Machine Learning-Based Android Malware Detection Using Manifest Permissions

J. Todd McDonald
Dept of Computer Science
School of Computing
University of South Alabama
jtmcdonald@southalabama.edu

Nathan Herron
Dept of Computer Science
School of Computing
University of South Alabama
nbh1001@jagmail.southalabama.edu

William Bradley Glisson
Cyber Forensics Intelligence Center
Dept of Computer Science
Sam Houston State University
glisson@shsu.edu

Ryan K. Benton
Dept of Computer Science
School of Computing
University of South Alabama
rbenton@southalabama.edu

Abstract

The Android operating system is currently the most prevalent mobile device operating system holding roughly 54 percent of the total global market share. Due to Android's substantial presence, it has gained the attention of those with malicious intent, namely, malware authors. As such, there exists a need for validating and improving current malware detection techniques. Automated detection methods such as anti-virus programs are critical in protecting the wide variety of Android-powered mobile devices on the market. This research investigates effectiveness of four different machine learning algorithms in conjunction with features selected from Android manifest file permissions to classify applications as malicious or benign. Case study results, on a test set consisting of 5,243 samples, produce accuracy, recall, and precision rates above 80%. Of the considered algorithms (Random Forest, Support Vector Machine, Gaussian Naïve Bayes, and K-Means), Random Forest performed the best with 82.5% precision and 81.5% accuracy.

1. Introduction

The Android operating system has consistently been a significant contender in the mobile operating systems market. As of June 2018, the Google Play store features over 3.3 million apps [1], and as of December 2018, Android boasts 54.2% of the global market share [2]. Due to Androids' consistent popularity, it has become a prime target for malware authors. In recent years, Android-powered devices have become increasingly targeted due in part to their increased use for business and financial tasks. Apps now routinely process sensitive financial and personal information as part of mobile banking, social media, and communication programs.

Norton Anti-virus (AV) defines malware as "software that is specifically designed to gain access to or damage a computer, usually without the knowledge of the owner" [3]. Norton further delineates types of malware as spyware, ransomware, viruses, worms, Trojan horses, and adware. In 2017, Kaspersky Labs reported the detection of 5,730,916 malicious installation packages, 94,368 mobile banking Trojans, and 544,107 mobile ransomware Trojans [4]. As such, it can be said that there exists a strong need for accurate and reliable commercial anti-virus (AV) tools in the Android environment and that malware in mobile devices can be a substantial threat [5].

While academicians are interested in detecting malicious activity [17,30-31], opportunities abound to improve Android malware detection accuracy in commercial AV. Zhou and Jiang [7] evaluated Android malware detection using the following anti-virus programs: AVG Antivirus Free v2.9 (AVG), Lookout Security & Antivirus v6.9 (or Lookout), Norton Mobile Security Lite v2.5.0.379 (Norton), and TrendMicro Mobile Security Personal Edition v2.0.0.1294 (TrendMicro). The anti-virus programs were used to scan separate devices afflicted with 1,260 samples of malware. Of the 1,260 samples, AVG was able to detect 689 samples (54.7%), Lookout 1,003 samples (79.6%), Norton 254 samples

URI: https://hdl.handle.net/10125/71460
978-0-9981331-4-0

(20.2%), and TrendMicro was able to identify 966 samples (76.7%). Nguyen et al. [17,30] reported similar results in a study of AV accuracy in detecting repackaged apps, where a newly repackaged botnet version of the popular Snapchat application was not detected by 12 different AV products including AVG, CM Security, Avast, Norton, Kaspersky, and others. In addition, the representative zero-day sample was not detected by online research engines such as Sandroid, AndroTotal, VirusTotal, and OPSWAT [17].

This overall environment spurs the need to improve the security of the large market share of end-users. This research addresses whether malware can be detected by analyzing permissions that accompany Android binaries, supplementing prior work with smaller test data sets and similar machine learning (ML) algorithms [25,38-40]. A case study analyses is performed on a corpus of 4597 known benign Android apps and 6000 malicious Android apps, with comparison to a popular AV engine (VirusTotal) based on four different single algorithm ML approaches. The balance of the paper provides background and related work (Section 2) and a description of the case study methodology (Section 3). Section 4 presents data from the analysis of commercial anti-virus (AV) engines, as well as effectiveness results of ML algorithms. Section 5 draws conclusions based on the results and identifies future work related to this research.

2. Background

The influx of mobile devices and applications is pressing the need for mobile security research. Android applications are deployed in an Android Package Kit (APK) format, which relies on traditional ZIP compression [4]. Repackaging is a significant threat because malicious reverse engineering of APK files is relatively easy given readily available standard open source tools [5]. In the traditional attack model, repackaged apps use cloned code along with inserted ads that redirect advertisement revenue [5-7]. Likewise, repackaged apps can have inserted malicious code on top of benign code that will be spread by unsuspecting end-users [8]. Malware detection approaches are broadly categorized as static and dynamic. Static approaches use parts of the APK file without running the application, while dynamic approaches require some type of sandbox or emulation environment to execute the app for collection of data [9]. Heuristic methods utilize rule-based inference to model apps as malicious or benign.

2.1. Android Architecture

The Android software stack, illustrated in Figure 2, provides a layered approach for supporting Android applications. Android app are compiled from source code, data files, and resource files using the Android Software Development Kit into an APK, an Android package, which is an archive file with a *.apk* suffix [9]. An APK file contains all the required content of an Android application and is the file that is used for application installation. All components of the application must be accounted for in a single *AndroidManifest.xml* file that resides in the APK archive.

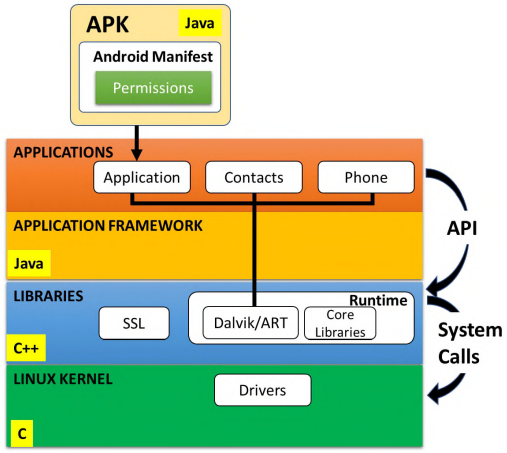

Figure 1: Android OS Architecture [23]

This research focuses on using the manifest file as input to chosen machine learning algorithms. The manifest file also stores a large amount of information such as permissions required by the application [10]. The exact contents of the manifest file vary based on the application: an example of an Android permission statement is shown in Figure 2. Developers may also define custom permissions.

```
<manifest ... >
    <uses-permission android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```

Figure 2: Manifset Permissions Labelling [9]

2.2 Android Malware

The landscape of Android malware is ever-changing. In March 2019, Kaspersky Labs released its annual evaluation of mobile malware showing that its services detected over 5.3 million malicious installation packages [11]. McAfee released a report in the first quarter of 2019, indicating that they had detected nearly 2 million samples of new mobile malware in quarter four of 2018 alone [12]. Some

common types of malware seen throughout 2018 were: Adware [13], Banking Trojans [14], Mobile Crypto Miners, and Repackaged applications [15-16]. The malicious datasets used in this research come from a wide variety of virus categories, but with no specific classification given to their family or type. Thus, the results of the research represent a true random sampling of potential Android malware that may be encountered in the wild. Section 3.1 provides more detail about malicious APKs used in this research study.

2.3 Static Detection

Static analysis is a process by which a program is analyzed without execution. The most common form of malware detection in AV programs is based on signature analysis. Signature-based detection works by scanning a file and then generating a unique identifier, a signature, for that specific file. The signature creation process varies between anti-virus programs. Typically, a hash of the file is created and compared against a table of the hashes of known malicious files. If the hash of the scanned file matches that of a known malicious file in the database, then it can be assumed to be malicious. Items that frequently undergo review in static analysis processes are as follows: source code, assets, manifest files, string patterns, and files that are known to be malicious. The shortcoming of signature-based detection methods is that if no signature exists for comparison, then nothing can be said about the file in question.

Schmidt et al. [18] developed a method of comparing application function calls with the function calls of malicious samples using the Prism, and Nearest Neighbor Algorithms (PART). Although they reported high accuracy for their static detection approach, they stated that in a real-world scenario, the tasks they performed would be impossible due to resource limitations. As a conclusion, they cited the need for further work to create time-efficient methods of detection.

Desnos [19] states that Android's open-source framework and its Java source code allow for malicious authors to tamper with Android applications easily. Therefore, they implement a method of static analysis using similarity distance to determine if an application is malicious or benign. Though further optimized than previous work, the authors state that there is a need for automated behavior analysis and that manual analysis and signature-based detection methods are insufficient.

Feng et al. [20] developed Apposcopy, a semantics-based method for identifying Android malware that aims to steal private user information. Apposcopy makes use of static analysis procedures as well as analysis of control flow to detect malware. Using 1027 malicious samples from 15 different malware families, Apposcopy reported an accuracy of 90% with 103 false negative (FN) and only 2 false positive (FP) results. They also analyzed 11,215 apps from the Google Play store, with only 16 reported as malware.

There is also current literature that bases static detection of malicious Android apps on machine learning in general, which is the focus of the research presented in this paper. Tam et al. [23] address issues about the sandboxed nature of Android applications. To access the system, all applications must be granted permissions by the Android Permission System during installation. Once installed and permissions have been given and enforced by the kernel, applications can interact with each other through the system with the use of API calls. Unfortunately, these rules apply to anti-virus applications as well and prevent applications from being inspected by other applications. Due to this reason, signature-based detection methods are the most viable [23].

Kang et al. [24] proposed a means of detecting and classifying Android-based malware using static analysis. Their study centered around three significant points. First, detection methods using static analysis should be associated with creator information. In particular, their approach used the serial number of certificates and analysis of app components such as permissions to determine malignancy. Second, a scoring algorithm was implemented that placed malware into families based on scores calculated from weights assigned to API calls, permissions, and system commands. Third, the proposed method was tested on live malware samples including 51,179 benign and 4,554 malicious samples. Their approach was able to correctly detect malignancy in 98% of all samples as well as correctly classify samples into corresponding malware families 90% of the time. Such results show that features based on certificate signatures, API calls, permissions, and system commands form a promising line of research.

2.4 Detection Based on Manifest Files

Our work is not the first to utilize manifest files as the source of static analysis or machine learning algorithms. The work of Milosevic and Dehghantanha [25] utilized static malware analysis techniques using both supervised and unsupervised machine learning methods. Of similarity to the work

in this paper, one of their four experiments utilized supervised machine learning algorithms where permission-based classification was in focus. For the training of their machine learning models, they made use of the M0Droid dataset, which includes 200 malicious and 200 benign Android applications. Algorithms tested in this experiment include SVM, Naïve Bayes, C4.5, and JRIP. The results of their permission-based analysis test are shown in Figure 3. The case study reported in this paper is similar because we also use Random forest, Bayesian networks, and SVM for classification based on Android permission features, but their result comes from a much smaller data set. Further comparison of results is provided in Section 4.

Algorithm	Precision	Recall	F-Score
C4.5 decision trees	0.827	0.827	0.827
Random forest	0.871	0.866	0.865
Bayes Networks	0.747	0.747	0.747
SVM with SMO	0.879	0.879	0.879
JRip	0.821	0.819	0.819
Logistic regresaion	0.823	0.822	0.821

Figure 3: Permission-Based Classification Using Single ML Algorithms [25]

In a 2016 study, Kumaran and Lee [38] report results of a study of 500 benign and 500 malicious apps. They focused on 183 features derived from requested permissions, declared permissions, and intent filters in the Android manifest file, which formed the basis of three data sets. Their reported best accuracies of various ML algorithms across the three sets were 63.6% for intents, 90.5% for permissions, and 91.7% for combined permissions. For the combined dataset, 6 different ML algorithms were studied (of which SVM is also considered in our study) and the following accuracies using ten-fold cross-validation were reported: Linear Discriminants (82.6%), Cubic SVM (91.7%), Weighted KNN (91.4%), Complex Tree (89.3%), Linear SVM (89.2%), Course KNN (79%). True Positive Rate (TPR)/False Negative Rate (FNR) of benign apps was reported as 94.2%/5.8% and TPR/FNR rates of malicious apps was reported as 89.2%/10.8%. In comparison, our case study uses a larger data set and three different ML algorithms, and only considers permissions. Further comparison of results is provided in Section 4.

In a 2017 study, Wang et al. [39] report development of Mlifdect, which provided a holistic study of parallel machine learning processing with combined classification of manifest information. Closer to the data set in this research, Mlifdect performance was evaluated on app dataset of 3,982 malicious and 4,403 benign apps using ML algorithms with 10-fold cross validation. Unlike this work, their approach fused information from 3 different ML algorithms (KNN, J48, and Random Forest) computing in parallel on 2 different features sets. One feature set used a combination of API calls, requested permissions, intents, and components and the other feature set contained command, hardware, protected strings, and network URLs. Our approach only focuses on permission and uses two of the same ML algorithms: K-means (similar to KNN) and Random Forest. However, we only consider the efficacy of a single algorithm instead of fusing combined results. Wang et al. [39] extracted 65,000 features covering eight categories of Android manifest permissions. Reported accuracy from Android benign and malware apps with Mlifdect was 99.7%.

In 2013, Sanz et al. [40] reported results of Android malware analysis using MAMA, based on manifest analysis. They consider both permissions and "uses" features found in the manifest file, where we only consider permissions in this study. They used variations of Naïve Bayes, Bayesian Network (K2 and TAN), Support Vector Machine (Polynomial and Normalized Polynomial Kernel), J48, KNN (K = 1, 3 and 5), and Random Forest (N = 10, 50 and 100). Figure 4 shows their reported TPR/FPR and accuracy results for permissions-only analysis. Best results were with Random Forest, using a population of 100 trees, achieving an 87% detection accuracy. Of note, their study down-selected a balanced set of 333 benign and 333 malicious apps for analysis, whereas our study is on a larger and more modern set of apps. Further comparison of results is provided in Section 4.

Algorithm	TPR	FPR	AUC	Accuracy (%)
Naïve Bayes	0.87	0.42	0.77	72.78
BN K2	0.90	0.41	0.79	74.49
BN TAN	0.91	0.41	0.85	74.99
SVM: Polynomial Kernel	0.74	0.07	0.83	83.34
SVM: Normalized-Polynomial Kernel	0.79	0.05	0.87	87.14
KNN K =1	0.80	0.06	0.93	87.28
KNN K =3	0.77	0.10	0.93	83.77
KNN K =5	0.76	0.10	0.92	83.14
KNN K =10	0.73	0.20	0.89	76.65
J48	0.72	0.10	0.87	80.90
Random Forest 10	0.80	0.06	0.94	87.07
Random Forest 50	0.79	0.05	0.95	87.28
Random Forest 100	**0.80**	**0.05**	**0.95**	**87.41**

Figure 4: Permission-Based Classification Using Single ML Algorithms [40]

2.5 Dynamic and Heuristic Analysis

Dynamic analysis methods are less frequent in commercial anti-virus scanners due to their intense

resource requirements and complex nature. Burgera et al. [21] developed an approach for automated dynamic analysis as a means of malware detection in Android devices, called Crowdroid. The detector relies on traces submitted from many users across many different types of devices. While Crowdroid did implement a method of automatic detection, the process still relied on signatures collected from users. In addition to being resource-intensive, the dynamic analysis also has the drawback that a large number of false positives may be generated.

Heuristic analysis is a relatively new approach [36] and makes use of a wide variety of methods to identify components of a program that can be used to create an inference as to the nature of the said program. A set of rules is established that determines the criteria for what flags a file as benign or malicious. The rules vary widely, depending on the author. However, they are usually generated by using pre-created algorithms and models that are used in data mining and machine learning. An example of such is the work of Suarez-Tangil et al. [22], which made use of vector space modeling and text mining to compare applications to generalized malware samples.

3. Machine Learning Methodology

The goal of this research is to examine efficacy of single-algorithm machine learning techniques on manifest permissions on a reasonably sized Android app data set. The aim is to evaluate performance in comparison to commercial AV products to see if single aspect detection (manifest permissions) using single ML algorithms provide as good or better results. Secondarily, the case study approach extends prior results on manifest-based feature detection by either examining a larger set of more recent apps and different ML algorithms. The methodology is divided into five phases, detailed next.

3.1 Phase 1: Setup and Data Acquisition

To begin, an environment is created in which analysis on malicious applications can be safely performed. For the test environment, experiments are executed on a Windows10 Pro Edition host machine running VMWare Workstation 15 Pro with an Ubuntu version 19.10, Eoan, operating system. This virtual environment provides the guarded sandbox in which all tests are conducted. Scripts used in experiments are written in Python 3.7 and Bash. Given the setup of the virtual environment, data sources are migrated to the environment which include a collection of 4597 benign applications from the Google Play store that have previously been run through anti-virus to verify their benign nature. These samples are part of a pre-existing internal private repository of APK files that were pulled from the Google Play store in 2017 and verified to be benign through VirusTotal. The benign samples act as the control set of data. In addition to the samples collected from the Google Play store, 6000 malicious samples are collected from the AndroZoo [28] repository. The samples are sourced from both the Google Play store and AppChina third-party market, and no sample exceeded 30 megabytes (MB) in size. The AndroZoo repository hosts a large collection of malicious Android applications with classifications of malware ranging from adware to repackaged applications.

All malicious samples are randomly selected applications that are no older than January 1st 2018 from the AndroZoo repository using the python library (AZ) that AndroZoo provides [28]. This sampling ensured the malicious set was representative of the current Android environment. For collection of the applications taken from AndroZoo, a script created in Python 3.7.3 is used that allows download from the repository. An API key must be provided to the script in order to run and can be requested by emailing *androzoo@uni.lu* and stating the name of the research institution and the name of the individual requesting access. All applications for AndroZoo access must be sent from a university or research institution email account.

3.2 Phase 2: Evaluating Commercial AV

As part of the research approach presented in this paper, performance of machine learnings algorithms is compared against performance of typical and current anti-virus engines used commercially. In order to evaluate the effectiveness of common commercial anti-virus engines, the malicious and benign data sets are submitted to VirusTotal [32] in their entirety. The VirusTotal free API is used for such uploads. The free API allows for submission of 1000 samples per day, and 30,000 samples per month. An API key can be requested from VirusTotal simply by navigating to the website [32] and signing up for a free account. The VirusTotal website provides a means to upload samples in batch by using the officially supported Windows or Mac application.

To facilitate the upload of the data set used in this research, the entire data set is split into smaller batches of roughly 900 samples each. This resulted in seven batches for the malicious set and five for the

benign set. It should be noted that this is solely for the purpose of uploading the samples to VirusTotal, and that the samples are not split into batches when chosen machine learning algorithms are applied during the algorithm evaluation phase.

3.3 Phase 3: Data Preparation

In phase three, APK files are decoded as part of data pre-processing for machine learning (ML) algorithms. The APK files, much like unextracted ZIP archive files, are largely useless in a raw form. To decode the APK, the Apktool version 2.4.1 application is used, which represents a common command line interface tool used for reverse engineering Android applications. Running Apktool requires its wrapper script, its JAR file, and a Java Runtime Environment (JRE) version of 1.8 or higher. Running Apktool against a given APK file produces several items: 1) an assets folder containing all the application assets; 2) a folder containing metadata; 3) SMALI files; and 4) the *AndroidManifest.xml* file (referred to simply as the *manifest* file). After retrieving the file of interest (i.e., the manifest file), the manifest file is converted from an XML document into a format that the ML algorithms can work with, which is a comma separated value (CSV) file containing no string data. All APKs in the benign and malicious sets are processed in this manner.

3.4 Phase 4: Algorithmic Choice

For testing effectiveness of various ML approaches, two options exist for selecting a set of features for a given manifest file: 1) manually selecting or 2) allowing an individual algorithm to place weight on the individual permissions. In this research, the latter approach was chosen, allowing ML algorithms to place weights on the individual permissions to create the set of features. In some ways, this approach is similar to the technique used by Sato et al. [26], where key malicious features identified by those researchers included permissions such as BOOT_COMPLETED, SMS_RECIEVED, and CONNECTIVITY_CHANGE. Additionally, custom permissions are included in this research, which are taken from the sample sets created for analysis. Additional inspiration for determining features is the list of commonly used permissions, which were adapted in this research from the work of Sarma et al. [27]. Figure 5 shows the twenty most common parameter sets. The algorithms utilized for this research are: Support Vector Machine (SVM), Gaussian Naïve Bayes, K-Means, and Random Forest.

Permission	Benign	Malicious
INTERNET	68.50 (1)	93.38 [1]
ACCESS_NETWORK_STATE	30.97 (2)	42.98 [8]
READ_PHONE_STATE	24.99 (3)	80.99 [2]
WRITE_EXTERNAL_STORAGE	24.14 (4)	59.50 [4]
ACCESS_COARSE_LOCATION	18.17 (5)	43.80 [7]
ACCESS_FINE_LOCATION	17.22 (6)	35.53 [12]
WAKE_LOCK	13.07 (7)	23.14 [18]
VIBRATE	12.84 (8)	23.14 [19]
ACCESS_WIFI_STATE	8.09 (9)	28.92 [16]
RECEIVE_BOOT_COMPLETED	7.99 (10)	23.14 [20]
READ_CONTACTS	7.50 (11)	47.11 [6]
GET_TASKS	5.32 (12)	5.78 [30]
CALL_PHONE	5.10 (13)	31.40 [14]
SEND_SMS	4.83 (14)	64.46 [3]
SET_WALLPAPER	4.75 (15)	30.57 [15]
CAMERA	4.35 (16)	5.78 [30]
GET_ACCOUNTS	4.31 (17)	4.95 [31]
RECEIVE_SMS	4.29 (18)	40.49 [10]
WRITE_SETTINGS	3.90 (19)	7.44 [27]
PROCESS_OUTGOING_CALLS	3.64 (20)	4.13[36]
READ_SMS	3.43 (21)	47.11 [5]
READ_HISTORY_BOOKMARKS	0 (113)	42.14 [9]
WRITE_HISTORY_BOOKMARKS	0(113)	37.19 [11]
WRITE_CONTACTS	1.99 (23)	32.23 [13]
MOUNT_UNMOUNT_FILESYSTEMS	1.25 (28)	26.44 [17]

Figure 5: Common Permissions [27]

Rationale for choosing these four ML algorithms among many other possible choices are delineated next. Naïve Bayes is one of the simpler classifiers, which often results in competitive performance despite the assumption of independence between attributes; this approach depends on Bayesian probability [34,35]. Random Forest has a history of strong performance, often leading to near optimal results; it also represents a tree-based learning paradigm [34,35]. SVMs have a strong theoretical basis for their operation, and like Random Forests, tend to provide strong results; it is based on optimization of hyperplanes [34,35]. K-Means is a clustering approach that allows for the discovery of natural groupings within the data; by assigning "clusters" the label of the majority class, it can be converted into an effective classification solution [34,35].

3.5 Phase 5: Evaluation

In the evaluation phase, the four chosen ML algorithms are exercised against the benign and malicious data sets. Results are taken from the iterations of testing, which produce a binary classification on whether each manifest file is either malicious or benign. A malicious file that is correctly classified as malicious is counted as true positive (TP), whereas a malicious file classified as benign is considered a false negative (FN). A manifest file from a benign APK that is classified as benign is considered a true negative (TN), whereas a benign sample categorized as malicious is counted as a false

positive (FP). These values form the basis of detection rates which are used in further statistical measures to assess effectiveness of a given algorithm. Detection rates from the four machine learning algorithms are compared against the detection rates of commonly used anti-virus software featured on VirusTotal, which includes over 70 anti-virus engines [33]. Data from all evaluations are used to determine which methods perform best in terms of specificity, sensitivity (recall), accuracy, precision, F1 score, and the output of their respective confusion matrices.

4. Experimental Results

This section presents results and analysis of each of the four chosen ML algorithms and the evaluation of samples using VirusTotal commercial anti-virus. Effectiveness of ML approaches are compared against this baseline of results for typical commercial AV applications, along with prior work.

4.1 Commercial Detection Engine Evaluation

To begin, the benign set of APKs are uploaded to VirusTotal. There were issues submitting some samples to VirusTotal, as files larger than 32mb in size are unable to be loaded. Therefore, these samples returned a null value and are excluded from analysis results. Of the 4,279 samples successfully submitted to VirusTotal from the benign set, there were a total of 789 unique false positives, with 2,879 total instances of a false positive instance occurring across all detection engines. 789 unique instances of false positives (FP) resulted in a false positive rate of 18.4%. 2,879 total instances of a false positive evaluation out of 253,105 total evaluations yields a 1.137% false positive rate against all instances of evaluation.

While VirusTotal offers support for over 70 engines in total, not all engines are able to process all samples. For a given sample, the number of total engines that were able to process the sample were recorded. Data collection included a mapping of samples to their evaluation engines, detection rates, anti-virus engine version numbers, and a full list of the SHA256s for each APK file. These raw results are stored in a private Github repository, with access granted and available upon request [37].

Next, the malicious set of APKs are uploaded to VirusTotal in the same manner. These resulted in a higher evaluation success rate, but out of 6,000 samples, only 98 were unable to be submitted due to the incompatibility with VirusTotal. The malicious data set resulted in 848 unique detections, resulting in a unique detection rate of 14.37%. Additionally, there were 3,856 total detections across all engines, resulting in a detection rate of 1.04% across all evaluations computed. Figure 6 provides a summary of the TP and FP unique detection rates for the benign and malicious sample sets.

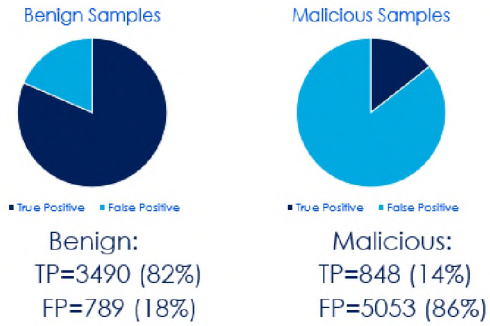

Figure 6: VirusTotal Detection Rates

4.2 ML Algorithm Evaluation

In this section, we detail the results of the computations generated from the four algorithms: Random Forest, SVM, Gaussian Naïve Bayes, and K-Means. All algorithms were implemented using Python 3.7, the *numpy* library (for computations), the *pandas* library (for data analysis and manipulation), and the *sklearn* library (for algorithm implementation). Training and test sets were taken from the benign and malicious APK sets (referred to in section 3.1). Algorithms read in content from pre-processed CSV files (referred to in section 3.3), stores them into a data frame (one for malicious one for benign), shuffles their respective contents, and then splits them evenly into two data frames. Hence, two sets of benign programs and two sets of malware programs are created.

Once the data frames have been shuffled and split, the benign data frame 0 is concatenated with the malicious data frame 0 to produce the learning set; the other two data frames produce the test set. For results, TP/FP/TN/FN counts are recorded (in a confusion matrix) and other key statistical values are computed as follows:

- **Sensitivity/Recall/TPR**: TP / (TP + FN)
- **Specificity/TNR**: TN/(TN + FP)
- **Precision**: TP / (TP + FP)
- **Accuracy**: TP + TN/(TP + TN+ FP + FN)
- **F1**: 2*(Precision*Recall)/(Precision+Recall)

Figure 7 provides a consolidated view of the confusion matrices produced by all four ML algorithms and the VirusTotal engine results. Figure 8 provides a summary of statistical factors produced

by each of the four ML algorithms. Figure 9 provides a receiver operating curve for all four ML algorithms.

Random Forest. The implementation of random forest successfully executed with no issues. The algorithm correctly labeled 2,577 samples as malicious, our true positive value. The algorithm also successfully labeled 1,698 samples as benign, our true negative value. Random forest incorrectly labeled 547 samples as malicious (false positives) and labeled 421 samples incorrectly as benign (false negatives). The precision for random forest was calculated at 0.8249, or roughly 83%. The recall was computed to be 0.8585, or roughly 86%. Accuracy was computed to be 0.8153, or roughly 81%. The F1 score was calculated to be 0.84188, or roughly 84% and the specificity is 0.7563 or roughly 76%.

Support Vector Machine (SVM). The implementation of SVM successfully executed with no issues. The algorithm correctly labeled 2,562 samples as malicious, our true positive value. The algorithm was also successful in labeling 1,626 samples as benign, our true negative value. Our implementation of SVM incorrectly labeled 619 samples as malicious (false positives) and labeled 436 samples incorrectly as benign (false negatives). The precision for our implementation of SVM was calculated at 0.8054, or roughly 80%. The recall was computed to be 0.8545, or roughly 85%. Accuracy was computed at 0.7987, or roughly 80 F1 score was computed to be 0.8292, or roughly 83%, and, the specificity is 0.7224 or roughly 72%.

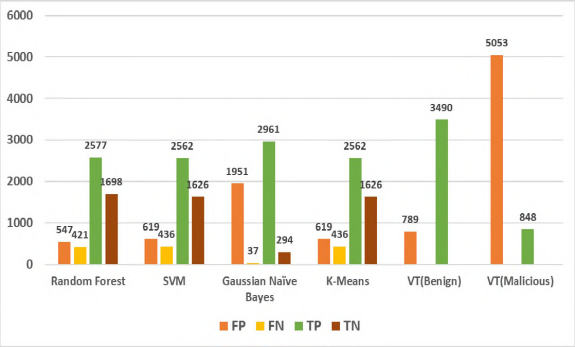

Figure 7: Comparative TP/TN/FP/FN Results

Gaussian Naïve Bayes. Our implementation of Gaussian Naïve Bayes was successfully executed with no problems. The algorithm correctly labeled 2,961 samples as malicious, our true positive rate. The algorithm was also labeled 294 samples as benign, our true negative value. Additionally, the algorithm labeled 1,951 samples incorrectly as malicious (false positives) and labeled 37 samples incorrectly as benign. The precision for our implementation of Gaussian Naïve Bayes was calculated to be 0.6028, or roughly 60%. The recall was computed to be 0.9876, or roughly 99%. Finally, the F1 score was calculated to be 0.7486, or roughly 75% and the specificity is 0.1313 or roughly 13%.

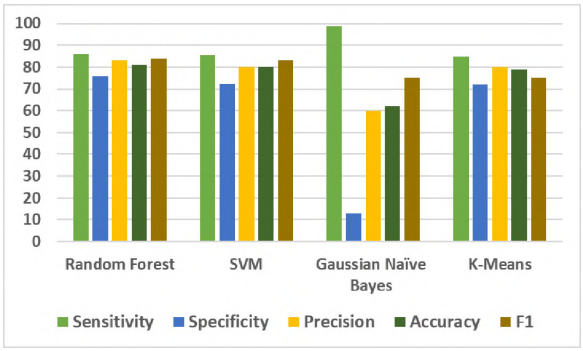

Figure 8: Comparative Performance Results

K-Means. Our implementation of K-Means was successfully executed without issue. The algorithm correctly labeled 2,562 samples as malicious, our true positive rate. The algorithm also correctly labeled 1,626 samples as benign, our true negative value. In addition, the algorithm incorrectly labeled 619 samples as malicious (false positives) and 436 samples as benign (false negatives). The precision for our implementation of K-Means was calculated to be 0.80541, or roughly 80%. The recall was calculated at 0.8545, or roughly 85%. Finally, the F1 score was calculated to be 0.7486, or roughly 75% and the specificity is 0.72428 or roughly 72%.

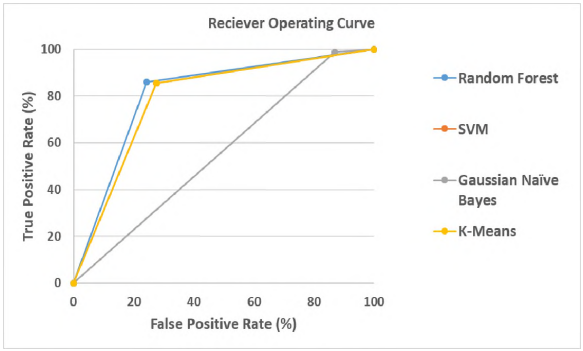

Figure 9: Reciever Operating Curve

In terms of comparison with other ML approaches that focus in some way on manifest-based permissions, Table 1 provides a summary of the experimental results of this work and a comparative with related studies. Of note, our sample size is larger than other comparative work and on a more recent set of generalized malicious Android apps. The focus of this work was to evaluate primarily the efficacy of single ML approaches on manifest permissions alone, whereas comparative work has achieved higher accuracy when ensemble methods or information fusion approaches are used [25,39].

6. Conclusions and Future Work

The substantial prevalence of Android powered mobile devices in the global market share makes Android platforms an attractive target for malware authors. The efficient and reliable detection of malware in the Android environment is a complex problem that doesn't seem close to being solved. The first goal of this research was to present an evaluation of commercial anti-virus tools and how well they perform at classifying samples as malicious or benign. The findings indicate that when presented with a sample set of recently discovered malware (under two years), commercial anti-virus software proves woefully inadequate in terms of detection rates. The malicious set, consisting of 5902 samples, had only 848 unique detections (14.37%). It should also be noted that the benign set of 4297 samples had a false positive rate of 18.4%, or 789 samples

Table 1: Comparative ML Approaches

	B	M	ML	Acc	Best
Milosevic, Dehghantanha [25] (2017)	200	200	6	89%	SVO SVM
Kumaran,Lee [38] (2016)	500	500	6	90.5%	Cubic SVM
Wang et al. [39] (2017)	4,403	3,982	8	99.7%	Fused
Sanz et al. [40] (2013)	333	333	13	87.4%	Random Forest
Case Study (2020)	4,597	6,000	4	81%	Random Forest

B = Benign Samples, M = Malicious Samples
ML = # of ML algorithms, Acc = Accuracy

The second goal of this research was to provide an evaluation of the effectiveness of several commonly implemented machine learning algorithms, namely, Random Forest, SVM, Gaussian Naïve Bayes, and K-Means, when applied to the APK manifest file alone. The findings showed that compared to the detection rates of anti-virus engines readily available, all of the ML algorithms that used just the manifest file alone offered significant improvements, with Random Forest having the highest precision (0.8249), accuracy (0.8153), and F1-score (0.8418), specificity (0.7563) as well as the second highest computed recall (0.8153). The only algorithm that offered less than desirable results was Gaussian Naïve Bayes. Gaussian Naïve Bayes had the lowest precision (0.6028), accuracy (0.6208), and F1 score (0.7486) and specificity (0.1313). Compared to relative work (seen in Table 1), this work also reaffirms that Random Forest and SVM are best at classification when permissions are in view.

Future work will be focused on the creation of a novel algorithm that is more finely tuned towards the detection of malware, as opposed to the use of general-purpose algorithms. Ensemble approaches and the use of additional ML algorithms are also in view for static detection using manifest permission features. There are also many other static features associated with APKs that are elaborated in the current literature that could be easily combined with a manifest file approach to form a greater feature set. From results of this research, a natural follow-on step would be to classify Android malware into families based on permissions. Finally, adversarial machine learning will be considered to account for attacks where adversaries target manifest permission features to deceive ML algorithms.

7. References

[1] Statista, "Number of available applications in the Google Play Store from December 2009 to September 2018," Statista, 2018. [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [Accessed 5 March 2020].

[2] Statista, "Subscriber share held by smartphone operating systems in the United States from 2012 to 2018," Statista, 2018. [Online]. Available: https://www.statista.com/statistics/266572/market-share-held-by-smartphone-platforms-in-the-united-states/. [Accessed 4 March 2020].

[3] Symantec, "What is malware and how can we prevent it?," Norton Anti-virus, 2018. [Online]. Available: https://us.norton.com/internetsecurity-malware.html. [Accessed 2 March 2020].

[4] Kaspersky Labs, "Mobile Malware Evolution 2017," Kaspersky Labs, 07 March 2018. [Online]. Available: https://securelist.com/mobile-malware-review-2017/84139/ . [Accessed 1 March 2020].

[5] Symantec, "What is anti-virus software?," Norton Antivirus, 2019. [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-antivirus.html . [Accessed 22 March 2020].

[6] Bazrafshan Z., H. Hashemi, S. M. H. Fard and A. Hamzeh, "A survey on heuristic malware detection techniques," *The 5th Conference on IKT*, Shiraz, 2013, pp. 113-120, doi: 10.1109/IKT.2013.6620049.

[7] Zhou, Y. and X Jiang, "Dissecting Android Malware: Characterization and Evolution," *IEEE Symposium on Security and Privacy*, 2012.

[8] Google Developers, "Platform Architecture," Google, 3 September 2018. [Online]. Available: https://developer.android.com/guide/platform/. [Accessed 2 December 2020].

[9] Google Developers, "Application Fundamentals," Google, 2018. [Online]. Available: https://developer.android.com/guide/components/fundamentals . [Accessed 14 April 2020].

[10] Dong, S. et al., "Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild", In: Beyah R., Chang B., Li Y., Zhu S. (eds) *Security and Privacy in Communication Networks. SecureComm 2018.* Springer, Cham, 2018.
[11] Kaspersky Labs, "Mobile Malware Evolution 2018," Kaspersky Labs, 05 March 2019. [Online]. Available: https://securelist.com/mobile-malware-evolution-2018/89689/. [Accessed 26 February 2020].
[12] McAfee, "McAfee Mobile Threat Report Q1, 2019," McAfee, 2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf [Accessed 26 February 2020].
[13] Erturk, E., "A Case Study in Open Source Software Security and Privacy: Android Adware," in *World Congress on Internet Security (WorldCIS-2012),* Guelph, ON, Canada, 2012.
[14] Ståhlberg, M., "The Trojan Money Spinner," in *Virus Bulletin Conference*, 2007.
[15] Jain, A., H. Gonzalez, and N. Stakhanov, "Enriching reverse engineering through visual exploration of Android binaries." In *Proceedings of PPREW-5*. ACM, Article 9, 1–9. doi: 10.1145/ 2843859.2843866.
[16] Ibotpeaches, "Apktool - A tool for reverse engineering Android apk files," 2016.
[17] Nguyen, T., J.T. McDonald, W. B. Glisson, and T. R. Andel, "Detecting Repackaged Android Applications Using Perceptual Hashing", *HICSS-53*, January 7-10, 2020, Grand Wailea, Maui, HI, USA.
[18] Schmidt, A., et al., "Static Analysis of Executables for Collaborative Malware Detection on Android," in *2009 IEEE Intl Conf on Comm*, Dresden, 2009.
[19] Desnos, A., "Android: Static Analysis Using Similarity Distance," in *2012 45th Hawaii Intl Conference on System Sciences*, Maui, 2012.
[20] Feng, Y., S. Anand, I. Dillig, and A. Aiken. "Apposcopy: semantics-based detection of Android malware through static analysis," In *Proc 22nd ACM SIGSOFT FSE 2014)*. ACM. doi: 10.1145/ 2635868.2635869.
[21] Burguera, I., U. Zurutuza, and S. Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for Android." In *Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices (SPSM '11)*. ACM, 15–26. doi: 10.1145/2046614.2046619.
[22] Suarez-Tangil, G., J. Tapiador, P. Peris-Lopez, and J. Blasco. "Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families," *Expert Syst. Appl.* 41, 4 (March, 2014), 1104–1117. doi: 10.1016/j.eswa.2013.07.106.
[23] Tam, K., A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro "The Evolution of Android Malware and Android Analysis Techniques," *ACM Comput. Surv.* 49, 4, Article 76 (February 2017), doi: 10.1145/ 3017427.
[24] Kang, H., J. Jang, A. Mohaisen, and H.K. Kim, "Detecting and Classifying Android Malware Using Static Analysis along with Creator Information," *International Journal of Distributed Sensor Networks.* doi: 10.1155/2015/479174.
[25] Milosevic, N., A. Dehghantanha, K. R. Choo, "Machine learning aided Android malware classification", *Computers & Electrical Engineering*, Volume 61, 2017, pp. 266-274, ISSN 0045-7906, doi: 10.1016/j.compeleceng.2017.02.013.
[26] Sato, R., D. Chiba, S. Goto, "Detecting Android Malware by Analyzing Manifest Files," *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, pp. 23-31, 2013. doi: 10.7125/APAN.36.4
[27] Sarma, H.P., N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. "Android permissions: a perspective combining risks and benefits," In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies (SACMAT '12)*. ACM, 2012, pp. 13–22. doi: 10.1145/2295136. 2295141.
[28] Allix, K., T. F. Bissyandé, J. Klein, and Y. Le Traon. "AndroZoo: collecting millions of Android apps for the research community," In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, 2016, pp. 468–471. doi: 10. 1145/2901739.2903508.
[30] Nguyen, T., J.T. McDonald, and W.B. Glisson. "Exploitation and Detection of a Malicious Mobile Application," in *50th HICCS*. 2017.
[31] Luckett, P., J.T. McDonald, W.B. Glisson, R. Benton, J. Dawson, and B.A. Doyle, Identifying stealth malware using CPU power consumption and learning algorithms. *Journal of Computer Security*, 2018, p. 1-25.
[32] VirusTotal [Online]. Available: https://www.virustotal.com/gui/home/upload [Accessed 15 May 2020]
[33] "How it works", VirusTotal [Online]. Available: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works [Accessed 28 Jun 2020]
[34] Aggarwal, C. C., *Data Mining: The Textbook,* Course Materials, First Edition, Springer, 2015.
[35] Han, J., M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Third Edition, Morgan Kaufmann, 2012
[36] Dua, S. and X. Du, *Data Mining and Machine Learning in Cybersecurity,* Auerbach Publications, 2011. ISBN-10: 1439839425
[37] Github, https://github.com/
[38] Kumaran, M. and W. Li, "Lightweight malware detection based on machine learning algorithms and the android manifest file", *REU Poster*, 2016, NYIT School of Engineering and Computing.
[39] Wang, X., D. Zhang, X. Su, W. Li, "Mlifdect: Android Malware Detection Based on Parallel Machine Learning and Information Fusion," *Security and Communication Networks*, article 6451260, 2017, Hindawi. doi: 10.1155/2017/6451260
[40] Sanz, B., I. Santos, C. Laorden, X. Ugarte-Pedrero, "MAMA: Manifest analysis for malware detection in android," *Cybernetics & Systems*, October 2013. doi: 10.1080/01969722.2013.803889