2022 2nd International Conference on Intelligent Technologies (CONIT)
Karnataka, India. June 24-26, 2022

Opcode and API Based Machine Learning Framework For Malware Classification

Hrishabh Soni
Dept. of C.S.E.
NIT Rourkela
Odisha, India
hrishabhsoni4764@gmail.com

Pushkar Kishore
Dept. of C.S.E.
NIT Rourkela
Odisha, India
monumit46@gmail.com

Durga Prasad Mohapatra
Dept. of C.S.E.
NIT Rourkela
Odisha, India
durga@nitrkl.ac.in

***Abstract*—Traditional machine learning (ML) based malware detectors depend on crafted human features that fail for recent malware. Deep learning (DL) based solutions solve the above issue but require a lot of training time. The real challenge is designing a malware detector with a higher F1-score for ML techniques. In this paper, we present a novel framework that classifies malware using the features named opcode and application programming interface (API) calls. First, API calls and opcodes are extracted using interactive disassembler pro (IDA pro) from the malicious samples' assembly language source code (ALSC) file. Then, the continuous n-gram technique is applied to the extracted API and opcode to create the dataset's features. The value of features in each row is based on its frequency in the concerned extracted behaviors. We scale the values in the dataset using the term frequency-inverse document frequency (TF-IDF) methodology. The best combination of n-gram and feature selection techniques is identified for API and opcode based datasets. The final label of the malicious samples is decided by the highest probability of the detection made by API and opcode based detectors. For analysis, an off-the-shelf dataset named Microsoft Malware is used. We achieve an F1-score of 96% for API-based detector and F1-score of 98% for opcode based detector. Our framework achieves an overall F1-score of 99.3%, better than the recent state-of-the-art techniques. Apart from attaining a higher F1-score, there is a reduction in training time due to using ML techniques instead of DL techniques.**

***Index Terms*—Malware detector, Opcode, API, Assembly language source code, Machine learning**

I. Introduction

These days cyber attacks are considered the topmost significant risk. Throughout history, cyberwarfare influenced major elections and businesses. The most dangerous espionage campaign against a political party named Democratic National Committee occurred in 2015 and 2016, where hackers released private and confidential information. Symantec [1] had reported that the count of new ransomware families found during 2016 tripled, and infections grew by 36%. In 2017, wannacry ransomware used EternalBlue[1] exploit to attack unpatched computers running Windows operating systems. Consecutively, a NotPetya cyberattack was planned on unpatched computers in the same year. The global malware industry is estimated to grow every year. The dark service providers distribute malicious software to criminals and hackers to exploit naive users. These days, hackers are stealing the user credentials, or computing power for mining cryptocurrencies [2].

Malware behaviour can be extracted using tools like interactive disassembler pro (IDA Pro) [2], Cuckoo SandBox [3], NITRSCT [4], Winitor [5], x64dbg[6], etc. The analysis is divided into two parts, namely static and dynamic. In the case of static analysis, the code is analyzed without being executed. While in the case of dynamic analysis, the code is executed, and its behavior is recorded for analysis. We use the benchmark dataset, Microsoft Malware Classification [3], to test our proposed framework. In the past, hand-crafted rules were sufficient to detect the presence of malware. These traditional signature based techniques' performance degrades over time due to exponential growth in malware families. Apart from that one, it was unable to detect the zero-day attacks. Consequently, machine learning (ML) techniques replaced signature based due to the generalization on unseen malware. Traditional ML techniques need a set of features which properly identify the sample. However, domain experts are consulted to select the appropriate features that provide the highest F1-score with the least training time. Nowadays, end-to-end learning algorithms have been introduced. They take the raw samples for analysis and predict whether they are malicious or benign. However, these algorithms fail for skewed or minor datasets. At last, deep learning (DL) is applied for malware classification and detection. They achieved higher F1-score but lagged in training on low-powered systems.

Some challenges currently persisting in the field of malware classification are:

1) Training time is higher due to malware classifiers based on deep learning [4].
2) A lot of features named application programming interface (API) calls, opcode, and byte code representations need to be extracted [4], [5].
3) The highest accuracy and F1-score are achieved using deep learning models only [4].

[1]https://attack.mitre.org/

[2]https://hex-rays.com/ida-pro/
[3]https://cuckoosandbox.org/
[4]https://github.com/pushkarkishore/NITRSCT
[5]https://www.winitor.com/
[6]https://x64dbg.com/

To resolve the above challenges, we fix our objectives as follows:

1) To select only two feature categories for creating datasets like API and opcode instead of all types of features.
2) To extract features without the assistance of pre-defined packages to achieve perfection.
3) To apply term frequency-inverse document frequency (TF-IDF) on dataset having features created using n-grams varied from 1 to 2.
4) To search for the best ML techniques which can beat the state-of-the-art methods.
5) To design a framework that can classify the malware with a higher F1-score.

Paper Organization: Section II sheds light on the related works. Section III discusses the steps involved in the proposed framework. Section IV provides information about the experimental setup and the obtained results. Section V compares our work with the present state-of-the-art techniques, and Section VI concludes the work with some future directions.

II. Related Works

Ahmadi et al. [5] used static analysis for malware classification. They combined feature types like entropy, 1-gram byte, string length, image representation, metadata, symbol opcode, API, section, and miscellaneous. Some features like Haralick and local binary patterns (LBP) that described an image's features were used for malware classification. An ensemble learning technique named XGBoost was used to achieve 99.76% accuracy. Training time complexity is higher due to ensemble and utilization of numerous feature types. Bidoki et al. [6] used dynamic analysis for malware detection. They detected malware requiring numerous processes to fulfill their goal. Collaborative processes share a common execution policy. They mainly applied reinforcement algorithm and combined API calls of collaborating processes to detect malware. Jain and Meena [7] proposed a non-signature based approach for detecting malicious code. They extracted n-grams with n = 1 to 8 from raw byte patterns. Class-wise document frequency was used for reducing feature space. Techniques named naive bayes (NB), AdaBoost, instance based learner, and J48 were used. However, it is tougher to deal with higher values of n-grams due to exponential possibilities. Thus, it is required to select smaller values of n for experimentation.

McLaughlin et al. [8] proposed a shallow convolutional neural network (CNN) to extract n-gram-like signatures from opcode sequences. Since the network is trained end-to-end jointly to learn appropriate features, thus it is not needed to consider millions of n-grams during training. They demonstrated their model's performance on the android operating system. Raff et al. [9] designed an end-to-end system to learn directly from the hexadecimal representation of executables. Their proposed architecture, MalConv, was based on the stacking of convolutional layers. But, the classification accuracy (96.4%) was lower compared to XGBoost [5]. Gibert et al. [10] represented the malicious program's content as an entropy stream, where each value represents the entropy of a small chunk of code in a specific location of the file. Categorization of the malware was based on the visual similarity between streams of entropy of the same family. The detector consisted of three-stage layers named convolutional, pooling, and activation. But, the classification accuracy (98.28%) is lower than XGBoost.

Zhang et al. [11] designed a malware detector based on multiple categories of static features named total lines of each section, operation code count, API, special symbols count, assembly (ASM) file pixel intensity feature, bytes file block size distribution, and bytes file n-gram. The detector was designed using XGBoost, ExtraTreeClassifier, and stacking method. But, the classification accuracy (99.74%) was lower than XGBoost [5]. Hassen and Chan [12] proposed a linear time function call graph vector representation based on clustering for malware classification. The above methodology reduced the time required to compute graph similarity. They combined graphical features with non-graphical features. However, the classification accuracy is lower than XGBoost [5]. Gibert et al. [13] visualized a malicious sample as a grayscale image due to its ability to capture minor changes while retaining global structure. They represented samples as 128 x 128 grayscale images and applied the CNN classifier to find similarities between images belonging to similar families. But, the classification accuracy (97.5%) is lower than XGBoost [5]. Gibert et al. [4] designed HYDRA, a novel framework to address malware detection and classification. They used feature types named API, bytes sequence, and opcode sequence. The classification algorithm was multimodal, which means that outputs from different networks were concatenated before sending to the softmax layer. However, the classification accuracy (99.75%) is lower than XGBoost [5].

III. Proposed Framework

Figure 1 represents the proposed framework for malware classification. The proposed framework is divided into six parts, namely: (a) Disassemble the samples using IDA-pro, (b) Extract API and opcodes trace from ASM files, (c) Apply n-gram on the API and opcode traces, (d) Calculate the frequencies of all n-gram tuples and apply TF-IDF, (e) Apply NB, logistic regression (LR), random forest (RF), support vector classifier (SVC) on API and opcode based datasets and (f) Select the best model and calculate prediction probabilities of samples for both API and opcode. The detailed description of the steps is described below.

A. Disassemble the samples using IDA Pro

Executables can be represented as a sequence of hexadecimal values for corresponding bytes of a binary file. The content of the binary file can be translated/reversed to assembly language. The above process is termed disassembling. We use IDA Pro for disassembling the malicious samples. Example-A row may look like: ”MOV BH,0 ;SET PAGE NO. FOR INT 10”.

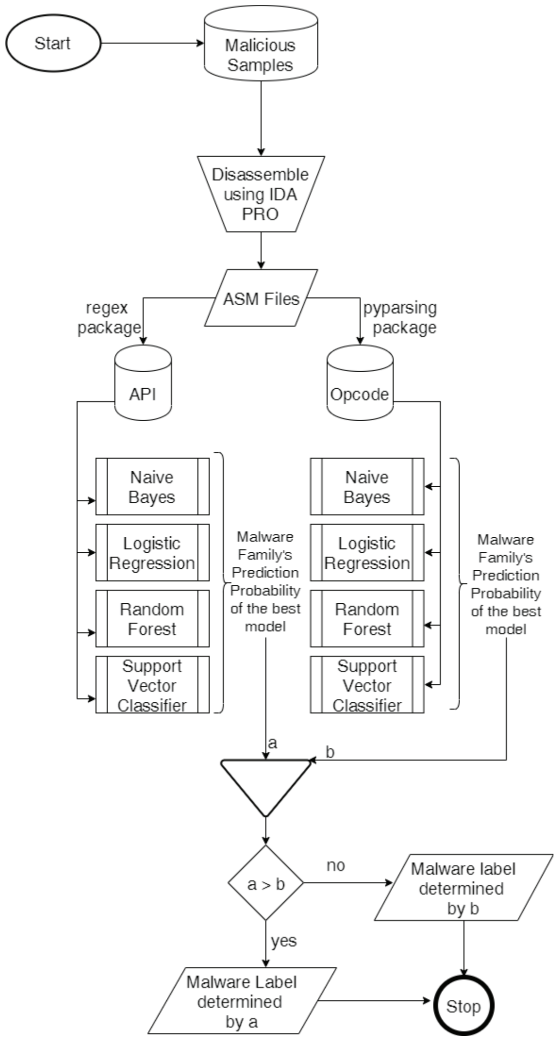
Start
Malicious Samples
Disassemble using IDA PRO
ASM Files
regex package
pyparsing package
API
Opcode
Naive Bayes
Logistic Regression
Random Forest
Support Vector Classifier
Malware Family's Prediction Probability of the best model
a
b
a > b
no
yes
Malware label determined by b
Malware Label determined by a
Stop

Fig. 1. Proposed framework for malware classification

B. Extract API and opcodes traces from ASM files

The assembly language source code contains the symbolic machine code of the executable. It consists of three types of statements:

1) **Instruction** defines the executable operation. The format is like [label] mnemonic [operands]. For example, in the case of instruction: (MOV TOTAL, 48), “MOV TOTAL” is the name of the instruction, and “48” is the parameter.
2) **Assembler** directives are commands not related to the processor instruction set.
3) **Macros** is a sequence of instructions that can be used anywhere in the program. Example - % macro name parameters <macro body> % endmacro.

First, we extract the API calls from the ASM files. In the case of packed malicious samples, only stubs of the import table are present. Thus, we searched for the dynamic link library (DLL) or function in the ASM file of the malware. The API calls or functions are prefixed with ’extrn’ in the ’.idata’ section in an ASM file. Upon extraction, we find numerous sets of API calls in the dataset. Since we will not require all of them for training, thus we consider those functions invoked at least twice in our training data. Finally, 4657 API calls are selected as the features of the dataset. Next, we extract the opcodes from the ASM files. The ’pyparsing’ package helps extract all the opcodes present in the ASM file. Example- Some opcodes extracted from ASM files are mov, push, call, add, mov, retn, etc.

C. Apply n-gram on the API and opcode traces

We apply a continuous n-gram methodology to the API and opcode traces. N-gram [14] based methods define an n-sized vector where the value of each vector is the count of its appearances in the trace file. In the case of API calls, n is frozen to one since there is no repetition of API calls in a single trace file. However, many opcodes are repeated within a single trace file. So, there is a provision for varying the range of n. We can alter the value of n from 1 to infinity. As the value of n increases, the number of solutions increases exponentially. After seeing the above problem, a minimal value of n should be selected, providing better performance among all values of n. Following that, we vary the value of n from 1 to 2 in the case of opcodes. Example- 1-gram API and opcode looks like [createfile, writefile, readfile] and [mov, retn] respectively whereas 2-gram opcode may look like [(mov,mov), (mov,retn)].

D. Calculate the frequencies of all n-gram tuples and apply TF-IDF

After the n-grams are fixed, we calculate the frequencies of all n-gram tuples within all trace files. In the case of API, a 1-gram dataset (1-gram API) is present. The values are 0 (absence of API call) or 1 (presence of API call). For opcodes, 1-gram (1-gram opcode) and 2-gram (2-gram opcode) datasets are present. There is much variation in the values of n-grams in the case of opcodes. To calculate the relevance of each n-gram in the dataset, TF-IDF [15] is used.

E. Apply NB, LR, RF, SVC on API and opcode based datasets

A total of 12 combinations, i.e., four techniques (NB, LR, RF, SVC) [15] and three datasets (1-gram API, 1-gram opcode, and 2-gram opcode) are tested. For NB, we use the multinomial function. In the case of LR, ’multi_class’ is set to multinomial, ’solver’ is set as lbfgs, and ’max_iter’ is set at 30000. At the same time, default hyperparameters are used for SVC and RF.

F. Select the best model and calculate prediction probabilities of samples for both API and opcode

We select the combination with the highest F1-score in the case of (4 techniques) x (1-gram API). Similarly, we choose the mix with the highest F1-score in the case of (4 techniques) x (1-gram opcode) + (4 techniques) x (2-gram opcode). Then, we calculated the malware prediction probability of the best combinations for API and opcode. If API prediction probability is higher than opcode, the final label is decided by API; otherwise, by opcode.

IV. Experimental Setup and Obtained Results

The experiment has a Windows 10 operating system, 64 GB RAM, 1TB Hard-Disk, Quadro RTX 5000 GPU, and anaconda installed on the system. The malicious samples used for the experiment are hosted on Kaggle and named Microsoft Malware Classification Challenge[7]. Table I represents the distribution of training and testing samples' families in the dataset. The number of families is inconsistent in the case of training and testing samples. But, total training and testing samples are almost the same. The metrics considered for the performance analysis of the models are precision, recall, F1-score, accuracy, area under curve (AUC), Matthews Correlation Coefficient (MCC), and Cohen's Kappa Coefficient (CKC). Precision is evaluated by dividing true positives by (true positives + false positives). The recall is evaluated by dividing the number of true positives by (true positives + false negatives). F1-score is the harmonic mean of precision and recall. AUC is considered the average of good and bad F1-scores evaluated at various thresholds. Since there is a class imbalance in the dataset, F1-score is preferred compared to the AUC. MCC can be defined for confusion matrix C for K classes. Equation 1 is used for evaluating MCC.

TABLE I
Malware dataset

Family Name	Type	Train samples	Test samples
Ramnit	Worm	1506	1515
Lollipop	Adware	2468	2465
Kelihos_ver3	Backdoor	2596	2600
Vundo	Trojan	235	241
Simda	Backdoor	31	39
Tracur	TrojanDownloader	294	289
Kelihos_ver3	Backdoor	62	64
Obfuscation.ACY	Obfuscated	1161	1171
Gatak	Backdoor	1010	1007
Total samples		9363	9391

$$MCC = \frac{cXs - \sum_k^K p_k X t_k}{\sqrt{(s^2 - \sum_k^K p_k^2)X(s^2 - \sum_k^K t_k^2)}} \quad (1)$$

where, t_k is the number of times class k truly occurred, p_k is the number of times class k is predicted, c is the total number of samples correctly predicted and s is the total number of samples. CKC is used for measuring inter-rater reliability for categorical items.

A. API based performance

Table II shows the values of the performance metrics for API based dataset. The highest accuracy (99%) is obtained for LR and RF models. Similarly, AUC (100%) and precision (99%) are highest for LR and RF. The highest recall (94%) is for LR, which is higher than the second best recall by 2%. The F1-score is highest (96%) for LR as its precision and recall are equal to or higher than other models. MCC and CKC are highest and similar (98.3%) for LR and RF. After observing all these statistics, we can infer that LR is the best model with the highest F1-score (96%), MCC (98.3%), and CKC (98.3%). Figure 2 depicts the confusion matrix for LR on API 1-gram dataset. The major source of errors (23.1%) is the misclassification of samples belonging to the simda family. Particularly, 9 out of 39 samples are misclassified. The least error (0.2%) comes from the samples belonging to the kelihos_ver3 family. Particularly, 5 out of 2600 samples are misclassified.

TABLE II
API 1-gram performance parameters

Models	Acc	AUC	Prec	Rec	F1	MCC	CKC
NB	92	92	84	87	85	90	90
LR	**99**	**100**	**99**	**94**	**96**	**98.3**	**98.3**
RF	**99**	**100**	**99**	92	95	**98.3**	**98.3**
SVC	98	99	98	88	92	97.5	97.5

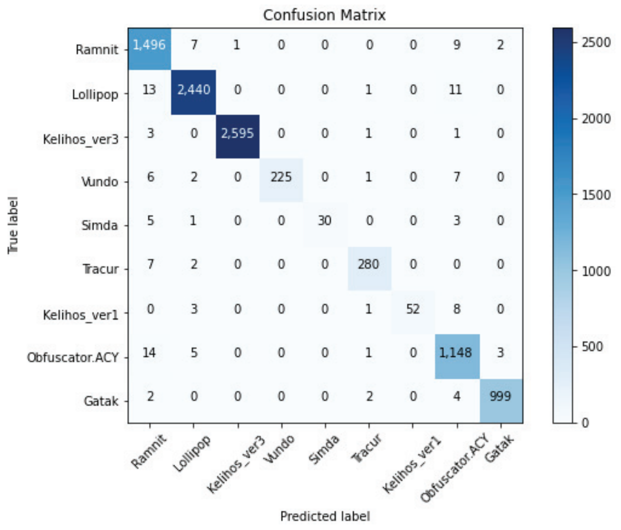

Fig. 2. Confusion matrix for LR on API 1-gram dataset

B. Opcode based performance

Table III shows the values of the performance metrics for the opcode 1-gram based dataset. The highest accuracy (99%) is obtained for the RF model. Similarly, AUC (100%), precision (99%) and recall (99%) are highest for RF. RF's F1-score is highest (98%) as its precision and recall are higher than other models. The MCC and CKC values of RF are equal to 99.2%, which is the highest among other models. After observing all these statistics, we can infer that RF is the best model with the highest F1-score (98%), MCC (99.2%), and CKC (99.2%). Figure 3 depicts the confusion matrix for RF on opcode 1-gram dataset. The major source of errors (10.3%) is the misclassification of samples belonging to the simda family. Particularly, 4 out of 39 samples are misclassified. The least error (0.1%) comes from the samples belonging to the kelihos_ver3 family. Particularly, 1 out of 2600 samples are

[7]https://www.kaggle.com/competitions/malware-classification/data

TABLE III
OPCODE 1-GRAM PERFORMANCE PARAMETERS

Models	Acc	AUC	Prec	Rec	F1	MCC	CKC
NB	70	95	56	40	41	63.6	60.2
LR	93	98	76	70	71	91.1	91
RF	**99**	**100**	**99**	**98**	**98**	**99.2**	**99.2**
SVC	97	99	97	94	95	96.2	96.2

misclassified.

Table IV presents the values of the performance metrics for

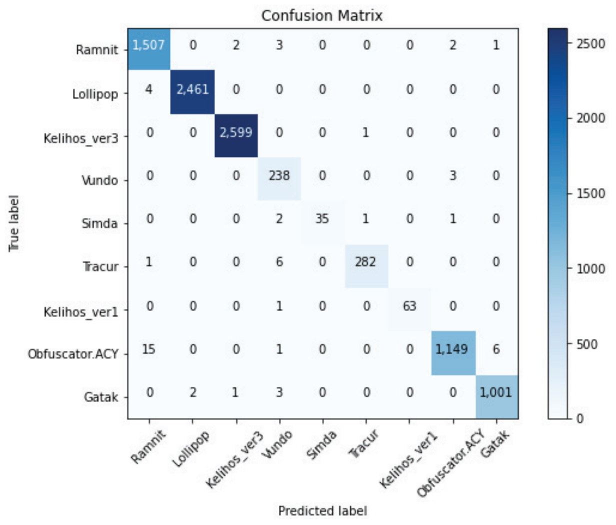

Fig. 3. Confusion matrix for RF on opcode 1-gram dataset

the opcode 2-gram based dataset. The highest accuracy (99%) is obtained for RF and SVC models. The highest AUC (100%) is obtained for LR, RF and SVC models. The precision of SVC is 99%, which is higher by 1% compared to the second best precision. The highest recall (99%) is for RF and higher by 1% compared to the second best recall. F1-score is the highest (98%) for RF and SVC as their precision and recall are higher than other models. On the other hand, the MCC and CKC values of RF are equal to 99.3%, which is the highest among other models. After observing all these statistics, we can infer that RF is the best model with the highest F1-score (98%), MCC (99.3%), and CKC (99.3%). Figure 4 depicts the

TABLE IV
OPCODE 2-GRAM PERFORMANCE PARAMETERS

Models	Acc	AUC	Prec	Rec	F1	MCC	CKC
NB	77	95	58	47	47	71.9	69.5
LR	97	**100**	97	83	86	96.2	96.2
RF	**99**	**100**	98	**99**	**98**	**99.3**	**99.3**
SVC	**99**	**100**	**99**	98	**98**	98.4	98.4

confusion matrix for RF on the opcode 2-gram dataset. The primary source of errors (7.7%) comes from misclassifying samples belonging to the simda family. Remarkably, 3 out of 39 samples are misclassified. The slightest error (0%) comes

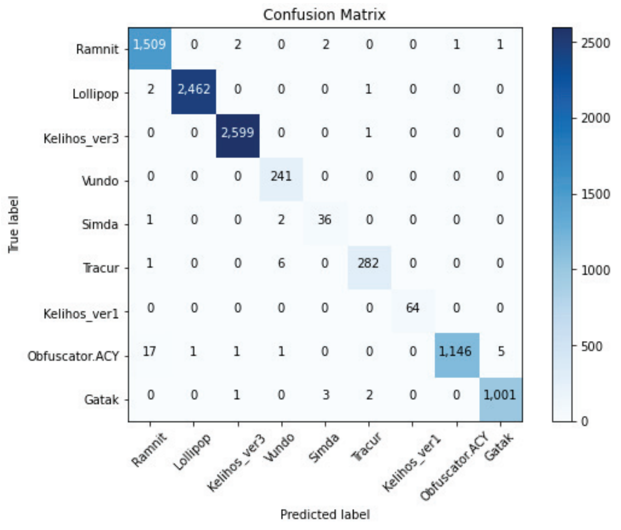

Fig. 4. Confusion matrix for RF on opcode 2-gram dataset

from the samples belonging to the vundo and kelihos_ver1 family. Remarkably, none of the samples are misclassified.

After comparing the best classifiers from opcode 1-gram and opcode 2-gram, i.e., RF and RF, we find that RF opcode 2-gram is best. Thus, the final framework consists of API 1-gram and opcode 2-gram.

C. *Proposed framework performance*

Table V shows the values of the performance metrics for our proposed framework. We enhance accuracy by 0.4% and F1-score by 1% compared to the second-best (RF, opcode 2-gram). The other parameters are similar to the best ones. F1-score, MCC, and CKC are the critical parameters for imbalanced dataset multi-classification problems. Our proposed model improves the parameters mentioned above compared to the second-best available results (RF: Table IV). Figure 5

TABLE V
PROPOSED FRAMEWORK PERFORMANCE PARAMETER

Models	Acc	AUC	Prec	Rec	F1	MCC	CKC
Proposed	99.4	100	99	98	99	99.3	99.3

depicts the confusion matrix for our proposed framework. The primary source of errors (7.7%) comes from misclassifying samples belonging to the simda family. 3 out of 39 samples are misclassified. On the other hand, the minor error (0%) comes from the samples belonging to the kelihos_ver1 family. Here, none of the samples are misclassified.

V. COMPARISON OF OUR WORK WITH PRESENT STATE-OF-THE-ART TECHNIQUES

Table VI presents the comparison of our proposed model with the existing state-of-the-art techniques. Gibert et al. [13] represented features as 128 X 128 grayscale image and applied CNN. But, the F1-score is 5% lower than our proposed model. Ahmadi et al. [5] represented features using a local

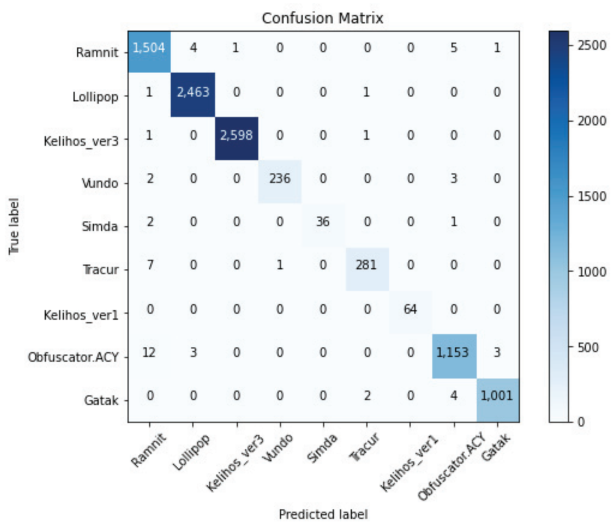

Fig. 5. Confusion matrix for proposed framework

binary pattern and applied XGBoost. The F1-score obtained using the classifier is 3.7% lower than our proposed model. Gibert et al. [10] considered the entropy of the malicious samples and applied CNN. The overall F1-score, as well as accuracy, are lower than our proposed model. McLaughlin et al. [8] considered opcode sequence for creating features and applied CNN. However, F1-score is still 1.57% lower than the proposed model. Gibert et al. [4] used APIs, bytes sequence, and opcode sequence for features and applied a multimodal deep neural network. The F1-score (98.5%) is the second-best compared to all other approaches. Raff et al. [9] used bytes sequence and applied the MalConv model for malware classification. But, the F1-score (88.94%) is the lowest among all the approaches. The F1-score is the lowest as they designed a malware classifier that works well for the higher number of instances (around 200,000). But, the number of training samples in the Microsoft Malware Dataset is 9363 only. Zhang et al. [11] used seven feature types and applied XGBoost for classification. However, the F1-score is 0.6% lower than our proposed work. Overall, our proposed model has the highest accuracy and F1-score.

TABLE VI
COMPARISON OF OUR PROPOSED APPROACH WITH STATE-OF-THE-ART APPROACHES

Approach	Feature type	Classifier	Acc	F1
Gibert et al. [13]	grayscale	CNN	97.5	94
Ahmadi et al. [5]	LBP	XGBoost	97.24	95.3
Gibert et al. [10]	entropy	CNN	98.28	96.36
McLaughlin et al. [8]	opcode	CNN	99.03	97.43
Gibert et al. [4]	multiple	DNN	98.7	98.5
Raff et al. [9]	bytes	MalConv	96.41	88.94
Zhang et al. [11]	multiple	ensemble	98.7	98.4
Proposed	multiple	ensemble	**99.4**	**99**

VI. Conclusions and Future Work

This paper presented a novel malware classifier that combined two feature types: API and opcode. We used interactive disassembler pro (IDA Pro) for disassembling the malicious samples. The extracted assembly (ASM) files were used for extracting the API and opcode sequences. API based dataset was organized into 1-gram representation. Similarly, opcode based dataset was organized into 1-gram and 2-gram representations. Then, we applied a pool of four classifiers to the three datasets. LR-1-gram was the best model for the API based dataset, while RF-2-gram was the best model for the opcode based dataset. At last, we predicted the final label by comparing the malware family prediction probability. If the API classifier's malware family prediction probability is higher than the opcode classifier, the label was decided by the API classifier. Else, it was decided by opcode. Overall, the prediction accuracy of the final framework was 99.4%, F1-score was 99%, Matthews Correlation Coefficient was 99.3%, and Cohen's Kappa Coefficient was 99.3%.
In the future, we will implement some new architectures, such as a deep neural network, to classify malware from API as its performance was lower than opcode. Our last line of research could be the study of explainable artificial intelligence techniques to interpret the results of machine learning models.

References

[1] K. Chandrasekar, G. Cleary, O. Cox, H. Lau, B. Nahorney, B. O. Gorman, D. O'Brien, S. Wallace, P. Wood, and C. Wueest, "Internet security threat report," *Symantec Corp*, vol. 22, p. 38, 2017.
[2] G. Cleary, M. Corpin, O. Cox, H. Lau, B. Nahorney, D. O'Brien, B. O'Gorman, J.-P. Power, S. Wallace, P. Wood *et al.*, "Internet security threat report," *Symantec Corporation, California, USA, Tech. Rep.*, 2018.
[3] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.
[4] D. Gibert, C. Mateu, and J. Planes, "Hydra: A multimodal deep learning framework for malware classification," *Computers & Security*, vol. 95, p. 101873, 2020.
[5] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. 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*, 2016, pp. 183–194.
[6] S. M. Bidoki, S. Jalili, and A. Tajoddin, "Pbmmd: A novel policy based multi-process malware detection," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 57–70, 2017.
[7] S. Jain and Y. K. Meena, "Byte level n–gram analysis for malware detection," in *International Conference on Information Processing*. Springer, 2011, pp. 51–59.
[8] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé *et al.*, "Deep android malware detection," in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 301–308.
[9] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[10] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
[11] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using multi-features and ensemble learning method for imbalanced malware classification," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 965–973.
[12] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 239–248.

[13] D. Gibert, C. Mateu, and J. Planes, “An end-to-end deep learning architecture for classification of malware’s binary content,” in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 383–391.

[14] W. B. Cavnar, J. M. Trenkle *et al.*, “N-gram-based text categorization,” in *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, vol. 161175. Citeseer, 1994.

[15] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. ” O’Reilly Media, Inc.”, 2019.