based on static features of PE files. However, they are not enough to discover real intentions of software. Dynamic features indicate real behaviors of PE files in operating systems. Multiple layers of malware detection is a key direction in information security. We may try to record behaviors of software in virtual OS and put them into dynamic intelligent systems as the 5th collaboration system to further improve the capability of information security.

References

1. Santos, I., Penya, Y.K., Devesa, J., Garcia, P.G.: N-grams-based file signatures for malware detection. In: ICEIS 2009 - Proceedings of the 11th International Conference on Enterprise Information Systems, Volume AIDSS, Milan, Italy, pp. 317–320 (2009)
2. Konstantinou, E.: Metamorphic virus: analysis and detection. In: Technical Report RHUL-MA-2008-2, Search Security Award M.Sc. thesis, 93 p. (2008)
3. Chan, P.K., Lippmann, R.: Machine learning for computer security. J. Mach. Learn. Res. **6**, 2669–2672 (2006)
4. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**, 2721–2744 (2006). Special Issue on Machine Learning in Computer Security
5. Ye, Y., Wang, D., Li, T., Ye, D.: IMDS: intelligent malware detection system. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1043–1047 (2007)
6. Chouchane, M.R., Walenstein, A., Lakhotia, A.: Using Markov chains to filter machine-morphed variants of malicious programs. In: Malicious and Unwanted Software, 2008. Proceedings of the 3rd International Conference on MALWARE, pp. 77–84 (2008)
7. Santamarta, R.: Generic detection and classification of polymorphic malware using neural pattern recognition (2006). https://www.semanticscholar.org/paper/GENERIC-DETECTION-AND-CLASSIFICATION-OF-POLYMORPHIC-Santamarta/5cda37f3fe61f1fa156752be27fdb7cc40983e84
8. Yoo, I.: Visualizing windows executable viruses using self-organizing maps. In: VizSEC/DMSEC 2004: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 82–89. ACM (2004)
9. Baldwin, J., Dehghantanha, A.: Leveraging support vector machine for opcode density based detection of crypto-ransomware. In: Dehghantanha, A., Conti, M., Dargahi, T. (eds.) Cyber Threat Intelligence. AIS, vol. 70, pp. 107–136. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73951-9_6
10. Bhatt, A.J., Gupta, C., Mittal, S.: iABC-AL: active learning-based privacy leaks threat detection for iOS applications. J. King Saud Univ. Comput. Inf. Sci. **33**(701), 769–786 (2021)
11. Zhang, H., et al.: Classification of ransomware families with machine learning based on N-gram of opcodes. Future Gener. Comput. Syst. **90**, 211–221 (2019)
12. Riasat, R., et al.: Onamd: an online android malware detection approach. In: 2018 International Conference on Machine Learning and Cybernetics (ICMLC), vol. 1, pp. 190–196. IEEE (2018)

13. Arora, A., et al.: Poster: hybrid android malware detection by combining supervised and unsupervised learning. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 798–800. ACM (2018)
14. Singh, A., Bist, A.S.: OSX malware detection: challenges and solutions. J. Inf. Optim. Sci. **41**(2), 379–385 (2020)
15. Gharghasheh, S.E., Hadayeghparast, S.: Mac OS X malware detection with supervised machine learning algorithms. In: Choo, K.K.R., Dehghantanha, A. (eds.) Handbook of Big Data Analytics and Forensics, pp. 193–208. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-74753-4_13
16. Tech-zealots.com. https://tech-zealots.com/malware-analysis/pe-portable-executable-structure-malware-analysis-part-2/. Accessed 26 May 2023
17. Wiki. https://wiki.osdev.org/PE. Accessed 01 Feb 2023
18. Nielsen, D.: Tree boosting with XGBoost why does XGBoost win "every" machine learning competition? Master's thesis, NTNU (2016)