A Static Analysis of Android Application Overpriviledged Permission Misuses, SSL API Misuses, and Interface Vulnerabilities

5/1/2018

Abstract

Android devices make up a huge percentage of the mobile market. Users are constantly downloading Android applications onto their devices. How can these applications be verified to be safe? In this article 2 datasets of android applications, 1 random and 1 IoT, are analyzed for security vulnerabilities. This will help to determine better methods of detecting true issues in applications so they can fixed or removed from public access. Static analysis shows that most applications are vulnerable. Those that weren't found to be vulnerable could very likely have easily leveraged security flaws that weren't searched for. There is still many inaccuracies surrounding automated analysis that prevent flaws from being uncovered.

1 Introduction

Android has grown over the years to be the largest mobile platform available. Part of its success can be attributed to the amount and wide variety of android applications available for download. These applications give users the features they want on their phones. How has the Android application experience become so well supported? One reason is that there is a very low barrier of entry to developing and publishing android applications. Google has made a lot of tools and tutorials for Android application development easily accessible. Android is both free and open source. It is also quite cheap, around $25, to publish an application on the Google's Play Store. It has become a very popular endeavor for people and companies to develop Android applications. An application could be developed by a single novice programmer or an entire development team working for a company. Due to its size and support, the Android application platform is well worth researching.

Android applications are downloaded by millions of users and have widespread use throughout the world. However they are not watched carefully for safety. The Play Store relies on user feedback to find issues with applications [8]. Security is an incredibly important issue to address for Android applications. The fact that anyone can easily put an application on the Play Store is dangerous to security in several ways. Inexperienced or lazy application developers won't be deterred from publishing a poorly developed application. Even though they do not intend to harm their users, security is hard implement. This also isn't limited to one man teams. Even big name applications with tons of people working on them have security flaws. Researchers have found many common malware programs that successfully attack many mobile banking apps. For example, Zitmo steals TAN codes sent by banks to customers through text message. Wrob replaces legitimate banking applications with similar looking trojans [1]. There is also the case of attackers purposely trying to harm users. Publishing a malware-infested application is quite easy. It has been found that although Google has strengthened its Play Store Security with their automated system, malware can still slip through [4].

It may seem like the problem is the developers. However, there is not much of a way of forcing them to implement proper security features. It can be encouraged, but at the end of the day this isn't a foolproof method. Attackers will continue to try to get malware through. Android could adopt the system that Apple uses for their App Store. Apple requires developers to have a subscription to their program. Their approval process is also much more strict, often taking several days as opposed to a couple hours for Android [6]. However this goes against Google's principle of having easy access. This refocuses the problem from creating secure applications to instead properly detecting those that are not secure. This is a big challenge due to the complexities of Android application analysis. Android transforms Java bytecode into DEX bytecode to run on the Android Dalvik VM [7]. It can be difficult to understand the actions of an Android application through this layer of abstraction. Better methods are needed to identify possible security risks in Android applications to protect users. There are many tools currently available to assist with Android application analysis. However many of these are built off the same framework [7]. The developers of these tools expect users to extend the functionality to be able to get meaningful data. Methods like taint analysis haven't been fully fleshed out. Static analysis that takes advantage of these tools in a methodical manner could really make breakthroughs in Android application security Anaysis.

2 Overview

This research experiment focuses on properly identifying potential security issues in android applications. There were three main research questions focused on relating to this problem:

RQ1: Do applications often ask for too many permissions?
RQ2: Are applications properly implementing SSL APIs?
RQ3: Are applications protecting their broadcast receivers and services from dangerous inter-application communication?

To answer these questions, data was gathered through static analysis. This means the code for the application sets were not run. There were four methods for gathering data: bash scripting, Androguard, Play Store API, and manual analysis. These produced data on what sort of security flaws each application had and the possible effects.

The dataset available consisted of two sets of application apks, a random set and a IoT set. It was predicted that the applications tested will ask for too many permissions. This can be dangerous because it could create potentially dangerous permission combinations. If an attacker gained control over an over-privileged application they would have a lot of options open up. For example, if an application had both the record audio and internet permissions they could eavesdrop on a user. It was predicted that around 20% of the android applications will exhibit SSL API misuse. It was predicted that 15% of the applications will have some sort of vulnerable interface misuse. It is also likely that the IoT application set will have more security vulnerabilities than the random application set because of its increased reliance on internet-related actions.

3 Design

The data analysis process is visualized with the flowchart in Figure 1. The first step was to convert the given apks for both sets into smali code. Smali code is necessary to generate for analysis because the .dex files android uses aren't in a very understandable format. The smali code

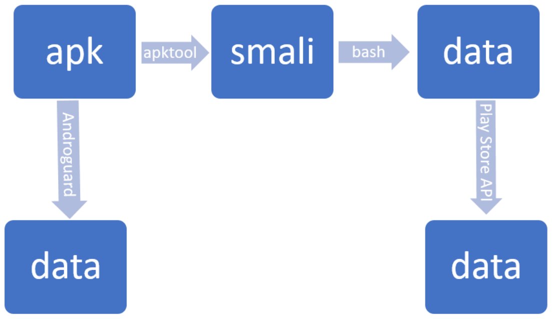

Figure 1: A flow chart of how the data was gathered

can be analyzed with a bash script and appear more similar to the source code.

The bash script was used to count instances of security vulnerabilities for the various applications. It traversed the smali code of each application looked for certain keywords or patterns that suggest security vulnerabilities are present. The first test it did dealt with over-privilege. It checked for GPS permissions in the manifest and if found, searched for class to GPS methods anywhere in the source code. The official library uses a Location class and methods that include the word "location" frequently in them. It can use the network provider or the device's internal GPS service. String patterns from this class was searched.

The next bash script method looked for dangerous permission combinations defined as such:

- Eavesdropping: RECORD_AUDIO and INTERNET
- Tracking: ACCESS_FINE_LOCATION and RECEIVE_BOOT_COMPLETE
- SMS Spam: SEND_SMS and WRITE_SMS

Next, SSL and interface vulnerabilities were searched for. This included recurring common methods for bypassing normal SSL procedures developers use. Options like allowing all hostnames, trust managers accepting all certificates, SSL Factory implementation accepting all certificates, using http instead of https for sensitive data, and sslErrorHandler bypassing were considered.

The last part of the script gathered data on services and broadcast receivers. Applications were checked for whether they were exporting receivers and calling services with implicit intents.

Androguard was used to compare target sdk averages for the random application set and the IoT set however the program kept crashing.

Certain applications that had more security vulnerabilities based on the bash script were manually analyzed. Using the package name obtained from the smali code, the official Play Store application could be looked up for many of the applications in the dataset. Previously unavailable data about the applications was available through a lookup on a Play Store web API (apptweak.io). This included screenshots, categories, ratings, and the application descriptions.

4 Evaluation

The results of the bash script analysis can be seen in Figure 2. For every category, the IoT dataset had equal or more applications with vulnerabilities. This agrees along the line of thinking that increased internet use causes more vulnerabilities. However it does not indicate that there is a higher rate. IoT applications could more secure compared to the average internet using application since the nature of the applications require more usage of data usage and transfer. If a random application had to implement these interfaces, they may take shortcuts if the application barely utilizes it.

One interesting difference between the IoT set and the random set is that although the IoT set asked for GPS Permissions much more often, both had the same amount of applications that didn't appear to use any GPS-related methods even though it was asked for. This seems counterintuitive since it would make sense that IoT applications call too many of these privileges. A developer for an IoT application may mindlessly call it assuming location is a necessity and not use it later. In the random application set, out of the 7 applications that asked for GPS permissions (ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION), 2 of them may not have been using the permission. Were these over-privileged or using some sort of external API? External APIs that access location information are also likely to have ”location” somewhere in their calls. It would be more accurate if the control flow was analyzed and actual uses of the GPS interface could be detected.

IoT applications also had a much higher rate of exporting their receivers as false. This also seems counterintuitive since inter-communication is highly emphasized in IoT.

Security vulnerabilities in general were high for both groups. This seems consistent with other research that found high rates and highlights the importance of being able to find and fix these vulnerabilities [2]. Only a couple vulnerabilities were scanned for and results showed up. HTTP misuse had the biggest misuse. However this problem may grow smaller as companies begin to enforce stricter https use requirements.

The Prize Claws application was the only random application with a trust manager vulnerability. It also had the possibility of eavesdropping and Http use over Https. Analyzing the Play Store information reveals it is a very popular application. It currently has over 400,000 reviews with close to a four star rating. It is free and offers in app-purchases. The application is an arcade style game that simulates claw machines found in arcades. Such a popular application having these vulnerabilities shows how any application can have them. Popular applications may request a large amount of permissions for collecting data. This application requests the GPS permission although it doesn't seem to need it to function. It does have ads which could be what the GPS data is used for.

5 Discussion

One reason that searching for over privileged applications to detect malware has a high false positive rate is because developers mindlessly ask for permissions to get their application working. As a result, almost all the alarms will not be true malware. However even if these applications aren't created to attack users, their vulnerabilities provide gateways for others to do so. Even one vulnerability can lead to information leaks.

Permission misuse is especially dangerous and hard to control. The Android Operating System actually has an auto-update application feature. The application could completely change how it operates, but it will still have the permissions granted earlier by the user. Simply catching permission misuse through application analysis may not be enough. Users should be aware of what an application can do.

Many of the issues surrounding the accuracy of the experiment could be addressed with dynamic analysis. Keywords and function calls can only evaluate so much of the application. In order to be certain permissions are utilized, the application needs to be run. Permissions are hard to track and dynamic analysis could assess these applications at runtime [3].

It would have been optimal to integrate Play Store data testing into the whole dataset. Many APIs are available that provide this information such as 42matters and AppTweak. These were tested and worked very well. However due to the financial constraints of this project, only the free version was tested and the amount of API calls allowed was too limited. Screen scraping is possible on the Google Play Store as an alternative. Applications can be found through The Google Play Store allows applications pages to be found through their application ID using the url: https://play.google.com/store/apps/details?id=. This ID can be found in the Android Manifest and through Androguard.

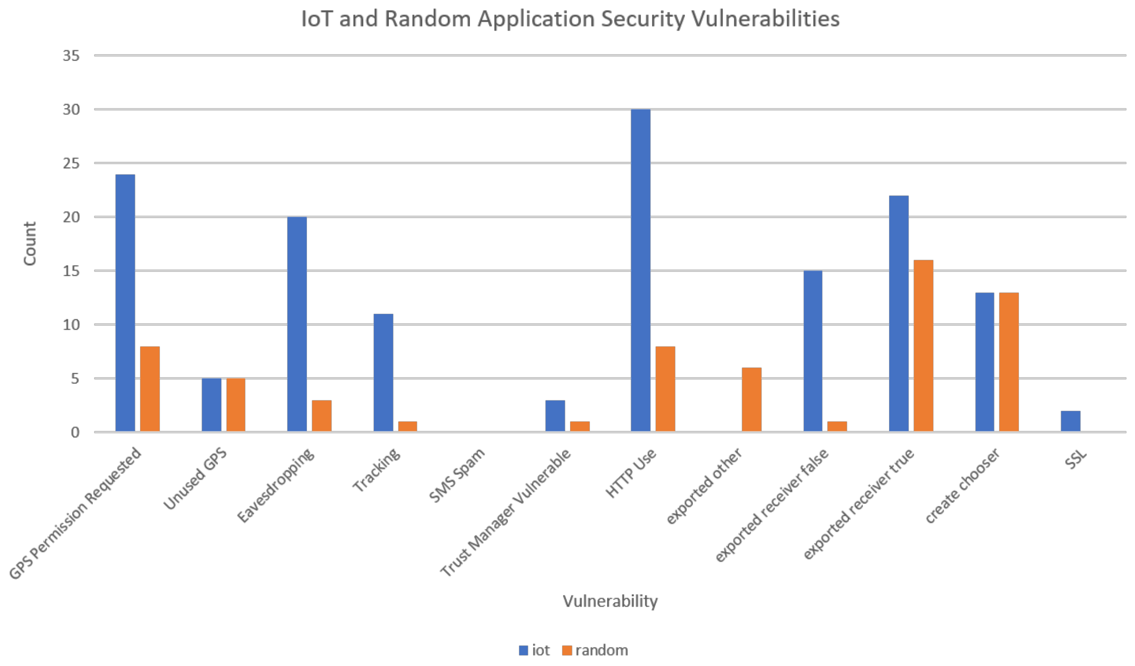

Figure 2: A flow chart of how the data was gathered

6 Related Work

One recent technique that could really boost Android application vulnerability analysis is machine learning. These prediction models were found to have a possible 3%-58% improvement over current models. Natural Language Processing can be used to detect certain features in the code that helps with vulnerability prediction [5]. If models are able to train off of previous vulnerabilities found, a very robust system could be made. One problem with dealing with cyber attacks is that they are constantly evolving. New malware can go undetected for a while. If the automated system evolves along with the attacks, more instantaneous responses could be made.

7 Conclusion

Static analysis for Android application vulnerabilities is growing better and better. Automating this process is becoming an extremely important topic as the Android platform grows bigger. As the Android platform is used to connect the world, the transfer of data through SSL must be monitored closely. Private user data is becoming less and less private. Ensuring that users have their permissions managed correctly could help combat the prevalent data selling going on.

References

[1] Issues and security measures of mobile banking apps. *International Journal of Scientific and Research Publications*, 6(1):36 – 41, 2017.

[2] A survey on https implementation by android apps: Issues and countermeasures. *Applied Computing and Informatics*, 13(2):101 – 117, 2017.

[3] A. Armando, G. Bocci, G. Chiarelli, G. Costa1, G. D. Maglie, R. Mammoliti, and A. Merlo1. Mobile app security analysis with the maveric static analysis module. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 5(4):103 – 119, 2016.

[4] E. Bridges, D. Kishore, and J. Pedo. Android app security analysis. 2012.

[5] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose. Automatic feature learning for vulnerability prediction. 2017.

[6] S. Liang and X. Du. Permission-combination-based scheme for android mobile malware detection, 06 2014.

[7] B. Reaves, J. Bowers, S. A. G. III, O. Anise, R. Bobhate, R. Cho, H. Das, S. Hussain, H. Karachiwala, N. Scaife, B. Wright, K. Butler, W. Enck, and

P. Traynor. droid: Assessment and evaluation of android application analysis tools. *ACM Computing Surveys*, 49(3):1 – 30, 2016.

[8] M. K. SU. Techniques for identifying mobile platform vulnerabilities and detecting policy-violating applications. In *Dissertations and Theses Collection*. 10 2016.