# Classification of Ransomware, Malware, and Benign Applications Using Permission-based Analysis

Meira Farhi mvf8@case.edu
William Kunz wfk16@case.edu
Ryan Marett rcm141@case.edu
Ian Post ijp13@case.edu

Case Western Reserve University
Department of Computer and Data Science
10900 Euclid Avenue, Cleveland, OH 44106

## ABSTRACT

As Android capability and use grows, as does the risk involved. Malicious Android applications can leak sensitive data and information and cause damage to a device. In this paper we consider three forms: general malware, ransomware and SMSmalware. Due to the complexity of Android security and ever-increasing availability of applications, the need for a comprehensive security solution is evident. Many proposed solutions do exist, with varying security techniques in each. In this paper, we will analyze the static analysis technique of permission analysis to look into the possibility of identifying and labeling benign, malware, and ransomware applications. This is to see if there is a cheap, and efficient way to identify malicious applications from benign applications.

## Keywords

Android; Malware; Ransomware; Benign; Static Analysis; Permission Analysis; Machine Learning; Security; SMSmalware; Permission Sets

## 1. INTRODUCTION

Android is the dominant mobile operating system with millions of users and applications and 82.8% of the market share as of 2015 [15]. With the popularity of Android, the mobile platform has become a target for malicious applications or malware. Since no other mobile operating system has more than 15% of the market share Android has become the largest target group for malware developers. There are many different kinds of malware, in this paper, we will discuss both general malware and ransomware. Ransomware is a subgroup of malware and can do devastating and expensive damage. With the dangers of malware and the growing risk due to the popularity of the Android operating system, it is clear why there must be an effective and efficient detection and prevention system.

One of the main concerns for Android security is that users can download applications off of the manufacturer stores which have security measures in place and from third-party sites. These third-party sites do not have the built-in security features of the manufacturer stores and thus the need for malware detection is even more pressing. Android has built-in security through the permission system which asks for user permission for each application to use certain system features such as INTERNET, SEND_SMS, and WAKE_LOCK. This feature allows for user discretion when allowing applications to use a given permission. However, users can be unknowledgeable about what they are allowing or apathetic towards their security. To combat this malware attempts to appear as benign in order to escape detection. Thus, a system that builds upon the built-in features is needed to truly protect the user against malicious software.

### 1.1 Prevalence of malware/ ransomware

As Android is the most common mobile operating system, there will be a lot of malicious applications designed specifically for Android OS. Kaspersky -- using user-reported data -- published data about the amount of malware that was detected in 2020. They discovered 5,683,694 malicious installation packages, 156,710 new mobile banking Trojans, and 20,708 new mobile ransomware Trojans. The new ransomware was severely down, as previously in 2018 and 2019, 60,176 and 68,362 ransomware installation packages were found [4]. Although ransomware is not as prevalent as normal malware, this type of malware can cause massive financial damages and irreparable damage if a person is unable to decrypt the files that have been encrypted. This is why our group focuses on normal malware and then specifically ransomware due to the devastating damage the latter can do.

### 1.2 Detection Methods

Static analysis examines the program -- such as a manifest file -- without running it, while dynamic analysis requires the code to be running. For example, in Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection, the authors detect ransomware by looking at if there are active UI widgets when a file is encrypted [5]. This is not possible to do in static analysis, as it requires the program to be running and an active analysis of what is happening live. This is different than in Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications where the authors used dynamic analysis to detect malware by stopping programs from performing update attacks, so applications were not able to upload malicious code [10]. Dynamic analysis is more expensive resource-wise as it requires the application to be running, but it can be an effective way to find malware. Static analysis at times can only require the manifest file to run analysis, which is a very small memory footprint.

## 2. BACKGROUND

Since Android dominates the mobile operating system market and is thus a hub for developers to produce applications, there are too many applications created to be fully screened. This combined with the multiple sources such as third-party and designated manufacturer stores such as the Google Play Store creates the need for an efficient and systematic approach to detecting and preventing malware. One of the techniques used in this study is static analysis. Static malware analysis is completed without running the application code. An example is checking the

permissions used in the Android Manifest file and using that information to predict if the software is malicious.

## 2.1 Definitions

The project focuses mainly on three specific types of applications (general malware, ransomware, and benign), with an added inspection into SMSmalware permission-based detection. First, malware applications are malicious programs that can harm the user's phone or information in a variety of ways, while ransomware is a specific type of malware that encrypts files or locks functionality to force payment by the user. This is a specifically dangerous type of malware as the user is able to lose their files forever if the files are not backed up and the user does not pay to decrypt the files. SMSmalware applications are malware that force the user to make unwanted calls, or texts to premium numbers which will charge the caller. Lastly, benign applications are specifically apps that are not malware and are normal applications -- such as Snapchat, Facebook Messenger, and other commonly used apps.

The following image is an example of WananCry Ransomware, where one has to pay $300 in bitcoin within three days, or the price to decrypt will go up to $600. As this is a large payment, one should try to avoid this as much as possible.


**Figure 1. Example of WannaCry Ransomware. [7]**

## 2.2 Android permission system

The Android permission system has a large variety of permissions, but not all of them are used. Olmstead and Atkinson found that out of the 235 permissions at the time of the study, only 10 were used by more than 20% of the apps, and that 147 of 235 apps were used by 0.09% of the total numbers in their data collection [8]. As most app permissions are not used by apps, there will be specific permissions that will pop up. For example, Olmstead et al. found that 83% of apps request full network access, and 69% request view network connections. In addition, they found that 165 permissions were hardware specific, while 70 were potentially related to a user's information [8]. Different apps would request different permissions, such as a flashlight app would use the flashlight and not need to modify usb storage, but ransomware may target the latter under the assumption that the files may contain important information. This does not mean that only ransomware would request this permission, but only applications that would want to modify usb storage would request that permission in theory. However, some applications over request permissions that they do not need. This is a limitation of our paper as our program assumes all permissions are related to the application, and this will be discussed further in the discussion section.

## 2.3 Extracting permissions from manifest

Android permissions for each individual application can be extracted from the Android manifest file which is located in the apk of the application. By first unzipping the apk file then parsing the manifest file for the "uses-permission" and "uses-feature" all requested permissions can be located and compiled into a readable data format. This is an example of static analysis as the code is not run in order to analyze the application.

## 2.4 Decision trees

Weighting of what permissions can be considered telling as to the status of an application (benign, malware, or ransomware) comes from the decision tree structures that our 2D-array data is all fed into. The decision trees used for this project come from the scikit-learn library for Python, a teaching tool on machine learning processes that has an ease of implementation with a strength of functionality that made it the immediate candidate selected to perform our analysis.

For some background for anyone unexposed to decision trees in elementary programming courses, the makeup of a standard decision tree is to have a series of linked nodes where the child of the parent nodes represent outcome states for that parent. As an example, we can think of a decision tree that helps one decide whether they will be rained on today. In it, our parents grow more specific as determining whether one will be rained on becomes more nuanced as the parents linking to parent nodes increasing building condition-upon-condition that must be met for any decision to be made on whether one will be rained on. Once the knowledge base (KB) of the system (any information we can provide on which decisions can be rendered) has been represented fully in predicate form (read: a logical statement that can spawn children of parent nodes) we know that our decision tree is fully formed and represents all of the states of the world.

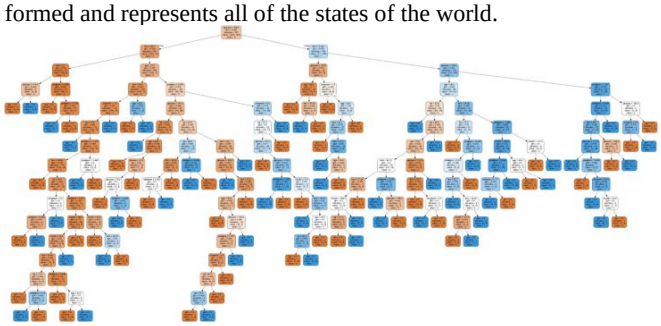

**Figure 2. Visualization of complex connected decision tree [2]**

From that peek into its AI consideration, we can explore how such a structure would be beneficial in a ML framework. Our KB here is the set of all permissions being studied for their predictive power of classification, with each apk being entailed by some combination of predicates from that KB. With each app defined as a leaf in this decision tree with its known classification, we sum over each classification to give us a probability at each parent of each app classification. These probabilities are then fed to a scikit regression that attempts to classify apk's of hidden app

classification to assess the performance of our decision tree methodology.

To get into the math defining the decision tree, not much more is occurring than a partitional recurse, if one remembers their Algorithms topics. Partitions of the KB are tested for their created impurity, with those where impurity is minimized being taken to formulate the branches in the decision tree [12]. We care about what branches occur where in the tree for the efficient traversal of the data structure. The smaller the average depth that has to be explored when attempting to reach a random leaf state, the more that can be done with our decision tree in analysis. Impurity is seen in our objective actually, the probabilities of classification at each parent, coming from Equation 1. In it, $p_{mk}$ represents the proportion impure at parent m with k children. An entropic implementation of it can be seen as how it plays into the partition decision, in this case mimicking nlog(n) cost.

$$p_{mk} = (N_m \sum_{y \in Q_m} I(y = k))^{-1}$$

**Equation 1. Impurity of a parent.**

Any regression on our tree from this is evaluated on Mean Squared Error, exhibiting standard formulation of the reciprocal node by the sum difference at any point from the mean. We see the overall efficiency of our decision tree defined as well by the sum of our impurity and the most costly node to reach.

For what we want to accomplish, a probabilistic measure of classifications based on a chained assessment of an available data type (here, permissions), using the linked list dominated decision tree is the obvious choice.

# 3. APPROACH

## 3.1 The Datasets

The application was trained and tested on benign, malware, and ransomware datasets. The total training dataset consisted of 2057 applications and the testing dataset consisted of 2,161 applications with no overlapping applications between the two. The training set had 623 benign applications, 591 malware applications, and 843 ransomware applications. The testing set had 626 benign applications, 663 malware applications, 670 ransomware applications from one dataset, 106 SMSmalware applications, and 96 ransomware applications from a separate dataset. All of the benign applications were split up randomly from one dataset as were the first malware and ransomware applications in order to protect against grouping of similar types within each category. The benign applications were from the Google Play store, the malware applications were from the Genome malware dataset, and the ransomware application were obtained with permission from the authors of Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection [5]. The SMSmalware and second ransomware datasets were solely used for testing purposes and were acquired from the Canadian Institute for Cybersecurity [3].

These datasets consist of individual, unique applications, however, some of them request identical permissions. In the benign training set there are 258 unique permissions requests and 278 in the benign testing set. There are 478 unique permission sets between the testing and training set meaning there are only 58 duplicates. There are 232 and 243 unique permission sets for the malware training and testing set respectively with 351 unique combinations across the two datasets. The ransomware dataset is the largest and has the greatest number of duplicates. The ransomware training set has 73 unique permission combinations and the testing set has 66 unique combinations. Across the two sets there are only 96 unique combinations. This shows that out of the 1,513 ransomware applications that there were only 96 unique combinations and thus many ransomware applications have the same permission requests. As ransomware is ever evolving and the ransomware dataset is over two years old that the permissions requested by ransomware may have evolved and diversified. There are 47 unique permission sets in the SMSmalware set and 40 in the second ransomware set. There are a total of 810 unique permission combinations across all 4,218 applications meaning that most applications request similar permission sets to others. This does not invalidate the datasets as all were taken as a sample of real-world applications and thus reflect the trends of applications in the wild. Further analysis on future datasets could aid in training the machine learning detection tool and thus produce more accurate and dependable detection results.

## 3.2 Formatting of permissions

Using Java code to unzip and analyze the apks of each application in order to compile the permissions used. First the applications to be analyzed apk's were put into a single folder. Linking the folder's path to the Android Manifest Analyzer the code goes through each apk individually first unzipping and locating the manifest file then parsing said manifest file in order to locate the "uses-permission" and "uses-feature" tags for permission extraction. After each permission is located it is added to a list that contains all the permissions for a given application. Once all of the permissions have been extracted, the list is added to a map where the key is the name of the apk and the value is the list of permissions for the given application. After all of the apks are analyzed in the provided folder they are then formatted into a list of binary values. Android allows for a total of 324 permissions to be requested. Thus a list of size 324 with each element being the name of a given permission. The dictionary is then iterated through and for each apk a list of size 324 is created. If the application requests a permission then this is denoted with a one, if not then it is denoted with a zero. The index of a given one or zero for a permission is the same as the index in the list of all Android permissions for that permission. Thus, a two-dimensional matrix of size number of applications by 324 is created where each row is an application and the 324 columns are the binary permission values. Finally, this matrix is exported to a csv file where it can then be loaded by the machine learning detection tool.

## 3.3 Sci-kit tool

To quote the developer's of scikit's website, "Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning." [13] To sweep broadly over its offering, it has every introductory tool one could hope for for starting data analysis in Python. An impressive tool that it offers is the visualization of most of its processes, allowing a window into the black box that is so often veiled, but it is the standard for ML in Python because its methods are very efficient, its toolset is diverse in supervised and unsupervised learning, and its algorithmic processes bring about a higher accuracy rate. [11]

Our utilization of scikit did not extend far beyond its simplest regression features; reaching our simple conclusion with means to reach it that are more complicated than needed brings distraction from what is found towards how it is found. We use its ordinary least square regression to assess the ability of different permission sets to predict classifications of benign, malware, and antimalware. I find the notation of how this occurs fascinating, so include it here as Equation 2.

$$\min_{w} ||Xw - y||_2^2$$

**Equation 2. Regression statement for linear regression**

As a regression, the heuristic it is performing is to iterate through lines of different slopes and different coefficients to attempt to minimize the absolute difference between the point of the line at the unmodifiable axis and the data point lying there.

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

Hide prompts and outputs

**Figure 3. Simplest example of the ordinary linear regression application. [14]**

## 4. RESULTS AND ANALYSIS

### 4.1 Accuracy Rates

An example of the output for one run of the tool on a testing set is included in figure 4. In the array, if our tool believes the APK to be malware it will label it as "1", and if it believes it to be ransomware it will label it as "2", and benign as "0". Finding accuracies was quite simple given the return matrices of the tool, but due to how the sci-kit tool works, each time the tool was run a slightly different result was returned. To account for this, the tool was run 20 times and the resulting frequencies were recorded and averaged for the most accurate results possible. The results for false positive rates are in table 1, with the total false positive rate being 8.0%. The results for false negative rates are in table 2, with the total false negative rate being 4.9%. Overall, malware was classified as ransomware 1.9% of the time, and ransomware was classified as malware 2.2% of the time, giving an overall misclassification rate of 2.1%. All the full accuracy rates are summarized in table 3. For all the ransomware we tested, we ended with an overall ransomware accuracy of 95.4%. The accuracy on all tested sets was 93.1% total.

The false positive rate was rather low for both malware and ransomware. That is, using only permission requests, our tool can reliably label benign applications as benign and not mislabel them as somehow malicious applications. It is also of note that the misclassification rates were very low: the tool does an excellent job of separating ransomware from malware on all datasets. This could be very useful for analyzing datasets where all the applications are known to be malicious of some kind, but it is unclear what classification of malicious apps they are. Our tool could be used to reliably diagnose future datasets in this way.

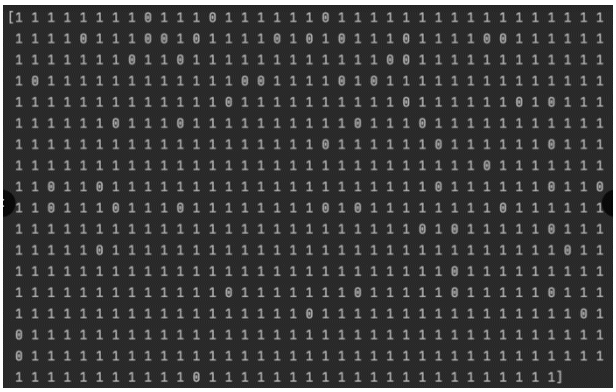

**Figure 4. Example output after running tool on malware test set**

**Table 1. False positive rates**

| Benign Tested Positive | Average Rate |
|---|---|
| Malware | 0.078 |
| Ransomware | 0.001 |
| TOTAL | 0.080 |

**Table 2. False negative rates**

| Dataset | Average Rate |
|---|---|
| Malware | 0.037 |
| SMSmalware | 0.219 |
| Ransomware 1 | 0.006 |
| Ransomware 2 | 0.139 |
| Total Ransomware | 0.022 |
| TOTAL | 0.049 |

**Table 3. Accuracy rates**

| Dataset | Average Rate |
|---|---|
| Malware | 0.948 |
| SMSmalware | 0.662 |
| Ransomware 1 | 0.990 |
| Ransomware 2 | 0.698 |
| Total Ransomware | 0.954 |
| TOTAL | 0.931 |

We, however, are more interested in false negative rates, given that is when users would mistakenly download malware or ransomware thinking it is benign. In our tool for ransomware detection, this occurs when ransomware is labelled as a 0 for benign. Putting both ransomware datasets together, this happened 2.2% of the time. While this is quite low, this number should

ideally be as close to 0 as possible. Analysis on why this number is greater than 0% is detailed in section 4.3.

## 4.2 Permission Combination Uniqueness

Although over 4,000 different Android applications were tested there were only 810 unique sets of permission requests. This phenomenon can be expected both in the training and testing sets and across all Android application marketplaces. As there are 235 permissions to be requested there are countless different unique combinations for applications to request. However, certain permissions are extremely common and others are not. According to Pew research only a fraction of the total permissions are requested by most applications. With the pattern of most requests permissions to percentage of applications requesting a given permission resembling a right angle with the corner in the top left [8]. This can be seen in figure 5 below.

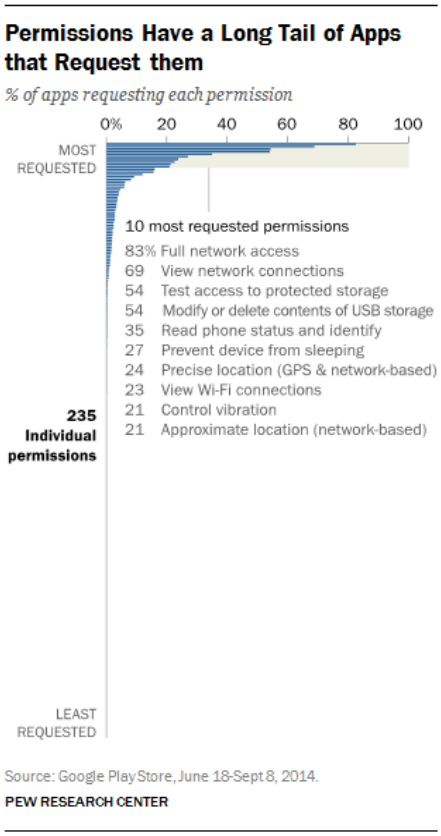


**Figure 5. Most requested permissions**

At the time of the study in 2014 there were only 235 total permissions for Android and of these 235 permissions only 10 were requested by more than 20% of applications. This helps to explain the phenomenon of relatively few unique permission combinations because only few permissions are generally used by any application. Many applications do variations of certain tasks such as a map application uses your location to route you to your desired location while a delivery application uses your location to know where to make the delivery.

## 4.3 Permission Frequency

For the datasets used in this study their average permissions used per application and the most requested permissions were compiled for each individual type of application. There is a clear pattern in the amount of permissions requested for each type of application. Benign applications request far fewer permissions on average than malware and ransomware as benign requests four average permissions while malware and ransomware both request an average of thirteen permissions. The specific permissions for each type of application also differ. The largest difference is between ransomware and the others. However, there are some differences between benign and malware applications.

**Table 4. Top permissions requested in benign apps**

| Permission | Requests |
|---|---|
| INTERNET | 1136 |
| ACCESS_NETWORK_STATE | 563 |
| READ_PHONE_STATE | 504 |
| WRITE_EXTERNAL_STORAGE | 353 |
| VIBRATE | 312 |
| ACCESS_COARSE_LOCATION | 308 |
| WAKE_LOCK | 214 |
| ACCESS_FINE_LOCATION | 178 |
| ACCESS_WIFI_STATE | 157 |
| READ_CONTACTS | 116 |

**Table 5. Top permissions requested in malware apps**

| Permission | Requests |
|---|---|
| INTERNET | 1648 |
| READ_PHONE_STATE | 1425 |
| ACCESS_NETWORK_STATE | 1266 |
| WRITE_EXTERNAL_STORAGE | 1081 |
| ACCESS_WIFI_STATE | 842 |
| READ_SMS | 797 |
| RECEIVE_BOOT_COMPLETED | 730 |
| WRITE_SMS | 660 |
| SEND_SMS | 564 |
| ACCESS_COARSE_LOCATION | 556 |

**Table 6. Top permissions requested in ransomware apps**

| Permission | Requests |
|---|---|
| RECEIVE_BOOT_COMPLETED | 2263 |
| WAKE_LOCK | 1481 |
| INTERNET | 1428 |
| GET_TASKS | 1192 |
| READ_PHONE_STATE | 1161 |
| ACCESS_NETWORK_STATE | 1094 |
| BIND_DEVICE_ADMIN | 1093 |
| SYSTEM_ALERT_WINDOW | 956 |
| KILL_BACKGROUND_PROCESSES | 835 |
| WRITE_EXTERNAL_STORAGE | 745 |

Malware and benign top ten requested permissions are quite similar with only VIBRATE, ACCESS_FINE_LOCATION, READ_CONTACTS, READ_SMS, WRITE_SMS, SEND_SMS, and RECEIVE_BOOT_COMPLETE being different. The top four requested permissions are the same in slightly different order. However, ransomware's top ten permissions differ greatly from the others with only four of the permissions being in the top ten of malware or benign applications.

Due to how the tool worked, we assumed any specific misclassifications were due to permissions requested by most of one training set not being requested by the specific apk in the testing set. To verify this, we examined the matrices returned by the Java apk analyzer, especially focusing on the ransomware applications most often misclassified. Again, due to the nature of the tool the apks were classified slightly differently upon each run. To compare the permissions requested by the misclassified apks to the most frequent ransomware permissions listed in table 6, we ran the algorithm a total of 20 times and averaged the results. On average, 85% of the ransomware applications that were labelled as benign requested at least 7 of the top 10 ransomware permissions requested in table 6. This makes us believe that there is a specific combination of other permissions that ransomware frequently requests that these specific apks did not request, mislabelling them as benign.

# 5. DISCUSSION

Through the use of static permission analysis the detection tool was able to identify benign, malware, and ransomware applications with general success. The success can be thought of in two different scenarios: overall accuracy and false negatives. For ransomware and other extremely harmful types of malware a false negative is dangerous as that malicious application will infect the users phone. The false negative rates are promising with only .6% and 13.9% for the two tested ransomware datasets. Both of these results, especially the first, show that the tool can be applied to detect ransomware and other malicious types of software with relative success. Since the tool does not have real time detection if ransomware infects the phone there is no way to stop the process and therefore if the ransomware encrypts files it will likely be impossible to recover the files without paying the ransom. Thus, if full protection against ransomware is needed a real time detection tool such as RansomProber is needed [5]. However, with the low execution time and memory footprint the detection tool can be effective at screening applications that are put on a third party marketplace.
Another application is in distinguishing between malware and ransomware. The tool was extremely effective at determining if an application was malware or ransomware given that it is known that it is not benign. With the lack of a comprehensive ransomware dataset this tool could be effective in helping create one for future use in studies and for the creation of new tools.

Additionally, across all of the datasets each type of application, malware, ransomware, and benign had relatively few sets of unique permissions meaning that the specific permissions each application requests are shared among many applications. This phenomenon allows for permission analysis to be especially effective. If the training set for the detection tool includes a diverse set of applications the probability of the test applications exact or similar permission set being in the training set increases. This can be seen in the detection rates of the two different ransomware test sets. The set where half of the dataset was split into testing and half into training is much more accurate then the solely testing dataset. Thus having a comprehensive training set would lead to higher accuracy.

In order to create a more comprehensive dataset understanding how permission sets evolve would be necessary. In addition, if permission sets evolve year to year it would be possible to have different training set depending on the year the application was created or last updated. This whole process would require development of comprehensive datasets specifically for ransomware as well as analysis on permission request evolution. In addition, the tool only checks if a permission is requested not if it is used. So, overprivileged applications may be falsely labeled if the permissions they request are determined to be malware or ransomware. Thus an addition to the tool for checking if a permission is used could solve this problem and increase accuracy. However, this would lead to an increased runtime which could limit the viability of tools on a larger scale such as analyzing a third party marketplace.

# 6. RELATED WORKS

Chen et al. [5] used a dynamic analysis approach to finding ransomware and was able to approach a detection rate of 99%, with only 9 false positives. As our approach was a static, permissions based approach and theirs was dynamic, it is not best to directly compare our results with the paper that used the first ransomware set. Other related papers focused on permission-based analysis only with malware and benign applications. For example, PAMD used static analysis. PAMD examined the permission requests listed in each application's manifest file and assigned each of these permissions a weight based on how risky that permission is. It is then fed through a decision tree classifier to denote whether the app is malicious based on those permissions or not. It ended up having an accuracy rate of 85%, with a false-negative rate of 16.13% and a false positive rate of 13.79% when run on 60 apps [9]. This is worse than our results, but this could be for a variety of reasons, including the decision tree improving in the last six years, our large sample size compared to their small sample size, and the fact they focused on specific permissions where we did not focus on any specific permissions.

Jiang et al. [6] was similar to PAMD, this solution only examines permission requests, but it takes them from both the manifest files and the code itself to ensure the application used the permission. They used a much larger sample size than PAMD and similar to ours, with 1700 benign and 1600 malware applications. They were able to achieve a true positive rate of 94.5% [6].

Another example of static analysis with an extremely large data is DREBIN. DREBIN had 5,560 malware samples and 123,453 benign apps. DREBIN is also specifically good for novel malware. It does use machine learning in its approach. The approach is based on feature extraction, taking as many features as possible from the manifest and bytecode, permissions requested, app components names, and intents. It then uses machine learning to output these features to a vector space, where a classifier is then run to decide whether the app is malicious or not. DREBIN was able to detect malware with 94% effectiveness and 1% false positivity rate. [1]

To do accurate permission-based prediction, it seems good to have a large sample size of applications that one tests on. In addition, it is important to look at the byte-code and features outside permissions declared in the manifest which DREBIN does.

## 7. CONCLUSION

Through the development of a static permission analysis we were able to identify benign, malware, and ransomware applications by analyzing their apks. By extracting and formatting the requested permissions as a list of binary values indicating if the application requested a specific permission with a one for requested and a zero from not. Creating three training sets for benign, malware, and ransomware applications. We were able to train our decision tree machine learning model from these sets. Lastly we tested our model on five different sets. Three of these sets are part of larger datasets that were divided into both the training and testing sets. Then there are two additional testing sets based on a separate ransomware dataset and a SMSmalware dataset. The two additional testing sets had lower accuracy as their permission sets differed from the training more than the other testing sets. This is expected as they are from different years and SMSmalware is a subtype of malware that may not have been well represented in the training set. The tool had accuracy rates of 94.8% and 99.0% for general malware and the first ransomware dataset respectively and accuracy rates of 66.2% and 69.8% for the SMSmalware and second ransomware dataset respectively. These rates show promise for the ability for the tool to detect malware and ransomware given the training set is comprehensive. In addition to detection, we characterized the most used permission for benign, malware and ransomware applications. We found similar popular requests between malware and benign applications and different popular permission requests for malware. Malware and ransomware also requested on average thirteen permissions while benign only requested four. In addition, a phenomenon was noticed that across all 4,218 applications there were only 810 unique sets of permission requests. This means many applications request the same permissions as other applications. Across all different sets: malware, ransomware, benign, and SMSmalware there were only 17 duplicate permission sets leading to evidence for the efficacy of permission analysis in identifying malicious applications. Thus, the tool developed can be used to identify benign, malware, and ransomware applications and the accuracy can be improved with a more comprehensive training set as well as further research into how permission use has evolved over time.

## 8. ACKNOWLEDGMENTS

We would like to acknowledge Professor Xiao from Case Western Reserve University for his help in the creation of this project as well as for providing the malware and benign datasets. In addition, we would like to thank the authors of *Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection* Chen J., Wang C., Chen K., Du R., and Ahn G. for providing the ransomware dataset. And we would like to thank the Canadian Institute of Cybersecurity for providing the SMSmalware and second ransomware dataset. Finally, we would like to thank the developers of Scikit, without their machine learning tool, we could not have created the decision tree model to predict benign, malware, and ransomware applications.

## 9. REFERENCES

[1] Arp, D. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings 2014 Network and Distributed System Security Symposium. (Feb. 2014).

[2] Avinash Navlani. 2018. Visualization of complex connected decision tree, DataCamp.

[3] Canadian Institute of Cybersecurity. 2017. Apks Of Android Applications by Type. (2017).

[4] Chebyshev, V. Mobile malware evolution 2020 (2021)

[5] Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., Ahn, G. Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection. IEEE Transactions on Information Forensics and Security. 13, 5 (December 2018), 1286–1300.

[6] Jiang, X., Mao, B., Guan, J., Huang, X. Android Malware Detection Using Fine-Grained Features. Scientific Programming. 2020, S.I. (article ID 5190138). (Jan. 2020).

[7] Jakub Křoustek. 2017. Wana_decrypt0r_2.0.png, Avast.

[8] Olmstead, K., & Atkinson, M. Apps permissions in the Google Play Store. (2015)

[9] Pham, G., Duc, N., Vi, P. Permission Analysis for Android Malware Detection. In *Proceedings of the 7th Vast-Aist Workshop "Research Collaboration: Review and Perspective".* (Nov. 2015), 2017-2216.

[10] Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In Proceedings 2014 Network and Distributed System Security Symposium. (Feb. 2014).

[11] Sanam Malhotra. 2020. Why Scikit-learn is Optimum for Python-based Machine Learning. (March 2020). Retrieved May 14, 2021 from https://artificialintelligence.oodles.io/blogs/scikit-learn-for-python-machine-learning/

[12] scikit-learn developers. 1.10. Decision Trees. Retrieved May 12, 2021 from https://scikit-learn.org/stable/modules/tree.html

[13] scikit-learn developers. Getting Started. Retrieved May 13, 2021 from https://scikit-learn.org/stable/getting_started.html

[14] scikit learn. Simplest example of ordinary linear regression application., scikit-learn developers.

[15] Song, S., Kim, B., Lee, S. The Effective Ransomware Prevention Technique Using Process Monitoring on Android Platform. Mobile Information Systems. 2016, (Mar. 2016).