Final Project Report

CPS 474/574 Software/Language-based Security

Fall 2022

Instructor: Dr. Phu Phung

Team Project: Android Malware Detection

Team Members:

Niharika Gadhave: gadhaven1@udayton.edu

Team management: https://trello.com/b/91ziU5MV/finalproject

Summary

For financial gain, data theft, and personal gain, mobile malware attacks Android devices (smartphone, smartwatch, and smart television). Malware for mobile devices can take the form of Trojans, worms, botnets, and root exploits. Most security professionals use both dynamic and static analysis to find malware.

We used static Analysis to determine the malware

Static analysis is a type of analysis that looks at the code of a malicious application and examines all of the activities within it over an extended period of time without actually executing it. The reverse engineering technique, which gets the entire code and further examines the application's structure and content, is the primary step of the static analysis procedure. As a result, this analysis may look at the code as a whole while using little CPU processing power and memory. The absence of the application

also makes the analysis process quick. This research also identifies unidentified malware with improved detection accuracy using machine learning techniques.

A permission-induced risk interface called MalApp Classification to spot privacy violations resulting from granting permissions during app installation is suggested in order to protect users' privacy.

It consists of a multi-step method that analyzes static data about apps according to their categorization.

- In order to create a Boolean-valued permission matrix, the notion of reverse engineering is first used to extract app permissions.
- To find the dangerous permissions across categories, ranking of permissions is done in the second step.
- Third, ensembling and machine learning approaches have been used to test the effectiveness of the suggested strategy on a set of data. Java and Python were successfully used to implement the interface, making it suitable with all Operating Systems.

##Inrtoduction

It is nothing much surprising that the mobile app industry is flourishing and growing immediately after 2008 when the first app was launched. In today’s era, everyone’s life revolves around smartphones, which has raised a serious concern for security and privacy. One of the biggest issues being faced by this trending technology is the protection of smartphone devices against various security threats and prevention of user’s privacy leaks. One of the most exciting features of smartphones is that their functionalities can be expanded by installing third-party applications called ‘apps’. Android is one of the most popularly used platforms for smartphones with billions of apps available on its official ‘Play Store’. As reported by buildfire statistics, there were 2.7 billion smartphone users in the world in 2019 and on an average, a smartphone user uses about 30 apps per month . Apps for the Android OS are mainly written in the JAVA and KOTLIN language. With the increase in the development of Android apps since 2008 when the first android app was launched, there are over 2.9 million apps available for download on the Google Play Store globally. However, this has also resulted in an increase of privacy breach of users. App usage rate is increasing at a steady rate with no signs of decline in the future. Forbes stated that the testing specialist researchers.

In conclusion, while smartphones give consumers a free platform to download third-party programs from play stores and enjoy themselves, they also pose a significant unknown risk to the security of the users' personal data. Users download the apps so easily without considering the harm they could do to their lives. According to studies, users often give apps permission to run inadvertently because they are in a rush and are unable to consider the potential danger. So that users may make educated judgments, it is the developers' responsibility to only ask for permissions that are necessary and to be clear.

Background of project

- In this work, a JAVA-developed interface that leverages Python as a backend to categorize unidentified harmful programs is shown. The interface analyzes the.apk file of an unidentified Android app and its category to determine whether it is safe to use before the app has ever been launched. • Employs a reverse engineering strategy using the "apktool" to extract app permissions from the AndroidManifest.xml file and create a Boolean-valued permission matrix (Pmxn), where m represents the number of apps under examination from different categories and n represents the number of permissions. For a total of 324 Android permissions, we examined around 60 apps from each category. systematically ranks each authorization from each category according to risk-relevance by utilizing the correlation coefficient. The top 20 dangerous permissions in each category were selected from 234 total permissions. • Applying a data heuristic technique, which gives permissions weights based on category and risk relevance by giving a greater heuristic value to a more risky permission. By doing this, the model performs better.
- uses machine learning and data ensembling methods to examine the actions of 409 dangerous applications and 404 benign apps. There have been used a number of machine learning classifiers, including naive bayes, SVM, decision tree (J48), random forest, and ensembling techniques including bagging, boosting, and voting. Several performance indicators, including True Positive Rate (TPR), False Positive Rate (FPR), Accuracy, and Area Under the Receiver Operating Characteristic (AUROC) curve, have been used to assess the suggested malware detection interface.
- Applying a thorough examination of performance indicators, the optimal classifier and permission matrix are selected for each category. • The suggested interface can test a fresh, untested software against a trained model and forecast whether the result will be benign or malicious.

Dataset

Numerous studies have been completed in the area of android app malware analysis, however very little analysis of apps based on their category has been done. Our study needed a sizable data set made up of the permission vectors of each app and a note of their respective categories in order to perform a comprehensive examination of the dangerous behavior of explicit and implicit Android permissions. The data set was manually created by downloading 10 apks for 14 Android categories, including art & design, augmented reality, auto & vehicles, beauty, books, comics, communication, house & home, lifestyle, libraries & demo, maps & navigation, personalization, videoplayers & editors, and weather, from unofficial third-party malware stores. Reverse engineering was done on a total of 10 benign and 10 malicious Android apks.

Permissions which can include both implicit and explicit permissions offered by the Android system—have been taken into consideration in this study. The permissions must be specifically requested by the app developer during the development phase by putting them in the AndroidManifest.xml file. The Manifest file in the app's folder contains a list of all the permissions. Static analysis on Android apps has been carried out in this work with the goal of analyzing the behavior of apps before they run by examining their code. All apks have been reverse engineered using apktool. Later, the AndroidManifest.xml file was read using DOM XML parser to retrieve all the rights contained inside.

Project Description

There are an abundance of white papers, journals, research pieces, and papers in this project work that explore the dangers and weaknesses connected with the use of Android permissions. This section gives a summary of a few studies in this field that have been linked in order to do behavioral analysis of Android apps.

1. Reverse Engineering The code, resources, signature, manifest, and various other files that must be executed in order to get the entire Android app up and running are all contained in an APK (Android Package), which is a comprehensive file that is most likely a ZIP file. The permissions requested by the application are listed in AndroidManifest.xml. The major aim was to obtain this file because it was known that security and Android permissions were closely tied. Their findings showed that the majority of malware apps insert malicious code and unauthorized access into the AndroidManifest.xml file. Dex2jar, Apktool, and Soot are three of the most well-liked Android app reversing tools that change the executable source into Jasmin, Smali, and Jimple ILs, respectively. Lipo Wang has modified these programs. Static analysis reverse engineering has been applied in this paper. Below is a description of some of the well-known static analysis research.

2. Static Malware Analysis To improve the effectiveness of detection, it examined malicious behaviors and permissions. On the basis of certain similarity scoring, the system may also categorize malicious programs. The project's findings show that the provided method performed with 88% and 90% accuracy for detection and classification, respectively.

3. Interface Java and Python, the two most widely used platform-independent programming languages among developers, were used to successfully build the interface, making it interoperable with all Operating Systems. An open source operating system is Android. The vast majority of Android applications created to date use Java. Therefore, using platform independent technologies makes it simpler to analyze harmful Android apps and afterwards employ static analysis to protect user privacy. Additionally, Java and Python are faster than other languages due to their heavily typed syntax and verbose coding style. Here, Python is used to develop the prediction model, while Java is utilized to design the user interface and extract the authorization vector for the new app.

Results

We studied on malware detection for creating the app where we can detect whether the .apk file has malware or not. We have categories the file into 14 different types. When we upload a file and select in what category it belongs to, while clicking on excute the result will display saying the malware is detected or not and is apk safe to install or not.

We can say that we were able to get output with 88-90% acurracy.

Contribution

Niharika Gadhave: My contribution towards this project is I have worked on doing reverse engineering of andoid app (apktool) and categorizing file types.

Hitesh Sakhare: My contribution towards this project is I have worked on static analysis and UI using Java. Also used above mentioned dataset to get more accuracy for the project.

Prototype

Bitbucket link https://bitbucket.org/lbs_gadhaven1/ss-lbs-team10/commits/

Result

https://bitbucket.org/lbs_gadhaven1/ss-lbs-team10/raw/c9a4f592e1ca6b14efc34f6077cca88db1f7661c/Result.png