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 such 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, it is important 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 some time 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 through vanilla SVM algorithm. The authors proposed an approach named Sec-SVM that aims to mitigate the effect of a features with very high weights in the final trained model. They proved that if the classification was heavily dependent on a few features, simply adding or removing those features contributes significantly to the classification outcome. Therefore, by restricting the feature weights to not exceed a certain threshold, one can mitigate the effect of the presence of a single feature on overall classification of the application. The paper also suggests that 10,000 of the highest weighted features from the DREBIN implementation suffice to train a model without significantly impacting its accuracy.

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 than benign applications. The authors acknowledged 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 help the classifier to be more robust to such carefully crafted attacks.

Some visualization approaches on android features include works by Hosseinkhani et al [7] who aimed to visualize the permission 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

For this project, I decided to work on two existing datasets. One is the original DREBIN dataset that was collected from the period 2010 to 2012. Analysis on the APK files in this dataset using the compilation date suggests that the application packages were compiled in a period spanning 2008 to 2012. This dataset shows a significant skew in the number of available samples for each year in the five year span. 2010 has the least number of applications in this dataset at 32 samples. 2012 has the largest number of applications in at 5535 samples. The DREBIN dataset is a good benchmark for a time aware analysis as it is a basis for several literature in the field of malware detection. The DREBIN dataset is also the first dataset that such feature abstraction and machine learning based malware detection was experimented on. Analyzing this dataset can give insightful observations on how much the feature abstraction has been affected since the authors first proposed the approach. Drebin Dataset summary:

2008: 302 samples
2009: 32 samples
2010: 571 samples
2011: 4579 samples
2012: 5535 samples

The second dataset is the VT Dataset that has class labelling and dating information extracted from VirusTotal.[10] This dataset contains applications spanning the years 2016 to 2019. In this four year window, the year 2019 has the least number of applications at 396 samples and the year 2017 has the largest number of applications at 1033 samples for this dataset. Similar to the DREBIN dataset, the VT dataset has been used to train a malware classifier using SVM. Information about the achieved accuracy and the optimal features selected by the SVM using dataset is readily available which makes it a convenient dataset for my time aware analysis. Visualizing this dataset along side the DREBIN dataset may uncover persistent or shifting trends in android features during the two disconnected timelines.

VT Dataset summary:

2016: 3077 samples
2017: 10033 samples
2018: 8983 samples
2019: 396 samples

Both of the datasets have roughly equal number of benign and malicious applications. This is important to remove unintended data bias towards either the benign or malware class. Labelling the datasets as either benign or malware was done with the help of AndroZoo [1] and flagging applications that have been labelled as viruses by multiple antivirus scanners in VirusTotal [10] as malicious. Applications not flagged by any of the antiviruses are labelled benign.

3.2 Extracting Features

After collecting the data, the features are extracted 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.

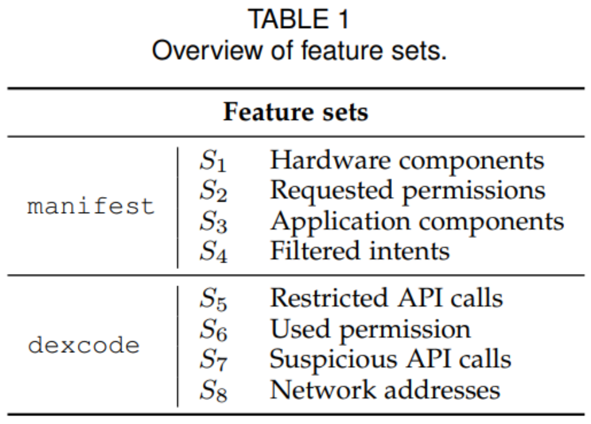

TABLE 1
Overview of feature sets.

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. Every android application package contains an AndroidManifest.xml file and one more executable DEX files. The presence of an andriod 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. Feature sets 1-4 in the table 1 can be retrieved using the DEX file of the APK and feature sets 5-8 can be retrieved by scanning the APK's manifest file. These features are collected using static analysis. A script that extracts the features is ran for each application to collect the features by searching for these special strings and merge the features found in all applications 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. Accuracy and Robustness measures of models that will be discussed in this project are based on such an abstraction. This abstraction is also used in the feature selection algorithms to visualize which features tend to be selected. This will be discussion in later sections.

3.4 Abstract Data

For visualization, every Android feature is a categorical attribute present in each of the samples in our dataset. The development year of the application is an ordered attribute for the dataset. Feature sets add a categorical abstraction to the features. The two classes of applications, namely benign and malware further categorize the applications in the dataset. Using these abstractions, derived attributes like feature count in an Android file, feature count in android families, maliciousness of features are taken as ordinal attributes in the dataset.

3.5 Tasks

Visualizing the Android features in the two datasets is aimed at accomplishing the following tasks. Note that these tasks may be refined as the project progresses.

1. Analyze feature trends if they exist
2. Contextualize the rate of growth of Android applications.
3. Infer important features that may contribute to the robustness of an SVM model
4. Explain the results of SVM model performance in terms of precision and recall
5. Explain the results of feature selection algorithms

4 Solution

4.1 Goal 1: Accommodate for a very large feature space

The aim of this study is visualizing DREBIN based features in the two datasets help understand how much android features have evolved over the years and the results of SVM classifiers. However, the sheer number of available features using the aforementioned feature space make this task challenging. Collecting thousands of android samples for each year from the two datasets increase the distinct features to close to 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 8 feature sets using a steam graph that spans the time range of the two datasets. Figure 2 shows such plot of the DREBIN dataset on an extended feature set list. In Figure 2, the hue channel is used to represent the extended feature sets and the length of the graph represents the number of features collected during some year of the dataset. It shows an expected usage rate increase for each feature set. One can infer that the feature set ActivityList shows the greatest increase in usage frequency in the 5 year period. In the year 2019, there is an expected skew becasue of the number of available data being small. This presents another challenge on how to make use of this dataset by mitigating the effects of the small number of samples collected for the year 2009.

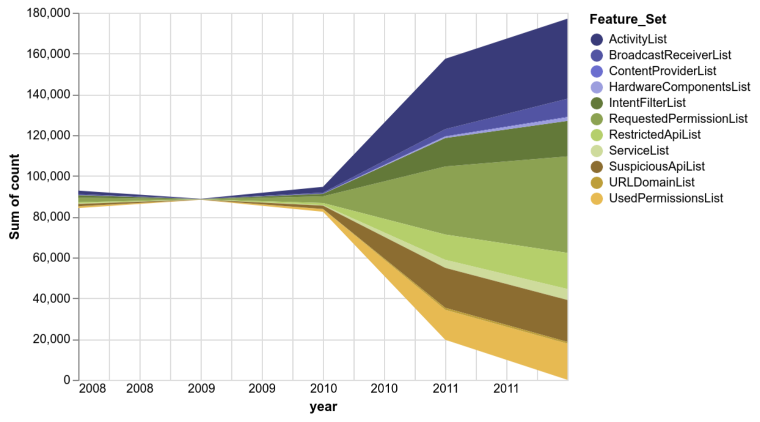

Fig. 2. Stream graph for evolution of the usage of feature sets on DREBIN Dataset

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. These important features are still 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).

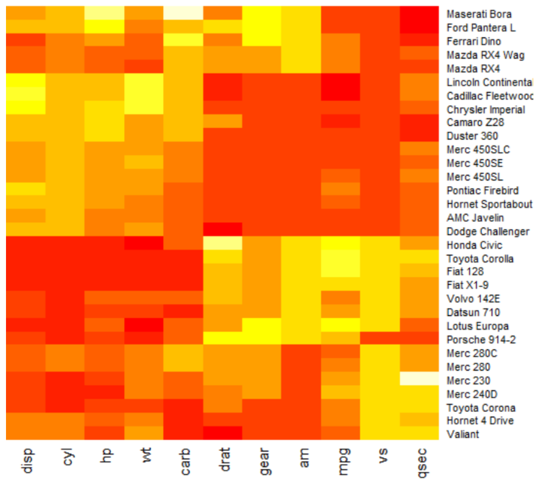

Fig. 3. Heat map to represent potential feature drift

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.

4.4 Goal 4: Visualize feature drift

Visualize the result of feature selection algorithms. One way to visualize such a selection is using scatter plots with point marks for the features and using hue to distinguish between selected and non-selected features. The horizontal axis will represent the development year as usual. The vertical axis encode maliciousness of the feature assessed by how often the feature appears in one class as opposed to the other class.

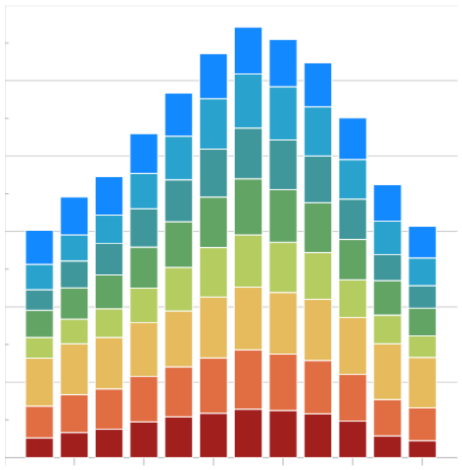

Fig. 4. Stacked bars for relationship between features and android families

4.5 Implementation

All the feature extraction and abstraction tasks are done using Python. I use Jupyter notebook to run codes. To construct many of these visualizations, I am using the python library altair since I will be writing python scripts for data collection and abstraction phase of this project. For some rudimentary analysis, I am also using the python library plotly. And further tweaks to the vis techniques given above will happen depending on the final data distribution.

5 Scenario

A user wants to understand how android applications have evolved over the years and what that means for existing android malware detection software. The user sees a plot on feature sets to get a perspective of the number of features available and how much each feature set contributes to the large feature count.

The user then sees a heatmap that shows how a selected number of features evolve in their malicious tendencies over a period of time. The user will have two separte visualizations using the two separate datasets as to see if the same trend persists in the two separate time frames. Using this visualization, the user can infer which features have consistently been more abundant in malicious applications and which have shifted their tendencies. The user will also be able to associate the features that exhibit a trend to one of the the android families. This way he/she can reason about which feature sets are actually important.

After looking at the interesting features. The user will be shown some feature selection algorithms and their outputs. The user will decide if any of these algorithms have successfully selected the aforementioned interesting features. He/she can reason about which selection is better this way.

After the feature selection visualization, the user will be presented with the results of the SVM classifier based on the different selected features. This will help the user understand if the feature selection algorithm that included the interesting features does indeed perform well in the case of classification using an SVM thereby consolidating the findings.

6 Milestones

Date	Task	hrs	Status
Nov 7	Literature reading	8	Done
Nov 7	Data Collection	4	Done
Nov 7	Apk to Features	4	Done
Nov 10	Get familiar with altair	2	Done
Nov 10	Get familiar with plotly	2	Done
Nov 16	Project update write-up	3	Done
Nov 16	Goal 1: Visualize feature sets	8	working
Nov 16	Project Updates	—	—
Nov 23	Heat map for Goal 2	5	To do
Nov 24	Post Update Meetings	—	—
Nov 23	Stacked bars for Goal 3	5	To do
Dec 10	Discussions on feature drift	8	To do
Dec 12	Feature selection algorithms	8	To do
Dec 14	Plot selection algorithm results	8	To do
Dec 14	Explain model performances	8	To do
Dec 15	Slide Prep and write-up	6	To do
Dec 15	Final Presentations	—	—
Dec 15	Report write-up	4	To do
Dec 17	Final Paper Submission	—	—

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". In: *IEEE Transactions on Dependable and Secure Computing* 16 (2019), pp. 711–724.

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

[8] Tamara Munzner. *Visualization analysis and design*. CRC press, 2014.

[9] Feargus Pendlebury et al. "TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time". In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 729–746. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[10] *VirusTotal*. URL: https://www.virustotal.com/gui/home/.