Visualizing Android Features Through Time: Project Proposal

Michael W. Tegegn
ECE department at University of British Columbia, Canada
Email: mtegegn@ece.ubc.ca

Abstract: Intentionally left blank

1 Introduction

Android is comfortably the most popular mobile operating system in the world with around 70 percent of the market share and close to 2.5 billion active users worldwide [5] . The open-source software has been the favorite mobile operating system since early 2010's for big mobile manufacturing companies like Samsung and LG. Able to run on this system, there are over 3 million android applications on Google Play alone.

Sadly, malicious applications exist even on secure application markets like Google Play itself. To detect these malicious applications, several approaches have been proposed including machine learning approaches. One effective machine learning approach is training a binary classifier for benign and malware applications using features extracted by examining android APK files.[2, 6, 4] Since android features may change through time, machine learning models that are based on APK features may lose robustness for application data from different versions or development years. To make full use of android features for malware detection algorithms, I believe one needs to understand how the android feature space has changed over time, including information on the features added, the relationship between certain features, trends in how the android feature space has changed, and distinctive features for the benign and malware classes. And to gain a deeper understanding of the android feature space and its evolution through time, I propose visualizing the kinds and quantities of android features over an extended period.

I am only currently being exposed to this area as I have only just started my studies as a Master's student. Through this project, I plan to acquire an insightful experience on android security. I am generally interested in safety and security of Machine Learning Applications, and so I think this is a great subway to be exposed to such research.

2 Related Work

Previous work have devised ways to use android features for machine learning applications. The popular 2014 tool DREBIN was one of the pioneers in this field.[2] DREBIN was a static malware detection tool that extracted features from the android application data and trained a Linear Support Vector Machine (SVM) based on these features. The features were extracted by searching for the occurrence of the strings corresponding to one of the eight feature families they discussed. More of this is discussed in the Data and Abstraction section. Their results were promising. DREBIN was able to achieve comparable results to the then top of the line commercial anti-viruses all.

Further research on this area suggested that adjusting the feature weights for the linear support vector machine to be bounded between some small interval aids in training a more robust classifier. [6] The authors proved this by carefully crafting malware applications to bypass DREBIN's approach of letting the model weights be determined completely by the vanilla SVM. The authors proposed an algorithm (Sec-SVM) that aims to eliminate the effect of a feature with high weight in the final trained model. Their argument was, if the classification was heavily dependent on a few features, then simply adding or removing those features contributes significantly to the classification. Therefore, by restricting the feature weights to not exceed a certain threshold, we can mitigate the effect of the presence of a single feature on overall classification of the application. This same paper also mentions how we can select 10,000 of the top features (features with the highest weights) from the DREBIN implementation without significantly impacting the accuracy of the model.

To further strengthen the robustness of such classifiers under attach, Michael Cao et al. [4] proposed an approach where more emphasis is given to features that are more common to malware applications than benign application. They built on the simple fact that it is easy for malware applications to appear benign by adding ineffectual features more commonly found in benign applications. Therefore, focusing on features that are more common in benign applications will hep the classifier to be more robust to such carefully crafted attacks.

Some visualization approaches on android features include works by Hosseinkhani et al [8] who aimed to visualize the permision component of android applications and Bacci et al [3] visualized the dynamic trace of potentially malware activity on android applications.

As far as visualizing the features themselves is concerned, i.e. not the feature weights of the SVM models or similarity between features, I am yet to find a good related work to find my project up on. From what I have discovered so far, the change in the number and type of features through a full decade worth of android OS history has not been extensively studied. And I believe visualization of the feature space through an extended period of time can prove to provide interesting insights for Machine Learning applications.

3 Data and Task Abstraction

3.1 Android Applications

I plan to collect real application data (APK files) from AndroZoo dataset [1] that currently has more than 16 million APKs registered. I have already applied for access to the AndroZoo dataset which will grant me access to their API enabling me to automate downloading multiple APK's per day. The applications found in the AndroZoo dataset are a mix of both benign and malware applications and are all free versions for obvious reasons. Using their API, the plan is to collect at least 1,000 APK samples for both malware and benign classes for each year starting from 2010 to 2021. The problem with this approach is that it is difficult to precisely tell when the applications were first developed. This is because any person can use an earlier version of an API to build an android application because of backward compatibility in Android OS. Hence, I found that the best metric to judge the assembly time of the application is by using its submission date on AndroZoo dataset. An alternative to this approach would be to use the tool proposed by Li et al [7] to infer the release date by looking at the usage of API's. Nonetheless, with a fast growing plethora of android application submissions, the AndroZoo dataset roughly matches the application development date to the submission date since it is highly likely for android applications to be submitted to this dataset once they are available on any market.

After the collection of the APK files, there is the task of labelling them either benign or malware before constructing visualizations. For this task, I plan to upload the files to VirusTotal website that has more than 25 antiviruses that check for malicious software. VirusTotal also has an API that can be used to automate the process. If an android application is labelled as malware by more 2 antiviruses in this tool, I will flag them as malware. The applications will be labelled as benign otherwise.

3.2 Extracting Features

After collecting the data, I will extract the features using DREBIN's definitions. In DREBIN, there are 8 families of android features. Table 1 is from Demontis et al [6] and summarizes the 8 feature sets/families from DREBIN. Each set can have thousands of features associated with it.

TABLE 1
Overview of feature sets.

Feature sets		
`manifest`	S_1	Hardware components
	S_2	Requested permissions
	S_3	Application components
	S_4	Filtered intents
`dexcode`	S_5	Restricted API calls
	S_6	Used permission
	S_7	Suspicious API calls
	S_8	Network addresses

Fig. 1. Available feature sets

An android feature in the case of DREBIN is nothing more than a functionality present in the android application. The presence of the feature can be easily inferred from either the DEX or Manifest files in the application bundle by simply looking for the presence of a particular string. I will run a script for each application to extract the features by searching for these special strings and merge the features found in all application to form a giant feature set. Below is 6 example features that can be found in an android application. The first three fall under the set S_5 and the next three fall under set S_4.

RestrictedApiList_android.widget.VideoView.setVideoPath
RestrictedApiList_android.location.LocationManager.isProviderEnabled
RestrictedApiList_android.widget.VideoView.pause
IntentFilterList_android.intent.action.ZC
IntentFilterList_android.intent.action.U
IntentFilterList_android.intent.action.Y

3.3 Features to Vectors

After the features are extracted, machine learning based malware detectors will need to represent every application as a feature vector with 0's and 1's for each feature. 1 for a feature means the application contains that feature and 0 for a feature means the application does not contain that feature. However, for the purpose of visualization, I plan to treat all applications as vectors of their available features. This means that applications are expressed using vectors with discrete data (feature string) but varying length.

4 Solution

4.1 Goal 1: Accommodate for a very large feature space

The aim of this study is visualizing DREBIN based features starting from early 2010's to help understand how much android features have evolved over the years. However, the sheer number of available features using the aforementioned

feature space make this task challenging. Collecting thousands of android samples for each year since 2010 can easily increase the distinct features to be more than a 100,000. Hence, my visualization should accommodate for such a large feature count. There is also the added challenge of feature updates that will add bias to the visualization of older vs newer features. The results upon successful completion of this task will include the number of features available for each class and each calendar year.

One visualization technique for this task is to represent the number of features for each of the 10 feature sets using a steam graph that spans from 2010 to the present. In figure 2, the hue channel can be used to represent the 8 feature sets and the length of the graph will represent the number of features for each class.

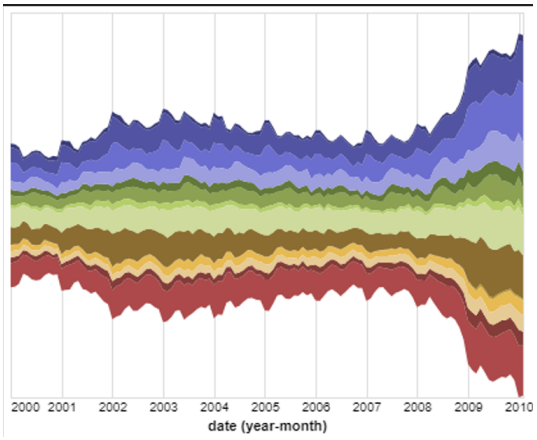

Fig. 2. Stream graph for evolution of feature sets

4.2 Goal 2: Visualize feature drift

After labelling the application data as either benign or malware, I plan to look for any feature drift/migration from one class to another. Some of the questions that a successful completion of this task could answer are the following. Do some features that have historically been more common in either benign or malware applications suddenly or gradually start becoming more common in applications of the opposite class? Are there certain features we can confidently say are always more common in apps of one of the two classes? Is there a time in android version history that the feature distribution in either of these classes has drastically changed?

One visualization technique for this task will be to select a handful of "important" features and construct a heat map as in figure 3. The right labels can be used to represent features and the bottom labels to represent the development years of the applications. The heat map entry will encode the information (percentage of malware apps containing this feature – percentage of benign apps containing this feature).

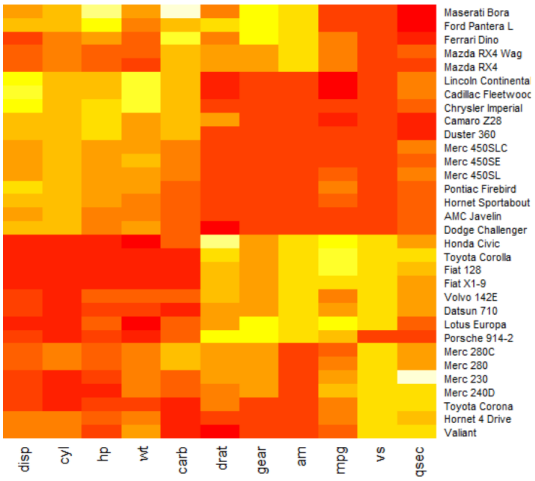

Fig. 3. Heat map to represent potential feature drift

4.3 Goal 3: Look for associations between features and malware families

Additional information about the applications can be found in public datasets like RmvDroid and VirusTotal. Grouping android applications into suitable families is a research topic and there is no officially recognized distinction. However, for the purpose of this study, I can use consistent labels for malware applications and visualize the feature distributions for each application family. This task is more appropriate for Malware applications since extensive studies mainly focus on them and I can resort to using one of the specified metrics to group my dataset of malware applications. Results for this section of the work should provide insightful explanations on the presence of any correlations between the subsets of features and malware families, i.e. do some features exist more abundantly on specific malware families?

One way to show this will be using stacked bars as in figure 4. The hue in the bars can represent the malware families and a bar will be constructed for each interesting feature.

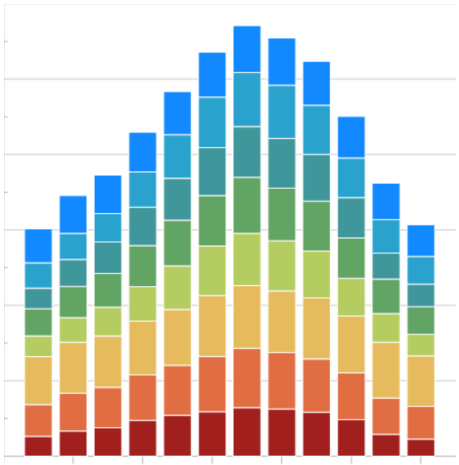

Fig. 4. Stacked bars for relationship between features and android families

To construct many of these visualizations, I plan to use the python library altair since I will be writing python scripts for data collection and abstraction phase of this project. And further tweaks to the vis techniques given above will happen depending on the final data distribution.

5 Milestones

Date to finalize	Milestone task
Nov 7	Data collection and Abstraction
Nov 16	Visualization on whole data set (Goal 1)
Nov 16: Project updates	
Nov 23	Visualization on feature drift (Goal 2 and Goal 3): Tentative
Nov 24: Post update meetings	
Dec 10	Visualization on feature drift (Goal 2 and Goal 3): Final
Dec 15	Project presentation and project write up
Dec 15: Final Presentations	
Dec 17: Final papers submission	

Fig. 5. Project milestones

References

[1] Kevin Allix et al. "AndroZoo: Collecting Millions of Android Apps for the Research Community". In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR '16. Austin, Texas: ACM, 2016, pp. 468–471. ISBN: 978-1-4503-4186-8. DOI: 10.1145/2901739.2903508. URL: http://doi.acm.org/10.1145/2901739.2903508.

[2] Daniel Arp et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." In: *NDSS*. The Internet Society, 2014. URL: http://dblp.uni-trier.de/db/conf/ndss/ndss2014.html#ArpSHGR14.

[3] Alessandro Bacci et al. "VizMal: A Visualization Tool for Analyzing the Behavior of Android Malware". In: Jan. 2018, pp. 517–525. DOI: 10.5220/0006665005170525.

[4] Michael Cao et al. "On Benign Features in Malware Detection". In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE '20. Virtual Event, Australia: Association for Computing Machinery, 2020, pp. 1234–1238. ISBN: 9781450367684. DOI: 10.1145/3324884.3418926. URL: https://doi.org/10.1145/3324884.3418926.

[5] David Curry. "Android Statistics (2021)". In: *Business of apps* (June 3, 2021). URL: https://www.businessofapps.com/data/android-statistics/ (visited on 10/21/2021).

[6] Ambra Demontis et al. *Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection*. 2017. arXiv: 1704.08996 [cs.CR].

[7] Li Li, Tegawendé Bissyandé, and Jacques Klein. "MoonlightBox: Mining Android API Histories for Uncovering Release-Time Inconsistencies". In: *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. 2018, pp. 212–223. DOI: 10.1109/ISSRE.2018.00031.

[8] Mona Hosseinkhani Loorak, Philip W. L. Fong, and M. Sheelagh T. Carpendale. "Papilio: Visualizing Android Application Permissions". In: *Computer Graphics Forum* 33 (2014).