Falcon: Malware Detection and Categorization with Network Traffic Images

Peng Xu[1], Claudia Eckert[1], and Apostolis Zarras[2]

[1] Technical University of Munich
[2] Delft University of Technology

Abstract. Android is the most popular smartphone operating system. At the same time, miscreants have already created malicious apps to find new victims and infect them. Unfortunately, existing anti-malware procedures have become obsolete, and thus novel Android malware techniques are in high demand. In this paper, we present *Falcon*, an Android malware detection and categorization framework. More specifically, we treat the network traffic classification task as a 2D image sequence classification and handle each network packet as a 2D image. Furthermore, we use a bidirectional LSTM network to process the converted 2D images to obtain the network vectors. We then utilize those converted vectors to detect and categorize the malware. Our results reveal that Falcon could be an accurate and viable solution as we get 97.16% accuracy on average for the malware detection and 88.32% accuracy for the malware categorization.

Keywords: Malware Detection, Malware Categorization, bi-directional LSTM, 2D image sequence Classification

1 Introduction

As the most popular mobile operating system globally, Android has become the main target for many attackers who seek to exploit new victims. These adversaries leverage malicious apps to infect mobile devices to carry out miscreants' nefarious activities, such as sending spam emails, spreading new malware, generating revenue from online advertisements by performing click-frauds, or even tricking users into revealing personal and private data. On the other side, both industry and academia work on the domain of Android malware investigation, which includes malware detection and categorization in an attempt to mitigate the aforementioned phenomenon [7, 10, 14, 18, 20]. Many of the proposed approaches utilize the contextual information of Android applications (primarily Android APKs code). Chen et al. [8] propose a technique that examines Android malware based on its static behavior that involves the use of components, permissions, and sensitive *Application Programming Interface* (API) calls. Li et al. [14] introduce a classifier based on the *Factorization Machine* (FM) architecture, in which they extract numerous Android app heuristics from both the

manifest files and source code. However, both methods analyze the Android application statically without running the program. Gibert et al. [11] present a way to convert the executable files into a 2D image and achieve malware detection based on the 2D image classification.

Meanwhile, several works either utilize the Android dynamic features, which are generated by running the Android application in a sandbox [26,29] or capture the network traffic to detect legitimate and malicious behaviors [2,16,17,28,32]. The first approach is expensive because it monitors those running applications in different level calls (system-level, function-level, etc.) and performs several low-level operations during their running activities. In contrast, capturing network traffic to analyze the application's behavior is cheaper. However, most of the existing network traffic research is based on the manual indicated rules and builds rule-based classic machine learning classifiers (network port, deep packet inspection, statistical, and behavior-based features) to detect and categorize Android malware. Still, those methods face a new challenge which is how to pick up the appropriate features.

Representation learning [5], which can learn features from raw data automatically, has increasingly attracted researchers and engineers. It can solve the above challenge with the manual indicated methods. Wang et al. [28] present a representation learning method for malware traffic classification, which converts the raw network traffic/flow data to image and takes the converted images as the input. Then, it uses a *Convolutional Neural Network* (CNN) to extract features from the raw network traffic. However, converting the network flows to images, and pre-train the 2D-gray-image-sequence-based multi-class classification model, cannot classify those malware or benign samples based on each 2D gray image. Normally, each PCAP file includes hundreds or thousands of raw network packets and network flows. Therefore, the malware classification issue converts to a continuous 2D image classification task. In other words, that is a 2D image sequence classification or sequential image classification [4, 15]. Most of the sequential image classification works combine *Recurrent Neural Networks* (RNNs) and CNNs, as they put the RNNs focus on the sequential task and the CNNs on the image features. Meanwhile, in the *Natural Language Processing* (NLP) field, in order to process the sequential issues with a pre-trained model, BERT [9], GPT (v2, v3) [6,23], and other transformers (e.g., ELMo [22], Transformer [27]) capture the sequence relationship by leveraging *Long Short Term Memory* (LSTM) or RNN networks.

In this paper, we present *Falcon*, a network-traffic-pattern-based malware detection and categorization framework. We operate *Falcon* as follows. First, we convert the network packets to 2D gray images and leverage CNNs to pre-train the classification network for the network traffic features. We then use a bi-directional LSTM network to process the continuous network traffic and perform malware classification similar to the 2D image sequence classification task. The results of our system are promising since *Falcon* exhibits 97.16% accuracy on average for the malware detection and 88.32% accuracy for the malware categorization.

In summary, we make the following main contributions:

- We introduce *Falcon*, a network-traffic-pattern-based Android malware detection and categorization framework.
- We design a bidirectional LSTM network to accomplish 2D gray image sequence classification, which takes the network packets (converted to 2D images) as input.
- We create a dataset, *AndroNetMnist*, which includes 3,255,391 2D gray images in five classes for network traffic classification.
- We evaluate the accuracy of our approach using real-world datasets.

2 Related Work

With the increasing popularity of Android smartphones in recent years, the topic of detecting Android malware and categorizing its families attracts several researchers' and engineers' attention. As with every malware detection system, Android malware detection can be classified into two types: the traditional feature-codes-based method and the machine/deep-learning-based methods. Regarding the conventional feature-codes-based approach, the detector checks the classic malicious behaviors. For machine/deep-learning-based methods, there are also multiple features based frameworks. Permission-based malware detection extract several types of permission features that are highly relevant to the manifest file and source code of each mobile application, including API calls and permissions [14, 21, 30].

Program-code-based malware detection methods extract features from the code itself. Technically those features include the API calls, N-gram, and control flow graph (CFG) based methods. API call based malware detection uses API calls to detect Android malware [1, 3, 14, 21, 30]. In general, this type of method first constructs two ranked lists of popular Android APIs. One is benign_API_list that contains the top popular APIs commonly used in benign apps, and the other malicious_API_list that contains the top popular APIs commonly used in malicious apps. N-gram-based Malware Detection is based on the n-gram opcode to detect Android malware [12, 18, 24]. Last but not least, Graph-based malware detection systems use graph structure to perform their detection [10, 19, 31].

Machine learning and deep learning techniques are heavily introduced into the network traffic analysis. Researchers use manual indicated features (e.g., port, deep packet inspection, statistical and behavior-based features) to recognize network traffic application patterns with traditional machine learning algorithms [2, 13, 17, 25, 28]. Finally, Gibert et al. [11] present a way to convert the executable files to 2D images and achieve malware detection based on the 2D image classification, which is different compared to the 2D image sequence classification problem.

3 System Design and Implementation

In our work, we consider that network packets are composed of many network flows. Those flows are counted as a binary representation and can be converted to 2D gray images. Therefore, we transform a malware detection and categorization problem into a continuous 2D image classification and categorization problem. For instance, randomly choosing one network packet from our dataset, it includes 3,329 network flows. *Falcon* converts those network flows to 3,329 2D gray images and then to 3,329 vectors to represent those network flows. Finally, we take those converted vectors into our classifier to accomplish the malware detection and categorization tasks.

3.1 Overview

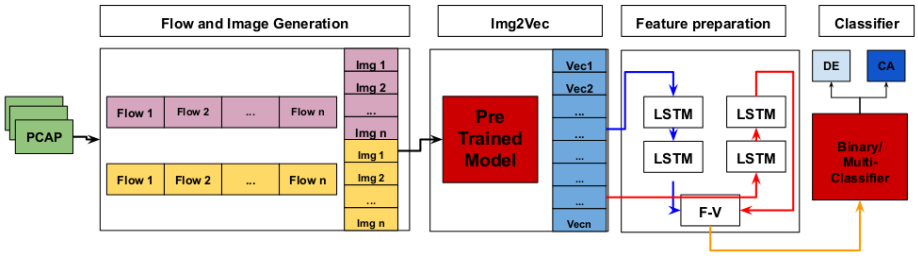

Fig. 1: The architecture of *Falcon*

The architecture of *Falcon* is presented in Figure 1. Our malware detection and categorization framework includes a bi-directional LSTM to prepare the feature vectors (F-V block in Figure 1) and a classifier to detect (DE block) and categorize(CA block) Android application. We input the *PCAP* files and convert each network flow contained in the PCAP file into a 2D image, and pre-train a model on 2D images with CNN network.[3] We use the pre-trained model to convert each 2D image to a vector and process the continuous network flows in a PCAP file as a 2D image sequence by a bi-directional LSTM network. We present this part in Section 3.2 in detail.

3.2 Features from Network Traffic

This section presents our method to convert network traffic to vectors based on image classification and transfer learning (see Figure 2). To compare to other works in this field, we have two challenges. The first challenge (`C1`) is how to classify each network flow (several network packets) efficiently, and the second one (`C2`) is how to classify the whole network packets based on the split flows.

[3] `https://wiki.wireshark.org/SampleCaptures`

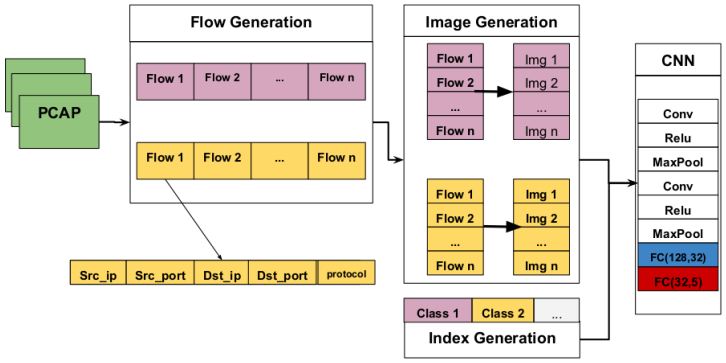

Fig. 2: Converting network traffic to vectors

Network Packets and Flows. For the network traffic analysis, there are three different granularity, raw packet level, flow level, and session level [28]. In our work, we take the network flow as our analysis target. All raw packets from the PCAP files are defined as a set $P = \{p^1, \cdots, p|P|\}$, and every packet is defined as $p^i = (x^i, b^i, t^i)$, where $i = 1, 2, \cdots, |P|$ and x^i stands for a 5-tuple, which includes source IP, source port, destination IP, destination port, and the protocol types (e.g., TCP, UDP), where b^i and t^i stand for the packet's size and the starting time of the packet, respectively. Network flow groups several packets that have the same 5-tuple. In this way, we solve the challenge `C1`. Meanwhile, for the network flow level analysis, it is shown as the flow generation in Figure 2. It is worth mentioning that we arrange all raw packets in the same network flow in time order.

Network Flows to Images. As we have previously mentioned, we split the network flow from the raw network packets. After getting network flow files, we convert them to 2D images like the image generation in Figure 2. Here we utilize trimming and padding methods to normalize all network flows that have the same size. If the network flow's size is larger than 784 bytes, we trim it to 784 bytes. If those flow files' size is smaller than 784 bytes, we pad them by 0x00 to 784 bytes. Finally, we convert those trimmed and padded files to 2D gray images. Each byte of the original file represents a pixel, such as 0x80 is gray, and 0xff is white. We also generate the class label in this step, which stands for the different network traffic classes. We define five different labels in our work because we have four various malware families and one benign group. That is reasonable for our malware categorization task. We pre-train the model indirectly for our malware detection task based on the previous malware categorization. In total, for the malware categorization task, we label all samples with five classes (four malware classes and one benign class) and label all samples with two classes for the malware detection task (malware and benign).

Transfer Learning and Feature Generation. In our work, we leverage an 8-layer convolution neural network to pre-train our converted 2D gray images. Our model has 70,213 total parameters. After the previous step, we transform our malware categorization and detection tasks into a 5-category classification problem.

$$\begin{aligned} Y^1 &= MaxPooling_{2*2}(Relu(conv2d_{3*3}(X_{28*28}))) \\ Y^2 &= MaxPooling_{2*2}(Relu(conv2d_{3*3}(Y^1))) \\ Y^3 &= FC_{128,32}(Y^2) \\ Y &= FC_{32,5}(Y^3) \end{aligned} \tag{1}$$

We use our 5-categories classification task to train the model. After getting the pre-train model, we take Y^3 that has a 32-bit vector as our features for the next step. We use *sparse_categorical_crossentropy* loss and Adam optimizer and set the learning_rate as 0.001 and epoch as 50. We use one dropout layer between MaxPool2 and FC1, and we set the dropout rate as 0.5.

Continuous Network Traffic Processing. So far, we have converted the network flows to images and pre-train the 2D gray image-based multi-class classification model. However, we cannot classify those malware or benign samples based on each 2D gray image for our malware detection and categorization task. Typically, each PCAP file includes hundreds or thousands of raw network packets and network flows. Therefore, the malware classification issue converts to a continuous 2D image classification task. In other words, that is a 2D image sequence classification or sequential image classification [4, 15]. Most sequential image classification works combine the RNN and CNN and put RNN focusing on the sequential task and CNN for the image features. Meanwhile, in the natural language processing (NLP) field, in order to process the sequential issues with the pre-trained model, BERT [9], GPT (v2, v3) [6, 23] and other transformers (e.g., ELMo [22], Transformer [27]) are introduced into to capture the sequence relationship by leveraging the LSTM or RNN networks. Therefore, in our work, to capture the network traffic's continuous characteristics, we introduce a bidirectional LSTM network on top of the pre-trained 2D-image classification model, which helps to extract image features from the converted network flows. This method can solve the `C2` effectively. Figure 3 presents our sequential image classification structure. The steps mentioned above prepare the image sequences and img2vec model, which replace each 2D gray image with a 32-bit vector. We take the 32-bit vectors from the second to last layer of the pre-trained CNNC model. We use a bidirectional LSTM network, and the input of LSTM has converted vectors with $(1, 32)$ shape. Both inputs for the forward and backward direction LSTM are the same. Furthermore, we concatenate the last hidden status $f_v, v \in N$ as our final output vectors, where N stands for the number of all PCAP files. After getting the f_v vectors for N PCAP files (N different Android samples), we use a full connection layer followed by a softmax layer to classify those raw network traffic into five different categories.

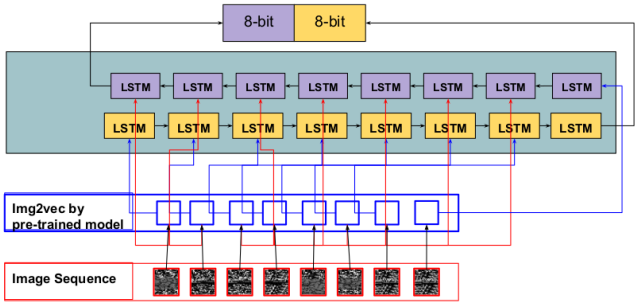

Fig. 3: 2D sequential image classification with bidirectional LSTM

3.3 Model Training and Prediction

After preparing the feature vectors by the bi-directional LSTM, we train and test our model by using the *sparse_categorical_crossentropy* loss function like Equation 2.

$$\begin{aligned} Loss &= -\sum_{N}^{i=1} y_{i_{label}} * log(y_{i_{pred}}) \\ &= -\sum_{N}^{i=1} y_{i_{label}} * log(< (< f_v, w_{i1} > + b_{i1}), w_{i2} > + b_{i2}) \end{aligned} \tag{2}$$

where $w_{i1}, w_{i2} \in R^p$ is the weight of the classifier and $b_{i1}, b_{i2} \in R^p$ is the offset from the origin of the vector space. In this setting, a converted vector f_v is classified into five categories.

4 Evaluation

4.1 Experimental Setup

We set up our experiments on our Euklid server, which runs on a Linux X86_64 platform and has 128 GB RAM and 16 GB GPU. Further, we trained our model with `Tensorflow 2.0.0-beta0`, `Keras 2.2.4`, and `Sklearn 0.20.0`. We also used the `SplitCap` tool to split the PCAP files.[4] Additionally, we used the `pillow 6.1.0` imaging library when we convert the network flows to images. Finally, we used other assistant libraries, such as `numpy 1.16.4` and `matplotlib 3.1.1`.

4.2 Dataset

For the train and evaluation dataset, we used the Android Malware CICMal2017 dataset [13, 25]. It includes 426 malware and 1700 benign samples and their

[4] https://github.com/Master-13/SplitCap

Table 1: Dataset explanation

Name	Description	Number
PCAP files	*All the raw network traffic files*	2,126
Network flows	*All network flows in Section 3.2*	3,255,391
Adware	*Adware network flows partition*	580,170
Ransomware	*Ransomware network flows*	382,279
Scareware	*Scareware network flows*	517,954
SMSmalware	*SMSmalware network flows*	245,691
Benign	*Network flows for benign applications*	1,529,297

corresponding network traffic raw files. Table 1 illustrates the number of various categories in detail. For the network traffic, we extracted 3,255,391 network flows in total from 2,216 PCAP files. Here, to pre-train our 2D gray image classification task, we created our dataset, *AndroNetMnist*, which provides a benchmark to network traffic analysis with the convolution neural network. We split the dataset with 80% training and 20% testing in our experiment.

4.3 Results Comparison

This section compares our results with other related works, both from the program code and network traffic-based field. We reimplemented (Droidmat [30] and CICMal2017 [25]) and reproduced (Drebin [3][5], Adagio [10][6]) other related works and compared them with our framework. We should mention here that the results of those frameworks differ a little from the original works because of the different datasets.

For *Falcon*, after preparing the dataset as CSV files, we used the *Random Forest* (RF) classifier by default to perform our malware detection and malware categorization. Our RF is defined as 1,400 trees in the forest and 80 as the tree's maximum depth. We set min_samples_split as five and the number of features to consider when looking for the best split as sqrt. Table 2 illustrates the malware detection (binary classification) performance.

Table 2 shows that *Falcon*-CNN gets the best performance, which catches up to 98% accuracy. However, this experiment processes the malware classification on *AndroNetMnist* similar to the digital handwriting classification on the MNIST dataset, which indirectly did the classification. That means we firstly extract and convert all network flows to images and then classify all images that belong to one class. For example, for a PCAP file, we extracted and converted network flows to images and got 3,329 samples. The 98% accuracy means 98% of 3,329 samples are classified correctly. However, we cannot determine the whole network flows characteristics because most malicious behaviors are hidden in a few network flows by sophisticated attackers. Even if we get a high performance of over 98%,

[5] https://github.com/alisakhatipova/Drebin
[6] https://github.com/hgascon/adagio

Table 2: Malware detection comparison

Classifier	**Accuracy**	**Precision**	**Recall**	**F1**
Drebin [3]	96.58	95.37	97.85	96.59
Adagio [10]	89.32	91.27	95.28	93.23
Droidmat [30]	89.87	90.89	88.28	89.56
CICAndMal2017 [13]	87.52	87.14	87.73	87.18
Falcon-CNN	98.04	98.09	98.05	98.06
Falcon	97.16	97.13	97.16	97.09

we cannot infer that this malware detection system can accurately determine the malware's network traffic. Therefore, we introduced *Falcon*, which converts all 2D images to a 2D image sequence for each PCAP file. With this method, *Falcon* gets 95.39% accuracy. The results are illustrated in Table 2 in detail.

In our experiment, the malware categorization task is a multi-class classification issue. Similar to the malware detection (binary classification) task, *Falcon*-CNN on *AndroNetMnist* gets the best performance on the image classification task indirectly. Take the same example with malware detection above; 97.23 accuracy means that 97.23% of images from the same PCAP file are classified to Adware class. However, we cannot determinedly infer that this PCAP is Adware network traffic. Additionally, we compared our results only with CICAndMal2017 [13] because most Android malware detection works, such as Drebin, Adagio, and Droidmat did not consider the malware categorization problem. Although FM [14] considers the malware categorization task, it converts the multi-class task to binary-class (i.e., if one malware sample belongs to a specific malware family, then the label is 1; otherwise, that is 0). *Falcon* on the multi-class classification task gets better results than CICAndMal2017. The primary reason is that essential patterns for various malware families represent the manually indicated features by CICAndMal2017 that lose some information. Our method can catch up with better malware families' features by representation learning. Table 3 shows the performance results of malware categorization.

Table 3: Malware categorization comparison (the average is weighted)

Classifier	**Accuracy**	**Precision**	**Recall**	**F1**
CICAndMal2017 [13]	86.85	85.92	86.85	84.82
Falcon-CNN	97.23	97.28	97.23	97.24
Falcon	84.70	80.22	84.70	82.39

Last but not least, besides the Random Forest (RF) classifier, we also consider the other four classifiers by Sklearn implementation for malware detection and categorization. The classifiers are described in Table 4. Besides the settings in

Table 4: Various classifiers settings

Classifier	**Settings**
RF	*n_estimators=1400, min_sample_split=5, max_features="sqrt", max_depth=80*
AdaBoost	*All default values*
GradientBoost	*lr=0.01, n_estimators=1500, max_depth=4, min_samples_split=40, max_features=4*
MLP	*sover="sgd", alpha=1e-5, hidden_layers_sizes=(400,400,200,100,10)*
DecisionTree	*min_samples_split=10, max_features="sqrt", max_depth=20*

Table 5: *Falcon*'s performance with various classifiers

Classifier	**Accuracy**	**Precision**	**Recall**	**F1**
RF	97.16	97.13	97.16	97.09
AdaBoost	93.13	92.81	93.13	92.85
GradientBoost	96.88	96.83	96.88	96.80
MLP	91.01	90.48	91.01	90.02
DecisionTree	93.66	93.64	93.66	93.65

Table 4, we used all default parameters. Due to the limited space, we only present results in Table 5 for the malware detection (binary classification) task. Table 5 shows that the RF classifier gets the best performance and then is followed by the GradientBoost classifier. MLP gets the worst in our framework.

5 Limitations

In contrast with other machine and deep learning based works in the malware detection field, *Falcon* can catch up with the dynamic information of the Android application. Our work has more time consumption than port-matching or permission matching systems to contrast with other rule-based methods, such as port-based malware detection with network traffic and permission-based Android malicious program detection. On the other hand, in contrast to other Android malware detection works, the dataset, especially the dynamic network-traffic dataset, is too small. Although our evaluation demonstrates better performance than its precedent, we need to increase the number of samples in the future.

6 Conclusion

In this work, we present *Falcon*, a network-traffic-pattern-based malware detection and categorization framework. We use the transfer learning method to extract features from the network traffic with pre-trained models. We treat the network-traffic-based classification as a 2D gray image sequence classification task and use a bi-directional LSTM to process image sequences. For the 2D gray image, we use an 8-layer CNN to pre-train the gray images, which stand for the network flows.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 883275 (HEIR) and No. 833115 (PREVISION).

References

1. Aafer, Y., Du, W., Yin, H.: Droidapiminer: Mining Api-Level Features for Robust Malware Detection in Android. In: International conference on security and privacy in communication systems (2013)
2. Arora, A., Garg, S., Peddoju, S.K.: Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices. In: International Conference on Next Generation Mobile Apps, Services and Technologies (2014)
3. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In: Network & Distributed System Security Symposium (NDSS) (2014)
4. Bai, S., Kolter, J.Z., Koltun, V.: Trellis Networks for Sequence Modeling. arXiv preprint arXiv:1810.06682 (2018)
5. Bengio, Y., Courville, A., Vincent, P.: Representation Learning: A Review and New Perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
6. Budzianowski, P., Vulić, I.: Hello, It's GPT-2–How Can I Help You? Towards the Use of Pretrained Language Models for Task-Oriented Dialogue Systems. arXiv preprint arXiv:1907.05774 (2019)
7. Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., Visaggio, C.A.: Effectiveness of Opcode Ngrams for Detection of Multi Family Android Malware. In: International Conference on Availability, Reliability and Security (2015)
8. Chen, C., Liu, Y., Shen, B., Cheng, J.J.: Android Malware Detection Based on Static Behavior Feature Analysis. Journal of Computers **29**(6), 243–253 (2018)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805 (2018)
10. Gascon, H., Yamaguchi, F., Arp, D., Rieck, K.: Structural Detection of Android Malware Using Embedded Call Graphs. In: ACM workshop on Artificial intelligence and security (2013)
11. Gibert, D., Mateu, C., Planes, J., Vicens, R.: Using Convolutional Neural Networks for Classification of Malware Represented as Images. Journal of Computer Virology and Hacking Techniques **15**(1), 15–28 (2019)
12. Kang, B., Yerima, S.Y., Sezer, S., McLaughlin, K.: N-Gram Opcode Analysis for Android Malware Detection. arXiv preprint arXiv:1612.01445 (2016)
13. Lashkari, A.H., Kadir, A.F.A., Taheri, L., Ghorbani, A.A.: Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In: International Carnahan Conference on Security Technology (2018)
14. Li, C., Zhu, R., Niu, D., Mills, K., Zhang, H., Kinawi, H.: Android Malware Detection Based on Factorization Machine. arXiv preprint arXiv:1805.11843 (2018)
15. Li, S., Li, W., Cook, C., Zhu, C., Gao, Y.: Independently Recurrent Neural Network (Indrnn): Building a Longer and Deeper Rnn. In: IEEE Conference on Computer Vision and Pattern Recognition (2018)

16. Malik, J., Kaushal, R.: CREDROID: Android Malware Detection by Network Traffic Analysis. In: ACM Workshop on Privacy-Aware Mobile Computing (2016)
17. Marín, G., Casas, P., Capdehourat, G.: DeepMAL–Deep Learning Models for Malware Traffic Detection and Classification. arXiv:2003.04079 (2020)
18. McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., et al.: Deep Android Malware Detection. In: ACM Conference on Data and Application Security and Privacy (2017)
19. Narayanan, A., Soh, C., Chen, L., Liu, Y., Wang, L.: Apk2vec: Semi-Supervised Multi-View Representation Learning for Profiling Android Applications. In: 2018 IEEE International Conference on Data Mining (ICDM) (2018)
20. Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G.: MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). ACM Transactions on Privacy and Security (TOPS) **22**(2), 1–34 (2019)
21. Peiravian, N., Zhu, X.: Machine Learning for Android Malware Detection Using Permission and Api Calls. In: IEEE international conference on tools with artificial intelligence (2013)
22. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep Contextualized Word Representations. arXiv preprint arXiv:1802.05365 (2018)
23. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving Language Understanding With Unsupervised Learning. Technical report, OpenAI (2018)
24. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware Detection by Eating a Whole Exe. arXiv preprint arXiv:1710.09435 (2017)
25. Taheri, L., Kadir, A.F.A., Lashkari, A.H.: Extensible Android Malware Detection and Family Classification Using Network-Flows and Api-Calls. In: 2019 International Carnahan Conference on Security Technology (ICCST) (2019)
26. Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: Copperdroid: Automatic Reconstruction of Android Malware Behaviors. In: Network & Distributed System Security Symposium (NDSS) (2015)
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention Is All You Need. In: Advances in neural information processing systems (2017)
28. Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y.: Malware Traffic Classification Using Convolutional Neural Network for Representation Learning. In: 2017 International Conference on Information Networking (ICOIN) (2017)
29. Wong, M.Y., Lie, D.: IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. In: Network & Distributed System Security Symposium (NDSS) (2016)
30. Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P.: Droidmat: Android Malware Detection Through Manifest and Api Calls Tracing. In: Asia Joint Conference on Information Security (2012)
31. Xu, P., Eckert, C., Zarras, A.: Detecting and Categorizing Android Malware With Graph Neural Networks. In: Annual ACM Symposium on Applied Computing (SAC) (2021)
32. Zulkifli, A., Hamid, I.R.A., Shah, W.M., Abdullah, Z.: Android Malware Detection Based on Network Traffic Using Decision Tree Algorithm. In: International Conference on Soft Computing and Data Mining (2018)