 doi:10.20944/preprints201808.0034.v1

*Type of the Paper (Article, Review, Communication, etc.)*

# AN ANDROID MUTATION MALWARE DETECTION BASED ON DEEP LEARNING USING VISUALIZATION OF IMPORTANCE FROM CODES

**Yao-Saint Yen [1] and Hung-Min Sun [2]**

[1] National Tsing Hua University; ad732bcaacd1@gmail.com
[2] National Tsing Hua University; hmsun@cs.nthu.edu.tw

**Abstract:** Using smartphone especially android platform has already got eighty percent market shares, due to aforementioned report, it becomes attacker's primary goal. There is a growing number of private data onto smart phones and low safety defense measure, attackers can use multiple way to launch and to attack user's smartphones.(e.g. Using different coding style to confuse the software of detecting malware). Existing android malware detection methods use multiple features, like safety sensor API, system call, control flow structure and data information flow, then using machine learning to check whether its malware or not. These feature provide app's unique property and limitation, that is to say, from some perspectives it might suit for some specific attack, but wouldn't suit for others. Nowadays most malware detection methods use only one aforementioned feature, and these methods mostly analysis to detect code, but facing the influence of malware's code confusion and zero-day attack, aforementioned feature extraction method may cause wrong judge. So, it's necessary to design an effective technique analysis to prevent malware. In this paper, we use the importance of word from apk, because of code confusion, some malware attackers only rename variables, if using general static analysis wouldn't judge correctly, then use these importance value to go through our proposed method to generate picture, finally using convolutional neural network to see whether the apk file is malware or not.

**Keywords:** Android; Malware; Convolutional Neural Network

## 1. Introduction

Nowadays, smart mobile device gets more popularity, many of people use mobile device instead of traditional phones. According to IDC (International data corporation) website, the whole worldwide smartphone market grows 1.1% every year.

In another side of rapid growth, this information let malware producers want to steal the private information from all users. Nowadays, the mobile malware become serious issue that many researches try to detect and to lower the influence of malware in smartphones[2].

Some research finds out that most of method that made malware is to copy similar malware code or makes minor alterations and turn into similar function malware[3], so they might contain similar malware code.

In this paper, we propose a deep learning based method to detect android malware by extracting code from apk, and then find the importance of each word in total contents and then digitized it. By coding confusion, some malware attackers only rename the variable, if using general static analysis wouldn't determine correctly, so we arrange the word's order by it's important value, then separate certain quantity word's important value into multiple groups, each group contain certain quantity words, then using these groups to convert into image, finally using convolutional neural network to see whether the apk is malware or not.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 August 2018 doi:10.20944/preprints201808.0034.v1



## 2. Materials and Methods

### 2.1 Algorithm used in this paper

#### 2.1.1 Simhash

In computer science, SimHash is a technique for quickly estimating two similar files. Google Crawler uses this algorithm to find near duplicate pages. SimHash is made by Moses Charikar[10].

SimHash's step roughly divided into five steps, participle, hash, weight, combine, dimensionality reduction.

Participle, give a sentence, doing segment, then gets effective feature vectors, and weight each feature vector set(if it is given a text, then the feature vector is the text of the word, the weight is the number of times the word can appear). The importance of the numbers in parentheses represents the word in the whole statement, the larger the number represents more important.

Hash, through hash function to compute each feature vector's hash value. In this way, the string becomes a series of numbers.

Weight, based on hash value, weighted all feature vector.

Dimensionality reduction, for cumulative results of aforementioned, which will get the SimHash value from that sentence, so that we can according to different sentence's SimHash value to judge their similarity degree. In Figure: 2.1, show that the simple example how SimHash algorithm works.

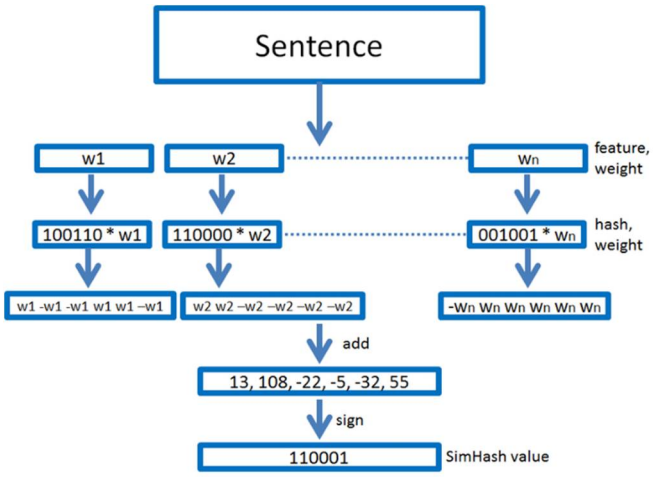


Fig 2.1: Example of SimHash

#### 2.1.2Djb2

Djb2[11] is a non-cryptographic hash function, made by Daniel J. Bernstein. It is based on module multiplication[12]. It is closely related to Rabin fingerprints[13], in fact, it can be seen as a special case of that.

### 2.2 System architecture

This part, we will introduce our overview of architecture, from turning apk file into code to generate image and through convolutional neural network to detect the result of training.

#### 2.2.1Apk file to code

To decompile apk, most popular method on internet belongs to Apktool[19], a tool to reverse engineering's 3rd party, closed, binary Android apps. Apktool can do decode resources to nearly original form and rebuild them even if making some modifications.

And dex2jar[20], a tool for converting Android's .dex file, which is in turn zipped into a single apk file on the device. .dex files can be created by automatically by Android translating compiled applications then written in Java programming language, to Java's .class file.

We use dex2jar to decompile apk file, after using dex2jar, apk file will turn into jar file(Java ARchive), which is a package file format that used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file [21]. When getting jar file, inside jar file, we extract the .class file, then we used Jad this tool to convert .class file to .java file, to complete this part. The process of this part is shown in Figure 2.2.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 August 2018 doi:10.20944/preprints201808.0034.v1



2.2.2Using code to generate image

We generated image from code by using the word importance from code, using method called TF-IDF(term frequency-inverse document frequency) to find out the importance value of word in every sample apk, when got every word's important value in apk, we packed words into multiple groups, the order of sequence inside groups and among groups is by the importance value of each word, when got whole groups in apk, we used aforementioned SimHash algorithm to generate x-axis and y-axis, and gjb2 algorithm to generate it's red, green and blue value. Finally, using the aforementioned data into convolutional neural network to observe how the result is, overall process shown in Figure 2.3.

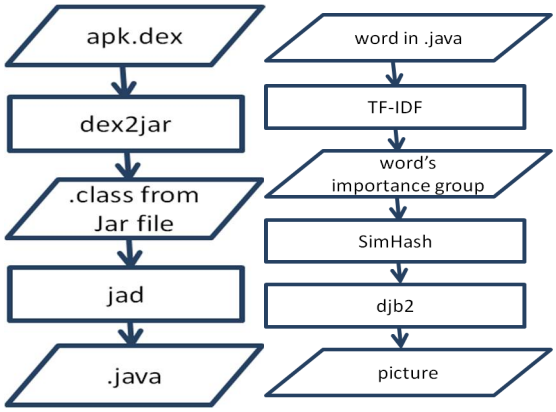


Fig 2.2: APK file to code




Fig 2.3: code to image

2.2.3TF-IDF

Term frequency-inverse document frequency is a commonly used weighting technique for information retrieval and text mining. It is a statistical method to evaluate importance of a word to one of the archives or one of the corpora. The importance of words increases in proportion to the number of times they appear in the archives, but at the same time declines inversely with the frequency of their occurrence in the corpus. TF-IDF weighted forms is often used by search engines as a measure or rating of the degree of correlation between files and user queries.

TF, term frequency, refers to the frequency of a given word in the file.

IDF, inverse document frequency, a given term can be divided by the total number of documents divided by the number of documents containing the word, and then resulting quotients is obtained by logarithms.

The high word frequency in a particular file, and the low file frequency in the entire set of documents, can produce a high weight TF-IDF. As a result, TF-IDF usually filter general words out and retain most important words.

2.2.4Implementation

In this part, we will go through flowchart of our proposed method. step by step. Figure 2.4 shows the flowchart of our paper.

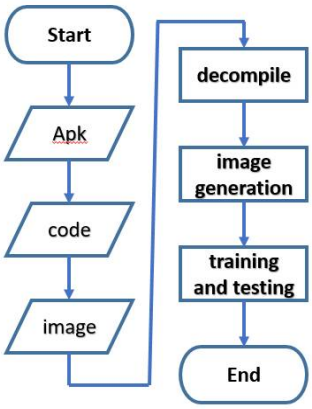


Fig 2.4: Flowchart

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 August 2018 doi:10.20944/preprints201808.0034.v1



Firstly, we collected total 1440 apk files, 720 for malware, and 720 for non-malware, we decompiled those apk files into codes using aforementioned method, dex2jar and jad tools, we put all apk files into directory, then using python script to successive executing dex2jar and jad tools for each apk file that in executed directory to generate the code we want, then put all java code that belongs to this apk into one txt file.

After generating all txt files in that directory, we used TF-IDF method to compute the important value of each word in txt file through whole txt files, in this part, we also use python script to do this work, after computing the important value of each word, we arrange the order by value's size, then we divide the whole important values from apk file into several groups, then take a certain amount of groups into next step, the step to produce pictures.

In this step to produce pictures, firstly, we take one by one group into Simhash algorithm, that algorithm will see that group as sentence, and gonna generate the X-axis and Y-axis of the picture. After generating the X-axis and Y-axis of the picture, then using the same group to go through djb2 algorithm to get red, green, blue value, forming pixel values, to provide a more convenient visual analysis, we put pixels around the pixel that generated coordinates that are recorded simultaneously, in Figure 2.5, shows how it works. If encounter overlap situation, we will choose the first one which comes earlier as the first priority, shown in Figure 2.6.

After all aforementioned steps, we got the image that we need, then we used convolutional neural network to train the network then test the network.

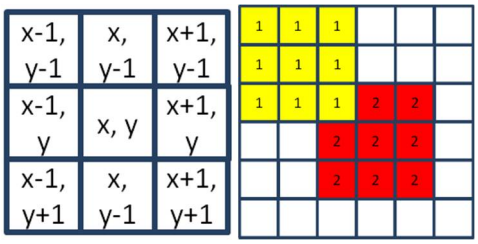


Fig 2.5: Nine pixels recording Fig 2.6:Overlapping pixels

The final image that go through aforementioned steps and came out like Figure 2.7 picture, we used lots of these pictures to train and to test the convolutional neural networks. We used 720 normal apk files, 500 for training, and 220 for testing, and 720 malware apk files, 500 for training and 220 for testing.

Fig 2.7: Example of generated image

## 3. Results

In this part, we will show the result of our proposed method, and compare to Ajit Kumar, K Pramod Sagar, K. S. Kuppusamy, and G. Aghila's method[22], which is used machine learning based method to detect android malware program that using analyze the visual representation of binary format apk file into several color formats.

Ajit Kumar, K Pramod Sagar, K. S. Kuppusamy, and G. Aghila's method uses several machine learning algorithms, such as decision tree[23], random forest[24], and k-nearest neighbors[25] algorithms to go through their proposed method. Then they observed that the best result is when using random forest, which can achieve average accuracy 86% for all color formats.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 August 2018 doi:10.20944/preprints201808.0034.v1



In this paper, we use Caffe[26], which is a convolutional neural network framework, with expression, speed, and modularity, it's developed by Berkeley AI Research. We used this tool to train and to test the convolutional neural network. In Caffe, it has to divide our data into two directories, training directory, and validation directory, show in Figure 3.1 and Figure 3.2.

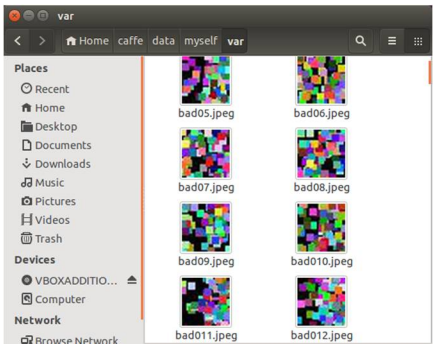


Fig 3.1: Directory of validation

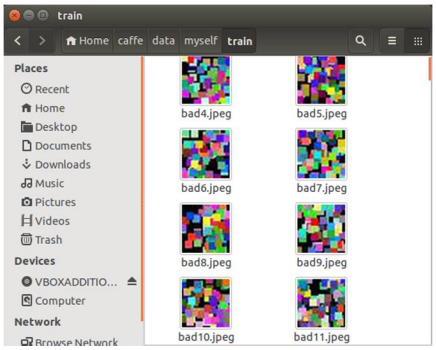


Fig 3.2: Directory of training

After dividing whole data into those directories, we need to tell Caffe which belongs to malware apk file's image, and which belongs to non-malware apk file's image by creating texture files, for testing, training, and validation, feed those information to Caffe.

After doing aforementioned steps, we start training our data, and for every 500 iterations will print the result of testing accuracy. Our result show in Figure 3.4, the first 500 iterations prints the accuracy is 80.33%, the second 500 iterations prints accuracy of 88.67%, and the third 500 iterations prints the accuracy is 91.55%, then it will gradually converge into 92.67%, better than Ajit Kumar, K Pramod Sagar, K. S. Kuppusamy, and G. Aghila's method. Figure 3.4 shows after iterating 1000 time's accuracy value, blue line is our proposed method, and orange line is Ajit Kumar's method.

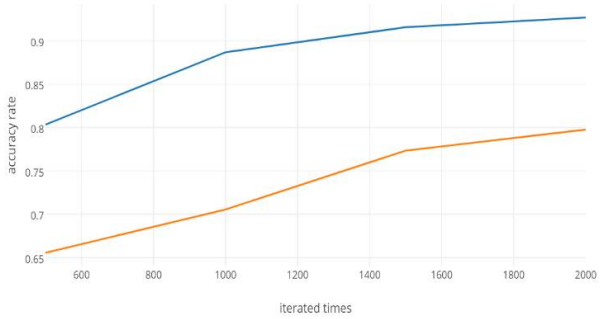


Fig 3.4: Testing result

 doi:10.20944/preprints201808.0034.v1



## 4. Discussion

Authors should discuss the results and how they can be interpreted in perspective of previous studies and of the working hypotheses. The findings and their implications should be discussed in the broadest context possible. Future research directions may also be highlighted.

## 5. Conclusions

This paper presented a method that can android malware using visualize the code's importance value into an image and then using convolutional neural network to train and to test the result, our result presented nighty two percent of accuracy, it's better than compared method, which using machine learning based method to detect android malware program that using analyze the visual representation of binary format apk file into several color formats, and it's result presented eighty six percent of accuracy..

## References


1. M. La Polla, F. Martinelli, and D. Sgandurra, "A Survey on Security for Mobile Devices," IEEE Commun. Surv. Tutorials, vol. 15, no. 1, pp. 446–471, 2013.
2. Cheng, Chien-Wen, "Identifying potentially malicious behavior inside android malware by static code analysis", 2013, retrieved from http://handle.ncl.edu.tw/11296/ndltd/44175699278460400644
3. G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. B. Alis, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," Expert Systems with Applications, 2013, in Press.
4. Q. Li and X. Li. Android malware detection based on static analysis of characteristic tree. In Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on, pages 84–91. IEEE, 2015
5. SimHash-Wikipedia, retrieved from https://en.wikipedia.org/wiki/SimHash
6. DJB2-Wikipedia, retrieved from https://en.wikipedia.org/wiki/DJB2
7. djb2-hashfunction, retrieved from http://www.cse.yorku.ca/~oz/hash.html
8. Rabin fingerprint-Wikipedia, retrieved from https://en.wikipedia.org/wiki/Rabin_fingerprint
9. K. Han, J. H. Lim, and E. G. Im, "Malware analysis method using visualization of binary files," Proc. 2013 Res. Adapt. Converg. Syst., pp. 317–321, 2013.
10. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," Proc. 8th Int. Symp. Vis. Cyber Secur., p. 4, 2011.
11. K. Kancherla and S. Mukkamala, "Image visualization based malware detection," Proc. 2013 IEEE Symp. Comput. Intell. Cyber Secur. CICS 2013 - 2013 IEEE Symp. Ser. Comput. Intell. SSCI 2013, pp. 40–44, 2013.
12. K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," Int. J. Inf. Secur., pp. 1–14, 2014.
13. S. Z. M. Shaid and M. A. Maarof, "Malware Behavior Image for Malware Variant Identification," 2014 Int. Symp. Biometric Secur. Technol., pp. 238–243, 2014.
14. Ajit Kumar, K Pramod Sagar, K. S. Kuppusamy, and G. Aghila, "Machine learning based malware classification for Android applications using multimodal image representations", Intelligent Systems and Control (ISCO), 2016 10th International Conference on, 2016
15. Decision tree-Wikipedia, retrieved from: https://en.wikipedia.org/wiki/Decision_tree
16. Random forest-Wikipedia, retrieved from: https://en.wikipedia.org/wiki/Random_forest
17. k-nearest neighbors-Wikipedia, retrieved from : https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
18. Caffe-Deep Learning Framework, retrieved from: http://caffe.berkeleyvision.org/