R2-D2: ColoR-inspired Convolutional Neural Network (CNN)-based Android Malware Detections

TonTon Hsien-De Huang†, Hung-Yu Kao*
Department of CM Security, Leopard Mobile Inc., Taiwan†
Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan† *
TonTon@TWMAN.ORG†

Abstract—Machine Learning (ML) has found it particularly useful in malware detection. However, as the malware evolves very fast, the stability of the feature extracted from malware serves as a critical issue in malware detection. Recent success of deep learning in image recognition, natural language processing, and machine translation indicate a potential solution for stabilizing the malware detection effectiveness. We present a colorR-inspired convolutional neural networks (CNN)-based Android malware Detection (R2-D2), which can detect malware without extracting pre-selected features (e.g., the control-flow of op-code, classes, methods of functions and the timing they are invoked etc.) from Android apps. In particular, we develop a color representation for translating Android apps into RGB color code and transform them to a fixed-sized encoded image. After that, the encoded image is fed to convolutional neural network for automatic feature extraction and learning, reducing the expert’s intervention. We have collected over 1 million malware samples and 1 million benign samples according to the data provided by Leopard Mobile Inc. from its core product Security Master (which has 623 million monthly active users and 10k new malware samples per day). It is shown that R2-D2 can effectively detect the malware. Furthermore, we keep our research results and release experiment material on http://R2D2.TWMAN.ORG if there is any update.

Index Terms—Deep Learning, Android Malware Detection, Convolutional Neural Network

I. INTRODUCTION

Nowadays, smartphone has become a daily necessity in our life. In the smartphone market, Android is the most commonly used operating system (OS), and it is still expanding its market share. According to the report by International Data Corporation (IDC) in 2016, the market share of Android in smartphone market increased from 84.3% in 2015 Q2 to 86.8% in 2016 Q3 (see Fig. 1). Android is featured by its openness; users can choose to download apps from Google Play or third-party marketplace. However due to the popularity and openness, Android has attracted attacker’s attention. In particular, malicious software (malware) can easily be spread and infects benign Android devices. The Security Report of AV-TEST Institute shows that while the number of malware increased from 17 million in 2005 to over 600 million in 2016, the percentage of Android malware had a significant increase from 3.19% in 2015 to 7.48% in 2016 Q2. Among them, Trojans targeting at stealing user data occupied 97.49%. We can also find that Android malware has dominated the market with 99.87% on the number of malware on smartphone platform [2]. Fig. 2 shows the statistics collected from our back-end system in January 2017. In countries such as the United States, United Kingdom, and France, etc., more than 50,000 users were infected daily. Moreover, according to the data provided by Leopard Mobile Inc. collected from its core product Security Master, it shows that the number of Android malware increased sharply from 1 million in 2012 to 2.8 million in 2014. In 2015, the number of Android malware detected was three times than the one in 2014 (over 9.5 million). In 2016 the number of Android malware achieved more than 17 million. In the first half of 2017, there was more than 10 million of the number of Android malware found in Security Master. It has been five years that the number of malware found in Android exceeded the number of malware found in Windows OS over the past 21 years [3]. It is obvious that we cannot neglect the security problem from Android nowadays. To deal with the serious security problem caused by Android malware, we proposed the colorR-inspired convolutional neural networks (CNN)-based Android malware Detection, R2-D2, to detect Android malware. R2-D2 is different from existing solutions. The R2-D2 detection is featured by its end-to-end learning process. More specifically, in contrast to the prior solutions that require manual process of feature selection and parameter configuration, R2-D2 can effectively decrease the resource inputs of manpower and computing. With R2-D2, we can now process and analyze tons of real-time data faster than before. Meanwhile, we can also detect unknown Android malware in a more effective way.

![Fig. 1. World Smartphones OS Market Share.](image-url)
II. RELATED WORK

A. Android Malware Analysis Background

First, Android Package (APK) is the format used by Android OS for distribution and installation of mobile apps. Essentially, APK files are a type of zip-formatted archive file. The structure of APK can be dissected as follows:

- META-INF/: contains the information description from Java jar file.
- res/: contains the resource document.
- libs/: contains the .so library from Android Native Development (NDK).
- AndroidManifest.xml: contains the configuration file about the authorization and service.
- classes.dex: contains the dalvik byte-code, i.e., Android execution file.
- resource.asc: contains the binary resource file after compilation.

Fig. 3 shows the compilation process of Android APK. First we compile all of the Java source code (including R.java and Java interface etc.), generating the .class files. After that, all of the .class files are transformed into dex format generating .dex files supported by Davik VM (Dalvik VM is alike Java VM). It is a bytecode compiler provided on Android phones, mainly developed by company such as Google etc., with the minimal requirement on the resource. In other word, Dalvik VM can be run on the mobile devices with limited computing and memory resource but with acceptable performance. The .dex file is a Java application program, which is in Dalvik executable format. It is all packaged as Android Package (.apk) file by apkbuilder.

One of the most common approaches for Android malware detection is static analysis (signature-based) method. It works through reverse-engineering tools such as Apktool, backsmali, dex2jar, and JD-GUI, to decompile Android App to access the source code from classes.dex. Furthermore, AXMLPrinter2 is used to analyze the AndroidManifest.xml to access the permission for using Android Apps. Another category of approach for Android malware detection is the dynamic analysis (behavior-based) method. The idea behind this category is to continuously track the system communications and network connections or messages by emulating Android apps through techniques such as sandbox (e.g. Droidbox), virtual machine and other operating environment in order to require the dynamic analysis of the behavior model. The security exports then analyze and inspect the records to see whether the malicious behaviors occur. If so, virus pattern and black/white-list are generated for pattern matching and for capturing the malware. However, anti-analysis techniques such as obfuscation, encryption, and anti-debugging are also proposed by hackers to hide the malicious behaviors to escape from the detection [4]. In fact, the above approaches can be useless when the malware programmers make modification on the initial components in malware, which can evade the detection.

Fig. 4 shows the popular class naming in benign Android apps (e.g. intellij.annotations and features2d). Fig. 5 shows the common way for declaring the data variables (e.g. paramBitmap and paramLoadedForm). However, to evade the detection, certain random naming (e.g. dshkji,jgfkjejkjh and nlsrkbgicl) will be used by “opfake” family (shown in Fig. 6). Fig. 7 shows the “fakeinst” family that can automatically encrypt variables (e.g. jj = 84f113ee155ba4f1e280b54401fffab and jj1 = 84f113ee155ba43f1e280b54401fffab).
Fig. 5. The common way for declaring the data variables.

```java
try {
    if (!paramActionTarget trovareTargetAction().orElse(paramAction.getTarget()))
        return;
    // Declaration of data variables
    double px[5];
    int n = 10;
    String s = "Hello world";
}
```

Fig. 6. Certain random naming format will be used by opfake family.

```java
// Declaration of variables
double px[5];
int n = 10;
String s = "Hello world";
```

B. Machine Learning-based Malware Detection

Machine learning is a technique for enabling the machine to learn the pattern, build the model, and make the prediction by seeing only the raw data. Currently, the most widespread malware attacks are still "exploit attack" and "privileged escalation". As a result, most of the machine learning-based malware detection still find the features such as the Android apps permissions, API invocation and control flow graphs (CFG) in order to distinguish between the benign and malicious apps. Usually, SVM and random forest are used to build the model to distinguish between benign and malicious apps.

For example, DroidMiner [5] has a two-step flow-chart to represent the behaviours of Android apps and capture the execution logs behind the apps. After that, rectors are clustered to identify the similar code snippet. Lei Chen et al. [6] extracted features from Android API invocation as a reverse engineering approach. They apply normalization procedure to the extracted featuring and perform logistic regression on the extracted feature to detect Android malware. Moreover, the dataset used by the current research-oriented machine learning-based Android malware detection is rather small. Most of existing solutions train their detection models sorely based on small datasets. This cause the practicality problem because the real world malware may have distinguishing distinctive behaviors and the existing detection models usually cannot successfully identify the malware by correlating the API invocation and Android permissions.

C. Deep Learning

Recent success in deep learning research and development attracts people’s attention [7]. In 2015, Google released Tensorflow [8], a framework of realizing deep learning algorithm. Deep learning is a specific type of machine learning. More specifically, deep learning is an artificial neural network, in which multiple layers of neurons are interconnected with different weights and activation functions to learn the hidden relationship between input and output. Intuitively, input data is fed to the first layer that generates different combinations of the input [9]. These combinations, after the activation function, are fed to the second layer, and so on. Under the above procedures, different combinations of the outputs from previous layer can be seen as different representation of features. The weights on links between layers are adjusted according to backward propagations, depending on the distance or less function between true output label and the label calculated by neural network. Note that deep learning can be seen as a neural network with a large number of layers. After the above learning process via multiple layers, we can derive a better understanding and representation of distinguishable features, enhancing the detection accuracy [10]. Also notice that the effectiveness of deep learning increases by the network size. In addition to deep neural networks, the most well-known deep networks are convolutional neural networks (CNN). The representation of CNN includes AlexNet, VGG, GoogleNet, and ResNet [11][12][13][14]. More specifically, CNN is composed of hidden layers, fully connected layers, convolution layers, and pooling layers. The hidden layers are used to increase the complexity of the model. If the same number of neural is associated with the input image, the number of parameters can be significantly reduced, adapting to the function structure much properly.

D. Deep Learning-based Malware Detection

Deep learning once is seemed as the cure for the above problem. However, a pre-processing step, such as feature engineering, is still needed before the model is learnt. Furthermore, the dataset for training the model usually cannot reflect real-world malware accurately. For example, [15] propose a malware detection in which the Windows API inquiry generates a corresponding ID, which is treated as the input of the deep-learning architecture (eg. stack of Auto-Encoders), and then it fine-tunes the model parameters. [16] is a method that it works on feature extracting first, such as contextual byte features, PE import features, string 2d histogram features, and PE metadata features. Then, the extracted features are fed to the deep neural network (DNN). With the training of two hidden layers, it is categorized. [17] uses static analysis to extract features such as required permission, sensitive API, and also uses dynamic
analysis to extract features such as "action dex class load", "action recent" and "action servicestart", from 500 samples for about 200 features as the input for the deep belief network (DBN). There is similarity between the execution logic of Android malicious apps and the order of functions being called.

Thus, in addition to the aforementioned solutions that apply DNN to malware analysis based on "exploit attack" and "privileged escalation", another category of malware detection relies on n-gram analysis on byte-code or op-code. For example, [18] and [19] first calculate the n-grams on the binary byte-code and then perform the malware detection based on k-nearest neighbor. [20] proposes to do reverse-engineering first and then analyze op-code. In addition, one more category of the malware detection relies on transforming malware into the images. For example, [21] proposes to first transform binary byte-code into grayscale image and then applies pattern recognition to the grayscale image.

All of the above methods achieve a certain level of detection accuracy. However, as mentioned in the introduction, the number of malware increased dramatically. Even worse, more and more anti-debugging techniques are discovered. The size of dataset used for training the model also has significant impacts on the detection accuracy and the computing efficiency in the training process. Here, we particularly note that despite the detection accuracy of the n-gram approach, N-gram approach consumes substantial computing resources and time for handling the dynamic growth of the model parameters required, implying the impracticality [22].

However, if we have limited computing resources and time, CNN is able to handle the explosive data growth because the increased number of parameters does not imply the growth of computing resource and time required. Recently, [23] also proposes deep learning-based malware detection, where the sequences of the op-code are encoded as one-hot vectors for the input of CNN. However, this method needs to dissemble the Android apps via reverse-engineering tools (backsmali) for deriving smali source code from classes.dex, and therefore cannot handle malware with encryption and obfuscation. It also requires a huge amount of human labor to be spent on feature engineering and detection modeling. To ease the model training, we adopt the above deep learning approach to construct an end-to-end learning-based Android malware detection. Our R2-D2 system possesses the following advantages:

- **R2-D2** translates classes.dex, the core of the execution logic of Android apps, into RGB color images, without modifying the original Android apps and without extracting features from the apps manually in advance. It can complete a translation from execution code to image within 0.4 second. Such translation is also featured by the fact that more complex information in the Android apps can be preserved in the color image with 16777216 colors (each sampling with 24 bit pixels) compared to the grayscale image with only 256 colors (each sampling with 8 bit pixels).

- With the fully connected network infrastructure of DNN, it can even deal with fast-changing malware with its large amount of parameters, however, the local receptive fields and shared weights of CNN make it more suited for more complex structure. It not only decreases the amount of parameters, but also reflects the complexity of Android malware, saving the time for huge computation with current method.

- The auto-image feature extraction in CNN do not extract features directly from image for pattern recognition. Instead, the raw pixels are represented by multi-dimensional matrices. Then through the calculation of filter and the size of stride in convolution layer, and non-linear activation functions in pooling layer, it can enhance the permutation relation between data.

- We only need the classes.dex in Android App, so if there is unknown new App appears, we only need to transform it to Android color image. The image size is about 10-50kb. Compared to uploading the App to the back-end
and then processing the extraction and identification, R2D2 can not only save the traffic loading on user side, but also reduce the resources and speed up the processing efficiency. We will further explain the system architecture and its process in section III.C and Fig. 13.

B. The Core Technology of Our Methodology

In the followings, we explain the algorithm procedures of R2-D2 in more details. First of all, we collected the Android apps which from our original back-end classification system, the Android apps were classified as benign and Trojan, RiskTool, HackTool, AdWare, Banker, Clicker, Downloader, Dropper, FakeAV, Monitor, SMS, Spy, Ransom, Exploit, and BackDoor. Then, we decompressed the apps to retrieve the classes.dex and presented as byte-code. We then mapped the hexadecimal from byte-code to rgb color code through the rule (e.g., 646578 = (R:100, G:101, B:120), 0A3033 = (R:10, G:48, B:51), 3500D1 = (R:53, G:0, B:209) and 4B1222 = (R:86, G:87, B:120) etc.) shown in Fig. 8. Finally, we reached an Android color image (shown in Fig. 9 and 10, which were four different Android benign and malware apps), and the images are fed to CNN and training a model to detection Android malware.

![Fig. 8. The result of present byte-code as rgb color code.](image)

Moreover, we have an image distance test on our Android malware image, where samples of the same Android malware family sharing similarity in their visual patterns are close to each other in the sense of distance from Levenshtein, RMS deviation and MSE image distance which validated our proposed Android color image algorithm that is suitable for the classification with CNN (e.g., 00ee9561c5830690661467cc90b116de of Android:Jisut-JY [Trj] from AVAST and cda4f446c3e1076ab48540e2283595ac of Android.Trojan.SLocker.IS from BitDefender are 54.59% similar, 0ce90908c2fb8f9b31da3afe05eb3427 of Trojan-Ransom.AndroidOS.Congur.aa of Kaspersky and ce2fed0ca9327b8d52388ec11ff3b4ca of Android.Trojan.SLocker.IS from BitDefender are 54.94% similar etc., shown in Fig. [11]). Although in the fine-grained sense is not accurate, Android malware image visual mode of this similarity will help us to quickly classify Android malware, greatly reducing labor costs.

However, we also found that two approaches might be used to escape our Android malware detection.

- Since the traditional filter size of CNN is 3*3 or 5*5, the uncorrelated bytecode might become correlated when we transform classes.dex into images. The malware may evade the detection by taking advantage of such a mismatch.

- Pooling is a common approach in CNN model to reduce the computation overhead significantly in traditional image recognition. The detection engine in our original research inherently uses pooling to achieve the speedup. However, Android color images are not natural images; instead, they are formed from Android source code. Thus, the pooling inevitably destroys the contexts and semantics of the malware code, causing the detection inaccuracy.

- Noted that CNN trained model is not suited to be embedded in Android App for malware detection due to the reason that the file is too large.

To address the above two issues, we did many experiments with CNN models (includes AlexNet, VGG, GoogleNet, and Inception-v3). We found the characteristics of 1x1 convolution in Inception-V3 (shown in Fig. [12]). It replaces few filters with a smaller perceptron layer with mixture of 1x1 and 3x3 convolutions, and 1x1 convolutions are specially used before 3x3 and 5x5 convolution to reduce the dimensions. In this way, we can add more non-linearity by having ReLU immediately after every 1x1 convolution and reduce the dimensions inside
this “inception module”. Based on [27], 1x1 convolution is equivalent to cross-channel parametric pooling layer, and this cascaded cross channel parametric pooling structure allows complex and learnable interactions of cross channel information. Cross channel information learning (cascaded 1x1 convolution) is biologically inspired because human visual cortex have receptive fields (kernels) tuned to different orientation.

C. The Architecture of Our Methodology

Fig. 13 shows our system architecture. Step 3-6 was mentioned in III.A. Users do not need to upload to consume network traffic. It can improve the efficiency of processing and reduce the computing resources. It can also get rid of the drawbacks of CNN mentioned in III.B while the training model is too large to run on user side.

- Step 1. User scans the Apps on the Android device.
- Step 2. If the Apps are all identified. The scanned results will be provided to the users directly.
- Step 3. If there is unknown App, the system will transform the classes.dex into Android color image.
- Step 4. Then upload the Android color image to the backend.
- Step 5. Feed the Android color image to the GPU computing pool of tensorflow.
- Step 6. Identify the Apps with the trained Inception-v3 model.
- Step 7. Send the results to user’s phone.

IV. EXPERIMENT RESULT
A. Experiment Environment and Datasets

Fig. 14 shows the hardware setting and software library used in our experiment. From our research cooperate partner’s Leopard Mobile Inc. It’s core products have reached 3,810 million installations globally with 623 million monthly active users each month by December 2016. We can collect 10k benign and 10k malicious samples daily in average. The data collection duration was from October 2016 to May 2017, among which we had a collection of approximately 2 million of benign and malicious Android apps for our experiments.

We particularly noticed that the malware may have different variants and mutations, depending on the factors such as the cellphone model, Android version, and the orographic regions. Fig. 15 shows the world trend of different malware families, and Fig. 16 shows the trend of different malware families particularly in China, Indonesia and India. Fig. 17 shows the market share of different cellphone models. Based on the above statistics, we confirm that even the same malware family will exhibit different behaviours in different geographic
regions. According to our experiment as follows that our dataset can handle these problems as above.

B. Evaluations of different deep neural network optimization

Based on our collected data, we evaluate the detection accuracy and performance with different network models (e.g., Alexnet, Googlenet, Inception-v3) and optimization. Noted that the learning rate is fixed to be 0.01, the optimization methods used are stochastic gradient descent (SGD), Nesterov Accelerated Gradient (NAG), AdaDelta and AdaGrad. From our experiment, we found that Inception-v3 (shown in Fig. 18) is almost always better than Alexnet (shown in Fig. 19) and other models. It also verified that adopting Inception-v3 can solve the drawbacks that mentioned before in our \textit{R2-D2} method. With such observation, we further fine-tune and compared Inception-v3 with AdaDelta and Inception-v3 with AdaGrad (see Fig. 20), and found that SGD is best suitable for our use (see Fig. 21). In particular, it resulted in the sharpest increase in accuracy and sharpest decrease in loss. As a result, we reach 98.4225\% and 97.7081\% accuracy (see Fig. 22).

C. Validation on Real Environment

To infer the capability of the \textit{R2-D2} system in the detection of unknown malware, we collected Android apps...
on Google Play in February 2017. More specifically, we collected Android apps in different categories (e.g., weather, business, finance, travel, etc.) from different countries (e.g., the US, UK, France and Germany, etc.). In addition, we also choose benign samples that none of VirusTotal (https://www.virustotal.com) vendors report malicious. We also selected malicious samples, the criteria for selecting malicious samples is that more than 30 VirusTotal vendors report malicious. We downloaded some malicious samples from contagio (http://contagiominidump.blogspot.tw). In sum, the above sources of benign/malicious samples are used for verifying the detection capability of our R2-D2 system. The evaluation metrics in our experiment include True Positive (TP), False Positive (FP), False Negative (FN), True Negative (TN), Accuracy (Acc), Precision (Prec), Recall (Detection Rate, DR), False Positive Rate (FPR) and F1-score (F-measure). The evaluation results are shown in Fig. 21 that with rapid generation and mutation of malware, even the detection model is trained based on our dataset with 1.5 million samples, the precision is higher than accuracy after the threshold 0.6. Moreover, when the threshold is over 0.7, recall and F1-score start to drop, meaning that the detection accuracy is not as good as before. This phenomenon can confirm that though R2-D2 is able to reduce the human labor and resource consumption, long-term sample collection and model updating are still necessary.

D. Real Case study

We further compare our detection results to the results reported by VirusTotal (Google’s malicious program detection website) and various anti-virus software engines on VirusTotal site. We collected 87 Minecraft apps that are verified by ESET as malware from Google Play. These apps had been active on Google Play since January 2017, and the numbers of installations have achieved more than 1 million. The most prominent feature of these apps after installation is the extra module for downloading, and the request for “Device Administrator permission”, promoting deceptive advertisements. Google Play has removed these apps. However, if they can be detected in the first place without feature extraction, Android malware can be mitigated.

From the results shown in Fig. 23, we can find that most of the above Minecraft malwares can be detected by approximately 16% vendors (10/60) on 2017/03/24. Until 2017/03/30, approximately 32% vendors (20/60) can detect these malwares. However, R2-D2 can detect more than 75% of these apps. Our method can not only reduce the resources that are consumed by traditional manual feature extraction method or machine learning method, but also can better detect unknown malware before other malware detection engines can.

E. Comparison with Existed Methodology

Finally, we compare our R2-D2 to DroidSieve [23]. The evaluation material includes the sample size, DR/FPR, acc., and detection time. From the results shown in Fig. 24, we find that though R2-D2 is slightly weaker than some others in DR/FPR and Acc. This is due to that our R2-D2 is trained based on a significantly larger dataset and is more adaptive to the real-world adoption. Others are trained with rather smaller dataset that cannot reflect the real environment. Moreover, R2-D2 has distinguishing advantage that it has fast detection time, 0.5 seconds for each coming samples. Besides, it does not need to go through manual feature extraction engineering. Asides from that, it takes only 0.4 second for R2-D2 to transform an Android app to color image.

V. CONCLUSION

This research adopts deep learning to construct an end-to-end learning-based Android malware detection and proposed a color-inspired convolutional neural network (CNN)-based Android Malware detection, labelled as R2-D2. The proposed
proof-of-concept system has been tested in our internal environment. The results show that our detection system works well in detecting known Android malware and even unknown Android malware. Also, we have published the system to our core product to provide convenient usage scenarios for end users or enterprises. The future work is to reduce the complex task and train for higher performance in confronting the Android malware, avoiding from a huge amount of computation burden. The experiment material and research results are shown on the website http://R2D2.TWMAN.ORG if there are any updates.

ACKNOWLEDGEMENT
This work would not have been possible without the valuable dataset offered by Leopard Mobile, Cheetah Mobile and Security Master (a.k.a CM Security, an Android application). Special thanks to Dr. Chia-Mu Yu for his support on this research.

REFERENCES
[1] International Data Corporation (IDC), (2016), Smartphone OS Market Share 2016 Q3 [Online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp.
[2] The AV-TEST Institute, (2016) Security Report 2015/16 [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV\_TEST\_Security_Report_2015\_2016.pdf.
[3] Trend Micro, (2015) Continued Rise in Mobile Threats for 2016 [Online]. Available: http://blog.trendmicro.com/continued-rise-in-mobile-threats-for-2016

Fig. 24. The result of comparing R2-D2 and existing research.