Attacking the classified ”StackDroid”

1st Segev Tzabar
Ariel University
Hagor, Israel
Segev95@gmail.com

2nd Sali Sharfman
Ariel University
Ariel, Israel
Salishar2@gmail.com

***Abstract*—This article was written as part of the ”Attacks on Android operating systems” course, where we took a classifier that identifies malicious apps using machine learning, which is a method of teaching computers to learn from data, without being explicitly programmed. In the context of malware and benign classification, machine learning algorithms are trained on a dataset of known malware and benign software, and then used to classify new software that did not appear to be malware or benign. This is done by extracting features from the software, such as the presence of certain keywords or certain behavior patterns, and using these features to train a model that can accurately distinguish between malware and benign software. Applications of this technology include real-time detection and blocking of malware on individual computers, as well as detection and removal of malware from large networks. The classifier we took is called ”StackDroid”, our main goal was to find weak points in the classifier in order to produce a malicious application attack and the classifier will identify it as a benign application**

I. Introduction

on the Android operating system, malware classification is used to detect and protect against malicious apps that may be installed on a user's device. These apps can potentially steal personal information, send premium SMS messages, make unauthorized phone calls, or perform other malicious actions. To protect against this, many security companies have developed anti-malware apps for Android that can scan the device for malware and remove it. These apps use machine learning algorithms to classify apps as either benign or malicious, based on features such as the app's permissions, code structure, and behavior.

In our work we use ”StackDroid” classification and we present how to bypass it using Feature-Space attack. A feature-space attack on an Android operating system refers to a type of attack where an attacker modifies the features of a malicious app in order to evade detection by malware classifiers. This can be done by modifying the code, changing the app's permissions, or altering its behavior.

We extracting features from Android APK files. And create a dictionary from the extracted features. After we train all the data (on the features), we pretict our accuracy score on the features and then we change the features randomly and see that the score change and the predict changed too. This demonstrates how to change the results of the classifier by changing the features.

II. Related Work

Sheikh Shah Mohammad Motiur Rahman and Sanjit Kumar Saha [4] discussing a technique called StackDroid, which is a multi-level approach for detecting malware on Android devices. It uses a method called stacked generalization, which combines multiple machine learning models in order to improve the accuracy of malware detection. The authors have evaluated the effectiveness of StackDroid by conducting experiments on a dataset of Android apps, and have found that it is able to detect malware with a high degree of accuracy. The authors also suggest that with the increasing number of malware in android devices, this approach is a promising solution for detecting malware in android devices.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck [1] discussing Drebin, a system for detecting Android malware on mobile devices. The authors claim that Drebin is effective in detecting malware, and that it is able to provide explanations for why a certain app is classified as malicious. The system uses machine learning to analyze the behavior of apps and identify those that exhibit malicious behavior. The authors have evaluated the performance of Drebin on a dataset of Android apps and have found that it is able to detect malware with a high degree of accuracy. The authors also suggest that the ability to provide explanations for the detection decision makes Drebin a more transparent and trustworthy solution compared to traditional malware detection systems.

Harel Berger, Dr. Amit Dvir and Dr. Chen Hajaj [3] discussing problem-space evasion attacks in the Android operating system. It is a survey of the current state of these types of attacks, which are used to bypass security measures and gain unauthorized access to a device or system. The authors provide an overview of the different techniques used in these attacks, as well as their potential impact and how to detect and prevent them. The authors also suggest that Android OS as a main operating system of most smartphones and its popularity make it a prime target for attackers, and the research needed to secure android ecosystem.

Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer and Yael Weiss [2] discussing Andromaly, a behavioral malware detection framework for Android devices. The authors claim that Andromaly is effective in detecting malware by analyzing the behavior of apps on a device and identifying those that exhibit malicious behavior. Andromaly works by

collecting data on the behavior of apps on a device, including system calls and network communications, and using machine learning to build models of that behavior. These models are then used to identify apps that deviate from normal behavior and are likely to be malicious. The authors have evaluated the performance of Andromaly on a dataset of Android apps and have found that it is able to detect malware with a high degree of accuracy

III. METHODOLOGY

A. Data Extraction

At first we extract information from Android apps. The size of the dataset is 14969. 5208 malwares and 9761 non-malwares. First uses a tool called apktool to extract the AndroidManifest.xml and DEXParser file from the APK file. Then parses it, and extracts various types of information such as requested permissions, activities, services, content providers, broadcast receivers, hardware components, and intent filters from the Manifest file. Second we extract from the DEXParser file the URLs. Finlay we stores the information in sets and save it as a file displayed on "Fig. 1", The classifier can read this file only.

```
permission::com.google.android.c2dm.permission.RECEIVE
permission::android.permission.INTERNET
permission::com.wTdtandroid.permission.C2D_MESSAGE
permission::android.webkit.permission.PLUGIN
permission::android.permission.ACCESS_NETWORK_STATE
service::com.wTdtandroid.Server.C2DMClientReceiver
activity::MessageViewer
activity::MainNavigationActivity
intent::com.google.android.c2dm.intent.RECEIVE
intent::android.intent.action.MAIN
intent::com.google.android.c2dm.intent.REGISTRATION
receiver::om.google.android.c2dm.C2DMBroadcastReceiver
```

Fig. 1. Example of the APK file extracted

B. Data Classification

After we have the file extract, We can classify the apps as malware or non-malware. We then reads a CSV file that contains the labels for each file and creates a dictionary that maps file names to their labels. Using this dictionary, We separates the file names in the dataset folder into the two lists according to their labels.

C. Machine Learning Model Training and Prediction

Then we uses the different machine learning classifiers from the sklearn library to train a model on the feature vectors of the Android apps and their corresponding labels (malware or non-malware). Then use this trained model to predict the labels for new unseen apps based on their feature vectors.

D. Model Evaluation

At this part the attack starts, We generates random numbers, and change the order of the test labels with malware and non-malware examples divided and a vector with class of each example. Then we can see the final prediction score, which is the accuracy of the model and the prediction score after features change, which is the accuracy of the model after randomizing the order of the test labels.

IV. RESULTS

The final results of the script show that the model has a high accuracy of 0.96593186 when predicting the labels of new unseen apps based on their feature vectors. This means that the model is able to accurately classify the apps as malware or non-malware with a 96.6

The accuracy of the model dropped significantly from 0.96593186 to 0.48563794 "Fig. 2".

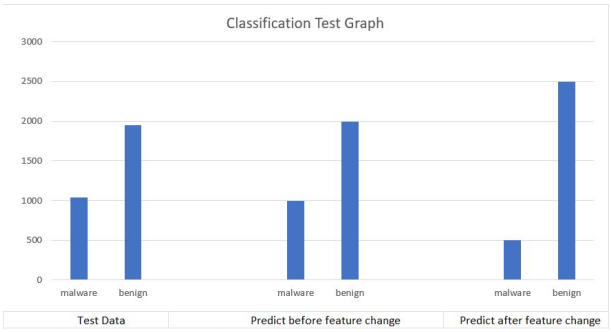

Fig. 2. Classification test graph

This suggests that the feature attack was successful in disrupting the performance of the model. The feature attack randomizes the order of the test labels, making it harder for the model to accurately predict the labels of the apps. This highlights the importance of developing defenses against feature attacks in order to improve the robustness and security of Android malware detection systems.

The solution we suggest is to use different feature extraction techniques to extract more meaningful and relevant features from the apps. This could help the model learn more general characteristics of the apps and improve its ability to classify new unseen apps accurately. Overall, it is important to keep in mind that the results of the model are highly dependent on the quality and diversity of the data used for training and evaluation, and that the use of various techniques and strategies can improve the performance of the model.

V. CONCLUSION

In this work we discussed the use of machine learning in the context of identifying and classifying malware on Android operating systems. The specific classifier used was "StackDroid," which employed a method called stacked generalization to combine multiple machine learning models to improve the accuracy of malware detection. The main focus was to present a technique called Feature-Space attack, which is a type of

attack where an attacker modifies the features of a malicious app in order to evade detection by malware classifiers. We use the classification to train a machine learning model on a dataset of known malware and benign apps and used it to predict the labels of new unseen apps based on their feature vectors.

The results of the experiment showed that the model had a high accuracy of 0.96593186 when predicting the labels of new unseen apps. However, when the feature attack was implemented, the accuracy of the model dropped significantly to 0.48563794, indicating that the feature attack was successful in disrupting the performance of the model. This highlights the importance of developing defenses against feature attacks in order to improve the robustness and security of Android malware detection systems.

We suggest that to improve the robustness and security of Android malware detection systems, different feature extraction techniques should be used to extract more meaningful and relevant features from the apps. This could help the model learn more general characteristics of the apps and improve its ability to classify new unseen apps accurately.

In future work, it would be interesting to test the proposed feature-space attack on different types of malware classifiers and evaluate their robustness against this type of attack. Additionally, it would be worth exploring other methods of evading detection and developing more robust defenses to protect against these types of attacks.

REFERENCES

[1] Malte Hübner Hugo Gascon Daniel Arp, Michael Spreitzenbarth and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 30:5–14, 2014.

[2] Panagiotis Andriotis Emiliano De Cristofaro Gordon Ross Enrico Mariconti, Lucky Onwuzurike and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018, 2018.

[3] Amit Dvir Harel Berger and Chen Hajaj. Problem-space evasion attacks in the android os: a survey. *Ariel Cyber Innovation Center*, 2020.

[4] Sheikh Shah Mohammad Motiur Rahman and Sanjit Kumar Saha. Stackdroid: Evaluation of a multi-level approach for detecting the malware on android using stacked generalization. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 8(2):68–75, 2017.