INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

ID3801: Open Ended Lab Project

Final Report

Android Malware Detection

Mentor:
Dr. Vivek Chaturvedi

Members:
Jerry John Thomas, 111901055
Joel Sam Mathew, 111901026

Contents

1 Introduction **2**

2 Background **2**

2.1 Dataset 3

2.2 Graphs Neural Network 3

2.2.1 Graph Convolution Network 3

2.3 Explainable AI (XAI) 4

2.3.1 XAI on GNN 4

3 Problem Statement **5**

3.1 Motivation 5

4 Literature review **5**

5 Proposed Design and Implementation **6**

5.1 Dataset 7

5.2 Model 7

5.3 Results 7

6 Conclusion **7**

7 Scope and limitations **7**

8 Future Work **8**

Abstract

The rapid advancement of technology, be it mobile or computer systems, has resulted in more advanced and efficient applications. But it has also increased the system complexity, which attackers may be able to exploit. In the smartphone realm, Android is the most popular operating system. Malware is one of the most serious security concerns for smartphones. This study looks into existing malware detection techniques and attempts to come up with a better solution for detecting unknown Android malware by taking advantage of both low and high computation models using GNN models and XAI concepts.

1 Introduction

With 73.0% of the smartphone market share in November 2021, Android is the most extensively used mobile operating system[20]. The open source nature of Android brings its users convenience but also makes it very susceptible to attacks and malware exploitations. Android is a platform that hosts roughly 99% of known malware to date, and is the focus of most research efforts in mobile malware detection due to its open source nature. [22]

Despite the fact that Google is constantly improving its defense against malicious assaults and established Google Play Protection (GPP), it is not dependable, and researchers have demonstrated that specially constructed harmful apps can easily avoid GPP detection. Malware approaches are evolving and the defense mechanisms must adapt to tackle them. Deep Learning has met with a lot of success over the recent years and it has been used in this domain. Previous deep learning-based malware detection systems have a high cost of computation because they combine several features to attain high accuracy and hence might not be feasible to run on resource constrained devices such as our phones and personal laptops. Hence the need for a low computational model for malware detection to detect new malware's that can be run on most of our personal devices.

There are two ways to test a program namely - static analysis and dynamic analysis. In static analysis we look throughout the code (or its build version) focusing on the internal structure and relationships between different parts of the code. Dynamic analysis tries to do the same when the program is executing and this testing, but this could potentially only find flaws from the code snippets that are actually being executed dynamically. Here we try to explore static analysis methods in order to find vulnerabilities, malwares etc.

In the sections to follow we go through the theory behind the concepts used, review the literature of Android Vulnerabilities and related works followed by attempts at improving existing state of the art solutions for Android Malware Detection.

2 Background

There are various fields related to Android Malware Defenses such as Malicious behaviour Analysis, Adversarial Learning Attacks and Protections, Repackaged/Fake App Detection etc. Of these the binary malware classification receives the greatest research interest of 68% [1]. Static analysis accounts for 73% of the current research publications, 17% for dynamic analysis and 10% for hybrid analysis. In static analysis reverse engineering can be done through various open source tools such as Androguard, APK-tool, soot etc to convert an APK file into its decompiled form. We can extract out the permissions, Java code, Bytecode, Program graph, App metadata, Network information, Intent filters, API calls, function call graphs etc from this, many of which are used as inputs for Deep Learning. Permissions and API calls are the most popularly used features followed by intent filters. [1]. These features are then encoded as categorical, text based, graph, image encodings etc. For the DL models, CNN and MLP are the most popular ones till date. In recent years, explainable approaches have been of keen interest where both global and local explanations are studied. A vast majority of the deployed applications are off device rather than on device and distributed.

2.1 Dataset

For collecting benign APK's Google play store is one of the easily available sources. Drebin [3], AMD [4], Genome [5] are popular sources to gather malicious applications. Although many use this as the backbone for this analysis most of these repositories are not updated and contain old APK files. To overcome this, newer sources such as AndroZoo [6] and VirusShare [7] are now used to collect recent malicious APK samples. Malnet[24] is one of the latest sources containing a large-scale dataset composed of both function call graphs (FCGs) and bytecode images extracted from over 1.2 million Android APKs. Malnet tiny is a smaller version that can be used as an indicator and for faster training to analyze trends of a model.

2.2 Graphs Neural Network

Let's look at graphs before we move on to GNN's. A graph consists of a set of vertices and edges. Edges between nodes can be presented as an adjacency matrix. What makes graphs difficult? A graph does not exist in Euclidean space, so it cannot be represented by any of our conventional coordinate systems, whereas other forms of data such as image, texts can be mapped. Unlike other models where the input size is always fixed, GNN's have to take in graphs of different shapes and sizes and hence must be size independent as well as permutation invariant

Node classification, graph classification and link prediction are some of the most extensively used applications of GNN's. GNNs are a cross between information diffusion and neural networks. Nodes update their states and exchange information by transmitting "messages" to their nearby nodes until they reach a stable equilibrium, which is defined as the information diffusion mechanism. There are different types of GNN's namely Graph Convolutional Network, Graph Attention Network, GraphSAGE etc.

2.2.1 Graph Convolution Network

Each node and edge has an embedding called node and edge embedding respectively. At a high level we iteratively take neighbours and mix/aggregate them and update the node embeddings, so eventually each node can be aware of more nodes in the graph. This method takes inspiration from the CNN models.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Here A is the Adjacency matrix [8]. $\tilde{A}$ is the one which is obtained after adding an identity matrix to A in order to add self loops. D is a row summation over the new Adjacency matrix. W^l is a trainable weight matrix σ refers to an activation function (eg. RELU). H^l is the matrix of activations in the l-th layer

Too many messenger layers will result in over smoothing, learn nothing new and eventually the nodes will be indistinguishable. Hence we must find a good balance to find the optimal number of neighbourhood hop's for the better node representation.

2.3 Explainable AI (XAI)

With the recent advancements in Deep Learning. Accuracy of different models have shot up. But the working of many of these models is still a black box. In order to explain a model and to get insights into what inputs are being given weightage with respect to the output, we use Explainable AI. The use of explainability is not completely standardized and can be defined differently as seen fit. Some applications of XAI are as follows:

- Debugging: If the model is relying on incorrect features to make this prediction the developers can attempt to fix it.
- Debiasing: Users should see what parameters are given weightage to make sure users themselves trust the model to use it.
- Re-coursing i.e. giving suggestions to change the label.
- Comparison of models i.e. evaluating which model is better based on insights from XAI.

XAI can be model agnostic or model specific. It can be used to explain a specific prediction (local) or the model as a whole (global). Lime, Shapley values, Grad-CAM, LRP, local integrated gradients, TCAV(testing with concept activation vectors) are some of the most powerful and frequently used methods. Most of these methods are used to explain classification and regression results on image and text data.

2.3.1 XAI on GNN

In comparison to the image and text domains, the explainability of graph models has received less attention, which is important for understanding deep graph neural networks. Most of the current work in this field are popular methods adjusted to work on graphs and not originally designed for graphs. For instance Level(local) Explanations[9] the popular methods include using the gradients/ feature (Grad-CAM), perturbations (GNN explainer, subGraphX), decompositions(LRP) and surrogate(Graph Lime). For Model level explanations the generation method(XGNN) is used.

GNNExplainer[10] aims to generate a minimal graph that explains the prediction for a node or a graph. The optimization problem is finding a subgraph in the computation graph that minimizes the difference in the prediction scores using the whole computation graph and the minimal graph. Or in other words it maximizes the mutual information between the minimal and computation graph. It learns soft masks for edges and node features to explain the predictions via mask optimization. The SubgraphX[11] method tries to find out the most probable explanation for the graph by reducing it to a subgraph via the Monte Carlo Search Tree Search Algorithm and node pruning; it also uses shapley values as rewards in its computation.

3 Problem Statement

With the rise of Android as the most popular mobile OS platform. We aim to make a low computation model for android malware classification with high accuracy so as to reduce the cost for detecting them and so that it can be run on any resource constrained device and protect the target devices.

3.1 Motivation

With the recent breakthroughs in this domain, there has been significant advancements in applying DL on binary android malware classification. But the limitations of existing methods are that they are computationally expensive. Hence they cannot be running on resource constrained devices and will have to depend on off device systems to check whether the APK they are installing is safe or not. Most of the earlier methods use a coarse grained feature selection, which might not be in the best interest of the model. Large feature loss of the graph structure is another important factor contributing to lower accuracy in these tasks. Hence we try to attempt to make a low computation model for the same.

4 Literature review

MaMadroid[12] was one of the earlier attempts to detect android malware by building markov chains of behavioral models. It uses the idea that malware uses different API calls for different tasks and in different order than benign applications. For instance android.media.MediaRecorder is used by any app with permission to record audio and only using it after calls to getRunningTasks(), which allows recording of conversations, may suggest maliciousness. From the APK, the call graph is extracted followed by sequence extraction and markov chain modeling which are then fed as inputs to various models for classification as benign or malware. Markov chains are memoryless chains, represented as a set of nodes with sum of all probabilities associated to all edges as exactly 1. This was then run through different types of models such as Random Forest, Svm etc. MaMaDroid requires a sizable amount of memory in order to perform classification and better input encoding than the markov chain could result in better performance metrics.

Many low computation models use the permissions of the APK which can be extracted as the sole input to the model. Although this can be seen as a low computational model its correctness is a point of concern. Like malware, many benign Android applications access sensitive user information and make extensive use of permissions that are also common in malware. As a result, permission-based malware detection systems may have a high percentage of false positives. [23]

MAPAS[13] is one of the recent approaches in low computation malware detection. This involves generating a dataset by extracting API Call graphs from both benign and malware APK's. This is done via tiant analysis tools (Flowdroid [14]). The graphs are then vectorised after which it is fed as input to a CNN. Following the learning phase, MAPAS employs Grad-CAM [21], a deep learning explanation approach, to identify high-weight API call graphs employed in malicious apps. Once these API calls are found, it uses a similarity model between the API call graphs of the APK and learnt high weight API calls to decide whether an APK is malicious or not. The resultant model is a very light weight model which can be used on resource constrained devices. The similarity used here is the Jaccard Algorithm. MAPAS identified applications 145.8% faster than MaMaDroid and utilized about ten times less memory. In addition, MAPAS detects malware with a better accuracy (91.27%) than MaMaDroid (84.99%). The limitation here is that the graphs are vectorised and converted into a format suitable for CNN and thereby loses many of the graph properties. If the graph structure of the API Call graph was maintained as such, better insights could have been obtained for explaining it. Better the explanations, better the lightweight model which is built on top of the explanations.

A Graph Neural Network method is also implemented with jumping knowledge[15]. The GNN used purely graph properties as node embeddings such as pageRank, in-out degrees, Node betweenness centrality. We try to inspect if we can use any other features as node embeddings for better results.

5 Proposed Design and Implementation

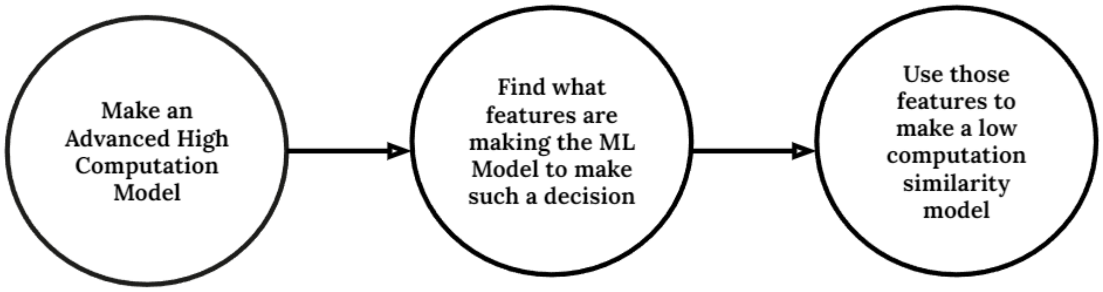
Make an Advanced High Computation Model
Find what features are making the ML Model to make such a decision
Use those features to make a low computation similarity model

Figure 1: High Level View of Frame Work

We propose to make a high computation model first, and identify the high weight API calls and make a computationally lighter model tackling the limitations of the recent approaches by using only API Call graphs as opposed to a combination of several features. In order to preserve the graph information in a better format GNN's have been proposed as the high computation model. Once the model is trained we can run GNN explainer methods such as GNNExplainer, subGraphX to find out the important part of the graph responsible for the prediction. Once the High weight subgraph is obtained we can run a similarity based comparison model for making our low computation cost model. A overview of this approach is visualised in Figure 1

5.1 Dataset

The dataset is obtained from a public dataset[16]. The dataset consists of 1361 PE (portable executable) files, 546 of which are benign and 815 of which are malicious. With the help of radare2, call graphs are extracted from these binaries. Each call graph is preprocessed by counting in each node the number of calls to the most frequent opcodes : ['mov', 'call', 'lea', 'jmp', 'push', 'add', 'xor', 'cmp', 'int3', 'nop', 'pushl', 'dec', 'sub', 'insl', 'inc','jz', 'jnz', 'je', 'jne', 'ja', 'jna', 'js', 'jns', 'jl', 'jnl', 'jg', 'jng'] (27 items). There is an attribution dictionary and neighbors dictionary. The initial node embeddings of each node consists of a 27 sized array where each position consists of the number of times the ith opcode is called for the corresponding node. The neighbors dictionary is converted into Adjacency Matrix. It is made into COO form so that custom datasets could be made in Pytorch[17]. The same can be found here[18].

5.2 Model

It consists of the following layers in order. Graph Convolution Network - GCN (27,256), RELU, Dropout, GCN (256,256), RELU, Dropout, GCN (256,256) , Dropout, Linear (256,1). Finally the Sign function is used to label the graphs. The optimiser used is Adam with a learning rate of 0.01. The loss function used is Binary Cross Entropy Loss with Logit Loss.

5.3 Results

The model attained a testing accuracy of 92.5%. The code with its performance matrix can be found here [19].

6 Conclusion

In this paper, we proposed a GNN model which uses deep learning to evaluate common aspects of API call graphs generated from malicious apps. It can also detect any sort of malware with a high degree of accuracy and demonstrates the effectiveness of a GNN-based approach in android malware detection tasks.

7 Scope and limitations

The explainability part of the proposed framework is still under active development but is not at an working stage at the moment. For models where explainability worked satisfactory accuracy of the model was not obtained.

This static analysis has a severe flaw in its ability to address obfuscation issues. Obfuscation techniques (e.g., polymorphic code, encryption) transform malware binaries into self compressed and uniquely structured binary files that are resistant to reverse-engineering approaches[1]. Obfuscation techniques increase code protection for Android apps, but they also make malware detection more difficult. Code reordering, for example, seeks to change the order of instructions in smali code while maintaining the original run-time execution trail, avoiding detection by malware defense tools.

Transformation of the APK to another format to be given as input for the models we build is being researched upon. APK embedding is very important. We have only extracted API call graphs, other methods use permissions and some combinations of other parameters. Unlike text, image or time series data, APKs are composed of multiple complex data. More often than not we only use a single feature or a small subset of features and vectorise depending upon our model specification. Embedding methods from CV, NLP are widely used. However, for APK files with complex structures, these solutions may be underdeveloped. We can hope to achieve a better APK embedding that can better capture the various aspects of an APK to be fed as input for our Deep Learning Models.

8 Future Work

Our current implemented architecture does not use the concept of explainable AI to provide better and more importantly low computation cost classification. It would be interesting to dive deeper into other GNN architectures to improve the accuracy of the model. Moreover the application of explainable AI would be helpful in understanding the working of the GNN and possibly highlighting suspicious subgraph paths. This can be used to make low cost similarity comparisons to make classifications. Better the high computation model, the better we can hope our low computation model will be due to better explainable insights from our XAI algorithm's. Exploring different XAI methods that are suitable to our model might also be helpful.

Since the availability of API call graph datasets are very limited, a new larger dataset of API call graphs extracted by reverse engineering apps from large sources like Androzoo[6], Drebin[3] etc would be quite important. Training on such a dataset would significantly boost the performance of the model and overcome problems of overfitting. Tools like AndroGuard[25] can be used to extract the API call graphs from a given APK.

We can also look into other features which can be used as node embeddings for our high computation model.

References

[1] Liu, Yue et al. "Deep Learning for Android Malware Defenses: a Systematic Literature Review." ArXiv abs/2103.05292 (2021): n. Pag.

[2] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In Proceedings of the 2014 ACM conference on SIGCOMM. 371–372.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In Ndss, Vol. 14. 23–26.

[4] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 252–276.

[5] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy. IEEE, 95–109.

[6] AndroZoo 2022. AndroZoo. https://androzoo.uni.lu/

[7] VirusShare.com 2022. Because Sharing is Caring. https://virusshare.com/

[8] Kipf, T., Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. ArXiv, abs/1609.02907.

[9] Yuan, H., Yu, H., Gui, S., Ji, S. (2020). Explainability in Graph Neural Networks: A Taxonomic Survey. ArXiv, abs/2012.15445.

[10] Ying, R., Bourgeois, D., You, J., Zitnik, M., Leskovec, J. (2019). GNNExplainer: Generating Explanations for Graph Neural Networks. Advances in neural information processing systems, 32, 9240-9251 .

[11] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," arXiv preprint arXiv:2102.05152, 2021.

[12] Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, E., Ross, G.J., Stringhini, G. (2016). MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). arXiv: Cryptography and Security.

[13] Kim, J., Ban, Y., Ko, E. et al. MAPAS: a practical deep learning-based android malware detection system. Int. J. Inf. Secur. (2022). https://doi.org/10.1007/s10207-022-00579-6

[14] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Traon, Y.L., Octeau, D., McDaniel, P.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices 49(6), 259–269 (2014)

[15] Lo, W.W., Layeghy, S., Sarhan, M., Gallagher, M., Portmann, M. (2022). Graph Neural Network-based Android Malware Classification with Jumping Knowledge. ArXiv, abs/2201.07537.

[16] Comparative Analysis of Feature Extraction Methods of Malware Detection, in International Journal of Computer Applications (0975 8887) Volume 120, June 2015, https://pdfs.semanticscholar.org/8ec8/858a25ce2fb76388d40ad07878b0ec79f580.pdf

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS.

[18] https://github.com/quarkslab/dataset-call-graph-blogpost-material.git

[19] https://github.com/JerryJohnThomas/OELP_Android_Malware

[20] https://gs.statcounter.com/os-market-share/mobile/worldwide

[21] Selvaraju, R.R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., Batra, D. (2019). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. International Journal of Computer Vision, 128, 336-359.

[22] https://www.proquest.com/openview/ae25689e24796eb68ef66538597f98f4/1?pq-origsite=gscholar&cbl=18750&diss=y

[23] Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 426–436. IEEE (2015)

[24] Freitas, S., Duggal, R., Chau, D. (2021). MalNet: A Large-Scale Cybersecurity Image Database of Malicious Software. ArXiv, abs/2102.01072.

[25] https://github.com/androguard/androguard