Permission and Header Based Android Application

Jayshree Alhate, Roshni Makhija, Rajashri Mahajan

MET Bhujbal Knowledge City,
Adgaon, Nashik

Abstract: **We propose a novel android malware detection system that uses a deep convolutional neural network Malware classification is performed based on static analysis of the raw opcode sequence from a disassembled program. To solve these challenges, we propose system to identify malware both efficiently and effectively. The novel technique enabling effectiveness is the Semantic-based deep learning. We use Long Short Term Memory on the semantic structure of Android bytecode, avoiding missing the details of method-level bytecode semantics. To achieve efficiency, we apply Multilayer Perceptron on the xml files based on the finding that most malware can be efficiently identified using information only from xml files.**

Index Terms**: Android Malware Detection; Deep Learning, Deep Refiner.**

I. INTRODUCTION

Unlike existing solutions, we build this work on the observation that fraudulent and malicious behaviors leave behind telltale signs on app markets. We uncover these nefarious acts by picking out such trails. For instance, the high cost of setting up valid Google Play accounts forces fraudsters to reuse their accounts across review writing jobs, making them likely to review more apps in common than regular users. Resource constraints can compel fraudsters to post reviews within short time intervals. Legitimate users affected by malware may report unpleasant experiences in their reviews. Increases in the number of requested permissions from one version to the next, which we will call "permission ramps", may indicate benign to malware (Jekyll-Hyde) transitions. Google play first releases its app in 2008.Since that it distributes applications to all the Android users. In Google Play Store, it provides services that user can discover the particular application, purchase those applications and install it on their mobile devices. Since Android is open source environment all the detail about the application users can be easily accessed by the application developers through Google play. In Google play 1.8 Million mobile applications are available and that is downloaded by over 25 billion users across the world. This leads to greater chance of installing malware to the applications that could affect users mobile devices. Google play store uses its own security system known as Bouncer system [6] to remove the malicious apps from its store. However, this method is not effective as testing some apps using virus tools many apps are found as malicious which are not detected by Bouncer system [6]. Fraudulent developers use search ranking algorithm to promote their apps to the top while searching. After downloading mobile applications from Google play users are asked to give the ratings and reviews about that particular downloaded applications. However fraudulent developers give fake ratings and reviews about their application promote their application to the top. There are two typical approaches used for detecting malware in Google Play. Thus are Static and Dynamic. The dynamic approach needs apps to be run in a secure environment to detect its benign. The static approach is not used as the need to give all types of attack in early stage itself but that is impossible as everyday attackers find the new way to inject malware on applications.

II. Literature Survey

Different from existing works (e.g., DroidMiner [11], MAMADROID [12] and DroidSIFT [13]) which extract semantics from data flows and control flows, DeepRefiner makes the effort to capture comprehensive bytecode semantics at both method-level and application-level by performing the combination of a Bytecode2Vec technique and

a Long Short Term Memory neural network [14]. Using the Bytecode2Vec technique, DeepRefiner captures methodlevel bytecode semantics by representing each line of bytecode with a dense Bytecode Vector according to its context (i.e., nearby bytecode in the same method). Two different lines of bytecode with similar functionalities or contexts are represented with similar Bytecode Vectors, and will be treated similarly during classification. Since typical obfuscation techniques affect no bytecode semantics within methods, method-level bytecode semantics encoded in Bytecode Vectors help DeepRefiner perform robust detection. After the process of Bytecode2Vec, DeepRefiner represents each application as a variable-length Bytecode Vector Sequence by combining all Bytecode Vectors according to the original bytecode sequence in source code.

DeepRefiner further applies stacked LSTM hidden layers on top of variable-length Bytecode Vector Sequence. During the process, LSTM hidden layers are used to capture historical information about the Bytecode Vector Sequence and iteratively update it to learn the application-level bytecode semantics without losing method-level bytecode semantics due to the special structure of LSTM called memory cell.

With this novel combination, DeepRefiner successfully captures the triggers of malicious behaviors even if they are separated far away from each other in the source code as shown in our experimental results.

Applying both method-level and application-level bytecode semantics to detect malware is proved to be highly effective in our experiments. For the 29,320 uncertain applications that cannot be reliably classified in the first detection layer, DeepRefiner further refines the classification results according to their bytecode semantics, and provides an accuracy of 96.14% and a false positive rate of 4.10%.

More importantly, with the help of bytecode semantics, DeepRefiner successfully detects 10,991 malicious applications from

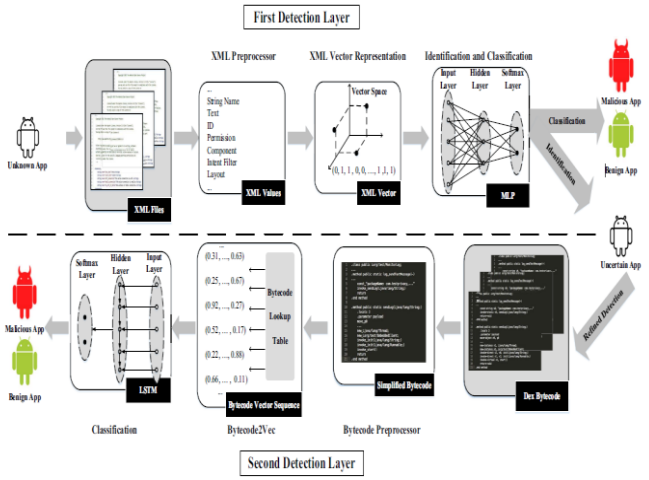
11,000 obfuscated malicious applications, which are generated by various obfuscation techniques.

III. Related Work

Over the past few years, Android malware detection has attracted extensive attentions in both academia and industry. In this section, we mainly review learning-based mobile malware detection systems which are more relevant to system.

A small portion of existing works integrate several classifiers to detect malware in different manners compared with DeepRefiner. Smutz and Stavrou [42] apply an ensemble classifier consisting of many basic classifiers. They perform
a diversity analysis to detect unreliable prediction results, and retrain their classifiers with unreliable observations so as to improve classifier accuracy. The basic classifiers are independent to each other, while the two detection layers in DeepRefiner are complementary. For unreliable observations captured in the first detection layer, DeepRefiner feeds them into the second detection layer for further inspection. DroidSIFT [13] extracts weighted contextual API dependency graph and constructs feature sets accordingly. With
graph-based feature vectors, DroidSIFT builds two classifiers, while the first classifier discovers zero-day Android

IV. Proposed Approach



The First Detection Layer: In the first detection layer, DeepRefiner performs efficient process to distinguish complexities of applications and classify a majority of malware from benign applications. The first detection layer consists of three phases, including XML Preprocessor, XML Vector Representation, and Identification and Classification.

XML Preprocessor: In XML Preprocessor, DeepRefiner decompiles apk files and performs lightweight preprocessing to retrieve xml values from AndroidManifest.xml and all xml files under /res/. The retrieved xml values provide detailed information about the resources required and included in an application, and show different patterns in benign applications and malware.

XML Vector Representation: Next, DeepRefiner represents each application as an XML Vector. To this end, DeepRefiner builds an XML Database according to the unique xml values retrieved from training dataset. If the size of XML Database is X, DeepRefiner defines an X-dimension vector space accordingly.

Identification and Classification:. In the last phase, DeepRefiner identifies uncertain applications from unknown applications, and classifies the rest of applications into malicious or benign according to the input X-dimension XML Vectors.

The Second Detection Layer:

This layer performs refined classification for uncertain applications identified in the first detection layer according to bytecode semantics at both method-level and application-level.

Bytecode Preprocessor:

In Bytecode Preprocessor, apk files are disassembled and dex bytecode is obtained by directly decompiling classes.dex file using apktool [27]. Dex bytecode contains more than 200 dex instructions, while a majority of dex instructions are similar with little differences, such as number of bits reserved for operands and operand type in operators.

Bytecode2Vec: In this phase, Proposed system represents each line of simplified bytecode with a dense and realvalued Bytecode Vector, which captures bytecode semantics by tracking its contexts within its method. For a target line of bytecode, its context is the nearby bytecode within a chosen window size in the same method.

Classification: In the last phase, DeepRefiner performs refined inspection on top of Bytecode Vector Sequences, and detects

malware from uncertain applications. Model Design. DeepRefiner builds the second detection model composing of multiple stacked LSTM hidden layers.

Mathematical model

I1: is the Set of Android Apks.
I2: is the Set of Malware Dataset.

Y is a set of techniques use for our system**.**

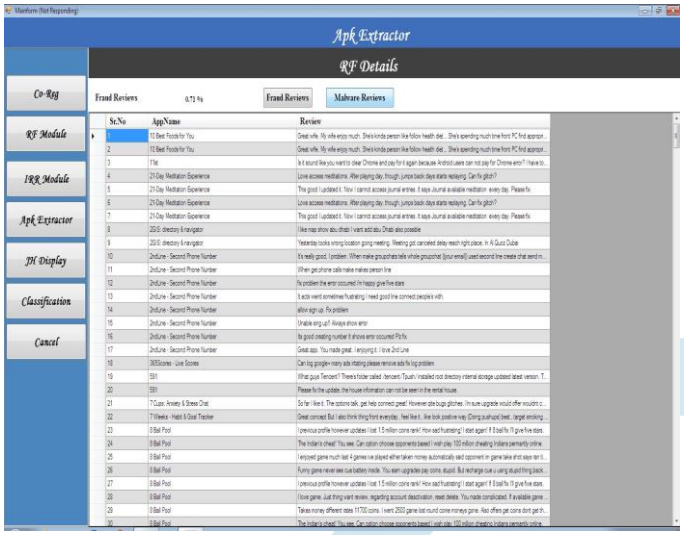
I={I1, I2} // Set of Inputs
Y={ KI, DE, JD, BC ,CR,CW }
DG: Dex File Generation.
FE: Features Extraction.
VS: Vectors Set.
CK: Cluster Kmeans.
CL: Classification
MD: Malware Detection.

F={f1, f2, ……., fn}
Fn1: Malware Files Dataset
Fn2: Apk Inputs
Fn3: Vectors Set.
Fn4: Cluster Kmeans
Fn5: Classification
Fn6: Malware Detection

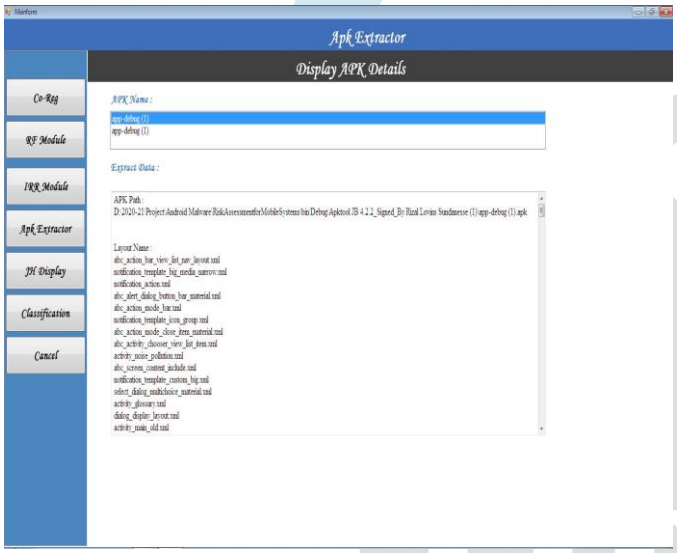
V. RESULTS

MainFrm

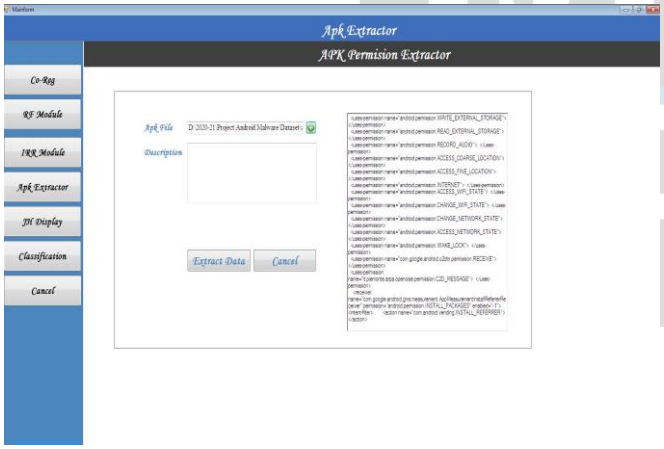
RF MODULE



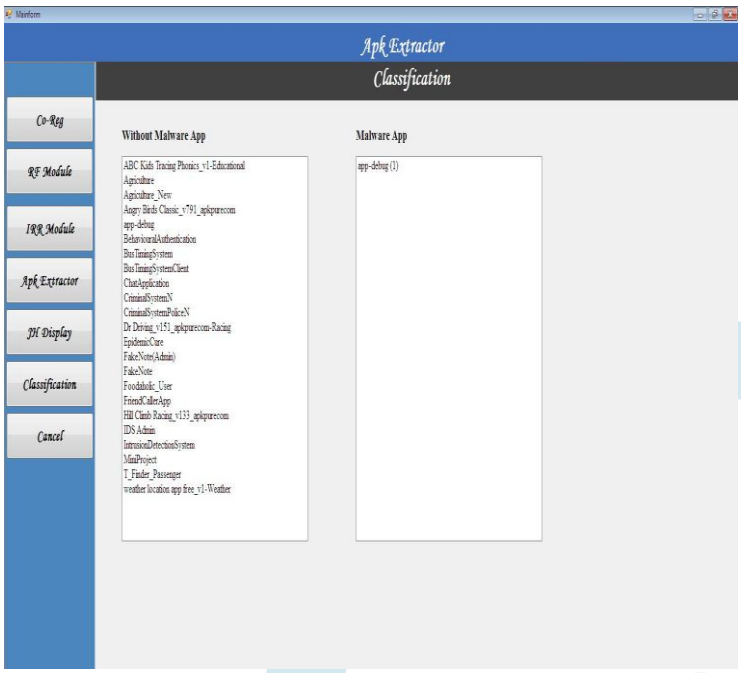
JH DISPLAY



APK EXTRACTOR



CLASSIFICATION



VI. CONCLUSION

We have presented a novel Android malware detection system based on deep neural networks. This innovative application of deep learning to the field of malware analysis has shown good performance and potential in comparison with other state-of-art techniques. Proposed system applies two different detection layers, where the first detection layer achieves efficient detection according to xml files and the second detection layer performs effective and robust detection based on comprehensive bytecode semantics at different scales.

REFERENCES

[1] [1] R. Vallee-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundare- ´san, "Soot-a java bytecode optimization framework," In Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research, p. 13, 1999.
[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: 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.
[3] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon, "Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis," In USENIX Security Symposium, pp. 543–558, 2013.
[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an informationflow tracking system for realtime privacy monitoring on smartphones," TOCS, p. 5, 2014.
[5] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in NDSS, 2014.
[6] V. Rastogi, Y. Chen, and X. Jiang, "Catch me if you can: Evaluating android anti-malware against transformation attacks," IEEE Transactions on Information Forensics and Security, vol. 9, no. 1, pp. 99–108, 2014.
[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural networks, vol. 2, no. 5, pp. 359–366, 1989.
[8] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "Triggerscope: Towards detecting logic bombs in android applications," In IEEE Security and Privacy, pp. 377–396, 2016.
[9] Github, "Open Source Development Platform," https://github.com/.
[10] PUDN, "Programmers United Develop Net," http://en.pudn.com/. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer:
[11] Automated mining and characterization of fine-grained malicious behaviors in android applications," Springer ESORICS, pp. 16 3–182, 2014.
[12] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," arXiv preprint arXiv:1612.04433, 2016.
[13] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," In CCS, pp. 1105–1116, 2014.
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[15] AndroidVersion, "Android Version Differences," https://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

[16] M. Sun, T. Wei, and J. Lui, "Taintart: A practical multi-level information-flow tracking system for android runtime," In ACM CCS, pp. 331–342, 2016.
[17] Kaspersky, "Mobile Threats Report," http://media.kaspersky.com/pdf/Kaspersky-Lab-KSN-Report-mobile-cyberthreats-web.pdf, 2016.
[18] D. E. 201, "AntiEmulator," https://github.com/strazzere/anti-emulator/ blob/master/slides/DexEducation/Anti-Emulation.pdf/, 2013.