A Generative Adversarial Approach for Malware Detection: Android Case Study

Prashant Kaushik
Department of Computer Science & Information technology
Jaypee Institute of Information Technology, Noida

ABSTRACT
Identifying infected Android apps relies on extracting key features from apps, both statically and dynamically. Static feature analysis offers a comprehensive view by examining all source code, including bytecode, C++ code, and permission-containing manifest files. Dynamic analysis complements this by observing app behavior in action, such as disk access, system calls, and network activity. Challenges arise when apps update, as feature sets evolve, potentially hindering classification accuracy. To address this, researchers developed a tool combining a GAN (Generative Adversarial Network) and automation to continuously gather and update feature sets for training. The GAN generates similar samples to enhance training and classification capabilities. The proposed classification cascaded with GAN model named TC-GAN, categorizes apps into three classes: malicious, benign, and inconclusive ("can't say"). Using TensorFlow Lite, the model achieved over 82% accuracy on a dataset of 12,000 apps and their variations, with 15 extracted and 10 GAN-generated features.

General Terms
Generative networks, deep learning.

Keywords
GAN, TC-GAN, malware applications, infected applications, deep learning, feature vector generation, malicious APK

1. INTRODUCTION
Almost all large and small Android application Markets are under continuous & timed attacks by cyber criminals. Also, there are many paid and free third-party stores and website which offer Free apps with malwares. Some of the onetime paid versions also have been on many stores that allows paid app as free. These free apps contain malicious code in them which allow miners to mine for illegal currencies on the victim's device without even being noticed by the user. Apart from stealing the private information, it can also be used to threatening the user for crypto coins. These attackers package the APK's and add many forms of spy ware, malicious code, crypto currency mining frameworks and reverse tcp tunnels for uploading the private contents on to dark web. By a report of Kasp Labs [1] there were 221,800 new mobile malware programs that emerged in the second quarter of 2018 alone, which is 1.8 times more than previous times. Google Play Store has also come up with play protect to solve this problem of identifying the malicious app using their own custom AI models. Also, other labs and authors have been trying to solve this problem of detecting the malicious and benign apps. Avast has built a human made dataset on it. Which we will be using its classification as our primary dataset for over 10k APK's [2]. Y.Zhou [3] has worked on features & machine extraction on API level, there are works done by authors to extract features based on system call and permissions [4]. AndroData is a good tool used to extract large number of static and dynamic vector-based features of android app [5]. Scandroid[6] is a Java/C++ program for dynamic feature extraction made for manifest file and it also performs vector feature extraction. Apposcopy is a fully dynamic tool to extract features based on API calls [7]. TaintDroid [8] are used for scanning privacy features and comDroid[9], a static analysis tool based on Bytecode of the decompiled APKs. So over all there has been many research work focused on feature extraction as well as algorithms for classification. The popular algorithm used varies from SVM to random forest [10] and also many CNN based feature extraction and the dataset creation from many aspects. But still there is no standard dataset for the practice or evaluation purpose.

Kim et al. proposed an intrusion detection tool utilizing a J48 decision tree for classifying malicious and benign applications. They extended their approach by incorporating JavaScript for dynamic feature extraction, creating an app that functions as a host-based intrusion detection system alongside existing antivirus systems. The study references various works in feature extraction, mathematical representation, algorithms, and training basic neural network models for effective APK classification. Dynamic analysis considers the number of system calls in emulators, incorporating system call information into the final dataset. The updated dataset, available on GitHub, serves educational purposes [11]. For classification, a neural network using the TensorFlow Lite framework is employed, featuring 10 layers with 512 perceptrons each. The choice of activation function impacts model accuracy. The project comprises three modules: generating the feature vector, creating a labeled dataset with human and Avast reports; training a GAN on this data for dataset enhancement; and employing a python-based CNN neural network to classify new APKs into three classes - malicious, benign, or undetermined. The code for TC-GAN and GAN is open for public review and evaluation [16].

2. NEW FEATUREVECTOR GENERATION USING TC-GAN
The TC-GAN generation process integrates both static and dynamic features to classify APKs as malicious, benign, or indeterminate. Static features are derived from the APK source code, decompiled using the Jadx tool [14]. These features are used to train TC-GAN, generating additional vectors. The static feature extraction module, implemented in Python using pandas and related modules, is depicted in Figure 1, illustrating the feature extraction process and the creation of the final data for TC-GAN. Training and labeling adhere to the AVG [15] report standards, and the resulting dataset will be accessible on GitHub solely for research purposes. Dynamic feature

extraction involves a C++ module running as a system application on rooted Android devices and emulators created on VirtualBox following Geny Motion guidelines. This module captures basic feature vectors, as outlined in Table 2. After TC-GAN training, a new set of features, labeled numerically, is presented in Table X. The amalgamation of static and dynamic features, coupled with TC-GAN's training, provides a comprehensive basis for classifying APKs, offering a nuanced understanding of their behavior and potential threats.

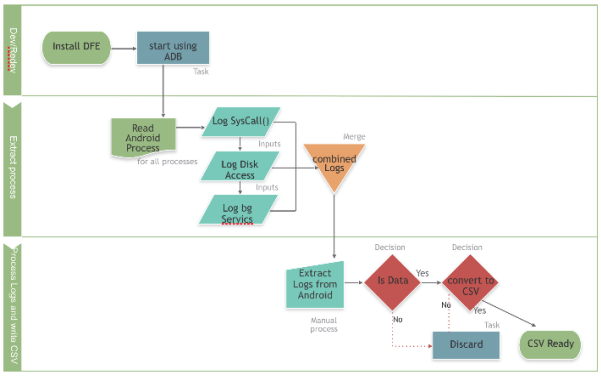

Fig. 1. Process of feature extraction

Once the dataset is labeled, it becomes ready for training the classifier to achieve the final classification of APKs. It's important to note that this dataset will continue to evolve, and the latest version will be accessible on our GitHub homepage. For legal compliance, the APK names have been deliberately removed. The dynamic feature extraction (DFE) module is crafted in C++ and is tailored for Android systems, functioning seamlessly on rooted devices. In contrast, the static module is designed in Python, offering versatility as it can operate on any system. This comprehensive approach, encompassing both static and dynamic features, coupled with the evolving dataset, ensures a robust foundation for training classifiers and enhancing the understanding of APK behavior in a security context.

Table 1. Sample dataset Generated using TC-GAN

S.No	F1	F2	—	Fn	Class
1	0.212	3	0.001	0	Malicious
2	0.45	4	0.003	0	Malicious
3	23	3	2	0	Benign
4	1.09	9	1	7	can't say

Table 2. Dynamic Feature Vectors [12]

S. No.	Feature vector name
1	System call
2	Disk Access time
3	Disk Access duration
4	No. of network URL
5	Data exchange per URL
6	Background activation time
7	background services
8	no. of permissions
9	no. of threads
10	native JNI libs
11	Reverse TCP payload
12	No. of HTTPS URL
13	No. of HTTP URL
14	Front Camera access
15	no. of Async tasks

Table 3. TC-Gan based generation feature vectors along with weights

S.No	F1	F2	F3	F4	F5	F6
W1	0.9	2	0.001	-	5	8
W2	0.12	7	-	-	9	8
W3	0.35	12.4	-	-	5.2	6.2
W4	0.1	6	-	-	1.5	-

3. SYSTEM MODEL

3.1 Hyper parameter optimization

The selection of the most effective neural network architecture is a crucial step in achieving optimal performance for both classification and generation tasks. In our approach, we employ TC-GAN and RNN for the entire system. The learning algorithm for the neural network is fine-tuned to attain an optimal learning rate, incorporating dynamic adjustments in error functions to minimize losses across a set of layers. This iterative process is repeated for both TC-GAN and RNN to obtain finely tuned and optimized parameters. The optimized parameters play a pivotal role in configuring the system model, a detailed explanation of which follows in the subsequent section. This meticulous parameter tuning ensures that the neural network models are poised to deliver the best possible results in terms of accuracy, robustness, and efficiency, aligning with the specific requirements of the classification and generation tasks at hand

3.2 Neural Network model and training

An eight-layered fully connected neural network is constructed using TensorFlow, with each layer comprising 1024 neurons of the RNN type. Notably, the RNN incorporates Long Short-Term Memory (LSTM), enhancing its capacity to capture dependencies over time. Both types of neural networks, standard RNN and LSTM-based RNN, are meticulously designed and subjected to training. The results are subjected to a comparative analysis to discern the performance disparities between the two architectures. The models are implemented in TensorFlow [17], with Figure 2 and Figure 3 depicting the structures of the RNN and LSTM-based RNN, respectively.

The fundamental distinction lies in the presence of a memory unit for short-term memory in each neuron of the LSTM, contributing to its heightened accuracy. The comparative analysis between the two architectures provides valuable insights into the trade-offs and advantages offered by standard RNN and LSTM-based RNN in the context of the specific task at hand. Additionally, the mathematical equation governing TC-GAN for feature vector generation, derived from the extracted APK features, is presented for a comprehensive understanding of the feature engineering process.

$$\mathbf{TC-GAN} = \mathbf{Quan}\,[\mathbf{min}\,\mathbf{max}\,[\mathbf{log}(\mathbf{D}(\mathbf{x})) + \mathbf{log}(\mathbf{1} - \mathbf{D}(\mathbf{G}(\mathbf{z})))] + \mathbf{error}(\mathbf{P1}|\mathbf{P2})\,]$$

In the TC-GAN methodology, the process involves taking the output from the Generative Adversarial Network (GAN) and introducing an error, forming an intermediate vector. This intermediate vector serves as input to the GAN, where it undergoes quantization to generate an output aligned with a specific class. The resulting output from this step is then utilized as a feature vector. This iterative process leverages the adversarial training dynamics of GANs, enhancing the discriminative capabilities of the generated features and contributing to the overall effectiveness of the classification task. The introduction of error and quantization steps within the GAN framework helps refine and optimize the feature vector for improved performance in subsequent classification and prediction tasks related to the APKs under consideration.

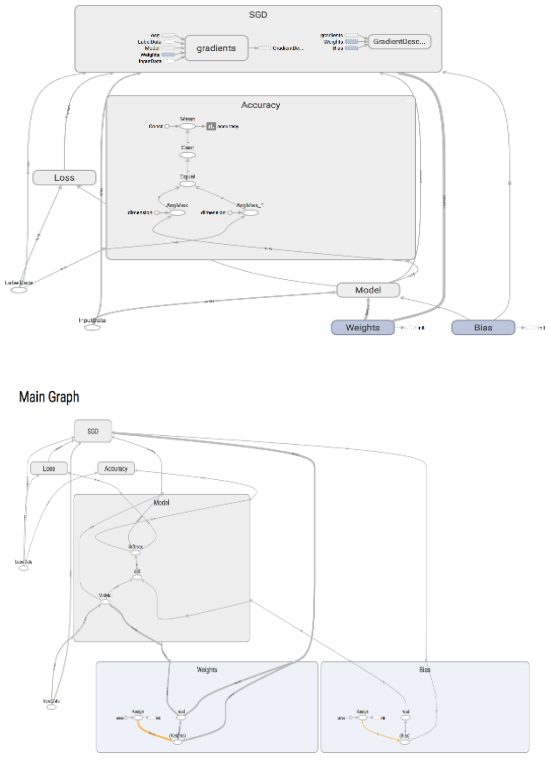

Fig. 2. Overall Model of this project

3.3 This paragraph is a repeat of 3.1

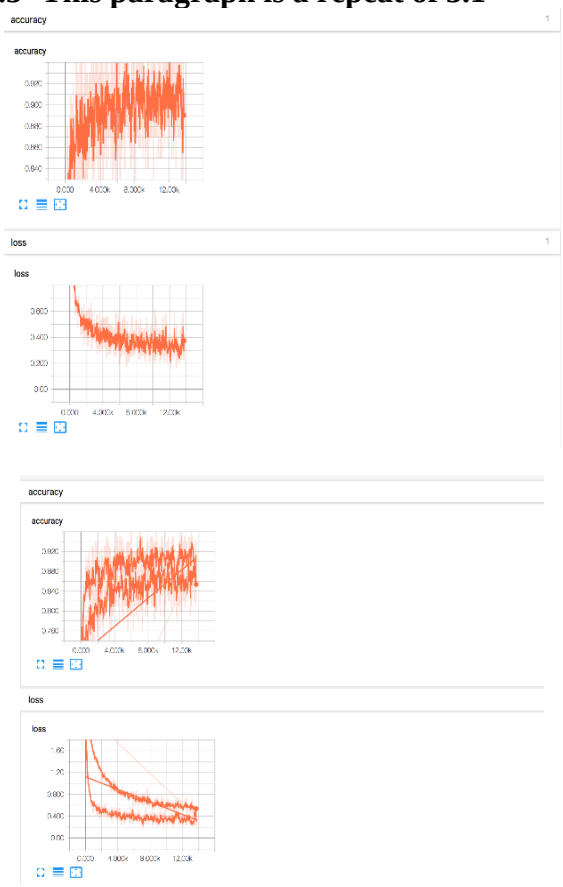

Fig. 3. Variation of Accuracy, Losses for TC-GAN

4. RESULTS

As outlined previously, the training and modeling process encompassed three types of neural networks. To evaluate the efficacy of these models, their results were compared with prior research conducted by various scholars, as documented in the papers we reviewed. table 4 visually represents the fluctuations in accuracy and losses, providing a comprehensive view of the models' performance over the training period. This comparative analysis not only validates the improvements achieved in our work but also contributes to the broader understanding of advancements in neural network-based approaches for the specific task at hand. The visualization in Tensor board serves as a valuable tool for tracking and interpreting the dynamics of accuracy and loss metrics throughout the training process, facilitating a more insightful assessment of the model's behavior.

Table 4. Comparison of Results with Kim et al [11]

S. No	Classifier	Train Acc	Test Acc	Time in Hours
1	J-48	81%	70%	0.10
2	Basic Bayes Net	68%	76%	0.13
3	RNN based models	81%	79%	9

4	TC-GAN(proposed model)	82%	98%	101

5. CONCLUSION

Following thorough training and experimentation, our results indicate significant improvements over traditional CNN and RNN-based neural networks. The utilization of TC-GAN-based neural models, which excel in feature extraction, holds promising potential for the classification and prediction of APK activities. This advancement enables more accurate and nuanced assessments of APK behavior. GAN models, which have evolved beyond their initial applications, are becoming instrumental in various dimensions beyond our specific use case. Their versatility positions them as valuable tools for enhancing the training of diverse models, including hybrid models that leverage the strengths of multiple approaches. This indicates a promising future for the integration of TC-GAN and GAN models in advancing the field of APK classification and prediction, offering more robust and effective solutions in cybersecurity.

6. FUTURE WORK

The utilization of TC-GAN has demonstrated its efficacy not only in generating values but also in overall feature generation. Our intention is to extend its application to address web-based activities, particularly in the mobile domain, focusing on harmful extensions that attach to browsers. This new endeavor aims to enhance our understanding of potential threats in the online environment. We invite contributions and insights from researchers worldwide through our GitHub page, fostering collaboration to build a benchmark dataset collectively. This collaborative effort is instrumental in creating a comprehensive and diverse resource, enriching the field of cybersecurity research and furthering our ability to identify and mitigate risks associated with web-based activities and malicious browser extensions.

7. REFERENCES

[1] "Kaspersky Lab Reporting: Mobile Malware Has Grown Almost 3-fold in Q2, and Cyberespionage Attacks Target SMB Companies." *www.kaspersky.com*, 18 May 2023,

[2] Funk, Christian. "Kaspersky Security Bulletin 2013. Overall Statistics for 2013." *Securelist*, 18 May 2021, securelist.com/kaspersky-security-bulletin-2013-overall-statistics-for-2013/58265.

[3] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang and P. Ning, Detecting malicious apps in o cial and alternative Android markets, Proceedings of the second ACM conference on Data and Application Security and Privacy,2012

[4] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp and J. Ho mann, Mobile- Sandbox: combining static and dynamic analysis with machine-learning tech- niques, International Journal of Information Security,14(2):141–153,2014

[5] Kiran Khatter, Sapna Malik: "AndroData: A Tool for Static & Dynamic Feature Extraction of Android Apps" in International Journal of Applied Engineering, Jan 2015.

[6] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid: Automated Security Certification of Android Applications," Technical report, University of Maryland, 2009.

[7] Y. Feng, S. Anand, I. Dillig and A. Aiken, Apposcopy: semantics-based detection of Android malware through static analysis,Proceedings of the 22nd ACM SIG- SOFT International Symposium on Foundations of Software Engineering,576– 587,2014

[8] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. P. Cox, J. Jung, P. Mc- Daniel and A. N. Sheth, TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones,ACM Transactions on Computer Systems, 32(2):1–29, 2014

[9] V. M. Afonso, M. F. de Amorim, A. R. A. Gregio, G. B. Junquera and P. L. de Geus, Identifying Android malware using dynamically obtained features,Journal of Computer Virology and Hacking Techniques, 11(1):9–17, 2015

[10] J. Abah, O. Waziri,M. Abdullahi, U. Arthur and O. Adewale, A machine learn- ing approach to anomaly-based detection on Android platforms, International Journal of Network Security and Its Applications, 7(6):15–35,2015

[11] H. Kang, J.-W. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," International Journal of Distributed Sensor Networks, vol. 11, no. 6, Article ID 479174, 2015.

[12] https://github.com/prashant343/APKinfectDetect/blob/master/APKdata.c

[13] Tensorflow. "GitHub - Tensorflow/Tensorflow: An Open Source Machine Learning Framework for Everyone." *GitHub*, github.com/tensorflow/tensorflow.

[14] Skylot. "GitHub - Skylot/Jadx: Dex to Java Decompiler." *GitHub*, github.com/skylot/jadx.