International Journal of Engineering & Technology, 7 (4.29) (2018) 101-103

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Deep Convolutional Generative Adversarial Networks for Intent-based Dynamic Behavior Capture

Salman Jan[1*,2], Shahrulniza Musa[1], Toqeer Ali[3], and Ali Alzahrani[3]

[1]Malaysian Institute of Information Technology,
[1]Universiti Kuala Lumpur
[2]University of Peshawar
[3]Islamic University of Madinah
**Corresponding author E-mail: jan.salman@s.unikl.edu.my*

Abstract

Malware analysis for Android systems has been the focus of considerable research in the past few years due to the large customer base moving towards Android, which has attracted a corresponding number of malware writers. Several techniques have been used to detect the malicious behavior of Android applications as well as that of the complete system. Machine-learning techniques have been used in the past to assess the behavior of an application using either static or dynamic analysis. However, for large scale Android malware analysis traditional machine learning techniques are not feasible. In this regard, many deep neural architectures have used static analysis. It has been shown that static analysis techniques can leave many malicious behaviors of an application unnoticed. In this paper, we used a new deep-learning architecture known as deep convolutional generative adversarial networks to measure the dynamic behavior of Android applications. More- over, we used the notion of Android intents as the parameter to measure the dynamic behavior of an application. We gathered a large set of intent-based behavior from more than 4,000 infected applications as well as 10 thousand applications' good behaviors on our modified Oreo version of Android. We received an F1 score of 0.996 and AUC curve of 0.993, which is almost the same as those received by many state- of-the-art works using machine learning.

Keywords: *Android security, Malware detection, Deep Learning, Generative Models, DCGAN,*

1. Introduction

The basic intension behind the development of malware was indeed to test the knowledge and technique employed for particular software. Before 1980s, malware was not developed to hide personally or organizationally sensitive details, nor were they profit-driven. Mostly they were developed by human without using tools to generate automated samples. A 2013 report disclosed that mobile malware developers were earning up to $12,000 USD each month [1]. These monetary incentives caused PC malware developers to pro- duce millions of viruses, while, until 2009, only 1,000 malware samples were known [2], [3]. Stolen information regarding vulnerabilities is sold [4]. As per report of Symantec, more than 430 million new malware were reported in the year 2015. Furthermore, as available on statistics page of VirusTotal, there are over a million of newly retrieved samples that had to be analyzed [5], [6]. Symantec disclosed that on average, 272 new malware programs and 5 new malware families targeting Android are discovered every month [7]. Further details regarding Android malware vulnerabilities can be found in [8], [9] and in Figure 1.

Currently, Android malware detection is based on static, dynamic and hybrid approaches. Many machine learning techniques are used to classify benign and malicious applications on Android platform. For example, DREBIN [10] one of the successful work done in the recent past to classify the behavior of the application on certain matrix, such as, permissions required by an application, API calls between the applications and middle- ware etc.

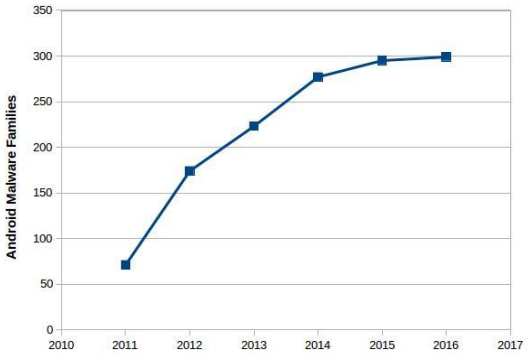

Fig. 1: Android Malware Families

They gathered a large dataset of 52 GB of behavior of 16 million benign applications' behavior and around 4,000 malware samples. However, our work is different from DREBIN in certain ways i.e., we are considering the dynamic behavior of the applications to classify intents while the DREBIN conducts static analysis of applications. In addition, we considered large-scale classification models that work on very large datasets. In contrast DREBIN worked on traditional classification models, such as, SVMs which cannot be scaled to very large datasets. TaintDroid [11] is another quite famous work done on behavior tracking at various levels of the Android software stack. TaintDroid use ma- chine learning model to identify the bad and good behavior. TaintDroid works on out-of-the- box analysis technique to measure the run-time behav-

ior of an app. However, one of our concerns regarding this kind of approach is its feasibility in the new Android version. Because of the many architectural changes in Android, such as its permission models, Delvik replacement with Android Run Time (ARM) etc., this technique is no longer workable. Similarly, this work is classifying the behavior on traditional machine learning model. A stepforward, there are few deep neural architectures models are recently employed that includes the Convolutional Neural Networks (CNNs) [12], Fully Connected Neural Networks (FCNN) [13], the deep belief networks (DBNs) [14], Deep Autoencoders [15], and recurrent neural networks (RNNs) [16] in the form of long short term memory (LSTM) [17] models that can detect malware with very low false positives rate. However, that works on static analysis techniques and is not designed for dynamic analysis. Another recent work done by Nauman et al. [18] has focuses on how to deal with large scale datasets. They utilized various deep leaning models to capture good and bad behavior. However, their work is also based on static techniques. This means that they are not executing the application and capturing its behavior based on certain parameters. In this paper, we have decided to capture the intents of an application to classify whether its behavior is malicious or benign. The selection of Intents as the behavior of an application instead of system calls or other matrices was made to keep the simplicity of architecture as well as due to the various quick architectural changes that have been made to the Android framework. As Intent is the very basic element of an Android framework, and this notion cannot be changed because of backward compatibility of the old applications as well as the fact that changing the programming paradigm for Android developers would be big a concern for Google Inc. Our contribution in this paper is twofold.

• We modified the recent Android OS (Oreo) to intercept the Intents of an application and generate very large dataset that is testable on deep learning model. Also, the Android was modified to capture the behavior of an application at run-time and detect whether an application is performing any malicious activity. If so, our system can generate an alarm to warn the user about unknown activity.

• Second, we utilized deep convolutional generative adversarial network (DCGAN) to generate a dynamic behavior analysis model of an Android application. Through our primarily results, we proved that DCGAN can be an effective model for detection of malware and its adversarial examples and can pro- duce high rate of accuracy on the Android platform. The manuscript provides a detailed back- ground in section II that essentially explains the evolution of malware, various malware analysis techniques, and malware analysis taxonomy. In section III, we explain the methodology and elaborate on the proposed machine learning DCGAN for Android security domain. We further provide experimental details and results in Section III (B) and conclude the paper in section IV.

2. Background

In this section, we provide background details regarding Android security. With the increase in Android malware samples, security experts provided appropriate remedial measures and developed a number of techniques over time. In the following section, we explain various methods for malware analysis.

2.1. Methods for Malware Analysis

Malware analysis is roughly classified into static, dynamic, and hybrid approaches [19], [20], [21], [22], [23]. In the following subsections, we review each of these approaches.

1) Static Malware Analysis: The static analysis does not execute the application for analysis purpose. One of the static analysis tools includes PEInfo [24] which can extract information or properties from malware code to characterize malware samples. Static analysis of android applications is also done through the use of application permissions. Permissions, such as SEND SMS are an important feature for analysis as most actions require particular permissions in order to be invoked [18], [25]. As an example, before accessing the camera, the Android system checks if the requesting application has the CAMERA per- mission [26]. These requested permissions must be declared within the AndroidManifest.xml. As the manifest is easy to obtain statically, many frameworks, such as PScout [27], [26], [28], use static analysis to evaluate the risks of the Android permission system and individual applications. However, intruders can manipulate or obfuscate the malicious code such that the extraction of information becomes difficult [29], [20]. Second, static analysis does not consider network activities and objects that are modified at runtime (also termed as reflections) since the effect of these modifications can only be viewed at run- time. An example of such a method is static taint analyser, [30], which detects privacy leaks among components in Android applications.

2) Dynamic Malware Analysis: Researchers have proposed dynamic behaviour analysis mechanisms that records traces of an activity during the execution of a malware sample in a controlled virtual environment. However, it is not guaranteed that a malware will execute un- expected in controlled environment as certain malware require a particular condition to occur or become unpack itself. A dynamic approach is complement to the static technique as it is less liable to obfuscation [31]. Both types of analysis help us understand the risks presented by and intentions of the attacker. One of the dynamic methods includes TaintDroid [11], which monitors the third party Android applications for possible misuse of private data. TainDroid keeps eyes on privacy sensitive data and how the downloaded applications use personal data of the users. Similar work is done by [32] known as FlowDroid and DroidScope [33], which monitors profile API-level activity, and track information leakage on the android while research conducted by [34] informs users about hidden behaviour pertaining to applications. SCANDROID [35] provides incremental approach for checking installed applications and confirming that the data flowing through these applications are intact.

3) Hybrid Malware Analysis: Approaches that utilize the strengths of both static and dynamic approaches are referred to as a hybrid analysis. In [36], [37], the authors in static manner included hooks in security sensitive functions and APIs in order to capture behaviour for onward dynamic analysis. While in [38], the authors statically used intents to know all possible paths of execution to examine behaviour. However, all of these approaches take up over- head over Android performance and cannot detect perturbations made to the benign and malicious datasets. In the upcoming sections, we provide details of our proposed solution which is a dynamic approach and consider Intents for malware analysis.

3. Proposed System

In the following section a detailed discussion on the proposed solution is provided which is further divided in two sections. The first explains the dataset collection for DCGAN classification and the second discusses proposed framework.

3.1. Dataset of Intents

Before elaborating how the Intents Dataset is generated, the role of Intents in Android is de- tailed. Intents are data structures holding descriptions about an operation which is required to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService (Intent, ServiceConnection) to communicate with a background Service. Intents facilitates in late runtime binding between the code among applications. Each intent contains action and data pieces of infor-

mation The Action refers to the general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc. The data part of an intent specifies the data to operate on, as an example users record in contacts database. Few more examples of Action and data includes e.g. ACTION_VIEW content://contacts/people/2: provides details of person with identifier "2". Similarly ACTION_DIAL tel: 333: Displays the phone dialler with the given number. Similarly we have a number of intents for carrying out various tasks on Android platform. Since our framework considers using Intents for classification of application, therefore, we require a dataset of Intents for dynamic analysis. A sample of few intents are numbered as shown in table 1. We explain how data set of intents is constructed in upcoming section. The widely used Drebin [10] and other similar Android

Table i: Mapping of Intents to its corresponding number value

Number	Intent	Description
…	…	…
1	Action_Main	Start as a main entry point, does not expect to, receive data.
160	Action_View	Display the data to the user.
150	Action_Attach_Data	Used to indicate that some piece of data should be attached
210	Action_Dial	Dial a number as specified by the data.
03	Action_Call	Perform a call to someone specified by the data
…	…	…

Datasets [39] do not possess runtime behavioural data and mostly have static information regarding applications, while this studies aims at analysing the dynamic behaviour of application on Android. For analysing runtime behaviour of applications, a dataset was generated based on Android Intents. The Oreo source code was downloaded, and hooks are utilized in order to capture all possible behaviours an application could exhibit during its runtime. Around 10 million application's data are collected that were downloaded using (GPlayCLI) [40]. We installed these applications over our modified Android OS to generate very large dataset of Intents generated by these apps. Hooks are placed in Android OS and stored the behaviour in log files to initially learn via our discriminative model. The collected data from various applications behaviour received around 5GB of intents sequences. We applied DCGAN model on our dataset by dividing it into training and test sets. The dataset is classified into training set, and test sets with ratio of 2/3 and 1/3 respectively at random. That is performed to ensure learning and to avoid memorization and look up table. If we use whole dataset as training set it will be like look up table and model will not perform well when real data is provided to it. Machine might have saved the results somewhere in memory. The k-fold cross validation concept was applied for providing more rigorous sampling and to overcome anomalies within datasets.

3.2. Proposed Framework:

Use of Generative Model We propose use of one of a cutting-edge concept in deep learning i.e. generative and discriminative models specifically the Deep Convolutional Generative Adversarial Networks (DCGANs) [41], [42], [43], which have recently gained marvellous acceptance in comparing adversarial examples. It has a pair of models the Generator and Discriminator as depicted in Figure 2.

A generative model tries to learn the joint probability of the input data and labels simultaneously, i.e. P(x, y). The generative model creates likely new (x, y) distribution samples, while the discriminative model maps labelled inputs (x) to class output labels (y).

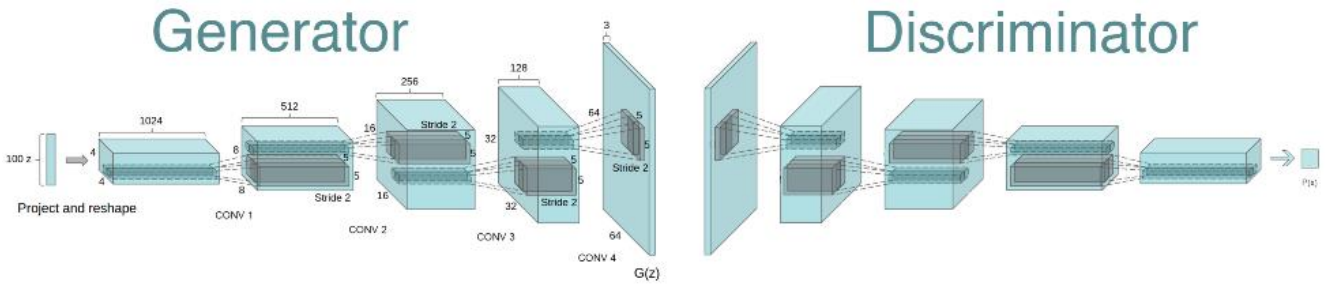

Fig. 2: Layers of generator and discriminator in Deep Convolutional Generated Adversarial Network [42]

In other words, they learn the probability distribution Prob (y | x). The generator network generates more realistic examples while the discriminator network learns to get more and better in recognizing the true data from generated one. Both the networks learn to get better using learning parameters whose values are updated through gradient descend algorithm used in the DCGAN. The discriminator update function is represented as:

$$\nabla\theta_d \frac{1}{m}\sum(i=1)^{m}[\log D(x^{(i)}) + Log(1 - D(G(Z^{(i)})))] \quad (1)$$

While the generator update mathematical function is:

$$\nabla\theta_d \frac{1}{m}\sum(i=1)^{m} \log(1 - D(G(Z^{(i)}))) \quad (2)$$

We propose to use the generator to receive sequence intents as a vector of input (x), learn parts of input representations, and generate specific data/intents that are similar to the input in order to generate variation in the input. The Discriminator which is trained on original inputs (in our case benign intents) verifies whether the input sequence is original or it's generated (possibly different patterns of malware). Polymorphic malware are mostly generated version of existing malware that are modified. Mathematically the cost function of the DCGAN is represented as [41]:

$$\text{minmaxV}(G,D) = E_{x\sim p}\log D(x) + E_{z\sim pz}z \log(1 - D(G(z))) \quad (3)$$

This study contributes by training a DCGAN on recorded benign sequences of intents i.e. benign behaviours in order to execute various permissions and to protect execution of unwanted combinations of sequences of intents to ensure security of applications and data. The DCGAN, which is proven to perform very well in the domain of vision, determines the underline patterns among various permissions and generates alarm in case of manipulated

intents or for those permissions that were not granted for specific tasks.

3.3. Experimental setup Required Tools:

The following mandatory con- figuration was made for implementation of DC- GAN:

1. Pandas and Numpuy: In order to handle data we used Pandas library, while the Numpy is used for numeric computations.

2. Theano: It has the capability to produce optimized implementations related to mathematical functions that are based on multi-dimensional tensors. Theano compiles C low level language code for symbolic computation. Based on the features it offers theano is configured in many domains in the literature for deep learning [44] TensorFlow is a similar library presented by Google [45].

3. Keras, a library that is used for Deep Learning and creation of Neural Networks. It accepts one of the two backends i.e. Tensorflow and Theano. We use Theano as it is more stable.

4. Matplotlib: For plotting and visualization.

4. Results

At last we might want to convey our best outcomes to take note. Using Deep Convolutional Generative Adversarial Networks, on a reasonably large dataset, we obtained the best possible results, which obviously beat other deep learning models trained on a dataset. Similarly, we claim that DCGAN is quite appropriate technique that is used to detect malicious behaviour deviating from normal for an Android app. DCGAN accomplished the best F1 score of 0.996, a score better than any state-of-the-art announced outcomes. Moreover, we obtained 0.988 accuracy, 0.993 precision and FPR of 0.002, which is quite acceptable. The malware detection rate was 0.991 and the AUC was 0.993 as represented in Figure 3.

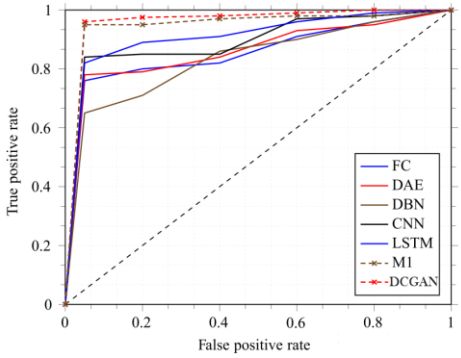

Fig. 3: Receiving Operating Characteristic(ROC) for Deep Learning employed models [18]

The results of employing DCGAN as compared to other approaches are provided in Table II.

Table II: Malware Analysis Results on Deep Learning Employed Models [18]

Metric	[47]	[2]	[24]	FC	DAE	DBN	CNN	LSTM	M1	DCGAN
Accuracy	–	–	0.858	0.792	0.811	0.807	0.895	0.936	0.989	0.998
Precision	–	–	0.892	0.784	0.798	0.801	0.832	0.919	0.982	0.993
F1 Score	–	–	0.874	0.790	0.804	0.803	0.873	0.924	0.986	0.996
FPR	–	0.01	–	0.11	0.10	0.09	0.03	0.019	0.002	0.002
Detec.rate	–	0.94	0.969	0.783	0.815	0.840	0.872	0.963	0.981	0.991
AUC	0.953	–	–	0.801	0.847	0.862	0.901	0.957	0.983	0.993

5. Conclusion

The Security and privacy of Android application has always been a top priority for service providers and consumers. A number of techniques have been employed for protecting Android devices and sometimes a complete OS stack. This paper contributes to design and implement a dynamic malware classification technique based Intents generated by an app. Our results showed that it provides high accuracy i.e. with maximum TP and minimum FP. The framework is real time deployable and is non-signature based malware detection technique. State-of-the-art employed machine learning approaches include the Neural Networks, RNN and ESN. Furthermore, the idea is presented that how we use Deep Convolutional Generated Adversarial Networks (DCGANs) for intents based malware analysis. The DCGANs have provided fine-tuned results in pattern recognition in various domains. Indeed, DCGANS are more robust as compared to the present machine learning techniques employed. We believe that our approach improves and produces more accurate results than previously provided machine learning techniques.

Acknowledgement

This studies is based on PhD research work of the principal author at Malaysian Institute of Information Technology, University Kuala Lumpur.

References

[1] T. Register, “Earn £8,000 a MONTH with bogus apps from Russian malware factories,” 2013, available at: https://www.theregister.co.uk/2013/08/05/mobile_ malware_lookout/.

[2] “McAfee Threats Report: First Quarter 2013,” 2013, available at: https://www.wilderssecurity.com/threads/ mcafee-threats-report-first-quarter-2013.348153/.

[3] McAfee, “McAfee Threats Report: 2014,” 2013, available at: https://www.mcafee.com/error-pages/404.aspx? url=https://www.mcafee.com/us/resources/reports/ rp-threats-predictions-2014.pdf.

[4] I. Week, “Cybercrime Black Market ,” 2014, avail- able at: https://ulasforensikadigital.weebly.com/home/ cybercrime-black-market.

[5] Symantec, “Internet Security Threat Report,” April 2016, https://www.symantec.com/content/dam/symantec/ docs/reports/istr-21-2016-en.pdf.

[6] Google, “VirusTotal. File Statistics,” https://www. virustotal.com/en/statistics/,July2017.

[7] Symantec, “Internet Security Threat Report 2014,” 2014, available at: https://issuu.com/ezenta-itsikkerhed/docs/ internet_security_threat_ report_201.

[8] J. J. Drake, Z. Lanier, C. Mulliner, P. O. Fora, S. A. Ridley, and G. Wicherski, Android hacker’s handbook. John Wiley & Sons, 2014.

[9] S. Ltd, “Android Malware Families,” 2009, avail- able at: http://developer.android.com/reference/java/ net/URLClassLoader.html.

[10] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explain- able detection of android malware in your pocket.” in Ndss, vol. 14, 2014, pp. 23–26.

[11] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “Taint- droid: an information-flow tracking system for realtime privacy monitoring on smartphones,” ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, p. 5, 2014.

[12] Y. LeCun, Y. Bengio et al., “Convolutional networks for images, speech, and time series,” The handbook of brain theory and neural networks, vol. 3361, no. 10, p. 1995, 1995.

[13] K. Hornik, M. Stinchcombe, and H. White, “Multi- layer feedforward networks are universal approxima- tors,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.

[14] R. Salakhutdinov and I. Murray, “On the quantitative analysis of deep belief networks,” in Proceedings of the 25th international conference on Machine learning. ACM, 2008, pp. 872–879.

[15] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," science, vol. 313, no. 5786, pp. 504–507, 2006.
[16] Z. Yang, Z. Hu, Y. Deng, C. Dyer, and A. Smola, "Neural machine translation with recurrent attention modeling," arXiv preprint arXiv:1607.05108, 2016.
[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735– 1780, 1997.
[18] M. Nauman, T. A. Tanveer, S. Khan, and T. A. Syed, "Deep neural architectures for large scale android malware analysis," Cluster Computing, pp. 1–20, 2017.
[19] M. I. Sharif, V. Yegneswaran, H. Saidi, P. A. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis." in ESORICS, vol. 8. Springer, 2008, pp. 481– 500.
[20] [A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual. IEEE, 2007, pp. 421–430.
[21] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Ki- raz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detec- tion on android," in Communications, 2009. ICC'09. IEEE International Conference on. IEEE, 2009, pp. 1–5.
[22] D. Kim, A. Majlesi-Kupaei, J. Roy, K. Anand, K. ElWazeer, D. Buettner, and R. Barua, "Dynodet: Detecting dynamic obfuscation in malware," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2017, pp. 97–118.
[23] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," Journal of Network and Computer Applications, vol. 36, no. 2, pp. 646–656, 2013.
[24] github, "PEInfor Service." https://github.com/crits/crits_ services/tree/master/peinfo_service,July2017.
[25] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifestand API calls tracing," in Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on. IEEE, 2012, pp. 62–69.
[26] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011, pp. 627–638.
[27] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 217–228.
[28] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012, pp. 31–40.
[29] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in Proceedings of the 10th ACM conference on Computer and communications security. ACM, 2003, pp. 290–299.
[30] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. Mc- Daniel, "Iccta: Detecting inter-component privacy leaks in android apps," in Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015, pp. 280–291.
[31] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in Security and Privacy, 2007. SP'07. IEEE Symposium on. IEEE, 2007, pp. 231–245.
[32] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flow- droid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," Acm Sigplan Notices, vol. 49, no. 6, pp. 259–269, 2014.
[33] L.-K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis." in USENIX security symposium, 2012, pp. 569–584.
[34] B. Davis and H. Chen, "Retroskeleton: retrofitting an- droid apps," in Proceeding of the 11th annual international conference on Mobile systems, applications, and services. ACM, 2013, pp. 181–192.
[35] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of android," Tech. Rep., 2009.
[36] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "Appguard–fine-grained policy enforcement for untrusted android applications," in Data Privacy Management and Autonomous Spontaneous Security. Springer, 2014, pp. 213–231.
[37] K. Z. Chen, N. M. Johnson, V. D'Silva, S. Dai, K. MacNa- mara, T. R. Magrino, E. X. Wu, M. Rinard, and D. X. Song, "Contextual policy enforcement in android applications with permission event graphs." in NDSS, 2013, p. 234.
[38] T.-H. Ho, D. Dean, X. Gu, and W. Enck, "Prec: practical root exploit containment for android devices," in Proceedings of the 4th ACM conference on Data and application security and privacy. ACM, 2014, pp. 187– 198.
[39] VXShare, "VirusShare," Accessed date 03 November 2017, available at: https://www.virusshare.com.
[40] Google Play Downloader via Command line, https:// github.com/matlink/gplaycli.
[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in neural information processing systems, 2014, pp. 2672–2680.
[42] Z. C. Lipton, "Deep Convolutional Generative Adversarial Networks," available at: https://github.com/zackchase/ mxnet-the-straight-dope/blob/master/chapter14_ generative-adversarial-networks/dcgan.ipynb.
[43] J. Burns, "Exploratory Android Surgery," in Black Hat Technical Security Conference USA, 2009, available at: https://www.blackhat.com/html/bh-usa-09/ bh-usa-09-archives.html.
[44] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Good- fellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improve- ments," arXiv preprint arXiv:1211.5590, 2012.
[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015," arXiv preprint arXiv:1603.04467, 2015.