Identifying Android Banking Malware through Measurement of User Interface Complexity

Sean McElroy[a]

[a]*Dakota State University, United States*

Abstract

This is a simple paragraph at the beginning of the document. A brief introduction about the main subject.

Keywords: Android security, malware detection, resource file, static analysis

1. Introduction

Research activity for designing new methods to detect Android malware has been sustained for over a decade, and a variety of novel strategies are available and effective.[7] Static analysis, which uses techniques to assess an Android application package, or APK, are beneficial to provide an assessment as to whether an APK is malicious before it is executed. Static analysis techniques can be implemented not only by mobile operating systems and antimalware solutions that execute on a user's mobile device, but they can also be employed by software repositories, such as Google Play, to scan for malicious packages before publishing and distributing them to end users. However, static analysis is susceptable to adversarial techniques such as obfuscation, which intentionally renders application code less legible and analyzable and reduces the efficacy of malware detection techniques.

While static analysis of an Android executable binary is a central and valuable strategy for classifying APKs as malicious, additional features are needed to improve the true and false positive detection rates. One such classifier, DroidSieve, addresses this need by incorporating additional features that are not susceptable to binary obfuscation techniques.[11] Such additional features include non-executable metadata elements, such as lists of mobile operating system permissions the application can request and malicious payloads disguised in or referenced by APK resource blobs. However, I propose Android malware is also detectable by an additional feature unconsidered by other classifiers – the complexity of user interface design in the Android application package.

Email address: `me@seanmcelroy.com` (Sean McElroy)

2. Related Work

2.1. Static Analysis

The static analysis of APK files has a relatively long research history, often using methods focused on the bytecode of an APK that would be executed on a target device. Such methods include reachability, data flow, call graph, and entry point analysis.[8] Matching static analysis findings against signatures of known malicious APKs or components common to malicious APKs of malware families has been supplimented by machine learning algorithms. While accuracy of some machine learning approaches to static analysis yield lower F-scores compared to more expensive and fragile signature and hand-coded heuristic approaches[6], they may more reliably detect new malware families.

Notably, Liu et al. [4] observed static analysis in machine learning methods yield improved detection rates when the characteristics of certain Android resources were incorporated into feature sets. Android resources are non-executable components in APK files which can provide or configure user interface elements, including image and sound assets and UI component layouts. Liu et al. [4] observed the number of resource images for each pixel density varied signficiantly between benign and malicious APKs. This UI feature alone is not significant as a malware classifier, but traditional static analysis features like intents and permissions observed in APK manifest files are becoming less sufficient and more significant static features are needed to improve detection models.[3]

2.2. Dynamic Analysis

Dynamic analysis of executing APKs yields unique features specific to the operating environment context, an area of research dating back nearly as long far as static analysis techniques.[9] Extracted dynamic analysis features can include Java and native method call logs and network packet captures obtained from running an APK in an emulated sandbox environment.[10] Rigid static analysis that is dependent on file hashes, signature and string matches, and specific call chains can be brittle to simple obfuscation techniques. In contrast, unlike static call chain analysis, dynamic analysis features may change with each execution, such as if they are affected by temporal, network, or other system state information. Many modern Android malware families employ evasion techniques to hinder their classification and observation of their runtime behavior to thwart dynamic analysis.[2]

However, static analysis of Android user interface control and data flows can vary widely from runtime observations in some contexts, leading Wang et al. [13] to suggest that this type of analysis of GUI components, in particular, may be unsound. Where static analysis methods are unsound, dynamic analysis could provide a more accurate classification of Android applications as potential malware. While dynamic analysis of UI features is well-researched in the area of automated UI testing, security research is somewhat limited to offensive use cases, such as activity hijacking and clickjacking attacks, and detection of the same.[1]

3. User Interface Resources

Android application package APK files contain executable code, bundled resources, and a manifest file which provides a mpaping to code and resources as well as other metadata, including configuration settings for how the APK will be presented and operate when invoked on an Android operating system. Bundled resources may include various non-UI assets, including certificates. Substantially though, bundled resources include images, themes and color values, localizable strings, and layouts. Layouts are XML files that contain symatic markup for how user interface elements are organized on a screen or menu.

The Android ecosystem is comproised of many different equipment manufacutures, each manufacturing an array of mobile devices of varying hardware specifications. Designing an Android application to operate well on myriad device screen sizes and display technology types often requires many resolution-specific versions of layout files to be designed and included in the bundle of resources within an APK. Moreover, complex user interfaces often require many elements in the XML layout file to nest UI components carefully to build up full-featured user experiences.

4. Methods for Measuring UI Complexity

Three methods are proposed to measure user interface complexity for the purposes of classifying APK files as malicious or not.

4.1. Density Versions of Bitmap Image Assets

Android applications may include scalable vector image formats which maintain clear and sharp characteristics on any screen size and pixel density as well as bitmaps, which vary in quality across different device models. When bitmaps are used, Android device compatability documentation provides guidance to generate multiple bitmaps for each type of density qualifier.These qualifiers range from ldpi for low-density screens at 120dpi to xxxhdpi for extra extra extra high-density screens which operate at 640dpi. Applications which do not conform to this guidance may scale bitmaps to match the device's density requirement at the cost of image quality. While hobbiest or low-quality apps may not take the time to follow this guidance, the lack of multiple density versions of bitmap image assets may be more likely to be observed in malware samples than legitimate applications, since malware designers ostensibly do not have the graphic design capacity to follow this guidance and may not have sufficient economic incentives to spend the effort to do so.

4.2. Layout XML Structure Metrics

Number of files (total, and menu vs layout) Number of elements per files Average number of attributes per element

Only applies to declarative UI's...

Webview-centric apps could muddle this

May not be a true positive indicator, may be a false positive reducer

May be susceptable to clone/recompile throwing this metric off.

4.3. Number of static values

May be susceptable to clone/recompile throwing this metric off.

5. Data Analysis

To test the proposed methodlogy, CICMalDroid 2020[5], a large and recent Android Malware labeled data set was processed and analyzed. Two sections of this data set were used, those APKs labeled Benign, which are not known by the data set maintainers to contain malware, and those labeled Banking, which are known to contain financial services-related malware.

Processing of each APK file in the dataset was performed first using apktool[12] to decompress APK packages into their constitutent resources.

References

[1] Bkakria, A., Graa, M., Cuppens-Boulahia, N., Cuppens, F., Lanet, J.L., 2017. Real-time detection and reaction to activity hijacking attacks in android smartphones (short paper), in: Proceedings of the 15th Annual Conference on Privacy, Security and Trust (PST), pp. 253–2535. doi:10.1109/PST.2017.00037.

[2] Bulazel, A., Yener, B., 2017. A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web, in: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, pp. 1–21. doi:10.1145/3150376.3150378.

[3] Kouliaridis, V., Kambourakis, G., Peng, T., 2020. Feature importance in android malware detection, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1449–1454. doi:10.1109/TrustCom50675.2020.00195.

[4] Liu, X., Dong, X., Lei, Q., 2018. Android malware detection based on multi-features, in: Proceedings of the 8th International Conference on Communication and Network Security, Association for Computing Machinery, New York, NY, USA. p. 69–73. doi:10.1145/3290480.3290493.

[5] Mahdavifar, S., Abdul Kadir, A.F., Fatemi, R., Alhadidi, D., Ghorbani, A.A., 2020. Dynamic android malware category classification using semi-supervised deep learning, in: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), pp. 515–522. doi:10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

[6] McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., Joon Ahn, G., 2017. Deep android malware detection, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Association for Computing Machinery, New York, NY, USA. p. 301–308. doi:10.1145/3029806.3029823.

[7] Razgallah, A., Khoury, R., Hallé, S., Khanmohammadi, K., 2021. A survey of malware detection in android apps: Recommendations and perspectives for future research. Computer Science Review 39, 100358. URL: `https://www.sciencedirect.com/science/article/pii/S1574013720304585`, doi:https://doi.org/10.1016/j.cosrev.2020.100358.

[8] Schmeelk, S., Yang, J., Aho, A., 2015. Android malware static analysis techniques, in: Proceedings of the 10th Annual Cyber and Information Security Research Conference, Association for Computing Machinery, New York, NY, USA. pp. 1–8. doi:10.1145/2746266.2746271.

[9] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y., 2012. "andromaly": A behavioral malware detection framework for android devices. J. Intell. Inf. Syst. 38, 161–190. doi:10.1007/s10844-010-0148-x.

[10] Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J., 2013. Mobile-sandbox: Having a deeper look into android applications, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, Association for Computing Machinery, New York, NY, USA. p. 1808–1815. doi:10.1145/2480362.2480701.

[11] Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L., 2017. Droidsieve: Fast and accurate classification of obfuscated android malware, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Association for Computing Machinery, New York, NY, USA. p. 309–320. URL: `https://doi.org/10.1145/3029806.3029825`, doi:10.1145/3029806.3029825.

[12] Tumbleson, C., Wiśniewski, R., 2010–2020. Apktool. `https://ibotpeaches.github.io/Apktool/`.

[13] Wang, Y., Zhang, H., Rountev, A., 2016. On the unsoundness of static analysis for android guis, in: Proceedings of the 5th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis, Association for Computing Machinery, New York, NY, USA. p. 18–23. doi:10.1145/2931021.2931026.