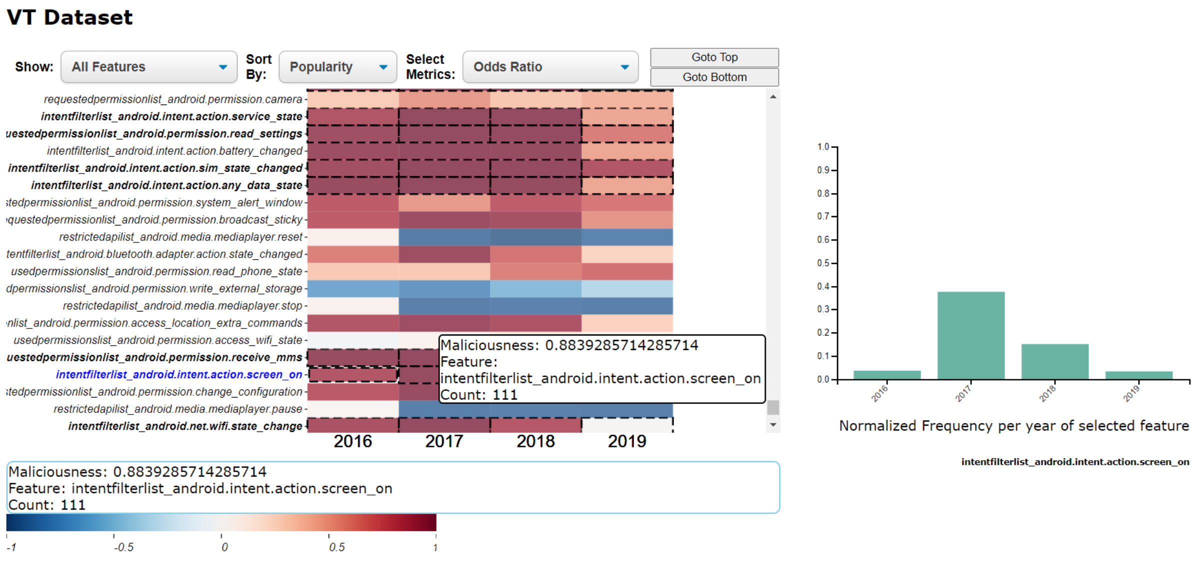
Visualizing Android Features Through Time

Michael W. Tegegn


VT Dataset
Goto Top
Goto Bottom
requestedpermissionlist_android.permission.camera
intentfilterlist_android.intent.action.service_state
uestedpermissionlist_android.permission.read_settings
intentfilterlist_android.intent.action.battery_changed
intentfilterlist_android.intent.action.sim_state_changed
intentfilterlist_android.intent.action.any_data_state
stedpermissionlist_android.permission.system_alert_window
questedpermissionlist_android.permission.broadcast_sticky
restrictedapilist_android.media.mediaplayer.reset
tentfilterlist_android.bluetooth.adapter.action.state_changed
usedpermissionslist_android.permission.read_phone_state
dpermissionslist_android.permission.write_external_storage
restrictedapilist_android.media.mediaplayer.stop
nlist_android.permission.access_location_extra_commands
usedpermissionslist_android.permission.access_wifi_state
uestedpermissionlist_android.permission.receive_mms
intentfilterlist_android.intent.action.screen_on
stedpermissionlist_android.permission.change_configuration
restrictedapilist_android.media.mediaplayer.pause
intentfilterlist_android.net.wifi.state_change
2016
2017
2018
2019
-1
-0.5
0
0.5
1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2016
2017
2018
2019
Normalized Frequency per year of selected feature
intentfilterlist_android.intent.action.screen_on

Show: All Features
Sort By: Popularity
Select Metrics: Odds Ratio

Maliciousness: 0.8839285714285714
Feature:
intentfilterlist_android.intent.action.screen_on
Count: 111

Maliciousness: 0.8839285714285714
Feature: intentfilterlist_android.intent.action.screen_on
Count: 111

Fig. 1. A snapshot of an android feature maliciousness visualization through heatmap, left, and a normalized frequency distribution visualization of a single feature, right.

Abstract— Different approaches have been proposed to generate Android malware detection systems. Some of these approaches use Machine learning methods by first statically extracting features from Android packages and building SVM classifiers for Benign and Malware classes. In training these models, applications used in training have been advised to temporally precede application data available for testing the models to remove temporal bias. Therefore, since the models are trained on a pool of features, selecting relevant features to train these classifiers is a crucial problem. In addition, since Android evolves over time, these features may exhibit shifting trends when analyzed through an extended period of time. This paper describes a visualization tool to visualize individual android features through an extended period of time in the aim of identifying feature trends and assisting feature selection. An interactive heatmap and a bar chart in a side-by-side multi-view aid in analyzing derived properties of features for Android feature space understanding and feature selection.

Index Terms—Android Malware Detection, Feature Selection

1 Introduction

Android is the most popular mobile operating system corresponding to around 70 percent of the market share with around 2.5 billion active users worldwide [6]. The open-source software has been a target operating system since early 2010's for big mobile manufacturing companies including Samsung and LG. The Google Play Store, the largest application store for Android, hosts around 3.5 million applications at the time of this writing.

Unfortunately, malicious Android applications, also called malware, exist even on secure application markets like Google Play. To detect 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 statically analyzing Android source and binary files [2] [7] [4]. This approach involves collecting and labelling Android applications, splitting them to training and testing datasets, extracting features from the training samples, training a binary classifier model and evaluating the model on the testing dataset.

In splitting the Android samples to the training and testing datasets, all the samples in the training dataset should be strictly temporally precedent to the testing ones [11]. That is, training should be done using Android applications from earlier years, and testing using applications from the later years. This is to remove bias in the classifier caused by including future information in the training phase. However, since android features may change through time, selecting features that exhibit consistent behavior over an extended period of time is crucial to ensure a reliable model performance. Hence, to make full use of Android features for machine learning based malware detection algorithms, it is important to understand Android features and how they evolve over time including information on the features added, features that are obsolete, trends in how the Android feature space has changed, and distinctive features for the benign and malware classes.

Thus, to gain a deeper understanding of the Android feature space and select temporally consistent features, we propose visualizing the kinds and quantities of Android features over some time period. We hypothesize that visualizing Android features over an extended time period will aid in identifying temporary consistent features to be selected for training classifier models.

• *Michael W. Tegegn is an MASc student in ECE department at the University of British Columbia. E-mail: mtegegn@ece.ubc.ca.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

2 Background

This section provides background information on Android, Android features Feature selection.

2.1 Android Applications

Android is an open source mobile OS with multiple active developers. Java and Kotlin are two of the most common languages through which android applications can be developed. Once an android application is compiled, it will composed of two types of files. A manifest file, which contains different configurations for running the android application including requested permissions, and a binary DEX file that is the compiled code able to run on Android infrastructure Virtual Machines.

2.2 Android Features

Android features are extracted from statically examining the two files that make up an Android application. Specifically, this work will focus on a feature sets described by DREBIN [2]. The features are grouped into eight feature sets extracted from manifest and DEX files. Figure 2 shows the overview of the available feature sets. We refer to these Android features as features throughout this writing.

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. 2. Available feature sets

Each feature set encompasses a number of features. For example, an application may request 10 different types of permissions, in this case it will have 10 features from the Requested Permissions feature set. Note that any application can be represented as a set of the features it contains. The application can be labelled benign or malware based on malicious activity

2.3 Feature Selection

In machine learning, feature selection is the process of selecting a subset of relevant features for developing a predictive model. Benefits of feature selection include model simplification, shorter training time, overfitting, reduction, noise elimination, etc. In the case of Android features, feature selection aims to identify relevant features that correctly capture test data distribution.

3 Related Work

In this section, previous works on android feature selection, android application visualization and android feature visualization are discussed.

3.1 Android Feature Selection

Previous works 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.

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. [7] 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.

3.2 Android Application Visualization

Rory et al [5] plotted complex features into dendograms as a visual way of distinguishing and thereby clustering android applications. In their work, circular dendograms are used where the outer leaves represent a hierarchical positioning of android samples based on their features. The samples are clustered as one moves into the center of the circular dendogram. Works by Santhanam et al [12] and Onwuzurike et al [10] proposed a visualization to systematically explore android java code to look for signs of potential malicious activity. Their visualization uses marks and edges to represent different methods and call sequences of the android application. They proposed that carefully tracking Android Framework API calls can hint on malicious behavior. Somarriba et al [13] have devised a similar visualization technique with the help of payloads of malicious activity where interactive dendograms in vertical layout are encoded to show potentially malicious activations. By coloring the edges to suspicious API calls, the authors are able to depict where the malicious payloads are activated.

3.3 Android Features Visualization

Some visualization approaches on android features include works by Hosseinkhani et al [8] 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. Gabriella Xiong and Michael Cao in a previous iteration of this course have proposed an interactive visualization where scatter plots show application similarity based on a selected number of features. By allowing users to select samples for training an SVM classifier, they visualized the weights assigned to each feature in the trained model to outline the importance of specific features for classification.

4 Data and Task Abstraction

4.1 Android Application Datasets

In this work, two separate existing datasets are considered. One is the original DREBIN dataset which contains samples collected before 2014. 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 implication of such an imbalance in the dataset is explained further in the Discussion section of this paper. 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 DREBIN feature space changed since the authors first proposed the approach.

The second dataset is the VT Dataset that has class labelling and dating information extracted from VirusTotal [1]. This dataset contains applications spanning the years 2016 to 2019. In this four-year window, 2019 has the least number of applications at 396 samples and 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. In summary, the two datasets contain applications that are both timestamped by development year and labelled as either benign or malware. In addition, 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. The DREBIN features extracted from each application were also available as part of the dataset. A summary about the number of application samples in the two datasets is shown in table 1 .

Dataset	Year	Number of Samples
DREBIN	2008	302
	2009	32
	2010	571
	2011	4579
	2012	5535
VT	2016	3077
	2017	10033
	2018	8983
	2019	396

Table 1. Table showing sample distribution of the two datasets.

4.2 Feature Selection Metrics

In addition to the android applications represented as DREBIN features, each dataset also contains a collection of six feature selection metrics and their results. The feature selection metrics rank the features based on specific scoring functions. The top 100 features subject to each feature selection metric are considered in this work. That is, for each feature selection algorithm, a list of 100 features that algorithm has selected is recorded.

The six feature selection metrics are common to machine learning applications, but only their results are considered in this work. The mathematical formulas and deeper insights about the available feature selection metrics are considered irrelevant for the purpose of this work. The short names for the six feature selection algorithms we have collected results from are Chi-squared, Correlation Coefficients, Mutual Information, Odds Ratio, Popularity Diff and Signed Information Gain.

4.3 Derived Attributes

Using the datasets explained in subsection 1, two attributes are derived.

- One is Maliciousness of an application. Maliciousness of an application measures the tendency of a feature to be more prevalent in malware applications rather than benign applications. Maliciousness for feature x is given by the following formula.

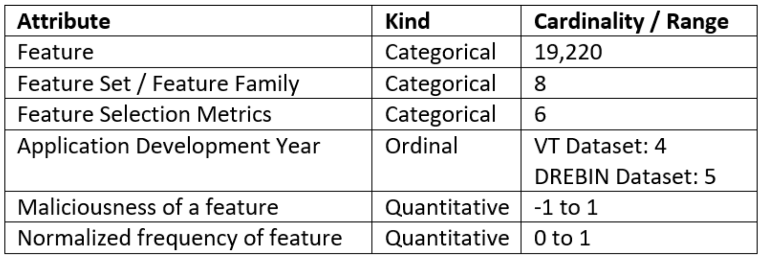
$$\text{Maliciousness} = \frac{\text{Malicious apps with x} - \text{Benign apps with x}}{\text{Total number of apps with x}}$$

- The second derived attribute is the normalized frequency of feature x. The normalized frequency of feature x measures how much feature x is prevalent in the available applications for that particular year. Normalized frequency for feature x is given by the following formula.

$$\text{Normalized Freq} = \frac{\text{Apps with feature x}}{\text{Total number of apps}}$$

4.4 Abstract Data

The table below shows the summary of the available dataset types considered for the visualization section.



Attribute	Kind	Cardinality / Range
Feature	Categorical	19,220
Feature Set / Feature Family	Categorical	8
Feature Selection Metrics	Categorical	6
Application Development Year	Ordinal	VT Dataset: 4 DREBIN Dataset: 5
Maliciousness of a feature	Quantitative	-1 to 1
Normalized frequency of feature	Quantitative	0 to 1

Fig. 3. Data abstractions

4.5 Task Abstraction

Overall, the visualization aims to allow researchers to analyze the feature trend shifts through an extended time period. Specifically, researchers should be able to identify which features shift their maliciousness property or their expected frequency over the visualized time period. The interactive visualization proposed in this work is aimed to assist researchers to achieve the following tasks:

1. Analyse the number of features available in each feature set and their properties

2. Analyze feature trends in feature maliciousness and frequency distribution over the available years

3. Analyze the properties of features selected by the six feature selection algorithms

4. Select relevant features that exhibit consistent behaviors over the time period to use for training a classifier model

5. Identify outlier features in terms of maliciousness and frequency properties

5 Solution

5.1 Goal 1: Visualize feature maliciousness over time

To visualize the maliciousness of features through the development years, we used a heatmap where features are listed in the y-axis and the development years in the x-axis. The individual cells of the heatmap encode the maliciousness of features through a blue-red diverging color map. This color scheme was selected after getting feedback from android malware detection domain experts. The blue color is associated with benign features while the red color is associated with malicious features in Android applications. Figure 4 shows this design decision and encoding. When a feature is not present in any year, the correspond-

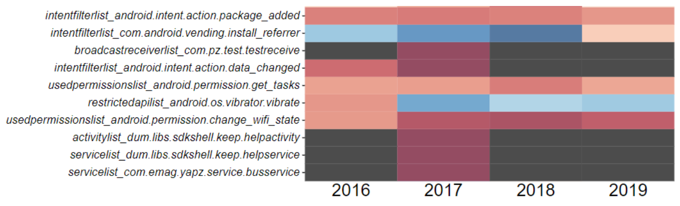
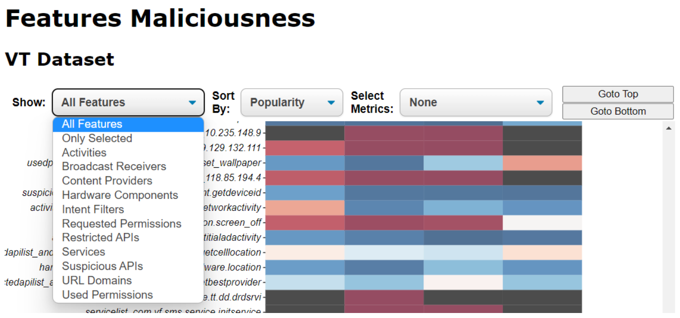


Fig. 4. Maliciousness encoding using heatmap

ing cell is colored black. we considered alternative visualization idioms for this task. One alternative encoding for the color scheme is using sequential color scale instead. However, this idiom will make the distinction between features common in unclear. Using the diverging color scheme helps to easily distinguish between features more common in malicious applications from ones mere common in benign applications. We have also considered using size to encode maliciousness where traditional bar charts can make comparing maliciousness values easier. We chose to go against this encoding because it is not scalable for the number of features available.

5.2 Goal 3: Accommodate for a very large feature space

The aim of this study is visualizing DREBIN based features in the two datasets to help understand how much feature drift is visible in the two datasets. However, the large 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 increases the distinct features to to tens of thousands. Therefore, the visualization should accommodate for such a large feature count. To enable users to easily identify features of interest, we designed methods to reduce the number of features displayed at one time. For this, we used filtering. The features can be filtered to display only features from a specific feature set, or features that have been selected by one of the feature selection metrics. Figure 5 shows the available options to filter the data. Other options considered to reduce the size of the features is to represent all features from the same feature set as a single feature thereby collapsing thousands of features into a 8 feature sets. However, this encoding will hide important details about the individual features while the chosen encoding will preserve information on individual features.



Show: All Features
Sort By: Popularity
Select Metrics: None

Fig. 5. Available filtering methods

5.3 Goal 3: Visualize normalized frequency of features

One of the attributes we chose to visualize is the normalized frequency of features during different development years. This information is encoded using multiple views visual encoding. The normalized frequency is visualized using bar graphs with the x-axis encoding development years and the y-axis encoding normalized frequency. This visual encoding allows easy comparison of the availability of features in different time periods. Figure 6 shows how the frequency bar chart is displayed as a side view for any selected feature. The feature name is displayed under the title for the bar chart. This side plot is displayed when users click at any place along the row of the feature in the heatmap. By showing this plot on demand, we are able to scale the bar chart approach to visualizing the normalized frequency of features.

5.4 Goal 4: Show selected features for the 6 feature selection metrics

To analyse the effects of maliciousness or frequency drift in the datasets, we decided to visualize the results of a few feature selection algorithms on the heatmap itself using Selection visualization design. The boundary of the individual cells in the heatmap are changed to dotted lines and the feature name itself is represented in bold face to highlight the selected features. Figure **??** shows how this is done on the heatmap.

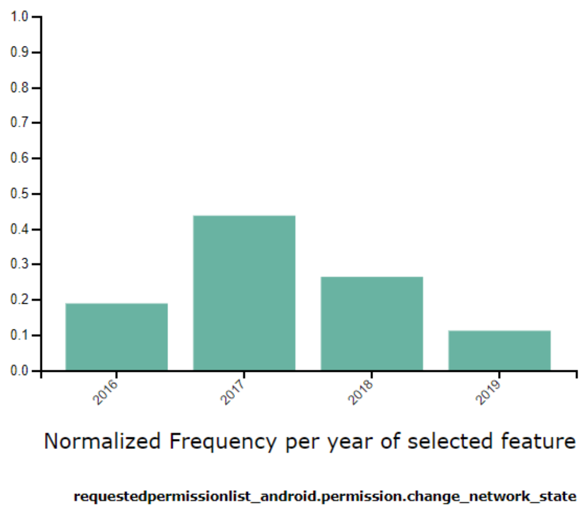


Fig. 6. Frequency chart for feature permission.change_network_state

This decision to embed the selection information on the main heatmap diagram is to allow users to do an in-place examination of the relationship between these selected features and other features. One other way to distinguish these selected features is to color the rows using a slightly different color encoding. However, this decision will interfere with the already encoded color channel and confuse users.

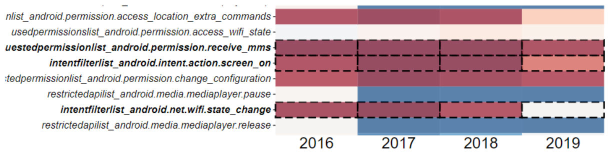


Fig. 7. Selected features shown using dashed borders and bold face

5.5 Goal 5: Allow users to easily find features they are interested in

For users to make an analysis of the available features, they need to be able to interact with the visualization for explorative tasks. Users may need to quickly find features they are interested in or group similar features together. To handle these tasks, we implemented view manipulation techniques where users can sort the displayed features based on either popularity, maliciousness or feature set. In addition two quick Goto buttons allow users to quickly scroll to the top or bottom of the heatmap to analyze feature with extreme values for any of the sorting attributes. Figure 8 shows features sorted using maliciousness. where the top of the heatmap shows only features that have the lowest maliciousness values.

6 Implementation

All the data preprocessing and derived attribute computation is done using Python in Jupyter Notebooks. We used Plotly and Altair at the start to examine different visualization idioms. For the visualization diagrams presented in this paper, we used the JavaScript Library D3 and no additional libraries.

The initial plan was to use plotly and altair for an interactive visualization. However, since more freedom of interaction was required, specifically in allowing vertically scrollable heatmaps, we used D3 as the final library,

7 Milestones

Table 2 shows a breakdown of the tasks involved in completing this project.

Date	Task	Info	hours(expected)	hours(actual)
Nov 7	Literature reading	Android malware detection, vis techniques	8	8
Nov 7	Data Processing	Derived attributes, Logs	4	6
Nov 10	Get familiar with altair	Tutorials	2	2
Nov 10	Get familiar with plotly	Turorials	2	2
Nov 16	Project update write-up	-	3	5
Nov 16	Goal 1: Visualize feature sets	Stream graph, bar graphs	8	7
Nov 16	Project Updates	—-	—-	——
Nov 23	Heat map using altair	Not completed, not suitable	5	3
Nov 24	Post Update Meetings	—	—-	—-
Nov 27	Get familiar with D3	Tutorials	4	4
Dec 1	Heat map visualization	Scrollable feature, color coding, ..	8	8
Dec 3	Heat map visualization	Show selected features	3	3
Dec 7	Heat map visualization	Filtering and sorting	4	5
Dec 10	Visualize feature frequency	Use multiview, encode bar charts	6	4
Dec 12	Train models functionality	Using flask, Not completed, Out of time	8	4
Dec 15	Slide Prep and write-up	-	6	6
Dec 15	Final Presentations	—-	—-	—-
Dec 17	Report write-up	-	4	12
Dec 17	Final Paper Submission	—-	—-	—-

Table 2. Milestone table showcasing the breakdown of tasks for this project

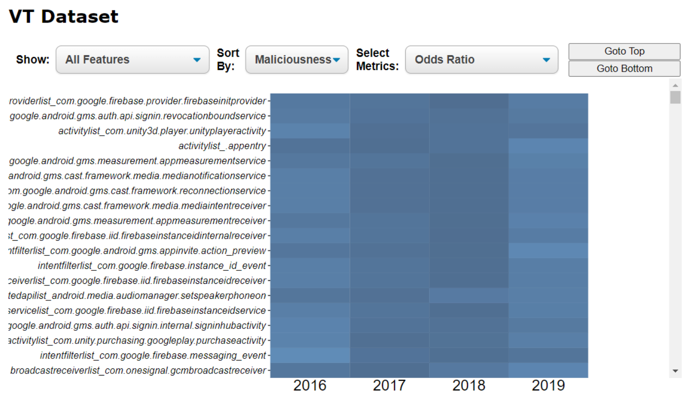

VT Dataset
Show:
All Features
Sort By:
Maliciousness
Select Metrics:
Odds Ratio
Goto Top
Goto Bottom
roviderlist_com.google.firebase.provider.firebaseinitprovider
google.android.gms.auth.api.signin.revocationboundservice
activitylist_com.unity3d.player.unityplayeractivity
activitylist_.appentry
google.android.gms.measurement.appmeasurementservice
android.gms.cast.framework.media.medianotificationservice
om.google.android.gms.cast.framework.reconnectionservice
ogle.android.gms.cast.framework.media.mediaintentreceiver
google.android.gms.measurement.appmeasurementreceiver
st_com.google.firebase.iid.firebaseinstanceidinternalreceiver
ntfilterlist_com.google.android.gms.appinvite.action_preview
intentfilterlist_com.google.firebase.instance_id_event
ceiverlist_com.google.firebase.iid.firebaseinstanceidreceiver
tedapilist_android.media.audiomanager.setspeakerphoneon
servicelist_com.google.firebase.iid.firebaseinstanceidservice
google.android.gms.auth.api.signin.internal.signinhubactivity
activitylist_com.unity.purchasing.googleplay.purchaseactivity
intentfilterlist_com.google.firebase.messaging_event
broadcastreceiverlist_com.onesignal.gcmbroadcastreceiver
2016
2017
2018
2019

Fig. 8. Features sorted by maliciousness display the most benign features at the top

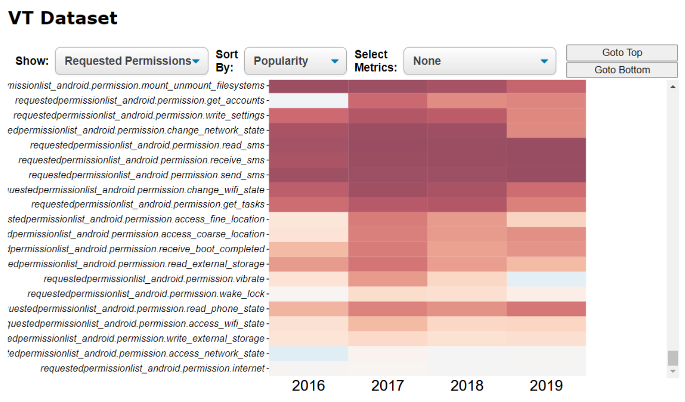

VT Dataset
Show:
Requested Permissions
Sort By:
Popularity
Select Metrics:
None
Goto Top
Goto Bottom
missionlist_android.permission.mount_unmount_filesystems
requestedpermissionlist_android.permission.get_accounts
requestedpermissionlist_android.permission.write_settings
edpermissionlist_android.permission.change_network_state
requestedpermissionlist_android.permission.read_sms
requestedpermissionlist_android.permission.receive_sms
requestedpermissionlist_android.permission.send_sms
uestedpermissionlist_android.permission.change_wifi_state
requestedpermissionlist_android.permission.get_tasks
stedpermissionlist_android.permission.access_fine_location
dpermissionlist_android.permission.access_coarse_location
permissionlist_android.permission.receive_boot_completed
edpermissionlist_android.permission.read_external_storage
requestedpermissionlist_android.permission.vibrate
requestedpermissionlist_android.permission.wake_lock
uestedpermissionlist_android.permission.read_phone_state
questedpermissionlist_android.permission.access_wifi_state
edpermissionlist_android.permission.write_external_storage
tedpermissionlist_android.permission.access_network_state
requestedpermissionlist_android.permission.internet
2016
2017
2018
2019

Fig. 9. Requested Permissions sorted by Popularity

8 Results

This section explains results of the visualization diagram proposed and presents a scenario of how researchers may use this visualization to analyze feature trends and select features that show consistent maliciousness and abundance characteristics. To do this, we refer back to the tasks defined in section 4.5 and present a scenario where a user tries to do these tasks.

8.1 Scenario

- A researcher wants to understand the number of features available in each feature set and their properties. In particular, the researcher wants to know the number and properties of requested permissions for the 4 year time span from 2016 to 1019. Therefore, the user thus loads the VT dataset and chooses to show ”Requested Permissions” feature set only. In doing so, the list of features is filtered out to show only the requested permissions. In addition, the researcher sorts the data based on popularity to see which permissions is requested by the most applications. As Figure 9 shows, the researcher learns that internet permission is the most requested permission amongst the samples in the VT dataset. The researcher also learns that internet feature has maliciousness close to 0, since it is color coded as close to white. The researcher also learns that most of the popular requested permissions are more associated with malicious applications as the heatmap appears red for the majority of applications in view.

- The researcher then wants to analyze feature trends in feature maliciousness and frequency distribution over the available years. The researcher explores the heatmap by filtering and sorting the features using the available metrics to identify key feature trends. Figure 10 Shows a visible trend in feature associated with SMS permissions. Using this visualization, the researcher can observe that SMS permissions are closely tied to malicious activity. The researcher can also deduce that by looking at the normalized frequency plots on the side, the features show very similar trends suggesting that the presence of one of these features means that the chances of seeing the other SMS related features is high.

- The researcher wants to analyze the properties of features selected by the six feature selection algorithms. The researcher can easily do this by selecting one of the feature selection metrics from the drop down menu on the right. In addition the researcher can filter the features displayed to only show the selected features. This gives insights on types of features are selected by which type of feature selection metric. Figure 11 shows features selected by the Odds Ratio feature selection metric. The researcher understands that only the Odds Ratio feature selection metric only selects features that appear to be more prevalent in malicious applications. Indeed, the Odds Ratio feature selection metric is a one-sided

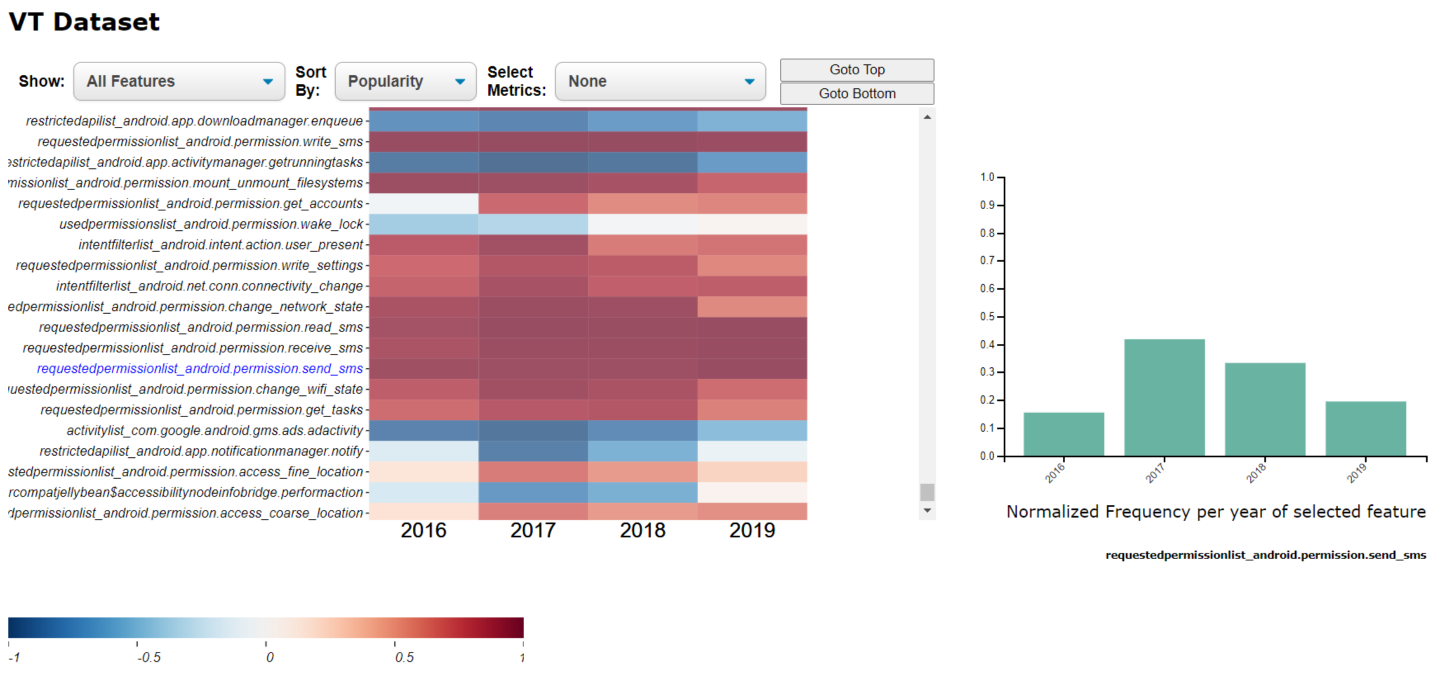

VT Dataset
Show: All Features
Sort By: Popularity
Select Metrics: None
Goto Top
Goto Bottom
restrictedapilist_android.app.downloadmanager.enqueue
requestedpermissionlist_android.permission.write_sms
restrictedapilist_android.app.activitymanager.getrunningtasks
missionlist_android.permission.mount_unmount_filesystems
requestedpermissionlist_android.permission.get_accounts
usedpermissionslist_android.permission.wake_lock
intentfilterlist_android.intent.action.user_present
requestedpermissionlist_android.permission.write_settings
intentfilterlist_android.net.conn.connectivity_change
edpermissionlist_android.permission.change_network_state
requestedpermissionlist_android.permission.read_sms
requestedpermissionlist_android.permission.receive_sms
requestedpermissionlist_android.permission.send_sms
uestedpermissionlist_android.permission.change_wifi_state
requestedpermissionlist_android.permission.get_tasks
activitylist_com.google.android.gms.ads.adactivity
restrictedapilist_android.app.notificationmanager.notify
stedpermissionlist_android.permission.access_fine_location
rcompatjellybean$accessibilitynodeinfobridge.performaction
dpermissionlist_android.permission.access_coarse_location
2016
2017
2018
2019
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Normalized Frequency per year of selected feature
requestedpermissionlist_android.permission.send_sms
-1
-0.5
0
0.5
1

Fig. 10. A plot showing a visible trend in SMS permissions

feature selection method, which means it only selects features that are more prevalent in malicious applications by formula design. Without knowing the formula involved, the clearly sees that features selected by the Odds Ratio feature selection metric are correlated with malicious applications.

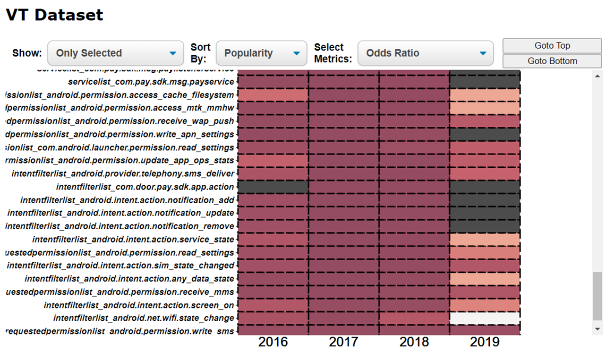

VT Dataset
Show: Only Selected
Sort By: Popularity
Select Metrics: Odds Ratio
Goto Top
Goto Bottom
servicelist_com.pay.sdk.msg.payservice
issionlist_android.permission.access_cache_filesystem
lpermissionlist_android.permission.access_mtk_mmhw
dpermissionlist_android.permission.receive_wap_push
dpermissionlist_android.permission.write_apn_settings
sionlist_com.android.launcher.permission.read_settings
rmissionlist_android.permission.update_app_ops_stats
intentfilterlist_android.provider.telephony.sms_deliver
intentfilterlist_com.door.pay.sdk.app.action
intentfilterlist_android.intent.action.notification_add
intentfilterlist_android.intent.action.notification_update
intentfilterlist_android.intent.action.notification_remove
intentfilterlist_android.intent.action.service_state
uestedpermissionlist_android.permission.read_settings
intentfilterlist_android.intent.action.sim_state_changed
intentfilterlist_android.intent.action.any_data_state
uestedpermissionlist_android.permission.receive_mms
intentfilterlist_android.intent.action.screen_on
intentfilterlist_android.net.wifi.state_change
requestedpermissionlist_android.permission.write_sms
2016
2017
2018
2019

Fig. 11. Features selected by the Odds Ratio Selection metric appear to all be extracted from malicious applications. This is information that gets encoded in the heatmap without knowing the feature selection formula for Odds Ratio metric.

- The researcher wants to select features that exhibit consistent behaviors over the time period to use for training a classifier model. To do this task, the researcher filters and clicks along a row to display the frequency bar graph of the corresponding feature. This interaction has been shown in Figure 10 where a user clicks on a row and the bar graph is represented on the side.

  The user then records features that are consistently malicious or consistently benign through the years. In addition the user filters from these features by looking at their normalized frequency bar chart by choosing only the features the are consistently present throughout the different development years visualized. For example the SMS features exhibit similar characteristics and are reasonably abundant, 15% to 45%, which suggests that selecting these features can improve model's performance.

- The researcher wants to identify outlier features in terms of malicious and frequency properties. The researcher can easily go to top or bottom after sorting the features by their maliciousness attribute to identify the most and least malicious applications. This interaction has been discussed in section 5.2.

9 Discussion and Future Work

This section will discuss about the strengths and limitations of the approach, lessons learned during this project and potential future work.

9.1 Strengths

This approach was first motivated in an effort to explain feature selection metrics' results on android features. Training SVM classifiers on feature selected using different feature selection metrics results in varying model performances. Understanding the difference in the features elected using these selection metrics allows researchers to better explain the results. In this approach, two attributes capable of exhibiting shifts in the datasets were analyzed. Maliciousness and abundance. Using a color scheme well known by domain experts also increases easy interpretation of heatmap to domain experts. Furthermore, encoding such derived attributes in a familiar heatmap encoding provides researchers with visual information on attributes that would have rather required mathematical computation.

For the exploratory task of finding consistently-behaving features, this visualization enables interactivity for researchers to easily find the features and analyze trends.

9.2 Limitations

One crucial limitation of the approach is the skew in the dataset. As table 1 shows, the two datasets used to construct the heatmap are severely skewed. Consequently, there is a threat to validity for the findings in the years that contain only a small number of samples. Although the frequency plot is normalized by the number of samples, the small number of features in some years still imply that some features do not even have samples associated with them. Consequently, a large amount of dark boxes are visible in areas where there are only a small number of features. This implies that our analysis maybe ill-advised durinig these years.

Another limitation is that choosing to analyze the two datasets separately, we have restricted the time span where feature trends can be observed. This means that we cannot make conclusive remarks about features disappearing because of Android version updates.

In addition, visualizing all available features still has scalability issues. The use of vertical scroll bars on the heatmap canvas showing all features does not suffice for researchers to make insightful observations. The number of available features is still too large and further abstraction is required.

9.3 Lessons learned

During the course of this project, we applied different visualization idioms discussed in class to find the best fitting encoding. We learned how to use Python visualization libraries like altair and Plotly. We also learned a JavaScript library D3 for constructing highly flexible visualization designs. Through different iterations of the project, we learned the process of refining the project's goal and implementation details to design a useful tool that we can use ourselves.

Moreover, using the tool to analyze the two datasets has given us insightful observations on the feature selection metrics themselves. In addition, through iterative feedback we have also learned the process of formal writing useful for writing scientific papers.

9.4 Future work

Because of the time constraints in doing the project, some planned functionalities were not included. One is the ability to train models using features selected by the researchers using the visualization tool and displaying results. This will allow researchers to better understand the implications of using features with certain feature trends. Another functionality not completed due to time constraints is allowing users to drag and drop a selected number of features. This functionality can allow users to group features they are interested in together to analyze the maliciousness of features throughout an extended time period.

The scrollable and interactive heatmap presented in this work can be further extended to encode any derived attribute for visual analysis of features in machine learning context.

10 Conclusion

In this paper, we present a visualization technique to analyze android feature maliciousness and abundance trends through time. The visualization design allows Android malware detection researchers to see how features of Android applications developed in different years change over time. In addition, the visualization allows researchers to identify features that exhibit consistent behaviour in maliciousness and abundance. The visualization focuses on visualizing the features through an interactive canvas to aid in explorative tasks of locating and analyzing the subset of features that the researchers are interested in.

References

[1] Virustotal.

[2] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of android malware in your pocket. 02 2014. doi: 10.14722/ndss.2014.23247

[3] A. Bacci, F. Martinelli, E. Medvet, and F. Mercaldo. Vizmal: A visualization tool for analyzing the behavior of android malware. pp. 517–525, 01 2018. doi: 10.5220/0006665005170525

[4] M. Cao, S. Badihi, K. Ahmed, P. Xiong, and J. Rubin. On benign features in malware detection. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, p. 1234–1238. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3324884.3418926

[5] R. Coulter, L. Pan, J. Zhang, and Y. Xiang. A visualization-based analysis on classifying android malware. In X. Chen, X. Huang, and J. Zhang, eds., *Machine Learning for Cyber Security*, pp. 304–319. Springer International Publishing, Cham, 2019.

[6] D. Curry. Android statistics (2021). *Business of apps*.

[7] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16:711–724, 2019.

[8] M. H. Loorak, P. W. L. Fong, and M. S. T. Carpendale. Papilio: Visualizing android application permissions. *Computer Graphics Forum*, 33, 2014.

[9] T. Munzner. *Visualization analysis and design*. CRC press, 2014.

[10] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Trans. Priv. Secur.*, 22(2), apr 2019. doi: 10.1145/3313391

[11] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 729–746. USENIX Association, Santa Clara, CA, Aug. 2019.

[12] G. R. Santhanam, B. Holland, S. Kothari, and J. Mathews. Interactive visualization toolbox to detect sophisticated android malware. In *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–8, 2017. doi: 10.1109/VIZSEC.2017.8062197

[13] O. Somarriba, U. Zurutuza, R. Uribeetxeberria, L. Delosières, and S. Nadjm-Tehrani. Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, 2016:1–17, 01 2016. doi: 10.1155/2016/8034967