International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-10 Issue-1, November 2020

Visual Detection for Android Malware using Deep Learning

Ahmed Hashem El Fiky

Abstract*: The most serious threats to the current mobile internet are Android Malware. In this paper, we proposed a static analysis model that does not need to understand the source code of the android applications. The main idea is as most of the malware variants are created using automatic tools. Also, there are special fingerprint features for each malware family. According to decompiling the android APK, we mapped the Opcodes, sensitive API packages, and high-level risky API functions into three channels of an RGB image respectively. Then we used the deep learning technique convolutional neural network to identify Android application as benign or as malware. Finally, the proposed model succeeds to detect the entire 200 android applications (100 benign applications and 100 malware applications) with an accuracy of over 99% as shown in experimental results.

Keywords: Android Malware; Malware Detection; Visual Analysis; Deep Learning; Image Processing.

I. INTRODUCTION

With the rapid development of mobile internet, mobile devices, especially smartphones, are not only as important tools for people to communicate with the outside world but also as personal digital assistants or enterprise digital assistants to organize or plan their users' work and also private life. So, mobile security has become increasingly important in mobile computing. The professional virus attacks over the year have been significantly upgraded, such as the abuse of system vulnerabilities, security reinforcement technology, especially social engineering. Although the number of malware is increasing every year, most of the variants are modified or generated based on the original malicious [1]. In most cases, hackers use automated or module reuse tools to generate malware automatically. These variants often share the same ancestor's work, resulting in a high score of similarity among variants. So, how to recognize the fingerprints among different variants is the main task we should pay more attention to. About the mobile operating systems, there are Android, iOS, Symbian OS, and Windows phones. Considering the open-source and widespread applications, in this paper, we mainly focus on the Android system.

Revised Manuscript Received on November 30, 2020.
***** Correspondence Author
Ahmed Hashem El Fiky *, Systems and Computer Engineering Dept., Faculty of Engineering, Al-Azhar University, Cairo, Egypt. Email: 0x4186@gmail.com

We introduce a new method to classify the android applications. For each android application sample, we decompile the application and extract the Opcode feature and map it into R channel of RGB image, API package feature into G channel and risk API function feature into B channel, we merge three channels into one combined feature image and use the deep learning algorithm for classification android applications. The rest of the paper is organized as follows: we give a brief introduction to the related work in Section 2. Our main contributions and image processing technique are described in Section 3 and 4. In Section 5, The experimental results are shown. Section 6 shows conclusion and future work.

II. RELATED WORK

The methods of malware detection can be usually divided into two categories: static analysis and dynamic analysis. In static analysis, the codes of the sample are examined comprehensively by decompiling or disassembling the malware binary files without executing it, which can prevent OS from malicious damages. The advantages of static analysis are low cost and require less time of analysis. But the disadvantages of static analysis are limited signatures database and can't detect zero-day malware. Hence, in the most, static analysis is a trivial task since hackers use code obfuscation, such as encryption, binary packers to evade detection. Dynamic analysis can analyze the behavior of the malware during executing it in a debugger. Currently, sandbox-based dynamic analysis is one of the most popular solutions. A sandbox executes a sample of malware in an Isolated environment that can record and monitor system calls and behaviors dynamically. The core limitations of dynamic analysis are more time and power consumption.

A. Static Analysis Approach

The static analysis method is an approach to determine the functionalities and maliciousness of a malware sample by analyzing and disassembling its source code, without executing the malware.

In [2] proposed DroidMat, the authors provide Android malware detection through Androidmanifest.xml file and API call tracing.

They extract the information (e.g., requested permissions, Intent messages passing, etc.) from each application's manifest file, and regards components (Activity, Service, Receiver) as entry points for tracing API calls related to permissions. They apply K-means and KNN algorithms to classify the Android application as benign or malware.

They collected 1,738 Android applications (1,500 Android benign apps and 238 malware apps) to test DroidMat. Finally, they achieved accuracy rate 97.87% for detecting Android Malware.

In [3] presented Permission Usage to detect Malware in Android (PUMA) as a new approach for detecting Android Malware APKs using machine learning by analyzing the extracted permissions from the application itself.

They collected 4,301 Android Malware samples and 1,811 benign apps.

They used several machine learning techniques for malware detection, including Naïve Bayes, Simple Logistic, Bayes Net, SMO, J48, IBK, Random Forest, and Random Tree.

At the end, they performed an analysis of the extracted permissions from Android apps and achieved accuracy 92% for detection.

In [4] proposed the combination of permissions and API calls and they used several machine learning methods for Android malware detection.

In their approach, the permission is extracted from each App's profile information and APIs are extracted from the packed App file by using packages and classes to represent API calls. They gathered 2,510 Android applications (1,250 benign and 1,260 malware). Their proposed model achieved a precision of up to 94.9%.

B. Dynamic Analysis Approach

In dynamic analysis, the detection phases and training happen during the execution of the app. Aside from the ability to detect unknown malware, this feature also allows zero-day attacks to be detected.

In [5], the authors main goal is to protect mobile device users and cellular infrastructure companies from malicious applications. So, they proposed a new behavior-based anomaly detection system for detecting meaningful deviations in a mobile application's network behavior.

The detection is performed based on the application's network traffic patterns only. For each application, a model represented its specific traffic pattern is learned locally.

They used semi-supervised machine learning methods for learning the normal behavioral patterns and for detection deviations from the application's expected behavior.

In [6], the authors proposed a combination of network traffic analysis with machine learning methods to identify malicious network behavior in highly imbalanced traffic and eventually to detect malicious apps.

They collected dataset from the Drebin Project. Where they captured traffic from over 5,560 mobile malware samples. So, they build a tool to convert mobile traffic packets into traffic flows.

Finally, the accuracy rate of machine learning classifiers can reach up to 99.9%. However, the performance of the classifiers declines when the imbalanced problem gets worse.

In [7], the authors proposed a lightweight framework for Android malware detection.

Their proposed method is a combination between network traffic analysis and a machine learning algorithm (C4.5) that is capable of identifying Android malware with high accuracy. Where, Network traffic generated by the mobile app is mirrored from the wireless access point to the server for data analysis.

During the evaluation process, they tested their model with 8,312 benign apps and 5,560 malware apps. So, their results show that the proposed model performs betters than state-of-the-art approaches.

Finally, when combining two detection mechanisms, it achieves a detection rate of 97.89%.

So, there have been some relative works on Android malware using static or dynamic methods. But different from the existing work, our contributions are as follows:

1) Proposing a new model recognizes Android malware without understanding its source code and execution behavior.
2) Using visualization techniques to detect the Android App as benign or malware.
3) Integrate the Opcodes sequence features, API Package Name features, and high risky API calls features to three different RGB channels.
4) Using Convolution Neural Network to train the Android apps to feature images and get excellent detection results.

III. VISUAL REPRESENTATION OF ANDROID APPLICATION

The authors proposed in [8], A method for visualizing and classifying malware using image processing techniques, which transform windows platform malware binaries to gray-scale images. In 2015, [9] won the championship of Kaggle, a famous Microsoft Malware Classification Challenge. Interestingly, the championship team with three people did not engage in security, and the methods used are very different from our common methods.

They use gray-images, n-gram, and PE-header features, and use machine learning to classify the malware.

Without understanding the malware's source code, it shows great potential that image-based methods for detection malware.

Based on this idea, we propose a method which mapping Android applications to RGB images.

Where, we extract three features: Opcode, sensitive API packages, and high level risky API functions. Next, integrate them into one RGB color image.

Finally, we will use image processing technique to classify Android malware.

A. Proposed Model Architecture

As shown in Figure 1 the working flow of our model. Firstly, we decompile the application and extract the Opcodes; Secondly, mapping the different Opcodes to pixels in the R channel of the RGB image, and then coloring some sensitive API packages in the G channel.

Highlighting the risky API functions in the B channel. Thirdly, merging the three R, G, B channels to generate the feature image.

Finally using deep learning convolution neural network to classify the android applications images as benign or malware.

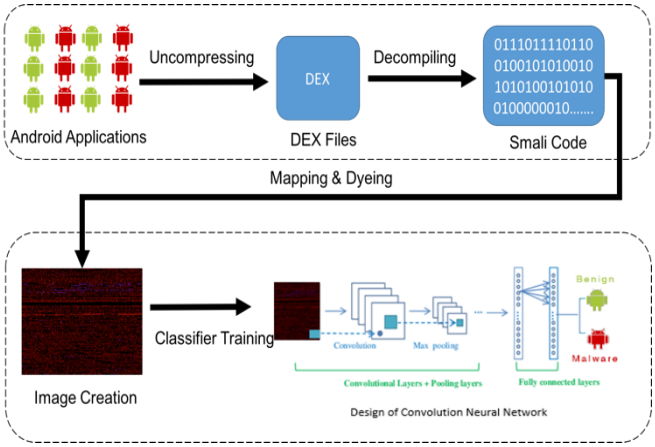

Figure 1: Android Malware Detection Model

B. Android APK Decompilation

An Android APK is usually written in java and compiled to Dalvik bytecode which is contained in a .dex file. This file can be just-in-time compiled by the Dalvik virtual machine or compiled once into a system-dependent binary by ART (Android Runtime) platform. The APK file contains the Dalvik executable .dex file. Androidmanifest.xml is used to describes the content of the package, including the permissions information. Also, it contains the digital certificates for authentication and the resources the app uses, for instance, image files, sounds, etc.

We used Apktool [10] to decompile an APK file. The decompiled code can be later modified and repacked again with the same tool. This process is represented in the diagram in Figure 2. Apktool outputs are Androidmanifest.xml file and class files of the application are also decompiled in a different format called smali [11] code (assembler/disassembler for the dex format used by Dalvik).

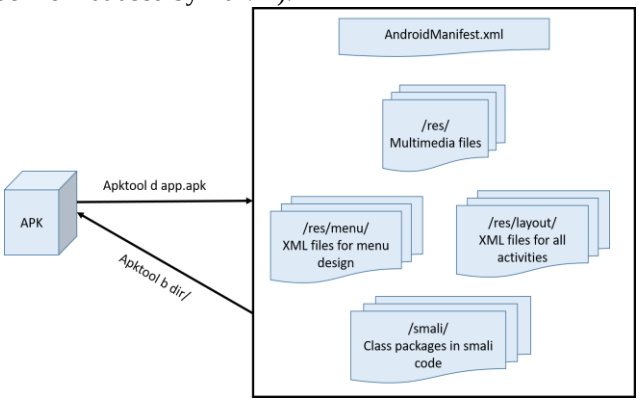

Figure 2: Diagram representation of Apktool decompile and compile option on any APK file

C. Opcode Feature Extraction

Feature extraction is the core for Android Malware Detection. We try to use an RGB image to describe Android Malware features in this paper. Besides that, once we got the featured images, we can use some image processing techniques, i.e., deep learning, to classify the Android images as benign or malware. First of all, we should map the Opcode into the R channel of the RGB image. Android OS has a total of 255 Opcodes coding from the 0x00 to 0xFF according to different functions [12].

Here, we adopt a method similar to [8], mapping the Opcode to pixels by converting its hex value (encoded in Android OS) to decimal value. An example is given in Figure 3.

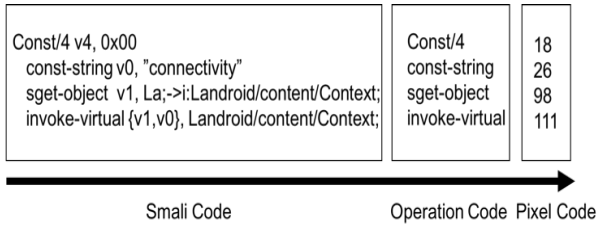

Figure 3: An example of mapping the Opcodes to pixel values in the R channel of an image

D. API Colouring

API (Application Programming Interface) is a set of protocols, functions, and tools for building software applications. API calls are applied in Android application development to implement functionalities conveniently. For example, if we want to get the phone number, we should call android.telephony.TelephonyManager → getLine1Number.

In Android OS, the API calls are usually given in parameters of a function call instruction as shown in Figure 4.

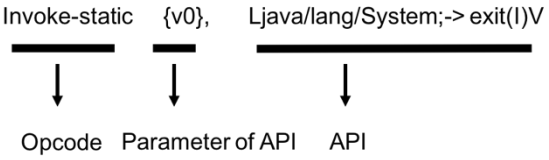

Figure 4: An API call example of Android Structure

To extract the API call features, we divide them into 58 classes by their packages. Among them, 18 classes are related to high-level security that is mapped to the G channel. In contrast, the other classes 40 are related to high-level risky methods that are mapped to the B channel of the RGB image as in [13].

E. RGB Image Creation

In the previous section, we extract Opcodes, API packages, risky API functions, and map them to different pixel values. After that, we should integrate all the three features into an integrated RGB image as combined features. Figure 5 shows an example of the RGB image Creation. While Algorithm 1 shows how to map Opcode to RGB image.

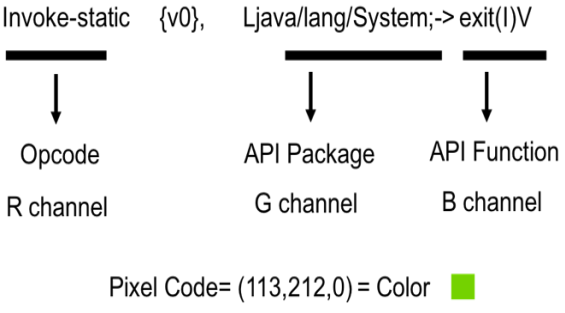

Figure 5: An example of RGB image Creation

Algorithm 1: Map Opcode to RGB image

```
Input Opcode, API Package, and API Function
Output RGB image
Initialization i=0
while Opcode file not end do
    RPixel = Call getRPixelValue(Opcode)
    if API Package is belong to high level security API Packages then
        GPixel = Call getGPixelValue(API Package)
    else
        GPixel = 0
    end
    if API function is belong to high level risky API functions then
        BPixel = Call getBPixelValue(API Function)
    else
        BPixel = 0
    end
    RGB pixel [i] = Call merge(RPixel, GPixel, BPixel)
    i++
end
```

IV. IMAGE PROCESSING TECHNIQUE

We adopt a Convolutional Neural Network (CNN) to classify the previous Android images as benign or malware in this paper. The CNN structure illustrated in Figure 6. Our architecture of CNN consists of two sets of convolutions, and pooling layers, followed by a fully-connected neural network classifier.

The convolution layer will learn 32 convolution filters, where each filter is of size 3x3. If each input feature image is shown as x_i, learnable weight value w_{ij}, then the output feature image is:

$$y_i = b_i + \sum_i w_{ij} * x_i$$

Where "*" is a convolution operator, and b is a learnable bias parameter.

The purpose of the convolution layer is to extract different features from the input layer. The first layer can only extract convolution low-level features such as lines, angles, edges, etc., more layers of the network can extract more complex features from low-level features in an iteration.

The output feature image adopts as activation function R = h(y) for non-linear mapping. In this paper, we use the ReLU activation function because it gives us much better classification accuracy. The ReLU is defined as:

$$f(y) = \max(0, y)$$

The activation function can enhance the non-linear characteristic of the decision function and the whole neural network but does not change the Convolution layer itself.

The ReLU activation function followed by 2x2 max-pooling in both x and y directions with a stride of 2 to reduce training parameters. It divides the input image into several rectangular regions and outputs the maximum value of each subregion. The max-pooling layer will constantly reduce the size of the data space, so the number of parameters and the amount of computation will drop. Also, it can control the overfitting during training to a certain extent. Typically, the CNN convolutional layer is periodically inserted into the pooling layer. After the second subsampling layer (S2), we flatten the output feature image to vector and link it to two fully connected layers whose dimensions are 128, 1 (binary classification) respectively. The first fully connected layer adopts ReLU as activation function and the last one (Loss Layer) uses Sigmoid, which is defined as:

$$s(x) = \frac{1}{1 + e^{-x}}$$

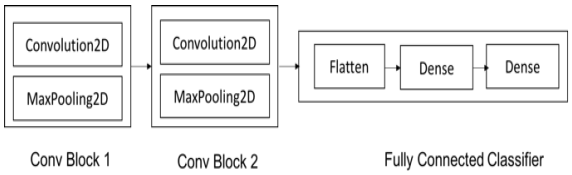

Figure 6: Structured of our used CNN network

V. EXPERIMENTS AND ANALYSIS

The experimental dataset is downloaded from the CICAndMal2017 dataset [14]. The dataset contains 5,491 Android applications (5,065 benign and 426 malware). We choose 200 Android applications (100 benign and malware) the malware category is from Ransomware. The experimental programs are written in Python, and the hardware environment is Intel Core i7 and 16 GB Ram. In the proposed Model, log loss is used as an evaluation measure. Log loss is the cross-entropy between correct labels and predicted labels. The exact formula for the log loss evaluation has been shown in the following:

$$Logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{M} y_{ij} \log(\rho_{ij})$$

Here, N is the number of samples and M is the number of classes. y represents the true label of the class and ρ is the probability of the given sample. A log that is mentioned in the above formula is Natural Logarithm.

Accuracy is another evaluation measure used for evaluating the performance of the proposed model mathematically accuracy can be defined as below:

$$\frac{TP + TN}{TP + FP + TN + FN}$$

In the above equation, TP and TN are the number of positive and negative samples, respectively which are correctly classified by the classifiers. Whilst, FP and FN are the number of positive and negative samples, respectively, which are misclassified by the classifiers.

Table I shows the performance of the proposed model in terms of Log loss and accuracy against 10 independent runs.

Table I: Performance of the proposed model

Epoch	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
1	0.2873	0.8970	0.5561	0.9500
2	0.1167	0.9600	0.6658	0.9000
3	0.0756	0.9776	1.7611	0.7000
4	0.0587	0.9821	1.1063	0.9000
5	0.0473	0.9853	1.0038	0.9500
6	0.0398	0.9868	0.9285	0.9500
7	0.0360	0.9880	1.0277	0.9000
8	0.0297	0.9900	0.9621	0.9500
9	0.0279	0.9905	0.8924	0.9000
10	0.0295	0.9895	1.1223	0.9500

Figure 7 illustrates the relation between the number of trials (Epoch) and loss related to the training process and the validation process.

In begin, train loss and validation loss stood at 0.23 and 0.55 respectively. Then, validation loss increased gradually from epoch 1 to epoch 2 and reached 0.6. In contrast, train loss decreased dramatically in the period of epoch and reached 0.1. Also, validation loss reached to the top 1.8 at epoch 3. While train loss decreased to 0.07 at the same epoch. Afterward, validation loss decreased gradually from epoch 3 to epoch 4 and reached 1.1. At the same time, train loss decreased dramatically to 0.05. Finally, validation loss fluctuated from epoch 4 to epoch 10. Whereas, train loss decreased and stable at 0.02 at the same time.

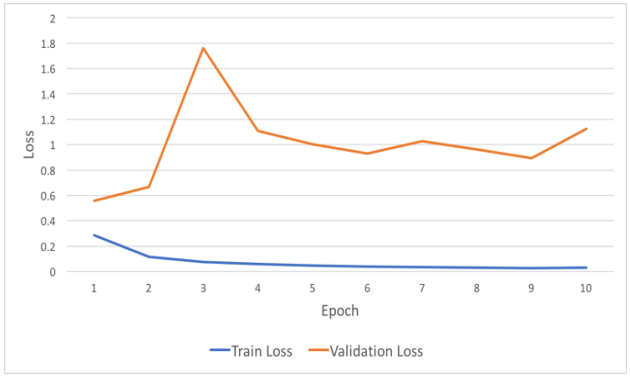

Figure 7: The relation between epoch and loss

Figure 8 shows the relation between epoch and accuracy-related to train process and validation process.

At first, train accuracy and validation accuracy stood at 0.89 and 0.95 respectively. While train accuracy increased dramatically and reached 0.96 from epoch 1 to epoch 2. In contrast, validation accuracy decreased gradually and reached 0.90 at the same time. Then, validation accuracy dropped sharply and reached 0.70. while train accuracy increased and reached to 0.97. After that, validation accuracy fluctuated from epoch 4 to epoch 10 and reached 0.95 at the end. Whereas, train accuracy increased dramatically and reached 0.99 at the same time.

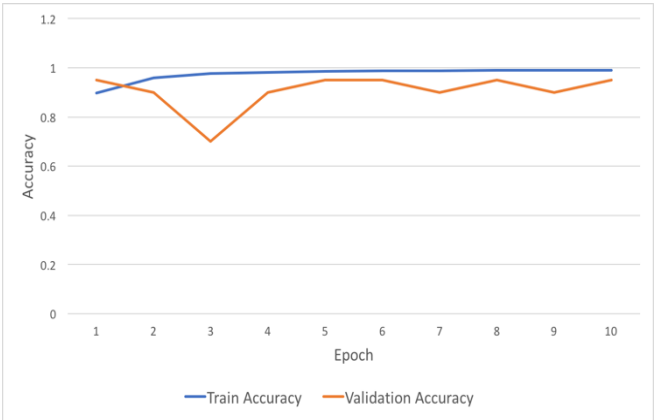

Figure 8: The relation between epoch and accuracy

VI. CONCLUSION

In this paper, we propose an Android malware detection algorithm that combines the Android application visualization method and deep learning techniques. First of all, we extract the Opcode features, API calls packages, and high-level risky API function features. Then we adopt a convolution neural network to train the images and classify the Android applications. The experimental results show that the classification accuracy can be 99%. About the future work, we can integrate the proposed static method with dynamic analysis to extend the robustness and adaptability of the detection system.

REFERENCES

1. M. Fossi, D. Turner, E. Johnson, T. Mack, T. Adams, J. Blackbird, S. Entwisle, B. Graveland, D. Mckinney, and J. Mulcahy, "2010 symantec internet security threat report," Volume, no. 5, pp. 277-278, 2011.
2. D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droid-Mat: Android Malware Detection through Manifest and API CallsTracing," AsiaJCIS, pp.62–69, 2012.
3. B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas, and G. A´ lvarez, "PUMA: Permission Usage to Detect Malware in Android," Advances in Intelligent Systems and Computing, vol.189, pp.289–298, Jan. 2013.
4. N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp.300–305, 2013.
5. A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile Malware Detection through Analysis of Deviations in Application Network Behavior," Comput. Se- cur., vol.43, pp.1–18, 2014.
6. Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, and B. Yang, "Machine learning based mobile malware detection using highly imbalanced network traffic," Inform. Sciences., vol.433-434, pp.346–364, 2018.
7. S.Wang,Z.Chen,Q.Yan,B.Yang,L.Peng,Z.Jia,"A mobile malware detection method using behavior features in network traffic," J. Netw. Comput. Appl., vol.133, pp.15–25, 2019.
8. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath," Malware images: Visualization and automatic classication," in Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 4, 2011.
9. https://www.kaggle.com/xiaozhouwang/competitions
10. Xdadevelopers Forum (2015). „Apktool v2.0.2.– A tool for reverse engineering of apk files", [Online]. Available: http://forum.xdadevelopers.com/showthread.php?t=1755243
11. Github. (2015). „smali" [Online]. Available: https://github.com/JesusFreke/smali
12. https://android.googlesource.com/platform/dalvik/+/kitkat-release/opcode-gen/bytecode.txt
13. Yong-liang Zhao and Quan Qian, "Android Malware Identification Through Visual Exploration of Disassembly Files" International Journal of Network Security, Vol.20, No.6, PP.1061-1073, Nov. 2018 (DOI: 10.6633/IJNS.201811_ 20(6).07) 1061
14. Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification", In the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada, 2018.

AUTHORS PROFILE

Ahmed Hashem El Fiky received the BSc degree in Computer Engineering Department Faculty of Engineering Helwan University May 2012 (Grade: Excellent and the first with honors) and Post-Grad Diploma Degree in Computer Engineering May 2015 (Grade: Very Good). He worked as a Teaching Assistant of Faculty of Engineering Helwan University Computer Engineering Department for 5 years ago (2013-2018). He is the author of several articles paper. Currently, He is an Information Security Team Leader at Tanmeyah Micro-Enterprise Services Company, Cairo, Egypt. and Also, He is a MSc student in Systems and Computer Engineering department of Faculty of Engineering Al-Azhar University. His main research interests are focused on Information Security, Network Security, Cryptography, Reverse Engineering, Malware Analysis, Digital Forensic and Space Science.