Journal of Signal Processing Systems (2021) 93:265–273
https://doi.org/10.1007/s11265-020-01588-1

A Method for Windows Malware Detection Based on Deep Learning

Xiang Huang[1] · Li Ma[1] · Wenyin Yang[1] · Yong Zhong[1]

Received: 29 June 2020 / Revised: 27 July 2020 / Accepted: 5 August 2020 / Published online: 2 September 2020

Abstract
As the Internet rapidly develops, the types and quantity of malware continue to diversify and increase, and the technology of evading security software is becoming more and more advanced. This paper proposes a malware detection method based on deep learning, which combines malware visualization technology with convolutional neural network. The structure of neural network is based on VGG16 network. This paper proposes the hybrid visualization of malware, combining static and dynamic analysis. In hybrid visualization, we use the Cuckoo Sandbox to carry out dynamic analysis on the samples, convert the dynamic analysis results into a visualization image according to a designed algorithm, and train the neural network on static and hybrid visualization images. Finally, we test the performance of the malware detection method we propose, evaluating its effectiveness on detecting unknown malware.

Keywords Cybersecurity · Malware detection · Malware image · Convolutional neural network

1 Introduction

In recent years, the Internet continues to develop rapidly and it plays an increasingly important role in the general public's work and life. In the meantime, malicious software (malware) on the Internet is also proliferating, posing a great threat to the information security of Internet users. The earliest (and still widely used) malware detection technology was based on "signatures", which are simply code sequences in a malware binary that uniquely identify it [1]. To detect malware, signatures are added to an antivirus program, which then matches them against the files it is scanning. If a match is found in a certain file, the file is determined to be malicious. Signature-based detection has the advantage of not easily producing false positives, because signatures are derived manually by professional malware analysts. However, it has a major drawback: it is only capable of detecting known malware [2]. When a new piece of malware appears, it will take time for the signatures to be extracted and added to the antivirus programs, leaving the users unprotected for a certain period. To address this problem, researchers proposed heuristic detection [3]. Heuristic detection was designed as a means to detect unknown malware by spotting suspicious characteristics in a program. For example, if a program is protected by some rarely seen "packer", then it is determined to be suspicious. This technique can identify some new malware, at the cost of higher false positive rates.

In light of the shortcomings of signature-based and heuristic detection, researchers and cybersecurity companies have turned their attention to machine learning, in the hope of finding a new way to detect malware in a timely and accurate manner.

Researchers have attempted to use machine learning in malware detection. Zhang [4] uses Naïve Bayes to detect malware. Lu [5] and Ravi et al. [6] extract opcode sequences from PE files and use *n*-gram together with KNN, SVM, and Decision Tree to classify malware. Zhang and Yang [7] extract representative permissions in Android apps, assign different weights to permissions, and classify malware using Naïve Bayes. Yang et al. [8] extract the icon of PE files and detect malware by calculating image similarity on the icons.

Recently, deep learning has been gaining momentum. Researchers have tried to apply it to various subfields in computer science, such as data privacy and protection [9, 10], vulnerability detection [11], data compression [12], and above

✉ Li Ma
molly_917@163.com

Xiang Huang
hx0755@qq.com

Wenyin Yang
yangwenyin1982@163.com

Yong Zhong
zhongyong@fosu.edu.cn

[1] School of Electronic and Information Engineering, Foshan University, Foshan, China

all, malware detection. Since image classification using CNN has been intensively researched, it is natural to try to convert the problem of malware detection into one about image classification. To this end, the idea of malware visualization has been proposed. Currently, some explorations into malware visualization have been made. Nataraj et al. [13] propose a method that converts malware binaries into greyscale images, extracts texture features using GIST, and finally classifies malware using KNN. Kalash et al. [14] also convert malware binaries into greyscale images, but they use CNN to classify malware. Li [15] visualizes Android .dex files as pixel normalized RGB images that are largely blue in hue, and then implements detection by using CNN and KNN. Fu et al. [17] convert PE files into RGB images based on entropy, byte value and relative size, extract features from the images, and then classify malware using machine learning algorithms. In addition to the above visualization techniques, which use static features, Shaid and Maarof [18] propose a method that visualizes binaries based on their dynamic features, i.e., runtime behavior; they assign more noticeable colors to more dangerous API calls.

Based on previous work in the field, this paper proposes a malware detection method based on visualization and deep learning. It stands apart from existing methods because:

(1) It incorporates both static and dynamic features in the detection, making it more robust in face of analysis evasion techniques; and by experimentation, we can conclude that this hybrid approach performs better than using static features alone.
(2) It explores a new static visualization algorithm that utilizes byte frequency information, rather than the commonly used entropy of a PE section.
(3) It visualizes malware as RGB color images, making use of all three color channels (as opposed to only one channel in greyscale images) to encode more information from the original file.

This paper describes a malware detection method based on deep learning, and evaluates its performance by experimentation.

2 Malware Visualization

2.1 Overview

Based on empirical knowledge gained through analyzing large amounts of malware, researchers have found that similarities exist among variants in the same family. For example, [13] showed that malware binaries belonging to the same family are visually similar. This is true because code is often reused between variants [19]. On the other hand, binaries which are unrelated to each other usually differ widely in their visualization images, which makes it possible for a deep learning algorithm to spot the differences between malware and legitimate software (cleanware).

To be able to use neural networks in malware detection, we first have to transform the input into a vector form, i.e., vectorize it. There are mainly two classes of vectorization approaches: (1) feature-based approach, and (2) image-based approach. This paper adopts the image-based approach, which is also known as malware visualization.

2.2 Static Visualization

In static visualization, we visualize malware as an RGB color image, based on static features. Each pixel in an RGB image has three channels: red, green, and blue. RGB images can accommodate (in the same image size) more information from the original file than greyscale images, thus helping malware detection.

We use the greyscale values used in [13] as the values for the blue channel in our RGB image, and then encode byte frequency information into the red channel. Byte frequency is directly related to the information entropy, as seen in the following definition of entropy.

$$H(X) = -\sum_{i=1}^{N} p(i)\log_2 p(i) \tag{1}$$

In eq. (1), $p(i)$ means the probability of the ith symbol (byte value) in event X's series of N symbols. A high entropy value tends to indicate the presence of encryption or packers [20]. In [17], malware is visualized based on information entropy, but its approach is to calculate the entropy of a PE section. Our approach uses byte frequency information, which is applicable to both PE and non-PE executables. The green channel of the image is unused for the moment and is filled with zeroes.

To statically visualize a piece of malware, its binary is first read in, and split into a vector of 8-bit unsigned binary integers. Then, convert the 8-bit unsigned integers in the vector into decimal values. The output of this step is known as a decimal vector. Next, construct a count vector based on the occurrence of numbers in the decimal vector (byte frequency). For example, a count vector <24, 32, … > means that in the decimal vector, 24 of its components are 0, 32 of its components are 1, etc. Now we are ready to construct a pixel vector, in which each component is a 3-tuple representing one RGB pixel. The blue channel value of the ith component is equal to the value of the ith component in the decimal vector. The red channel value of the ith component is related to the count vector, as seen in eqs. (2)–(5). The green channel is unused and is set to zero for all components. Once the pixel vector is constructed, reshape it into a pixel matrix of width 512, and save it as an image. Finally, scale the image to 512×512

Table 1 Api category-color mapping table.

API Category	RGB value
Notification	(0,0,128)
Certificate	(0,0,255)
Crypto	(255,255,128)
Exception	(0,64,0)
File	(255,128,0)
IExplore	(128,64,0)
Misc	(64,128,128)
NetAPI	(255,0,128)
Network	(128,0,128)
OLE	(0,255,0)
Process	(255,0,0)
Registry	(128,0,0)
Resource	(128,255,255)
Services	(255,128,128)
Synchronisation	(192,192,192)
System	(128,64,0)
UI	(255,255,255)

pixels in order to be fed into a neural network.

$$\text{atanNorm}(\boldsymbol{x}) = \arctan(\boldsymbol{x}) \times \frac{2}{\pi} + 1 \tag{2}$$

$$\text{minMaxNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \tag{3}$$

$$\boldsymbol{\beta} = \lfloor \text{minMaxNorm}(\text{atanNorm}(\boldsymbol{\alpha})) \times (-255) + 255 \rfloor \tag{4}$$

$$\text{R}(i) = \boldsymbol{\beta}_i \tag{5}$$

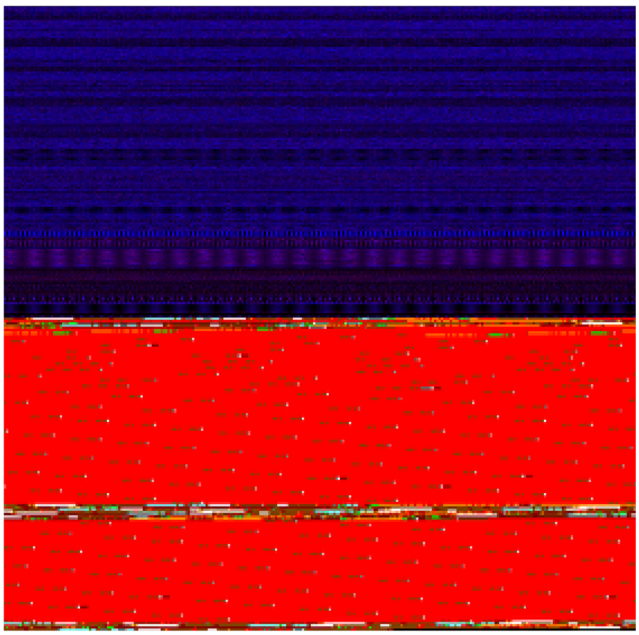

Figure 1 Example of an image generated by hybrid visualization.

In the above equations, the arctan function, arithmetic operations, and the floor function are all applied component-wise if the input involves a vector. The vector $\boldsymbol{\alpha}$ is the count vector already described. R(i) means the red channel value of the ith component of the pixel vector. $\boldsymbol{\beta}_i$ means the ith component of vector $\boldsymbol{\beta}$.

We now explain the meaning of the equations we developed. In order to derive the red channel value, we need to scale the count vector to the range [0,255]. To do this, the functions atanNorm and minMaxNorm are first applied to the count vector to normalize its components to the range [0,1]. One reason why atanNorm is applied is that in a typical legitimate binary, not all byte values occur with similar frequencies. In fact, some values (e.g., byte 0) can occur far more frequently than others. The arctan function grows more and more slowly as its input increases, so it is applied in hope of reducing the influence of possible outliers. If we multiply a value in the interval [0,1] by 255, we obtain a value in the range [0,255], which is already fit for use as a channel value of a pixel. However, we wish to highlight more informative regions in the malware binary with a brighter red in the image. Recalling knowledge from information theory, one knows that the less likely an event is to occur, the more information it carries. Therefore, after multiplying by 255, we further multiply the value by −1 and then add 255 to it, so that less frequently occurring byte values get a higher red channel value. Lastly, the floor is taken simply because the pixel value needs to be an integer.

2.3 Hybrid Visualization

Hybrid visualization combines static and dynamic approaches. In dynamic visualization, a program's behavior is visualized as an image. On modern operating systems, when a program wishes to request service from the OS, e.g., accessing the hard disk, it has to initiate a system call. Therefore, by monitoring the system calls that a program makes, we can learn about its important behavior.

On Windows, system calls are known as Windows APIs. Many sandbox solutions exist that can monitor the API calls of a chosen program. In this paper, we use Cuckoo Sandbox, which divides APIs into 17 categories. To visualize program behavior, we assign a color to each category of APIs, convert a program's API call sequence into an API category sequence, and then derive an image from it. When we were designing the mapping table from API category to color, we followed this principle: more potentially dangerous API categories should be assigned warm colors, and relatively harmless API categories should be assigned cold colors. The mapping table we finally designed is as follows (Table 1).

Now we describe the algorithm for dynamic visualization used in this paper. First, run a specified program in the

Table 2 Datasets sizes.

	Training	Validation
Static Images	494 (malware), 1048 (cleanware)	123 (malware), 262 (cleanware)
Hybrid Images	410 (malware), 710 (cleanware)	102 (malware), 178 (cleanware)

sandbox, and obtain its API call sequence. Each API call in the sequence is then converted to its corresponding category. The result is known as an API category vector. Next, map each API category in the API category vector to its corresponding RGB value, and the resulting vector is known as a pixel vector. Once the pixel vector is obtained, reshape it into a pixel matrix of width 512, and save it as an image. Finally, scale the image to 512 × 512 pixels.

With hybrid visualization, we incorporate the static and dynamic images in one image. By doing so, we encode both static and dynamic features of the original file into a single image. Static features reflect the structure and layout of the original binary, which are similar across malware variants and therefore helpful in identifying malware. However, static analysis can easily be frustrated by the use of obfuscation and code transformations. Dynamic analysis is significantly less susceptible to these than static analysis [21]. Obfuscation or code transformations aim to produce more structurally complex programs that remain semantically equivalent. This means that the programs produced should not change much (if at all) in their behavior. By using dynamic (behavioral) analysis, we are able to deal with obfuscated or transformed code more effectively than static analysis can. The dynamic approach is not without its weaknesses. Techniques have been proposed and implemented that try to circumvent behavior-based analysis. For example, malware can detect if it is being run in a monitored environment; if so, it will change its behavior [22]. The static approach is unaffected by these evasion techniques. Therefore, our hybrid approach, combining the static and the dynamic, overcomes the limitations of both methods. It will be more robust when it deals with malware that only tries to evade one of these two kinds of analysis.

To visualize malware using the hybrid approach, one first has to perform both static and dynamic visualization on the original binary. If dynamic visualization fails (e.g., because the binary fails to run in the sandbox), the hybrid approach fails. Otherwise, proceed to place the dynamic image below the static one, forming a new image with the size 512 × 1024 pixels. Finally, scale the new image to 512 × 512 pixels to obtain the hybrid image.

Figure 1 demonstrates an image generated by the above hybrid approach algorithm.

Rich information is encoded in the hybrid image. Figure 1 is clearly divided into the upper (static) half and the lower (dynamic) half. In the upper part, there is a distinct texture: alternating bands of blue and black regions. The blue regions correspond to different PE sections, and the black ones are the gaps (null bytes) between them. At the bottom of the upper part, we see purplish blocks next to each other. These are in fact resources (strings, icons, etc.) embedded within the executable, and they are visualized with a redder hue. This sets the region apart because it has a different byte frequency from others. The texture and color of the upper half reflect the structure and layout of the binary, and are usually characteristic of a certain malware variant (if it is malware). In the lower part, we see large, red regions, indicative of a number of potentially dangerous system calls. In

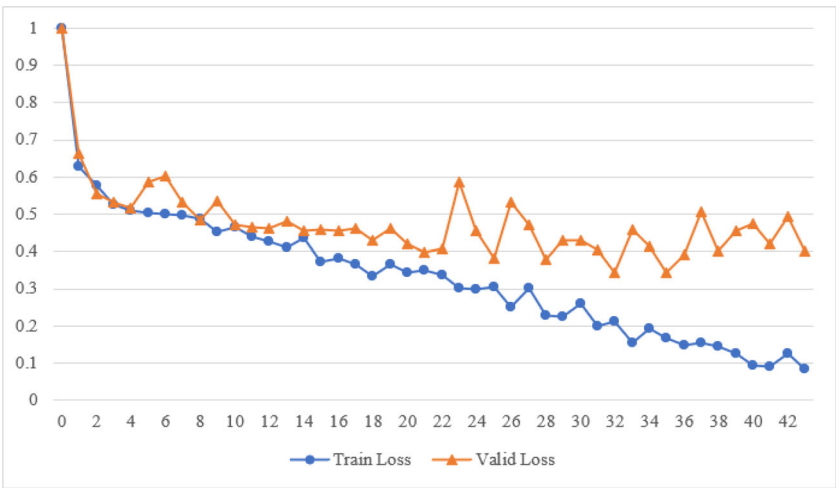

Figure 2 Loss for Model A on the training set and validation set.

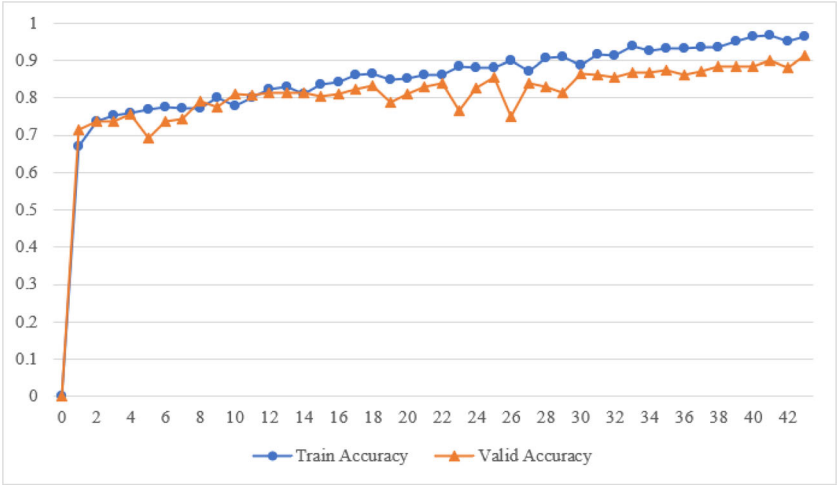

Figure 3 Accuracy for Model A on the training set and validation set.

particular, these calls are process-related operations, such as process creation, or remote code injection. It is typical malware behavior to inject malicious code into other processes. A legitimate program may carry out process-related operations, but usually not in such huge numbers or with such frequency. Therefore, the lower half suggests to us that the binary in question is highly suspicious. Generally, the lower half of a hybrid image indicates the "riskiness" of the binary's behavior. The static and dynamic information encoded in the image provide basis for a deep neural network to differentiate between malware and cleanware.

3 Experimentation

The network architecture used in this paper is based on the VGG16 network. Input to the network is a visualization image; output of the network is a 2D vector, whose first component is the probability of the original file being malicious, and whose second component is the probability of the original file being legitimate.

3.1 Datasets

Two datasets were used in the training process: the training set and the validation set. All images in these two sets were divided into two classes: malware and cleanware, and the images were derived from sample binaries we collected. Samples of the malware class were collected from the free malware package provided by virussign.com on March 12, 2020. Inside the package, there were 613 malicious PE files that run on 32-bit systems, and 4 that run on 64-bit systems. Samples of the cleanware class, totaling 1310 files, were from the system programs that come with Windows 7 and legitimate software installed on the author's computer.

After visualization, 617 static images of malware and 1310 static images of cleanware were generated. For some programs, it was impossible to generate their hybrid images, because, for example, they failed to run in the sandbox.

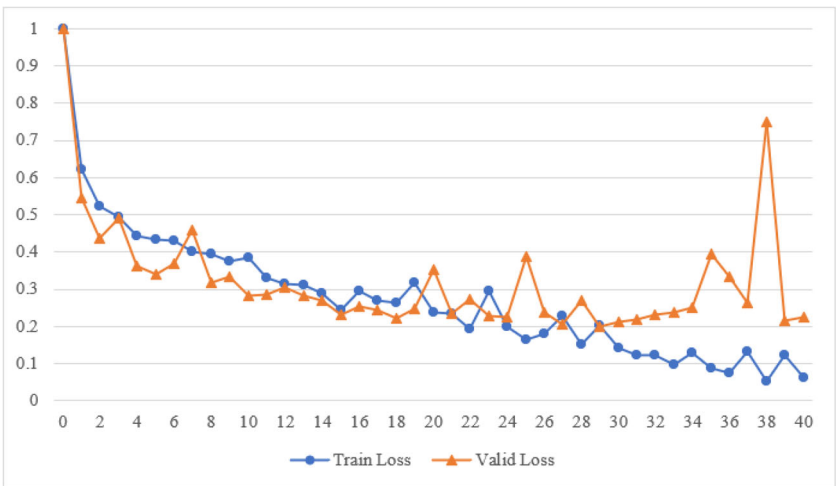

Figure 4 Loss for Model B on the training set and validation set.

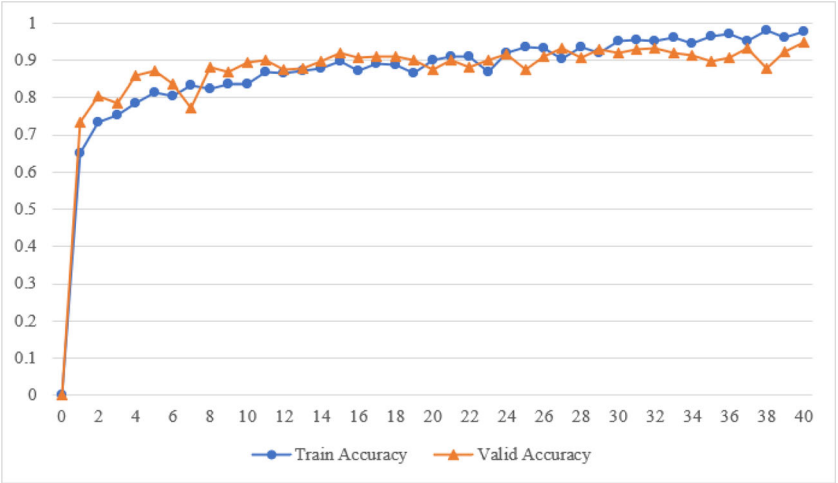

Figure 5 Accuracy for Model B on the training set and validation set.

Therefore, there were fewer hybrid images than static images. 512 hybrid images of malware and 888 hybrid images of cleanware were generated.

We randomly selected 80% of the samples in each class for training, and the remaining 20% for validation. The final sizes of the datasets are shown in Table 2.

3.2 Model Training

In this paper, we trained two models: Model A based on static visualization, and Model B based on hybrid visualization. Both models are based on VGG16 network. The weights were initialized randomly. Input to the network is an image of the size $512 \times 512 \times 3$. At the end of the network, we used a global average pooling layer and a Softmax layer in place of three fully connected layers to reduce computational cost. The parameters of the network were learned using Adam algorithm, with a learning rate of 1e-4. We used binary cross-entropy as the loss function. Batch size was set to 24. We trained Model A for 43 epochs and Model B for 40. We preserved the models with the highest accuracy during training.

The loss and accuracy for Model A on the training set and validation set versus training epoch are plotted in Figs. 2 and 3.

The loss and accuracy for Model B on the training set and validation set versus training epoch are plotted in Figs. 4 and 5.

Table 3 Validation accuracy of the models.

Model	Accuracy (Validation)
Model A	91.41%
Model B	94.70%

3.3 Evaluation

We carry out evaluation of the models in the following ways:

(1) Evaluate the accuracy on the validation set, i.e., how well do the models perform on samples similar to those in the training set.
(2) Compare our models with other similar approaches in terms of accuracy.
(3) Evaluate the accuracy on a newly selected test set, i.e., how well do the models perform on completely unknown samples.

Table 3 demonstrates the accuracy that the two models achieved on the validation set.

The results show that Model B outperforms Model A on the validation set.

Next, we compare our models with other approaches that use machine learning or deep learning. The results are shown in Table 4.

Model B outperforms the methods used in various works. It should be noted, however, that in the works of Li [16] and Xia [17], the datasets are based on Android malware, where as in this paper, we use Windows malware.

Finally, we evaluate the accuracy of our models on a new test set. Samples of the test set are from the free malware packages provided by virussign.com from March 8, 2020 to March 18, 2020 (without duplicates with the training set and validation set). From the packages, we randomly selected 200 files as our test set. The evaluation results are shown in Table 5.

We can conclude from the results that both models perform worse on the test set than on the validation set, which is expected because the test set contains samples completely new to the models. In addition, Model B still outperforms Model A on the test set, and its accuracy does not drop much from that on the validation set.

Table 4 Comparison of accuracy of different models.

Model	Method	Accuracy (Validation or Test)
Model A (ours)	Static Visualization	91.41%
Li [16]	Pixel Normalized RGB Image	92.37%
Xia [17]	API Sequence + Text CNN	93.90%
Cheng et al. [23]	Native API Sequence + SVM	94.37%
Mosli et al. [24]	Number of opened handles + RF	91.40%
Raff et al. [25]	Byte (Mapped) Sequences + CNN + RNN	94.00%
Siddiqui et al. [26]	Assembly n-grams + RF	94.00%
Model B (ours)	Hybrid Visualization	94.70%

4 Conclusions and Future Work

This paper proposes a malware detection method based on deep learning and malware visualization. Firstly, we generate static visualization images based on the static features of sample files. Secondly, we run the samples in our VM, obtaining a behavior analysis report generated by Cuckoo Sandbox, which is then used to derive dynamic visualization images. Thirdly, we merge the static and dynamic images, forming hybrid images. Finally, we train two detection models on the static and hybrid images respectively. This paper also evaluates the two models we train and concludes that the hybrid approach performs better than the static approach alone.

In the experimentation, we have also found that Model B performs badly on the following two types of malware:

(1) Older malware (e.g., from ten years ago);
(2) QQ password stealer trojans.

Since Model B is not adequately trained on past malware binaries, it fails to detect most of them. We may conclude that Model B, although able to predict unknown malware to a certain extent, requires regular (but perhaps not frequent) updates in order to adapt to currently prevalent malware. The reason why Model B is not so effective on type (2) malware may be that QQ password stealer trojans require certain environment (e.g., having QQ, an instant messenger application, installed) to be present in the virtual machine (VM) to function normally. We haven't installed QQ in our VM, which presumably prevented the trojans from showing malicious behavior. Model B, based on a hybrid approach, relies partly on program behavior to detect malware.

Table 5 Test accuracy of the models.

Model	Accuracy (Test)
Model A	82.50%
Model B	92.50%

Considering the above weaknesses of Model B, we can improve it in the future in the following respects:

(1) Improve its scalability, in order to more easily update it to adapt to current malware;
(2) Install commonly used applications in the VM, allowing malware to show full behavior.

Finally, we hope to utilize the currently unused green channel in our static visualization algorithm in the future to encode more meaningful information from the original file.

Acknowledgments This work was supported by grants from the Natural Science Foundation of Guangdong Province No.2018A0303130082, The Features Innovation Program of the Department of Education of Foshan under Grant Numbers 2019, Basic and Applied Basic Research Fund of Guangdong Province No.2019A1515111080, and Natural Science Foundation of China No.61802061.

References

1. Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques, 13*(1), 1–12.
2. Alazab, M., Venkataraman, S., Watters, P. (2010, July). Towards understanding malware behaviour by the extraction of API calls. In *2010 Second Cybercrime and Trustworthy Computing Workshop* (pp. 52-59). IEEE.
3. Ye, Y., Li, T., Adjeroh, D., & Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR), 50*(3), 1–40.
4. Zhang, F. (2003). *Implementation Technology and Research on Unknown Virus Detection Method and System* (Master's thesis). Northwestern Polytechnical University, Xi'an, Shaanxi Province, China.
5. Lu, Z. (2013). *A Study of Static Malicious Code Detection Method Based on Opcode Sequneces (Master's thesis)*. Harbin: Harbin Institute of Technology.
6. Ravi, C., & Manoharan, R. (2012). Malware detection using windows api sequence and machine learning. *International Journal of Computer Applications, 43*(17), 12–16.

7. Zhang, R., & Yang, J. (2014). Android malware detection based on permission correlation. *Journal of Computer Applications, 34*(5), 1322–1325.
8. Yang, P., Zhao, B., & Shu, H. (2019). Malicious code detection method based on icon similarity analysis. *Journal of Computer Applications, 39*(06), 1728–1734.
9. Qiu, H., Qiu, M., & Lu, Z. (2020). Selective encryption on ecg data in body sensor network based on supervised machine learning. *Information Fusion, 55*, 59–67.
10. Qiu, H., Noura, H., Qiu, M., Ming, Z., Memmi, G. (2019). A user-centric data protection method for cloud storage based on invertible DWT. IEEE Transactions on Cloud Computing
11. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., ... & Zhong, Y. (2018). Vuldeepecker: A deep learning-based system for vulnerability detection. Paper presented at 25th Annual Network and Distributed System Security Symposium (NDSS 2018), San Diego, California, USA, 18–21 February 2018.
12. Qiu, H., Zheng, Q., Memmi, G., Lu, J., Qiu, M., & Thuraisingham, B. (2020). Deep residual learning based enhanced JPEG compression in the internet of things. *IEEE Transactions on Industrial Informatics.*, 1.
13. Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: Visualization and automatic classification. In Proceedings of the 8th international symposium on visualization for cyber security (pp. 1-7).
14. Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., Iqbal, F. (2018, February). Malware classification with deep convolutional neural networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (pp. 1-5). IEEE.
15. Li, Y. (2018). *Detection of Malicious Software Based on Data Visualization* (Master's thesis). In *Xidian University, Xi'an*. Shaanxi Province: China.
16. Xia, X. (2018). Research on Android Malware Detection Method Based on Image and Text Feature with Deep Learning (Master's thesis). Harbin Institute of Technology, Harbin.
17. Fu, J., Xue, J., Wang, Y., Liu, Z., & Shan, C. (2018). Malware visualization for fine-grained classification. *IEEE Access, 6*, 14510–14523.
18. Shaid, S, Z, M., & Maarof, M, A. (2014, August). Malware behavior image for malware variant identification. In 2014 International Symposium on Biometrics and Security Technologies (ISBAST) (pp. 238-243). IEEE.
19. Zhang, J., Chen, B., Gong, L., & Gu, Z. (2019). Research on malware detection technology based on image analysis. *Netinfo Security, 19*(10), 24–31.
20. Lyda, R., & Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy, 5*(2), 40–45.
21. Moser, A., Kruegel, C., Kirda, E. (2007, December). Limits of static analysis for malware detection. In Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007) (pp. 421-430). IEEE.
22. Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR), 44*(2), 1–42.
23. Cheng, Y., Fan, W., Huang, W., & An, J. (2017, September). A shellcode detection method based on full native api sequence and support vector machine. In IOP Conference Series: Materials Science and Engineering (Vol. 242, no. 1, p. 012124).
24. Mosli, R., Li, R., Yuan, B., & Pan, Y. (2017, January). A behavior-based approach for malware detection. In IFIP International Conference on Digital Forensics (pp. 187-201). Springer, Cham.
25. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C, K. (2018, June). Malware detection by eating a whole exe. In Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence.
26. Siddiqui, M., Wang, M. C., Lee, J. (2008, April). Detecting trojans using data mining techniques. In International Multi Topic Conference (pp. 400-411). Springer, Berlin, Heidelberg.

Xiang Huang received the B.E. degree from Foshan University, China in 2020. He will be pursuing an M.S. in Computer Science in the fall of 2020. His research interests include software security and systems security.

Li Ma received the B.E. degree from Central China Normal University, China in 2001 and M.S. degree from South China University of Technology, China in 2008. She is currently an associate professor in college of electronic and information engineering of Foshan University, China. She has published more than 20 peer-reviewed journals or conference papers. Her research interests include access control in social network, formal specification of the security system, information security theory and cloud computing security.

Wenyin Yang received the B.E. and M.S. degrees from Jinan University in Guangzhou, China in 2005 and 2007, respectively. She is currently a doctoral candidate in Central South University and a lecturer in college of electronic and information engineering of Foshan University, China. Her research interests include social network security, security in cloud computing and big data.

Yong Zhong received the B.E. degree from Beijing University, China in 1992 and M.S. degree from Xi'an Jiao Tong University, China in 1999. He received the Ph.D. degree of Computer Science from Nanjing University of Aeronautics and Astronautics, China in 2005. He is currently a professor in college of electronic and information engineering of Foshan University, China. He has published more than 60 peer-reviewed journals or conference papers. His research interests include social network security, access control, security in cloud computing and big data, and computing logic.