International Journal of Research in Engineering, Science and Management
Volume 5, Issue 1, January 2022
https://www.ijresm.com | ISSN (Online): 2581-5792

Android Malware Detection

Sayi Rosshhun Gadde[1*], J. C. Pavan Kaushal[2], T. Vijay Rao[3], N. Srikanth[4], Sayi Khushhal Gadde[5]
[1,2,3,4]*Department of Computer Science and Engineering, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering & Technology, Hyderabad, India*
[5]*Department of Computer Science and Engineering, CMR Technical Campus, Hyderabad, India*

***Abstract*: For the past few decades, the growth in usage of mobile phones has been increasing abnormally. Recent surveys hypothesize most of the mobile phone market segment is benignly dominated by Android Operating System and this made the Android OS (Operating System) the most vulnerable Operating System; as more users are adopting to use Android OS (Operating System) most often, malware attacks on Android operating systems have been increasing, this can be considered as one of the significant issues and a security threat for every mobile phone users. For the past decade or so, we have been seeing many malware detection software which has adopted a technique called Signature-Based malware detection, which is used to detect malware in Android applications, as the name describes that software extracts a string called the signatures or package name from the input app or APK (Android application package) and tries to predict the presence of malware. However, this approach is limited to identifying only a few known malware. In short, the malware detection software will extract the signature from the Android application and compare it with a set of publicly available databases where package names of known malware apps are available, which contains a list of package names of popular malware applications. The most efficient way of identifying unknown malware is to extract more information regarding the apk. So the point is how we can extract the data within the scope of user permission? So, any tool or a script can find this information in the Android manifest file of the target APK (Android application package). Usually, every android app has this file to let OS know what kind of permissions are requested, and it also stores metadata of the application. So, from the Android Manifest File, the signatures and the approvals defined in that file are then being extracted and compared with the dataset through an artificial neural network; this model will be trained from a huge malware dataset and the input apk, by this way the neural network is capable of identifying the malware by analyzing the extracted permissions and strings.**

***Keywords*: Android, SIGID, permissions extraction, feature extraction, artificial neural network, APK, intents.**

1. Introduction

According to a research study conducted by a semantic scholar, Google Play Store accounts for 67.2% of malicious app installs. In another research, data of 7.9 million apps from 12 million smartphones have been collected for four months and concluded that only 10.1% of malicious app installs are from app stores other than Google Play Store.

Firstly, we have conducted an initial study, where we have analyzed trends in the distribution of malware through apps and the causes of the malware when it gains control over the system. After analyzing all the trends, we have formulated that there are numerous activities a hacker or a malware can do, and they are:

1) Extraction of information

The smartphone can be hacked, and personal data such as the IMEI number and the user's information can be stolen.

2) Computerized phone calls and text messages

The cost of making calls and sending SMS to select premium lines is increased.

3) Root exploits are a type of root exploit

The malware will obtain root access to the machine, control it, and change the data.

4) Search Engine Optimizations

To raise the income of a search engine or the traffic on a website, artificially search for a phrase and mimic clicks on specific pages.

5) Dynamically downloaded code

It refers to an existing application that downloads malicious code and deploys it on mobile devices.

6) Covert channel

A flaw in the gadgets makes it easier for information to escape across operations that aren't supposed to exchange it.

7) Botnets

Command and Control servers are in control of a network of compromised mobile devices with a Bot Master (C&C). Deliver spam and launch DDoS attacks against the host devices.

By observing these trends, we have concluded that most malware attacks are exhibited with the help of user permissions to the application. So, to counter the malware attacks, there are many paid and free tools available, but many of them detect malware based on the application’s signature, which is inefficient.

To counter the malware attacks on Android devices, we have developed a website where users can obtain the APK version of the application and upload it to the website. The backend consists of a trained model which extracts the list of the required permissions and the signature from the android manifest file of the app.

2. Related Work

A literature survey is a comprehensive overview of prior research on a particular subject. A literature review examines scholarly articles, books, and other sources that are pertinent to

*Corresponding author: sayirosshhun@gmail.com a specific field of research. It should provide a theoretical foundation for the study and assist you (the author) in determining the scope of your study.

[1] The SigPD technique is described in the publication. Instead of gathering and evaluating all Android permissions, the model uses data mining to determine the most important permissions that can be classified. After that, SigPID uses machine learning-based classification methods to classify the data. Only 22 permissions are found to be significant in the evaluation. SigPID is more effective, finding 93.62 percent of malware and 91.4 percent of unknown malware samples in the dataset.

[2] From API data, the model creates Boolean, frequency, and time-series data sets. Three detection models for Android malware detection, API calls, API frequency, and API sequence aspects are built using these three data sets. Finally, an ensemble model is constructed and tested using 10010 benign and 10683 malicious applications. The model has an accuracy of 98.98 percent.

[3] The study offers DL-Droid, a deep learning system that uses dynamic analysis and stateful input generation to detect fraudulent Android applications. According to the evaluation, DL-Droid can achieve a detection rate of up to 97.8% (with dynamic features only) and 99.6% (with dynamic Plus static features), which is better than traditional machine learning techniques.

[4] According to the article, the Random Forest classifier outperforms the SVM classifier, which delivers roughly 94 percent accuracy with 97.24 percent accuracy, demonstrating that it is a viable technique for Android malware detection.

[5] The model described in the paper was created by combining four different machine learning algorithms, including deep learning, farthest first clustering, Y-MLP, nonlinear ensemble decision tree forest approach, and rough set analysis as a feature subset selection algorithm, to detect malware from real-world apps with an accuracy of 98.8%.

[6] In comparison to SVM and Naive Bayes, the research claims that Random Forest has the highest accuracy. Random Forest, SVM, and Naive Bayes classifiers are also commonly employed.

[7] To propose an effective technique in Android malware detection, the study suggests employing SVM for classification and PCA for feature selection.

[8] MADAM monitors Android at both the kernel and user levels at the same time to detect actual computer deformities and discriminate between normal and malicious behaviour using machine learning techniques. MADAM's prototype can detect a variety of real-world viruses. Due to the limited number of false positives created during the learning phase, MADAM has no effect on the device's usage.

[9] Machine learning is used to detect harmful code. This research combines a taxonomy based on statistically generated criteria for identifying the acquisition techniques of new harmful code into Mechanical Learning (ML) approaches. The taxonomy is then used to categorise research on this topic and to identify key open research concerns as a result of rising dangers. File representation and selection methods, classification algorithms, weighted ensembles, inequality, practical learning, and chronological assessment are all discussed in this article.

[10] On the Linux side of Android, the article discovered roughly 105 helpful usable and linkable (ELF) formats. These applications' calls are subjected to statistical analysis. The analysis results can be compared to the recently installed software to see whether there are any major differences. Furthermore, certain work calls signal a potentially risky action. As a result, we propose a simple decision tree for determining the application's suspicion level. Our findings pave the way for the first step in detecting fraudulent apps on Android smartphones.

[11] Dynamic Examination runs the application in a sandbox, which intervenes and records low-level system interactions for further analysis. Both sandbox and diagnostic algorithms can be placed on the cloud, allowing for easy access to and distribution of suspicious software through a mobile app store such as Google's Android Market. A Sandbox can also be used to improve the performance of older antivirus software on the Android operating system.

3. Existing Model

There are quite a few existing models available in the market, but from what we have analyzed is that most of them are subscription-based, and few of them are available for free; from those available models, we have tested 200 apps and websites that are available on play store and the internet and came to a conclusion that most of the available malware detection softwares are try to find the existence of malware in android OS and in-an app by extracting a string, which is a package name, this package names will be in the android manifest file of every app. Then those extracted strings are compared with the existing dangerous package names. Nevertheless, this method is proven inefficient by many researchers.

4. Proposed System

Given an existing model, this proposed model is an attempt to overcome the limitations of the existing model and provide better results with the help of ANN.

Table 1
Key critical features of the proposed model

S. No.	Features of the project
1.	Fast and Precise detection of Malware in an application
2.	Learns from the analyzed from app
3.	Security and protection for side loading apps by users
4.	Root level permission detection.
5.	Metadata visibility.

There are different ways for implementing malware detection in Android applications, but we choose static analysis. Static analysis is a way of extracting the details from the applications, including permissions, intent, uses-feature, API calls, and application details.

There are two main modules in this project, the front end, and the back end; the front end module is a visual representation with a feature to upload an APK, and the results will be displayed on the screen. The processing and extracting details of the application (statistical analysis) happens in the back-end module, where the model will analyze the uploaded APK.

From the uploaded APK, AndroGuard will extract MD5 Signature and other permissions with the help of AndroGuard. AndroGuard is a Python-based tool that is used to disassemble and decompile Android apps. It is conducive for the Static Analysis of an application.

Choosing informative, discriminating, and independent features is a critical stage in classification for improving detection performance. So, we have chosen the top 20 used permissions of 86798 malicious applications as a feature.

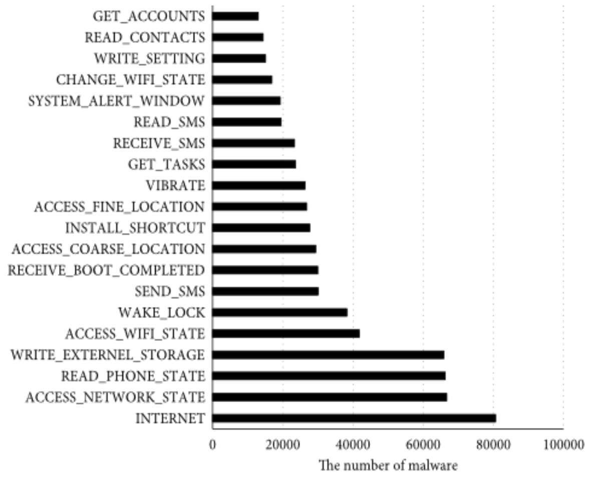


Fig. 1. Top 20 used permissions of 86798 malicious applications as a feature

The extracted features are fed to the artificial neural network, trained with a large set of malicious applications that will detect whether the app file contains malware or not by simply analyzing the extracted permissions and package name.

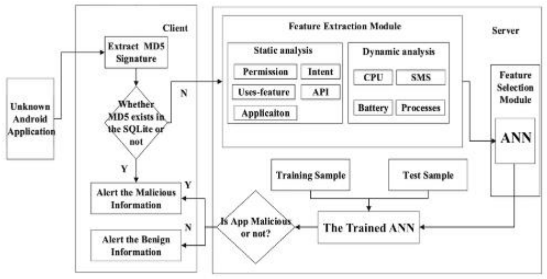
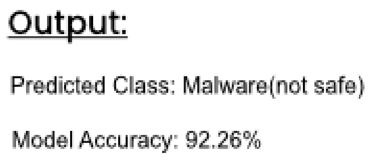


Fig. 2. Proposed system architecture

5. Implementation and Testing

With everything in hand and proper implementation of ANN, we created an HTML page that facilitates a user to upload an APK file and send a request for classification, and the above functionality can be achieved through the ANN and the web page which designed with the help of a templating language called Jinja2 beside flask framework.

After Uploading the APK file, the features are extracted and passed to the ANN classifier. The classifier responds with data relevant to the app along with the classification, which is demonstrated below. The output block of the HTML page displays relevant information about the variety, namely the Predicted class and Model accuracy.



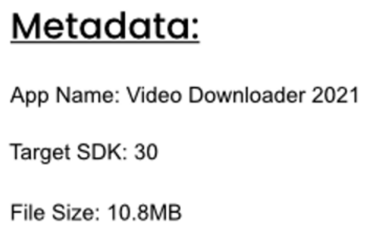
Output:

Predicted Class: Malware(not safe)

Model Accuracy: 92.26%

Fig. 4. Metadata of the input apk

The metadata block of the HTML page displays relevant information about the uploaded APK file, namely the App name, Target SDK Version, and File size.



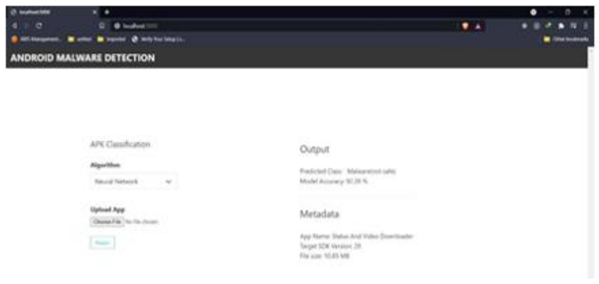
Metadata:

App Name: Video Downloader 2021

Target SDK: 30

File Size: 10.8MB

Fig. 5. Metadata of the input apk



ANDROID MALWARE DETECTION

APK Classification

Algorithm

Neural Network

Upload App

Choose File No file chosen

Output

Predicted Class : Malware(not safe)

Model Accuracy: 92.26 %

Metadata

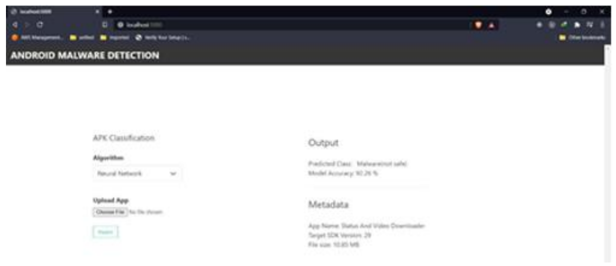
App Name: Status And Video Downloader

Target SDK Version: 29

File size: 10.83 MB

Fig. 6. HTML page for a benign classification

Using the Classify function, we are attempting to determine if the application downloaded is safe or not. The permissions and metadata of the files are accessed in the classifier function. Then the software predicts if it is safe or not to use with the help of an artificial neural networks model.



ANDROID MALWARE DETECTION

APK Classification

Algorithm

Neural Network

Upload App

Choose File No file chosen

Output

Predicted Class : Malware(not safe)

Model Accuracy: 92.26 %

Metadata

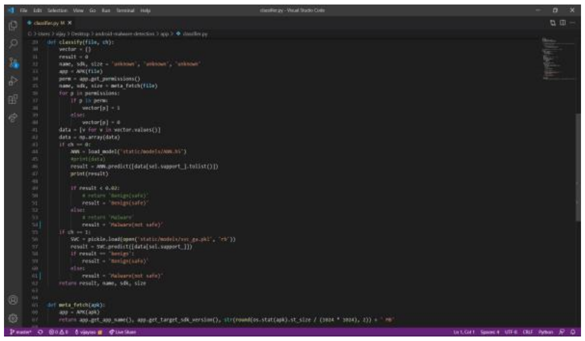
App Name: Status And Video Downloader

Target SDK Version: 29

File size: 10.83 MB

Fig. 7. HTML page for a classification

This ANN model was trained using Keras, an open-source software package including Dense, Dropout, Activation, and Embedding features. A specific activation function is given to the model using Dense, and a particular threshold value is specified using Dropout. Then the model is allowed to fit and predict the score.

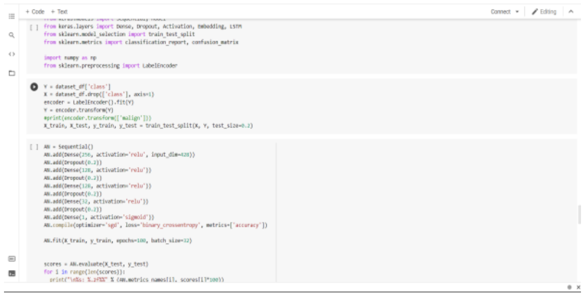


```python
def classify(file, ch):
    vector = {}
    result = 0
    name, sdk, size = 'unknown', 'unknown', 'unknown'
    app = APK(file)
    perm = app.get_permissions()
    name, sdk, size = meta_fetch(file)
    for p in permissions:
        if p in perm:
            vector[p] = 1
        else:
            vector[p] = 0
    data = [v for v in vector.values()]
    data = np.array(data)
    if ch == 0:
        ANN = load_model("static/models/ANN.h5")
        #print(data)
        result = ANN.predict([data[sel.support_].tolist()])
        print(result)

        if result < 0.02:
            # return 'Benign(safe)'
            result = 'Benign(safe)'
        else:
            # return 'Malware'
            result = 'Malware(not safe)'
    if ch == 1:
        SVC = pickle.load(open('static/models/svc_ga.pkl', 'rb'))
        result = SVC.predict([data[sel.support_]])
        if result == 'benign':
            result = 'Benign(safe)'
        else:
            result = 'Malware(not safe)'
    return result, name, sdk, size


def meta_fetch(apk):
    app = APK(apk)
    return app.get_app_name(), app.get_target_sdk_version(), str(round(os.stat(apk).st_size / (1024 * 1024), 2)) + ' MB'
```

Fig. 8. Flask code sample for classification



```python
from keras.layers import Dense, Dropout, Activation, Embedding, LSTM
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import numpy as np
from sklearn.preprocessing import LabelEncoder
```

```python
Y = dataset_df['class']
X = dataset_df.drop(['class'], axis=1)
encoder = LabelEncoder().fit(Y)
Y = encoder.transform(Y)
#print(encoder.transform(['malign']))
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

```python
AN = Sequential()
AN.add(Dense(256, activation='relu', input_dim=428))
AN.add(Dropout(0.2))
AN.add(Dense(128, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(128, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(32, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(1, activation='sigmoid'))
AN.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

AN.fit(X_train, y_train, epochs=100, batch_size=32)


scores = AN.evaluate(X_test, y_test)
for i in range(len(scores)):
  print("\n%s: %.2f%%" % (AN.metrics_names[i], scores[i]*100))
```

Fig. 9. Model for classification

6. Conclusion

In this paper, we build an Android malware detection system based on ANN, which, unlike existing detection methods, can detect unfamiliar Android applications using machine learning. We extract several characteristics using the static method. Our experimental results demonstrate that the novel method outperforms established detection approaches such as signature detection, with a greater detection rate and a lower error detection rate. Android malware and detection methods are both developing. As a result, we predict that comparable future studies will be required to address these growing dangers and detection technologies.

7. Future Scope

The proposed model is limited, and it is a web-based application with which a user can only upload a single app and check whether malware exists or not; this kind of behavior is called static analysis.

This work can be further extended by adding a dynamic analysis module with which a user can directly use the developed ANN model live on the device, and the model will have access to the metadata of every app present on the device, and users need not have to provide an input apk. The model will automatically determine whether an app is malicious or not by analyzing the behavior of every application and identifying which app consumes resources like battery, internet, and others from their native operating system, i.e., Android OS. Currently, the classifier works based on multilayer perceptron's, but according to a few research papers, genetic algorithms provide better results when compared to the current model.

Aside from malware, if a developer makes a mistake, it makes the lives of hackers simple to discover and exploit these vulnerabilities. As a result, ML-based approaches for detecting source code vulnerabilities may be built.

References

[1] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, Heng Ye, Significant Permission Identification for Machine-Learning-Based Android Malware Detection.

[2] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, Jianfeng Ma, A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms

[3] Mohammed K.Al Zaylai A, Suleiman Y.Yerima, SakirSezerc, DL-Droid: Deep learning-based android malware detection using real devices.

[4] Md. Shohel Rana, Sheikh Shah Mohammad Motiur Rahman, Andrew H. Sung, Evaluation of Tree-Based Machine Learning Classifiers for Android Malware Detection.

[5] Arvind Mahindru, A. L. Sangal, MLDroid—the framework for Android malware detection using machine learning techniques.

[6] Prerna Agrawal, Bhushan Trivedi, Machine Learning Classifiers for Android Malware Detection.

[7] Long Wena, Haiyang Yub, An Android malware detection system based on machine learning.

[8] Reverse engineering using Androguard, International Journal of Innovative Science and Research Technology.

[9] Dini, Gianluca, Martinelli, F. and Sgandurra, A Multi-Level Anomaly Detector for Android Malware.

[10] Shabtai, A., Moskovitch, R., Elovici, Y. and Glezer, Detection of malicious code by applying machine learning.

[11] Schmidt, A.D. Schmidt, H.G., Clausen, J., Yuksel, K.A., Kiraz, O., Camtepe, A. and Albayrak, Enhancing the security of Linux-based Android devices.

[12] Blasing, T., Schmidt, A.D., Batyuk, L., Camtepe, S. A. and Albayrak, An android application sandbox system for suspicious software detection.