A Novel Image-Based Malware Classification Model Using Deep Learning

Yongkang Jiang, Shenghong Li, Yue Wu(✉), and Futai Zou

School of Electronic, Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
{jiangyongkang,shli,wuyue,zoufutai}@sjtu.edu.cn

Abstract. Nowadays, the vast volume of data which needs to be evaluated potentially malicious is becoming one of the major challenges of antivirus products. In this paper, we propose a novel image-based malware classification model using deep learning to counter large-scale malware analysis. The model includes a malware embedding method called YongImage which maps instruction-level information and disassembly metadata generated by IDA disassembler tool into an image vector, and a deep neural network named malVecNet which has simpler structure and faster convergence rate.

Our proposed YongImage converts malware analysis tasks into image classification problems, which do not rely on domain knowledge and complex feature extraction. Meanwhile, we use the thought of sentence-level classification in Natural Language Processing to establish and optimize our malVecNet. Compared to previous work, malVecNet has better theoretical interpretability and can be trained more effectively.

We use 10-fold cross-validation on Microsoft malware classification challenge dataset to evaluate our model. The results demonstrate that our model can achieve 99.49% accuracy with 0.022 log loss. Although our scheme is less precise than the winner's, it makes an orders-of-magnitude performance boost. Compared with other related work, our model also outperforms most of them.

Keywords: Malware · Embedding · Classification · Deep learning

1 Introduction

Malware is a term for all software that intend to cause harm or inflict damage on computer systems. Recently, with introducing of polymorphic and metamorphic techniques, malicious software gets explosive growth on both quality and quantity. Malware classification has always been a concerned field in recent decade, which is an issue of giving a malicious sample i, calculating a family label j from knowledge base. Hence, malware classification can associate a fresh variant to a known family, which is meaningful to malware detection.

The research of Microsoft [11] indicates that the vast volume of data which needs to be detected is becoming one of the major challenges of anti-malware

T. Gedeon et al. (Eds.): ICONIP 2019, LNCS 11954, pp. 150–161, 2019.
https://doi.org/10.1007/978-3-030-36711-4_14

companies. Traditional signature-based and behavior-based malware analysis techniques are difficult to meet the demand, a more effective method is needed [19]. Deep learning is a good choice, under the gpu acceleration, a model can be easily trained and the detection is even more efficient. In addition, inspired by the excellent performance of convolution neural network in image field, we decide to explore a simpler model, converting malware classification tasks into image classification problems.

In fact, recent research of malware analysis, both static and dynamic, is moving from traditional aspects to deep learning. Ronen *et al.* [15] make a comparison between research papers using Microsoft malware classification challenge dataset (BIG2015). The results suggest that none of 12 papers in 2016 introducing deep learning, but 5 of 17 papers in 2017. Although most of them still rely on solid domain knowledge, researchers are exploring an end-to-end model to extract and fuse malware features automatically.

Nataraj *et al.* [13] propose first malware embedding method called NatarajImage based on binary file in 2011. Although NatarajImage is believed vulnerable to obfuscation and packing techniques, their work has been followed by many others [1,7,8,18]. Andrew *et al.* [2] propose another malware embedding method named AndrewImage based on disassembly file at Black Hat conference 2015. As far as we know, AndrewImage has not been introduced in any research paper. Compared with NatarajImage, AndrewImage embeds instruction-level information, which has better robustness and interpretability. Unfortunately, AndrewImage uses so much zero padding that the accuracy still remains a challenge.

To summarize, this paper has the following contributions. 1. We propose a novel image-based malware classification model including a malware embedding method called YongImage and a simple deep neural network named malVecNet. YongImage directly embeds hexadecimal instruction and other metadata into vector space. MalVecNet has better theoretical interpretability and can be trained more effectively. Our model converts malware analysis tasks into image classification problems that do not rely on domain knowledge and time-consuming feature extraction. 2. We successfully train the model on Microsoft malware classification challenge dataset, the results indicate that our model outperforms most of the related work. As far as we know, it is the state-of-the-art solution for large-scale malware classification tasks. 3. We make the code for malware embedding and training based on the model described in this paper, now is available in github.[1]

2 Malware Embedding

Formally, malware embedding maps malicious software into a vector space, helps learning algorithms to obtain better performance in malware analysis tasks. Similar to word embedding [12] in Natural Language Processing (NLP), this choice has several advantages—simplicity, effectiveness and the observation that

[1] https://github.com/jyker?tab=repositories.

some image-based models trained on huge malware dataset outperform most of traditional signature-based and behavior-based methods. Differently, malware embedding now focuses more on how to choose atomic units.

2.1 NatarajImage and AndrewImage

NatarajImage [13], as shown in Fig. 1, chooses 8-bit malware binary as atomic units, maps whole Portable Executable (PE) file or only *.text* section into a gray image vector.

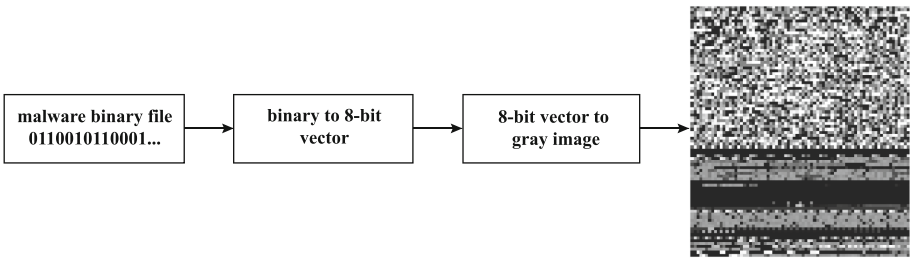


Fig. 1. NatarajImage embedding process

However, even without considering obfuscation and packing techniques, Ahmadi *et al.* [1] find that the texture pattern of NatarajImage in different malware families may be exactly similar. Therefore, models based on NatarajImage are vulnerable to attackers.

Differently, AndrewImage [2], as shown in Fig. 2, chooses hexadecimal instruction in disassembly file as atomic units, embeds malware instruction into a black-white image vector.

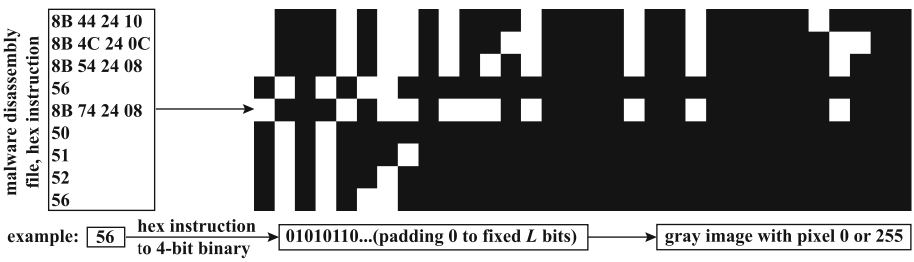


Fig. 2. AndrewImage embedding process

In fact, AndrewImage has excellent semantic features —one line instruction, one row vector. Unfortunately, it is also this excellent visual interpretability that uses a large percentage of invalid zero padding, which makes output image vector too large to train, and high accuracy still remains a challenge.

2.2 YongImage

Inspired by previous work, we propose YongImage. As Fig. 3 shows, a PE malware disassembly file generated by IDA Pro[2] contains two aspects of information—hexadecimal instruction, and corresponding metadata, *i.e.* section name, address of instruction, opcode and operand.

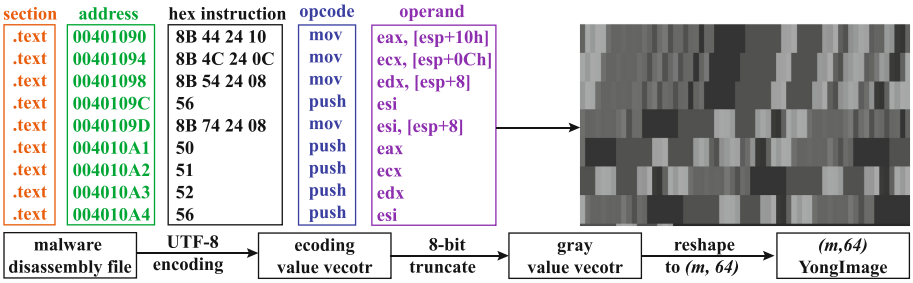


Fig. 3. YongImage embedding process

We embed these information as following steps: 1. Encode malware disassembly file with UTF-8 [20]. 2. Obtain gray vector by truncating each encoded value from a high order to 8-bit. 3. Reshape the gray vector to $(m, 64)$. Intuitively, the visual interpretability of YongImage is not as well as AndrewImage. In fact, YongImage retains instruction-level interpretability by reshaping the gray vector to $(m, 64)$.

Why (m, 64). Firstly, Let us prove the optimal padding length in AndrewImage is $L = 64$.

The Intel 64 and IA-32 architectures instruction encodings are subsets of the format shown in Fig. 4, which specifies the maximum length of an instruction is 15 *bytes*, in general, it does not exceed 11 *bytes* [5]. For example, the hexadecimal instruction **{8B 44 24 10}** in Fig. 2 is only 4 *bytes*.

Instruction Prefixes	Opcode	ModR/M	SIB	Displacement	Immediate
Prefixes of 1 byte each (optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes or none	Immediate data of 1, 2, or 4 bytes or none

Fig. 4. Intel 64 and IA-32 architectures instruction format [5]

[2] https://www.hex-rays.com/products/ida/.

Therefore, the first idea is to pad binary digits to 120-bit to cover all instructions or just truncate binary digits to 88-bit vector to support most. However, in that case, the output vector is too large to train.

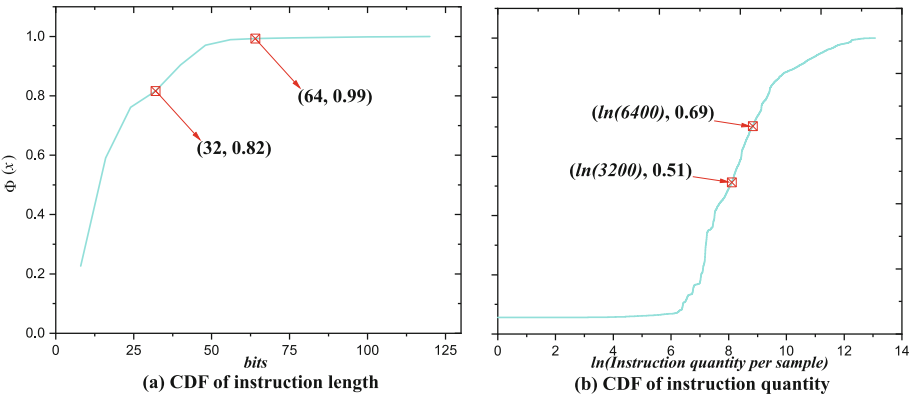


Fig. 5. The cumulative distribution function of instruction in BIG2015 dataset

By analyzing samples in Microsoft malware classification challenge dataset (BIG2015) [15], we obtain the cumulative distribution function (CDF) of instruction length in samples, as shown in Fig. 5(a), which indicates that 99% of instructions do not exceed 64 *bits* and 82% of instructions do not exceed 32 *bits*. Therefore, $L = 64$ is chosen to cover almost all instructions while maintaining smaller vector size. Differently, YongImage does not truncate each line vector to 64-bit, as an alternative, we reshape the vector to $(m, 64)$. Since 64-bit is almost sufficient for a line of disassembly file encoded by UTF-8, m can be approximated as number of instruction lines, *i.e.* instruction quantity.

Another problem of YongImage is how many lines of instructions should be embedded, *i.e.* how to choose m. Certainly, the larger m can lead to better accuracy without considering performance. However, when the model reaches a certain accuracy, m should be as small as possible. Figure 5(b) indicates that the instruction quantity in disassembly file varies greatly, 50% of files contain no more than 3200 instructions, and 69% of files contain no more than 6400. In our experiment, 3200 and 6400 are two candidate values of m.

3 Model Definition

Kim *et al.* [9] propose a simple novel convolution neural network (CNN) architecture with little hyperparameter tuning and static vectors, which achieves excellent results. Inspired by their work, we propose a variant architecture named malVectNet, as shown in Fig. 6.

Malware embedding outputs an image vector with size $(m, 64)$, where m represents the number of instructions. Channel transformation is designed to

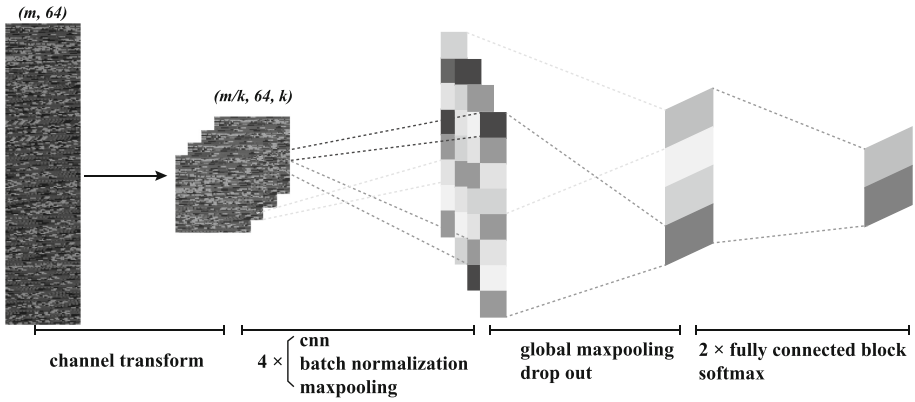


Fig. 6. Image-Based malware classification model architecture

turn vector into a new shape of $(\frac{m}{k}, 64, k)$, in which k represents the number of channels.

Next, we use a special value $k = 1$ to illustrate our malVecNet.

First, let $S_j \in \mathbb{R}^{64}$ be the 64-dimensional instruction vector corresponding to the j-th instruction in m. Then, a malware sample X_i is represented as:

$$X_i = [S_1, S_2, ..., S_j, ..., S_m]. \tag{1}$$

Each convolution layer includes several filters $w \in \mathbb{R}^{hc}$, which apply a window with size (h, c). For instance, $c_{i,t}$ is a vector generated from a window $X_{i:i+h}(t : t + c)$ by

$$c_{i,t} = w \cdot X_{i:i+h}(t : t + c) + b, \tag{2}$$

where $b \in \mathbb{R}$ is bias. When a row convolution is finished, we obtain a new row feature vector as follows:

$$c_i = [c_{i,1}, c_{i,2}, ..., c_{i,64}], \tag{3}$$

Similarly, when whole convolution is completed, a new abstract instruction vector is generated, *i.e.*

$$c = [c_1, c_2, ..., c_i, ..., c_m]. \tag{4}$$

Then, batch normalization (BN) [6] is applied on c. Details of BN on mini-batch are in Algorithm 1. Recent research has shown that BN can smooth the objective function [16], which is significant to accelerate deep neural network.

After that, we use a non-linear activation function f and max-pooling on e_i, which aims to reduce the dimension of the feature vector.

$$e = maxpooling(f([e_1, e_2, ..., e_i, ..., e_m])); e_i = BN_{\beta,\gamma}(c_i) \tag{5}$$

So far, we have described the process of one CNN block in malVecNet, which uses four similar CNN blocks. Therefore, the input of global max-pooling is a normalized, activated and pooled feature vector, *i.e.*

Algorithm 1. The algorithm of batch normalization on mini-batch

Input: c value on each mini-batch $\varphi = \{c_{1...n}\}$
1: parameters to learn γ, β
Output: $e_i = BN_{\beta,\gamma}(c_i)$
2: calculate mean of mini-batch $\mu \leftarrow \frac{1}{n}\sum_{i=1}^{n} c_i$
3: calculate variance of mini-batch $\sigma_{\varphi}^2 \leftarrow \frac{1}{n}\sum_{i=1}^{n}(c_i - \mu_{\varphi})^2$
4: standardize $\hat{c_i} \leftarrow \frac{c_i - \mu_{\varphi}}{\sqrt{\sigma^2 + \varepsilon}}$
5: scale and shift $e_i \leftarrow \gamma\hat{c_i} + \beta \equiv BN_{\beta,\gamma}(c_i)$

$$e = [e_1, e_2, ..., e_g]. \tag{6}$$

where g is determined by the specific parameters of the before layers. In order to preserve the most important features (one with highest value) while reducing the dimension of the final feature vector, we apply global max-pooling on e and take $\hat{e} = max\{e\}$ as the final instruction-level features vector.

Finally, we use two fully-connected blocks and a softmax layer to get:

$$y = [y_1, y_2, ..., y_n]. \tag{7}$$

in which y_i is the probability of malware family i, n represents the number of malware families.

We use the idea of sentence-level classification to construct the entire model. In theory, it is more suitable for instruction-level malware embedding methods, such as AndrewImage and YongImage.

4 Experiments and Results

4.1 Dataset

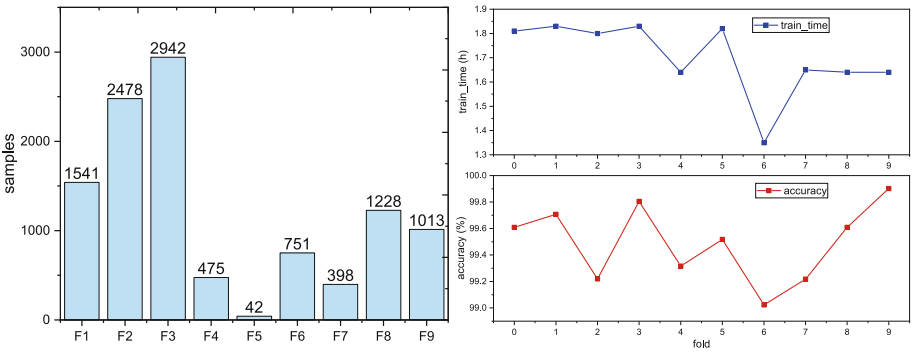
BIG2015 [15] contains 21741 malware samples of 9 different families, *i.e.* Ramnit(F1), Lollipop(F2), Kelihos_ver3(F3), Vundo(F4), Simda(F5), Tracur(F6), Kelihos_ver1(F7), Obfuscator.ACY(F8) and Gatak(F9).

Since only 10868 training samples in BIG2015 is labeled, we choose this part as experimental dataset. As shown in Fig. 7(a), the dataset is extremely unbalanced. Therefore, we combine the following two methods to eliminate the impact of this unbalance on the model. Firstly, F4, F5 and F7 are randomly up-sampled to 500. Secondly, a loss function weight that is inversely proportional to the class frequency is set in the input data [10].

4.2 Platform and Environment

We evaluate malVecNet on the platform environment presented in Table 1. More model details can be found in our github.[3]

[3] https://github.com/jyker/zklearn.



(a) Family distribution in the dataset

(b) YongImage 10-fold cross-validation on the dataset

Fig. 7. The distribution of dataset and cross-validation results

Table 1. The platform environment of experiment

Platform	Content
Hardware	2 × GeForce GTX 1080 @ 8 GB
	1 × E5-2630 v3 @ 2.40 GHz
	32 GB of memory
Software	CentOS 7.4
	python 3.6.5
	tensorflow-gpu 1.8.0
	Keras 2.2.0
	scikit-learn 0.19.1
	numpy 1.14.3
	Pillow 5.1.0

Loss function of the model is cross entropy, which is defined in Eq. (8),

$$loss = -\frac{1}{M}\sum_{i}^{M}\sum_{j}^{N} y_{ij}\log p_{ij}. \tag{8}$$

where M is the number of samples in min-batch, N is the number of malware classes, y_{ij} is 1 if sample i is in class j, otherwise, y_{ij} is 0, and p_{ij} is the predicated probability of sample i in class j.

At the same time, we choose accuracy, precision, recall and f1-score[4] to evaluate the performance of our model.

[4] https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics.

4.3 Hyperparameter

In this section, we begin to discuss hyperparameters in our model. Firstly, it is the instruction quantity m. Experiment results shown in Table 2 indicate YongImage outperforms AndrewImage on all metrics regardless of m. In particular, when m reaches 6400, the accuracy of YongImage is increased, and AndrewImage decreased. One potential reason is that as m increases, the large number of invalid zero padding in AndrewImage causes more interference.

Intuitively, a larger m covers more instruction information, which should achieve a higher accuracy. In fact, when m increases to 6400, the accuracy of YongImage is only slightly improved, however the training time almost increases by half. Therefore, our model takes $m = 3200$.

Table 2. The impact of instruction quantity m

Method	m	Accuracy (%)	Precision (%)	Recall (%)	Train_time (h)
AndrewImage	3200	97.87	98.08	97.87	1.95
	6400	96.16	97.01	96.16	3.31
YongImage	3200	99.49	99.51	99.51	1.70
	6400	99.55	99.57	99.56	3.23
Note: BIG2015 tenfold cross-validation results; k = 1					

Second hyperparameter is k, which represents the initial number of channels. In fact, the initial idea is to analyze the correlation between instructions by stacking k instructions in the channel direction to obtain higher model accuracy. However, the experimental results in Table 3 indicate that this design is only beneficial to accelerate model training. In order to achieve better accuracy, we finally choose $k = 1$.

Table 3. The impact of channel parameter k

Method	k	Accuracy (%)	Precision (%)	Recall (%)	Train_time (h)
YongImage	1	99.49	99.51	99.51	1.70
	4	98.87	98.89	98.87	1.01
	8	98.54	98.57	98.54	0.69
Note: BIG2015 10-fold cross-validation results; m = 3200					

We use tenfold cross-validation on BIG2015 to evaluate the above hyperparameters. Particularly, the detail results of YongImage with $k = 1$ and $m = 3200$ is shown in Fig. 7(b), where the average accuracy is 99.49%, and the average training time is 1.70 h.

4.4 Comparison with Other Work

In this part, we compare malVecNet with several methods that have performed well on BIG2015 in recent years. Certainly, due to differences in experimental platforms, time metric is only a certain degree of reference.

The results in Table 4, suggest that only the solutions of Kaggle Winner [17] (BIG2015 winner) and Ahmadi [1] are slightly more accurate than our malVecNet. However, they all rely on time and labor consuming feature engineering, which is inefficient during both training and detecting phases. Hence, compared with them, malVecNet makes an orders-of-magnitude performance boost.

Garcia *et al.* [4] introduce random forest based on NatarajImage to classify variant malware. Unfortunately, it is believed that NatarajImage is vulnerable to obfuscation and packing techniques, so their solution is less robust.

Raff *et al.* [14] propose a novel valid distance metric named Lempel-Ziv Jaccard Distance (LZJD) to classify malware in raw data, which obtains greater performance improvement than Normalized Compression Distance (NCD). However, models combining distance metrics with clustering algorithm are easy to train but time-consuming to detect.

Drew *et al.* [3] introduce strand gen sequence to malware classification, which achieves 98.89% accuracy and requires only 0.75h to train. Unfortunately, the detection time of strand gen sequence classifier is still too long for large-scale malware.

In fact, only the method of Yan *et al.* [18], which stacks VGG (based on NatarajImage), LSTM (based on opcode), achieves similar performance as our malVecNet. However, our model still has obvious advantages despite the differences in the experimental platform. Firstly, we use YongImage as the only input feature vector, model preprocessing and training are relatively simple, certainly, detection is faster. Secondly, benefit from instruction embedding and sentence-level modeling, our solution is more robust and interpretable.

Therefore, as far as we know, our malVecNet is the advanced solution for large-scale malware classification tasks.

Table 4. The comparison with other work

Methods	Accuracy (%)	Process (h)	Train (h)	Detect (s)
(2015) Kaggle Winner [17]	99.83	72	1	13649
(2016) Novel Features [1]	99.77	21.86	—	4096
(2016) Random Forest [4]	95.62	—	—	—
(2017) LZJD [14]	97.10	1.35	—	—
(2017) Strand Gene Sequence [3]	98.59	—	0.75	307.20
(2018) VGG LSTM [18]	99.36	—	2.91	30.72
malVecNet	99.49	0.22	1.70	4.79
Note: detection time is measured on 1024 samples.				

5 Conclusion

This research aims to explore a simple and practical model to convert malware classification tasks into image classification problems.

We propose a novel image-based malware classification model including a malware embedding method called YongImage and a simple deep neural network named malVecNet. YongImage directly embeds hexadecimal instruction and other metadata into vector space. MalVecNet has better theoretical interpretability and can be trained more effectively. Our model does not rely on solid domain knowledge and time-consuming feature extraction.

We successfully train the model on Microsoft malware classification challenge dataset, the results indicate that our model outperforms most of the related work. To the best of our knowledge, it is the state-of-the-art solution for large-scale malware classification tasks.

Acknowledgment. This work is supported by the 2019 Science and Technology Project of SGCC "Security Protection Technology of Embedded Components and Control Units in Power System Terminal, No.2019GW-12", National Key Research and Development Program of China (No. 2017YFB0802300), NSFC-Zhejiang Joint Fund for the Integration of Industrialization and Informatization (No. U1509219), and National Key Research and Development Program of China (No. 2018YFB0803500).

References

1. Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G.: 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, pp. 183–194. ACM (2016)
2. Andrew Davis, M.W.: Deep learning on disassembly data. Internet (2015). https://www.blackhat.com/docs/us-15/materials/us-15-Davis-Deep-Learning-On-Disassembly.pdf
3. Drew, J., Hahsler, M., Moore, T.: Polymorphic Malware detection using sequence classification methods and ensembles. EURASIP J. Inf. Secur. **2017**(1), 2 (2017)
4. Garcia, F.C.C., Muga, F.P.: Random forest for malware classification. Cryptography and Security (2016). arXiv
5. Intel: Intel® 64 and ia-32 architectures software developer's manual, volume 2: Instruction set reference. Internet, September 2016. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf
6. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
7. Kebede, T.M., Djaneye-Boundjou, O., Narayanan, B.N., Ralescu, A., Kapp, D.: Classification of malware programs using autoencoders based deep learning architecture and its application to the Microsoft Malware classification challenge (big 2015) dataset. In: 2017 IEEE National Aerospace and Electronics Conference (NAECON), pp. 70–75. IEEE (2017)

8. Kim, H.-J.: Image-based Malware classification using convolutional neural network. In: Park, J.J., Loia, V., Yi, G., Sung, Y. (eds.) CUTE/CSA -2017. LNEE, vol. 474, pp. 1352–1357. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7605-3_215
9. Kim, Y.: Convolutional neural networks for sentence classification. In: Empirical Methods in Natural Language Processing, pp. 1746–1751 (2014)
10. King, G., Zeng, L.: Logistic regression in rare events data. Polit. Anal. **9**(2), 137–163 (2001)
11. Microft: Sam cybersecurity engagement kit. Internet (2018). https://assets.microsoft.com/en-nz/cybersecurity-sam-engagement-kit.pdf
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
13. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.: Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, p. 4. ACM (2011)
14. Raff, E., Nicholas, C.: An alternative to NCD for large sequences, Lempel-Ziv Jaccard distance. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1007–1015. ACM (2017)
15. Ronen, R., Radu, M., Feuerstein, C.E., Yomtov, E., Ahmadi, M.: Microsoft Malware classification challenge. Cryptography and Security (2018). arXiv
16. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? In: Advances in Neural Information Processing Systems, pp. 2483–2493 (2018)
17. Xiaozhou Wang, J.L., Chen, X.: Microsoft Malware classification challenge (big2015): First place team: Say no to overfitting. Internet (2015). https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf
18. Yan, J., Qi, Y., Rao, Q.: Detecting Malware with an ensemble method based on deep neural network. Secur. Commun. Netw. **2018**, 16 (2018)
19. Ye, Y., Li, T., Adjeroh, D.A., Iyengar, S.S.: A survey on Malware detection using data mining techniques. ACM Comput. Surv. **50**(3), 41 (2017)
20. Yergeau, F.: UTF-8, a transformation format of ISO 10646. Technical report (2003)