# Plain Source Code Obfuscation as an Effective Attack Method on IoT Malware Image Classification

Hayato Sato*, Hiroshi Inamura*, Shigemi Ishida*, Yoshitaka Nakamura†
*School of Systems Information Science, Future University Hakodate, Japan
†Faculty of Engneering, Kyoto Tachibana University, Japan

***Abstract*—IoT malware is rapidly increasing due to variants easily generated from publicly available source codes. Malware image classification capable of fast and accurate malware identification attracts attention. Since the classification by imaging is affected by malware binary changes, a binary modification without behavioral changes can be a potential attacking method to the classification by imaging. There are concerns that by combining the publicly available malware source code with readily available source code obfuscation tools, it is possible to construct an effective attack that bypasses image classifiers relatively simply. In this study, we show the effectiveness of the attack by source code obfuscation and the possibility of defense against the attack. The contribution of this research is twofold. 1) We showed that Obfuscator-LLVM (oLLVM) code obfuscation could be used as an attack method on malware image classification. The obfuscated malware binaries made by oLLVM were misclassified by VGG16-based image classifier for all the attacked malware families including Mirai, Lightaidra, and Bashlite. 2) We showed that classifier training with obfuscated samples could address this attack method. We confirmed that the malware image classifier trained with obfuscated malware binaries made by oLLVM could classify with an accuracy of 100% the malware family with obfuscation as the obfuscated original malware family.**

***Index Terms*—Malware Image Classification, Code Obfuscation, IoT Malware.**

## I. Introduction

In recent years, the number of observed IoT malware has increased. Sonicwall cyber threat report [1] recorded 112.3 million attacks in 2022, an 87% year-over-year increase. However, the attack volume remains high, so it is necessary to continue to monitor IoT malware trends.

Malware family classification is helpful because many IoT malware results from variant generation using publically available source code. Malware belonging to the same family has similar functionality, so there is less need for new analysis. Usually, malware analysis is a manual process [2]. Manual analysis is often a time-sensitive task and can take several hours to several weeks [3]. With the increase in malware, it has become impossible to analyze all of them manually.

Malware image classification is suitable for IoT malware family classification. Many IoT malware is statically linked, and some binaries have also had their symbol tables removed [4]. More than 85% of the IoT malware we collected was statically linked, and about half of them had their symbol tables removed. In the case of malware with statically linked and symbol table removed, it is not possible to use malware classification methods based on linked library functions [5], as is done on Windows malware. There are known classification methods that do not use the information on linked functions, such as detection methods that use system-call sequences obtained by dynamic analysis and deep learning [6], and techniques that use n-grams of instruction sequences that exist in binaries [7]. Still, the cost of these analyses is high. Malware image classification is a method that does not require information on linked functions and is reported to be more accurate than conventional methods [8]. It does not require code execution or disassembly, and feature extraction can be automated, contributing cost reduction.

Malware image classification is affected by binary changes in malware. Therefore, making binary changes without behavioral changes of the program can be a potential attack method against image-based classification. As a means of such binary modification, we consider code obfuscation utilizing the publicly available source code for IoT malware.

There are concerns that by combining the publicly available malware source code with readily available source code obfuscation tools, it is possible to construct an effective attack that bypasses image classifiers relatively simply. This research assumes obfuscation applied to source code as an attack method, and its effects and countermeasures are examined. When a malware family's classification is attacked, it is necessary to analyze all newly detected malware variants, which increases the burden on analysts and prevents them from responding to a rapid increase in malware. To improve image-based malware classification technique , it is important to consider countermeasures against possible future attacks such as those described above.

The contribution of this research is twofold. 1) We showed that Obfuscator-LLVM (oLLVM) [9] code obfuscation could be used as an attack method on malware image classification. The obfuscated malware binaries made by oLLVM were misclassified by VGG16-based image classifier for all the attacked malware families including Mirai, Lightaidra, and Bashlite. 2) We showed that classifier training with obfuscated samples could address this attack method. We confirmed that the malware image classifier trained with obfuscated malware binaries made by oLLVM could classify with an accuracy of 100% the malware family with obfuscation as the obfuscated


doi: 10.1109/COMPSAC57700.2023.00125

original malware family.

## II. Related Work

Research on malware image classification has been actively conducted [10]–[14]. Many of them utilize a Convolutional Neural Network (CNN) based classifier. However, the effect of code obfuscation on malware image classification has not been verified as we have investigated. In this research, we facilitate Obfuscator-LLVM [9] as a code obfuscator to verify the effectiveness of the attack and the possibility of countermeasure with it.

### A. Malware Image Classification with CNN and Use of Obfuscation

In Kalash et al.'s study [10], image classification using CNN was performed on Windows malware, with higher accuracy than image classification methods using traditional machine learning algorithms. The CNN-based classification method automatically acquires more effective features than human-designed methods. When applied to malware image classification, it obtains malware-specific image regions that indicate common functions of the malware family.

Yakura et al. visualize the specific regions of malware families in an attention map by adding an attention mechanism to the CNN [11]. Analyzing malware based on the attention map reduces the work of manual analysis. They report that the specific regions of malware correspond to other malware belonging to the same family; and that it is possible to respond even if the location of the specific regions changes.

Nataraj et al. used conventional image-based malware classification to perform family classification for Windows malware and reported that packed malware can also be classified [12]. Since packed malware often contains common sequences [11], it could be classified by image-based classification for the same reason. In this research, we examine the effect of code obfuscation on attacks and evaluate countermeasures. Code obfuscation is presumably difficult to deal with because it allows modification at the code level without the restriction of modification at the binary level.

Su et al. [13] evaluated a CNN-based image classification for IoT malware. They found that malware targeting IoT devices is often a lightweight program, so a more compact model than the CNN used for Windows malware classification could still classify them with high accuracy. In their view, the study on complex cases such as malware obfuscation is considered to improve the detection rate of IoT malware, affirming a direction similar to our study.

Huang et al. [14] apply image classification methods to Android malware. Their method uses color images rather than grayscale images for malware classification. While obfuscation is used in the production of legitimate software in Android applications, there are reports of its application to malware as well [15]. Just as malware obfuscation is widespread in mobile applications, the use of advanced defenses such as code-level obfuscation in IoT malware can be expected to become commonplace.

Regarding transformation techniques that preserve runtime semantics and transform representations, there is research on the Android platform that targets the misidentification of malware families by injecting adversarial samples through transformations at the operation code level [16]. It preserves runtime semantics while composing adversarial examples. In this research, a One-pixel attack is applied to the data area, and the identity of the execution result cannot be perfect. In the case of IoT malware, for which source code is publicly available, it is possible to apply sophisticated obfuscation techniques to transform the entire code while preserving runtime semantics globally. Because, at the same time, it is easily implementable into the existing malware development pipeline, the practical impact is more significant.

### B. Obfuscation Functions in Obfuscator-LLVM

Obfuscator-LLVM is an obfuscation tool based on LLVM [17]. LLVM creates an intermediate code called LLVM IR (Intermediate Representation) at compile time, optimizes, and then translates the compiled IR into an architecture specific executable binary file. oLLVM obfuscates against LLVM IR and can generate different binaries each time from the same source code. oLLVM is highly flexible because it is essentially language-agnostic. It can be applied to programming languages such as C/C++, Swift, and Go, which have a language processing system that uses LLVM as the back end, and its target processor supports various architectures such as x86, ARM, and x86_64.

oLLVM has three obfuscation functions: control flow flattening, instructions substitution, and bogus control flow insertion. In control flow flattening, all control statements are divided into blocks and changed to a program controlled by switch statements and variables. Instructions substitution replaces standard binary operators with functionally equivalent but more complex instructions. Bogus control flow insertion splits all the basic blocks and changes them into a program controlled by switch statements and variables. The binary image of the malware before and after obfuscation using oLLVM is significantly different. Details are given in Section III-A2.

## III. Code Obfuscation Attack and Countermeasure

Since the classification by imaging is affected by modification of malware binary, a binary modification that does not change the program's meaning can be considered a potential attack method to the classification by imaging. In this study, we propose an attack method for malware image classification using code obfuscation. Assuming that obfuscation is added to generating variants of open source IoT malware, we examine the effectiveness of the attack and the possibility of dealing with it.

In the following sections, we provide an overview of the Obfuscator-LLVM-based code obfuscation attack, describe the malware binary imaging technique, and depicts the attack technique using the binary image changes caused by obfuscation

**Algorithm 1** Transform Malware Binaries into 2D images

**Require:** $binary$
**Ensure:** $image$
1: $s \leftarrow \sqrt{\text{sizeof}(binary)}$
2: $image[s][s] \leftarrow 0$
3: **for** $y = 0, \cdots, s-1$ **do**
4:  **for** $x = 0, \cdots, s-1$ **do**
5:  $image[y][x] \leftarrow binary[y \times s + x]$
6:  **end for**
7: **end for**
8: $image \leftarrow \text{resize}(image, (224, 224))$
9: **return** $image$

with oLLVM, and how to deal with it through training with obfuscated samples.

### A. Obfuscator-LLVM-based Code Obfuscation Attack

In this study's malware image classification, the collected malware is imaged and used to train CNN. It is assumed that the malware's source code targeted by the attack is available. We use oLLVM to generate the target malware binary with code obfuscation from the source code. Obfuscated malware is visually different from the original malware when imaged because the binary representation is altered. The CNN classifier trained with collected malware will be misled by obfuscated malware binary.

The implementation of the attack involves creating a baseline malware image classifier of the IoT malware to serve as the evaluation standard and verifying the collected malware test data by replacing it with binary images of obfuscated samples.

*1) Imaging Malware Binaries:* The malware executable binary is converted to a square grayscale image with one byte being one pixel, depending on the file size of the executable binary. The imaging algorithm is shown in **Algorithm 1**. Each malware generates a different size image, but the CNN needs a uniform input size, so it is scaled into the size of $224 \times 224$, which is the size required by the input of VGG16, the CNN used in this implementation.

*2) Altering Binary Image:* The imaged malware binaries are visually altered by applying the Obfuscator-LLVM obfuscation functions. The Figure 1 shows an example of IoT malware Mirai built with GCC, Clang, and oLLVM, respectively, and converted to a binary image using the method described in Section II-B. Clang and oLLVM use LLVM as the backend compiler, and the section layout is different from that using GCC. This has resulted in global changes that can be seen briefly when comparing builds using GCC with builds using Clang and oLLVM. The malware collected in this study confirmed that most of the malware prevalent in the market are specimens built using GCC.

The global changes in the binary image due to the rearrangement of sections through LLVM and the changes in image detail due to the obfuscation process could disrupt the malware image classifier. Obfuscated binary images by obfuscation function are shown in the Figure 2. oLLVM has three obfuscation functions:control flow flattening, instructions substitution, and bogus control flow insertion. The obfuscation process combines each of these functions and able to generate different binaries each time from the same source code by randomly changing obfuscation parameters. The binary image without obfuscation is equivalent to the build with Clang in the Figure 1, and the differences can be observed. Flattening the control flow adds an additional flattening process early in the process, so the code portion is larger. The code section is the large block at the bottom of the binary image. Instruction substitution and false control flow add redundant code to the entire binary, resulting in a larger malware file size and finer texture throughout the image. Although it is difficult to see the effect on the image alone by checking the build image by oLLVM of the Figure1, in which all obfuscation functions are applied once, we can see that the granularity of the entire image is finer.

### B. Countermeasure by Training with Obfuscated Samples

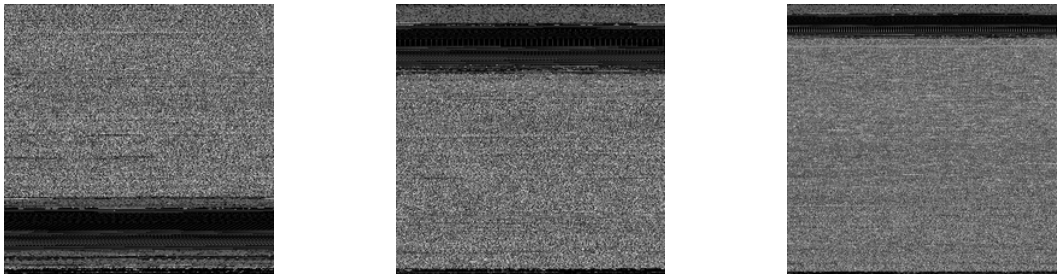
For the attack effect on the malware image classification by obfuscated samples, it is considered that it can be dealt with by training using the obfuscated sample. When training an image classifier, obfuscated samples are used in addition to the malware used for learning, and the obfuscated samples are trained as a new malware family. We will train including samples generated without obfuscation using oLLVM, so that we can respond to attacks that are built based on LLVM but are not obfuscated.

## IV. Evaluation

Obfuscation processing is applied to the source code of IoT malware to generate a binary. It is discriminated using a baseline malware image classifier trained with the collected malware, so that the attack effect of the obfuscation processing is verified. As a countermeasure to this attack, we created an image classifier for malware trained using obfuscated samples, made them discriminate samples with and without obfuscation, and could classify obfuscated samples into the correct malware family. The evaluation was performed using Mirai, Lightaidra, and Bashlite, IoT malware whose source code is open to the public.

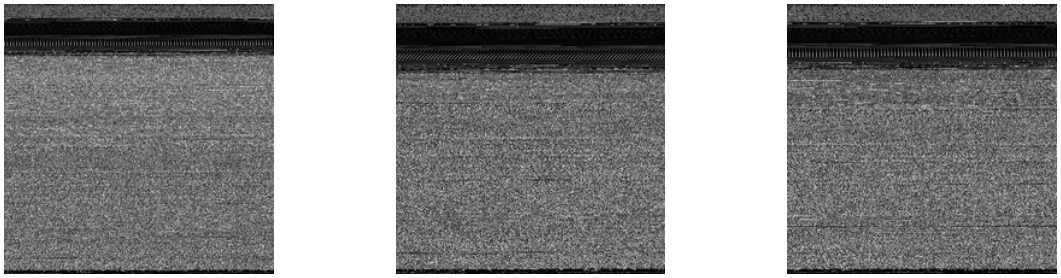
### A. Dataset Used in Experiments

As the data set used for the experiment, the collected malware, the obfuscated samples for attack experiments, and the obfuscated samples for countermeasure experiments were used. The collected malware and obfuscated samples for countermeasure experiments were labeled and divided by malware families, and 80% for training, 20% for testing, and 10% for training data were used as verification data. Since all malware for attack experiments is used by replacing the test data of the baseline malware image classifier, it is not necessary to divide by each family.



(a) Built with GCC (b) Built with Clang (c) Built with oLLVM

Fig. 1. Binary images by build method



(a) Control Flow Flattening (b) Instructions Substitution (c) Bogus Control Flow Insertion

Fig. 2. Binary images by obfuscation function

TABLE I
RATIO OF EACH ARCHITECTURE OF MALWARE COLLECTED

| Architecture | #.of Samples | Ratio (%) |
|---|---|---|
| ARM | 16468 | 30.55 |
| x86 | 9702 | 18.00 |
| MIPS | 9318 | 17.29 |
| PowerPC | 4228 | 7.84 |
| SuperH | 4141 | 7.68 |
| Motorola 68k | 4026 | 7.47 |
| SPARC | 4026 | 5.63 |
| x86-64 | 2741 | 5.09 |
| (others) | 242 | 0.45 |

TABLE II
PERCENTAGE OF EACH LINK METHOD FOR ARM ARCHITECTURE TARGET SAMPLES

| Linking Methods | #.of Samples | Ratio (%) |
|---|---|---|
| Static | 14014 | 85.10 |
| Dynamic | 2454 | 14.90 |

*1) Collected Malware:* From the malware sharing site VirusShare [18], we obtained a dataset that summarizes the ELF format malware confirmed from 2014 to 2020. Similar malware is available from "VirusShare_ELF_20190212" and "VirusShare_ELF_20200405".

Since many IoT devices are equipped with the ARM architecture CPUs and most IoT malware is static links, only ARM architecture and statically linked samples are extracted and used for the experiments. The table I shows the percentage of collected malware by architecture. If the architecture is different, the characteristics of the images generated from the binary will also be significantly different, so in this experiment, only the samples targeting the ARM architecture, which has a sufficient number of samples, were used. The table II shows the ratio of IoT malware targeting ARM architecture by link method. Since the proportion of samples is low, this study excluded dynamically linked samples.

TABLE III
LABELING BREAKDOWN

| Malware Families | #.of Samples | Ratio (%) |
|---|---|---|
| Mirai | 4846 | 38.77 |
| Lightaidra | 2612 | 20.90 |
| Bashlite | 2301 | 18.41 |
| Occamy | 1466 | 11.73 |
| Mploit | 597 | 4.78 |
| Skeeyah | 318 | 2.54 |
| Yakuza | 134 | 1.07 |
| Tsunami | 118 | 0.94 |
| Berbew | 106 | 0.85 |

For the labeling of malware families, the judgment results provided by Microsoft via VirusTotal [19] were used to obtain malware families with more than 100 samples. The number of samples for each malware family is shown in the table III. The number of malware samples collected was 12498.

About 80% of IoT malware is either one of Mirai, Lightaidra, Bashlite, and the source code of these malware are available on GitHub [20]. Therefore, these codes were used to generate the obfuscated sample.

*2) Obfuscated Samples for Attack Experiments:* We used samples generated by Clang and an obfuscation sample generated by oLLVM. To individually confirm the effects of section relocation through LLVM and obfuscation processing using

oLLVM, these attacks were established separately.

For the obfuscated sample build, control flow flattening, instruction substitution, and fake control flow addition were applied. The total number of obfuscated samples was prepared as many as the test data for each malware family.

This combination of obfuscation functions is used because it does not deviate from the file size of the collected malware, all obfuscation functions are effective, and the build time is relatively short.

*3) Obfuscated Samples for Countermeasure Experiments:* Obfuscated samples generated by oLLVM were used, including samples without obfuscation to handle Clang attacks. The obfuscation function of oLLVM applied the flattening of the control flow, the instruction substitution, and the addition of the fake control flow 0 to 1 times each. So, there are eight combinations of obfuscation functions. Since we prepared 300 obfuscated samples for each type, the obfuscated samples are 2400 samples for each malware type. Since there are three types of IoT malware used for attack and defense, the total number of obfuscated samples was 7,200.

### *B. Creating a Baseline Image Classifier*

Mirai, Lightaidra, and Bashlite, the targets of this attack, are classified with 87% accuracy in average by our baseline classifier. The confusion matrix of the baseline classifier is shown in the Figure 3 (a). The vertical axis is the true malware family, and the horizontal axis is the inferred malware family. The accuracy of Tsunami and Berbew is low due to the small number of samples, but it can be solved by using a large data set. Occamy, Mploit, and Skeeyah are considered Mirai in some of the classification results, suggesting the existence of two classes in their code similarity to Mirai and Occamy, but are shown here as the same according to the ground truth. Since this research aims to verify the attack effect by obfuscating the source code and evaluating its countermeasures, we will not discuss this problem in-depth.

TensorFlow and Keras were used to create the CNN. The CNN model was based on VGG16 [21] . We trained with an epoch number of 25, an optimization function of SGD, and a batch size of 6. The learning rate was changed according to the number of epochs and was reduced to 0.01 at the beginning of learning and to 0.001 when the number of epochs was 15 or more. The accuracy of the baseline classifier trained using the collected malware was about 78%.

### *C. Validating Obfuscator-LLVM-based Obfuscation Attack*

The obfuscated malware binaries made by oLLVM were misclassified by VGG16 based image classifier for all the attacked malware families, Mirai, Lightaidra, and Bashlite. We attacked the baseline malware image classifier by replacing the obfuscated images with test data for each malware family targeted. The figure 3 (b) and the figure 3 (c) show the confusion matrix after the code obfuscation attack. The classification performance of the baseline malware classifier before the attack is compared with that of the Clang and oLLVM attacks. The baseline classifier can classify the attacked malware families with 87% accuracy in average before the attack, but when the two attack methods are applied, the attacked malware families are all misclassified for both methods (circled on the matrix).

Since the attack effect using Clang is recognized, it is considered that the impact of section relocation through LLVM is significant. However, all attacks by Clang are classified into the same family. Since there is no randomness and only the same binary can be generated, it is presumed that the attack performance by itself is low.

This experiment confirmed that the obfuscation process applied at compile time could have a specific effect as a straightforward attack on the image classification method of malware.

### *D. Validation of Countermeasures by Training with Obfuscated Samples*

We confirmed that the malware image classifier trained with obfuscated malware binaries made by oLLVM could classify with an accuracy of 100% the malware family with obfuscation as the obfuscated original malware family. Since the effectiveness of the obfuscation process attack against the image classification method was confirmed, we set up a dataset of binary images of the collected malware and obfuscation samples. and evaluated the countermeasure by training, including the obfuscated samples. The confusion matrix of the classifier trained with obfuscated samples are shown in Figure 4. Mirai_o, Lightaidra_o, and Bashlite_o are new malware families trained with obfuscated samples. Obfuscated Mirai, Lightaidra, and Bashlite were classified with 100% accuracy by training with obfuscated samples (circled on the matrix). Furthermore, the classification of the obfuscated samples did not affect the classification results of existing malware families.

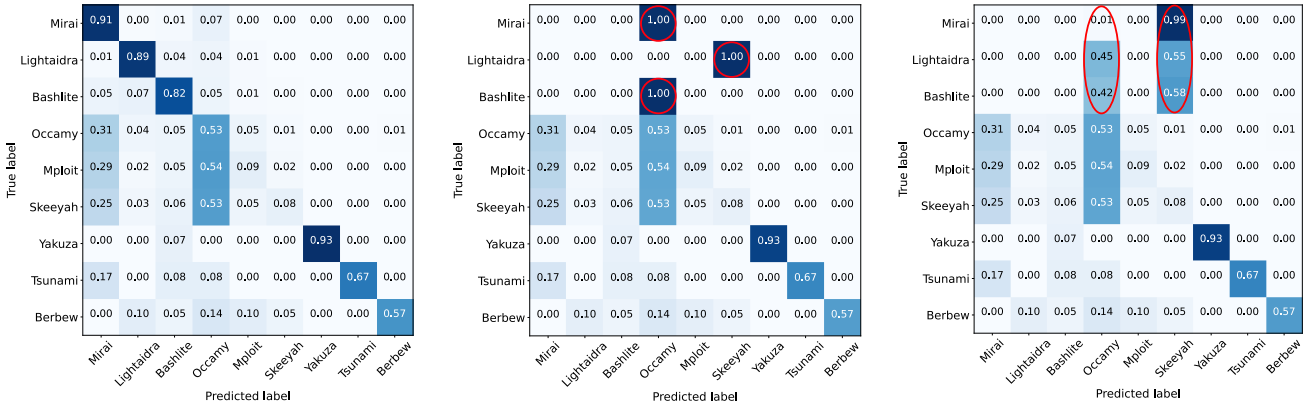
## V. Conclusion

By combining the publicly available malware source code with readily available source code obfuscation tools, it is possible to construct an effective attack that bypasses image classifiers relatively simply. This paper presented an attack on malware image classification methods by obfuscation applied to source code and a countermeasure with training using obfuscated samples.

Obfuscation applied at compile-time is effective as an attack method on IoT malware image classification. Since it is easy to add oLLVM obfuscation to the malware production process, it is conceivable that malware with obfuscation protection will increase. Since this attack can be addressed by training with obfuscated samples, when using a classifier based on image classification, it should be considered in advance to train with obfuscated samples.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP20K11772 and JP21K11847.




(a) Pre-attack Baseline Classifier (b) Attack by Clang (no obfuscated) (c) Attack by oLLVM (obfuscated)

Fig. 3. Confusion Matrix Before and After the Code Obfuscation Attack

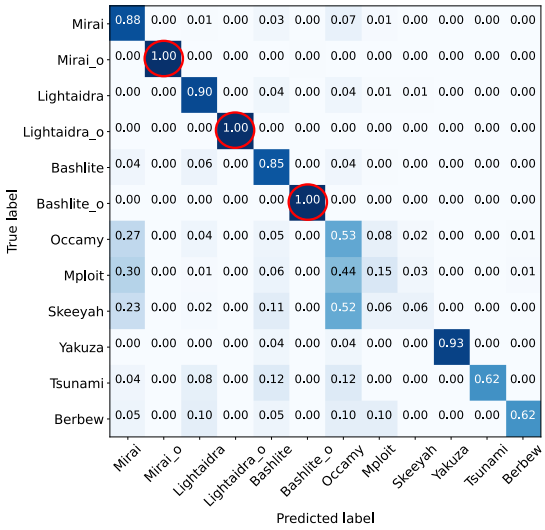



Fig. 4. Confusion Matrix of Trained Classifier with Obfuscated Samples

## REFERENCES


[1] SonicWall, "2023 sonicwall cyber threat report: Threat intelligence," https://www.sonicwall.com/2023-cyber-threat-report, accessed: 2023-04-27.

[2] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 2007, pp. 231–245.

[3] B. Anderson, C. Storlie, M. Yates, and A. McPhall, "Automating reverse engineering with machine learning techniques," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, ser. AISec '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 103–112. [Online]. Available: https://doi.org/10.1145/2666652.2666665

[4] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 161–175.

[5] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," in *Proceedings of the 2010 ACM symposium on applied computing*, 2010, pp. 1020–1025.

[6] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian joint conference on artificial intelligence*. Springer, 2016, pp. 137–149.

[7] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild." *Journal of Machine Learning Research*, vol. 7, no. 12, 2006.

[8] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 21–30.

[9] P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin, "Obfuscator-LLVM – software protection for the masses," in *Proceedings of the IEEE/ACM 1st International Workshop on Software Protection, SPRO'15, Firenze, Italy, May 19th, 2015*, B. Wyseur, Ed. IEEE, 2015, pp. 3–9.

[10] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 2018, pp. 1–5.

[11] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Neural malware analysis with attention mechanism," *Computers & Security*, vol. 87, p. 101592, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404819300264

[12] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[13] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, vol. 2. IEEE, 2018, pp. 664–669.

[14] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2633–2642.

[15] X. Zhang, F. Breitinger, E. Luechinger, and S. O'Shaughnessy, "Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations," *Forensic Science International: Digital Investigation*, vol. 39, p. 301285, 2021.

[16] S. Gu, S. Cheng, and W. Zhang, "From image to code: Executable adversarial examples of android applications," in *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, ser. ICCAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 261–268. [Online]. Available: https://doi.org/10.1145/3404555.3404574

[17] llvm.org, "The llvm compiler infrastructure project," https://llvm.org, accessed: 2023-04-27.

[18] CorvusForensics, "Virusshare.com," https://virusshare.com, accessed: 2023-04-27.

[19] Google, "Virustotal," https://virustotal.com, accessed: 2023-04-27.

[20] F. Ding, "Iot malware," https://github.com/ifding/iot-malware, 2017, accessed: 2022-03-14.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.