Choosing the Weapon: A Comparative Study of Security Analyzers for Android Applications

Ryan B. Joseph
University of New Orleans, USA
rbjoseph@uno.edu

Minhaz F. Zibran
Idaho State University, USA
MinhazZibran@isu.edu

Farjana Z. Eishita
Idaho State University, USA
FarjanaEishita@isu.edu

***Abstract*—This study compares security focused static code analyzers for Android applications. Android operated hand-held devices (e.g., smart phones, tablets) are used in the modern computing world for nearly every need. Banking, email, health care, and other sensitive dealings are completed through the Android applications. Hence, Android application security must be held to the same level of scrutiny as traditional application security. This study compares two open-source security analyzers, MobSF and MARA, against two benchmark datasets and 20 live Android applications. We highlight the strengths and weaknesses of each analyzer and reveal security vulnerabilities found in the Android applications.**

***Index Terms*—Security, Vulnerability, Cybersecurity, Android, Static Analysis, Empirical Study**

I. Introduction

Mobile devices have quickly outpaced the use of more traditional computers in recent years. Smartphones and their applications have largely replaced the use of desktop and laptop computers. According to a Pew Research Center study on Americans, 81% of own a smartphone and 74% own a desktop/laptop [5]. This trend will not slow down; mobile devices are closing the gap in general computing usage every year. Online banking, healthcare, shopping, email, and other sensitive transactions are now completed on mobile devices. No matter the digital device, software security concerns remain the same. Mobile application developers must be careful not to carry over the common security vulnerabilities from much older paradigms in the field.

This paper compares security analyzers for Android applications. The Android operating system is found in smartphones, tablets, and home entertainment systems. StatsCounter, an online statistics collection service, shows that Android has an 75.85% market share versus iOS at 22.87% in November 2019 [22]. Such a large market share and smartphone use has attracted bad actors with the intent of stealing private information from Android based devices. Many of the same software vulnerabilities found in more traditional applications have carried over into the Android landscape.

Malware purposefully written for Android devices is a growing threat. A 2011 report from Fortinet claims that there are approximately 2000 Android malware samples that belong to 80 different families [4]. Another worrisome trend is the amount of information that mobile applications collect on users. Many applications track users for targeted advertisements, but there have been many cases of abuse. Privacy concerns regarding mobile applications have moved into spyware territory.

Greater security auditing of mobile software is needed. There are some major differences in the underlying architecture and design that prevents applying older security analysis tools to mobile applications. Multiple open-source tools exist for security analysis on software. These tools have matured over many iterations, changing in step with newer vulnerabilities and hacker techniques. Security tools that focus on mobile applications are slowly catching up. In this study, we compare two security analyzers for Android applications. These tools should do the following:

- Identify potential security vulnerabilities within an Android application with contextual framework information
- Automatically reverse engineer APKs (Android Application Packages) to a human readable state
- Generate usable reports with sufficient supporting information that benefits the secure development lifecycle.

The rest of the paper is organized as follows. In Section III, we discuss studies related to this work. Section IV describes the methodology of our study. The findings derived from qualitative and quantitative analyses are presented in Section V. In Section VI, we discuss the possible threats to the validity of the results. Section VII includes a discussion and finally, Section VIII concludes the paper.

II. Background

A. *The Android Operating System*

Android is an open-sourced mobile operating system (OS) developed and maintained by Google Inc. Applications in the Android ecosystem are typically written in Java but go through multiple compilation transformations before the code can be run natively on devices. The Android SDK compiles Java source code and other resources into a compressed Android Package (APK) known as .apk file [23]. The Android Package is comprised of the AndroidManifest.xml (manifest file that lists Android Permissions and other data), a classes.dex file (Dalvik bytecode ran by the DVM), and other XML or graphic resources [23].

B. *APK Reverse Engineering*

Fully assessing security vulnerabilities in an Android Package requires that Java source code can be reviewed and parsed by static analysis tools. The application's APK technically does not contain Java source code. It has been compiled into Dalvik bytecode by the Android SDK at this point. Luckily, tools have been made that can reverse engineer the Dalvik Executable and produce Java source code. Baksmali is tool that disassembles .dex files into various .smali files. Each .smali contains the equivalent information of a Java .class file [23]. Without using extensive obfuscation techniques, Java .class files are relatively easy to reverse engineer.

III. Related Work

There have been many studies on general Android security and purpose-built security analyzers over the past few years. Many of them focus on separate parts of the Android security model and how they may introduce vulnerabilities. Software security traditionally aims at analysis of source code, but the Android Framework has a few aspects that don't exactly live in running code. Configuration settings such as Android Permissions are prime examples of how vulnerabilities can be introduced into APKs. Chiluka, Singh, and Eswaranka covers the misuse of dangerous Android permissions that can lead to private data leakage. The researchers identified many commonly used Android Permissions and categorized them on risk level. Android Permissions are typically found in the AndroidManifest.xml file; they are used by developers to declare which access rights their application needs in order to be run [6]. These permissions are used to access low-level APIs similar to system calls other operating systems. The problem lies in the fact that many permissions are broad in the power granted to user-land applications.

The Android OS and supporting frameworks are related to other operating systems. After all, the Android Kernel is a modified version of Linux. Yet, there are enough differences when taking a holistic view of the Android ecosystem. Joshi and Parekh completed a survey of Android vulnerabilities found in the National Vulnerability Database (NVD) and the Open Source Vulnerability Database (OSVDB) over an 8-year period. They identified 12 vulnerabilities that Android applications are susceptible to [16].

Other studies have examined the use of security analyzers for Android applications. Using static/dynamic analysis tools on software to find bugs that cause security vulnerabilities are common practices in software engineering. Since mobile applications run on specialized hardware with an array of software stacks on top, new tools had to be written and tested. Rangnath and Mitra studied the effectiveness of multiple free security analysis tools for Android applications. They tested the security analysis tools on a custom benchmarking collection, Ghera, that localizes specific vulnerabilities found in Android applications. The researcher found that many of the tools could not detect vulnerabilities across the board. Some of the tools were accurate at detecting some of the vulnerabilities in Ghera's sub-test categories, while completely failing at others [21]. Security analyzers for Android applications still need to go through development and refinement to meet current needs.

There are studies on the security and quality assessment of source code and other artifacts [10], [11], [14], [12], [13] of software systems in general using different security scanning tools. Recently, Daniel et al. [19] conducted a study to assess the efficacy of security scanner plugin for WordPress websites. This study of ours is completely different from all these studies in its objective and procedure. Instead of assessing the software artifacts or source code, we assess the efficacy of the security scanner tools in detecting vulnerabilities in Android applications.

IV. Methodology

A. Analyzer Selection

Android application analyzers are seemingly still an emerging field of development and research. There are not that many all-in-one analyzers that do not require some sort of pre-processing during the reverse engineering phases. In this study, we include two open-source analyzers, MARA and MobSF. Open-source allows for nearly any developer or security professional to install the software package and start incorporating it into their workflow. The scanners were installed on an Ubuntu 18.04 LTS (Bionic Beaver) virtual machine hosted by an Oracle VM VirtualBox v6.0 type-2 hypervisor.

1) MobSF v2.0: The Mobile Security Framework (MobSF) is an automated mobile-application penetration testing, malware analysis, and security assessment framework. MobSF is capable of static and dynamic application analysis of Android, iOS, and Windows binaries. The framework also provides REST APIs to support continuous integration and continuous delivery (CI/CD) pipelines for developers to automatically test their applications on each build. MobSF v2.0 is written in Python 3.7 and it is open-source under the GNU Public License v3.0 [3].

The installed version used in this study was a Docker container provided and maintained by MobSF's authors. This provided a simplified installation process and the ability to tear down or build the service on demand. When the Docker container successfully spins up, users can direct their web-browser to localhost:8000 to interact with MobSF's graphical user interface. MobSF is capable of dynamic analysis, but this aspect was not tested or configured in this study. The dynamic analysis functionality of MobSF requires a cloud-based Android virtual machine service called Genymotion.

2) MARA Framework v0.2.2: The Mobile Application Reverse Engineering & Analysis Framework (MARA) [17] attempts to be a one-stop-shop for a user's Android application reverse engineering and security analysis needs. It is open-source under the GNU Public License v3.0. MARA is mostly a bash script that combines multiple commonly used Android reverse engineering and vulnerability analysis tools into one project. The goal was to create an effortless pipeline for security researchers and developer to use. MARA is capable of dynamic and static analysis without any extra set-up post installation. MARA does not have a GUI component; it was built for terminal user interfaces only. Many of the sub-tools are written in different versions of Python [17]. This study focuses on the static analysis portion of MARA due to the other analyzers' limitations.

B. Benchmarking Applications

The Damn Insecure and Vulnerable App (DIVA) [15] for Android is a purposefully vulnerable Android application that was created as a learning tool for developers. DIVA was built to show case many of the common vulnerabilities that are present in mobile applications. It has built in coding vulnerabilities that cover insecure logging, insecure data storage, input validation errors, and access control issues. DIVA is not a core benchmarking package; it is really meant to be used by penetration testers and developers to learn how to exploit or recognize vulnerabilities in source code. Its use in this study is exploratory in nature, servicing as a touchstone of how the analyzers work.

Ghera [18] is a repository of Android applications that exhibit specific vulnerabilities. Benchmarking software for Android security analyzers has yet to mature, Ghera attempts to rectify the situation. Ghera is a collection of 60 separate benchmarks that tests for specific software vulnerabilities that can be found in real-world applications. Updates and additions to the benchmark collection are frequent. The benchmarks are split into multiple categories such as Crypto, Inter-component Communication (ICC), Networking, Permission, Storage, System, and Web. Each benchmark is comprised of three .apk files, an exploitable benign version, a malicious version, and a patched secure version.

The benign version has the targeted vulnerability built in, while the malicious version has the ability of exploiting the benign version. Secure versions of the benchmarks provide a patched implementation of the exploitable benign case [18]. It is an open-source project under the BSD-3 License. Ghera was built by Joydeep Mitra and Ventkatesh-Prasad Ranganath of Kansas State University [21]. In this study, subsets of Ghera benchmarks found in Table I are used.

C. Live Applications

Testing the security analyzers against benchmarks provides a sound basis to reporting and functionality. Using real-world examples provides thorough testing of the reverse engineering capabilities of the analyzers and may reveal malicious content on the Google Play marketplace for Android applications. Normally, Google Play requires a registered Android device to install applications. There are known work arounds, such as third-party sites that host a reflection of application currently on Google Play.

APKPure.com is a third-party distributor of free Android applications that can be found on the official market. Developers can submit their applications to the site through approved channels. Users can then download their choice of applications and run them in emulators or natively on their Android device [1]. This provides a great way to acquire applications immediately, without using an Android device and isolating the .apk files for transferring them to analysis workstations. The live applications in this study, as listed in Table II, can be divided into the following broad categories: Flashlight Apps, Messenger Clones, Fitness Trackers, Home Automation, Photo Filters, TV Apps, and Other (e.g., social networking/business/special events).

V. Analysis and Findings

A. Qualitative Results

1) Report Output: The presentation of results to users is an important aspect to choosing a robust analysis tool. The results should be viewable in a human readable format that is concise and provides enough contextual information so that a full security assessment can be done. MARA and MobSF arguably provide many of the same findings, but in inconsistent and problematic ways.

MARA is an all-in-one tool, built from many other Android reverse engineering and security analysis tools. It is the "Frankenstein's Monster" of Android security analyzers. This becomes a significant setback in sub-tool usability. Each of these different tools have their own expected inputs and produced outputs. In the reverse engineering phases, the tools work well together in a typical workflow that is representative of how they would be used separately. Analysis, post reverse engineering, is a dizzying array of text files from multiple analysis tools which are spread across multiple file system directories. These text files are often terminal based redirects of what would have been presented to the user by one of MARA's sub-tools. The file naming conventions made it relatively easy to ascertain what type of information was in each file, but knowing which tool was responsible for the output is challenging.

Worst yet, many of MARA's sub-tools do not seem to be producing the expected output. MARA claims to have a smali control-flow graph generator built in by using Eugino Delfa's Smali-CFGs [8]. The control flow graphs are nowhere to be found when parsing through MARA's output directories. There are also cases when MARA simply fails to produce results. In the case of WeChat v7.0.5, the tool hit multiple runtime exceptions during the reverse engineering phases. Data could not be collected from running MARA on WeChat v7.0.5.

The Trueseeing Report is the most dependable report being produced. This report is the focal point in comparisons to MobSF's reports. The Trueseeing Report categorizes its findings based on severity levels: Critical, High, Medium, Low, and Info. Each vulnerability found is given cursory information such as Synopsis, Description, Risk Factor (severity), CVSS Temporal Score, and Instances. The Synopsis and Description sections are not always filled in with information.

MobSF is a relatively easy application to use for in-depth APK analysis. Users are directed to upload the application of their choice, and a loading bar appears as the application goes through the entire reverse engineering and analysis process. The interface switches to a simple, yet elegant, webpage that shows all of MobSF's security focused findings with logical subsections and info pages. MobSF's report format allows the user to take a quick glance to assess the severity of an

TABLE I: Ghera Benchmarks

Category	Name	Vulnerability
Crypto	• BlockCipher-ECB	Block cipher algorithm in ECB mode
	• BlockCipher-NonRandomIV	Block cipher algorithm in CBC mode, uses non-randomized initialization vector
Permission	• UnnecessaryPermissions-Priv Escalation	Unnecessary permissions used for privilege escalation attacks
	• WeakPermission-UnauthorizedAccess	Weak permissions used to export services, offers little protection.
Storage	• ExternalStorage-DataInjection	External storage susceptible to data injection
	• ExternalStorage-InformationLeak	External storage susceptible to data leaks
	• InternalStorage-DirectoryTraversal	Internal storage susceptible to directory traversal
	• InternalToExternalStorage-InformationLeak	Transfer of internal to external storage may cause information leaks
	• SQLite-ExecSQL	SQLiteDatabase.execSQL() method is susceptible to injection attacks
	• SQLlite-RawQuery-SQLInjection	SQLiteDatabase.rawQuery() method is susceptible to injection attacks

TABLE II: Live Applications Used for Assessment

ID	Application	Version
App-1	Free TV Shows App News Series Episode Movies	5.20
App-2	UNO Social	5.60.0_9168
App-3	MyRunTracker	4.3.1
App-4	Running Fitness Calorie Sport tracker	3.0
App-5	Running Tracker	1.8
App-6	Free TV shows series movies news sports	1.0
App-7	Photo Camera HD for Instagram	1.8.3.v7a
App-8	Decora Digital Dimmer Timer	1.5.6
App-9	Screen Flashlight	1.2.6
App-10	Super flashlight LED 2019 super lightnes	3.4
App-11	Messenger	1.15.2
App-12	The Messenger App Free for message chat	3.1.3
App-13	InstaSweet Retro Vintage Photos Filter Camera	1.0.1
App-14	Information to Feed Your Mind	1.0
App-15	DXC Onboard Me	2.0.3
App-16	WeChat	7.0.5
App-17	My Leviton	2.0.75
App-18	Color Flashlight	3.8.8
App-19	Messenger for Messages Text and Video Chat	2.95
App-20	New Orleans Jazz Festival	17.1

applications security issues. An Average CVSS Score, Security Score, and Trackers total is given to the user immediately. De- obfuscated source code files can be retrieved from the webpage. Download links are provided for the AndroidManifest.xml, Java source code, and Smali code.

This report covers all the information found in the various text files of MARA's output directories. MobSF's major sections covers the Code Signers Certificate, Android Permissions, Android API Calls, Security Analysis, Malware Analysis, and Recon to name a few. Many of these major report sections are subdivided into smaller groupings. Security Analysis is subdivided into Manifest, Code, and File analysis. Each of these smaller groupings contains vital information that is given a severity score. The level of detail far exceeds MARA's Trueseeing Report.

2) Findings Categorization: Using MARA and MobSF to analyze the same benchmark and live APKs allowed a baseline of how each tool worked and its propensity of error. The tools reported their vulnerability findings extremely differently which made direct comparison difficult. MARA has its Trueseeing Report, which is the only dependable report that was produced out of the sub-tools. The Trueseeing Report categorizes general findings and vulnerabilities in broad strokes. It categorizes general findings such as null-terminated strings, URLs, library usage, and others in the Information category. Problematically, it also categorizes some important findings such as Permissions in the Information category. Which Permissions are present is one of the core aspects of assessing an APK's security. MobSF categorizes its general findings and vulnerabilities in a much higher granularity. MobSF also applies severity categorizations to many of MARA's informational findings such as Permissions and Broadcast Receivers/Services. This led to a much higher reporting of sever vulnerabilities in MobSF's report.

3) Tool Documentation: Proper documentation is an important aspect of using any software tool. Android application analyzers are no different. MARA's documentation is fundamentally incomplete. Its GitHub page covers the "front-end" bash script that runs the sub- tool pipeline, but how these tools work together and how the output is formatted is not covered in detail. Essentially, users have to track down information on each sub-tool and make an educated guess to where its output goes. Another problem is that using MARA's bash script removes any fine-grained control of how the sub-tools function. Users cannot pass optional flags from MARA's "front-end" to a sub-tool. MARA is a wrapper for all of the other tools; users are forced to hunt for multiple wiki forms that may not even apply to their issues.

MobSF's documentation is more complete and straightforward. Most of the necessary documentation can be found on MobSF's GitHub wiki. Two exceptions are setting up MobSF for dynamic analysis and CI/CD integration. Dynamic analysis requires using an Android virtual machine, the maintainers recommend installing and using Genymotion. This is a third-party tool from MobSF's perspective, yet it provides documentation on how to integrate it into the analyzers framework once installed. MobSF's documentation on CI/CD integration is merely a link to a Medium.com article written by Omer Hevroni [9].

B. Quantitative Results

Due to the vast categorization differences in reporting, MARA and MobSF's categorization range is compressed to the following: High, Medium, Low, and Info. This was done to fairly compare the two analyzers. MARA's Trueseeing Report has a higher severity categorization, Critical. MobSF's Signature categorization does not compare to any within MARA's Trueseeing report. This categorization was extremely rare to see and was dropped from the results.

a) Performance on the DIVA Benchmark: Figure 1 presents the number of vulnerabilities at different severity levels (i.e., high, medium, low, info) detected in the DIVA benchmark by MARA and MobSF. This figure shows the differences in the characterization/categorization of severity of vulnerabilities by MARA and MobSF. MobSF correctly identifies and categorizes dangerous Android Permission usage as High. MARA collects the same information for the AndroiManifest.xml, but groups permission findings in the 'info' category. This trend is also seen in the results for the Ghera benchmark and for the live applications.

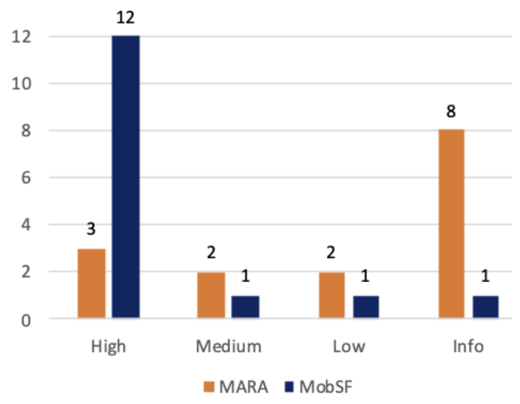


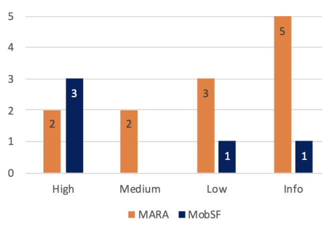
Fig. 1: Vulnerabilities Detected in the DIVA benchmark

b) Performance on the Ghera Benchmark: Figure 2a and Figure 2b present the vulnerabilities detected by MARA and MobSF in the *benign* versions of Ghera's BlockCypher-ECB and BlockCipher-NonRandomIV benchmarks. A manual investigation reveals that MARA and MobSF correctly found the vulnerabilities in question but categorized them differently. Some vulnerabilities classified as 'medium' by MARA are categorized as 'high' (severity) by MobSF.

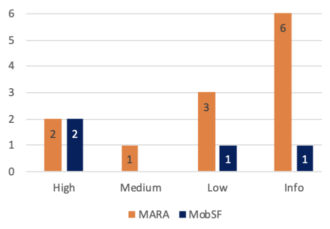
Figure 3a and Figure 3b present the number of false positives reported by MARA and MobSF in the *secure* versions of Ghera's BlockCypher-ECB and BlockCipher-NonRandomIV benchmarks. As seen in the figures, MARA reports more false positives compared to those of MobSF.

For the rest of the Ghera benchmark, we found mixed results. On the benchmark under Storage category, both MARA and MobSF fail to detect many of the vulnerabilities. Only MobSF is able to detect some vulnerabilities in the SQL-based benchmarks (i.e., SQLite-ExecSQL and SQLlite-RawQuery-SQLInjection). Both analyzers correctly identified the vulnerabilities in Ghera's Permission based benchmarks (i.e., UnnecessaryPermissions-Priv Escalation and WeakPermission-UnauthorizedAccess). However, the detected vulnerabilities are categorized differently with respect to severity. MARA places all permission-based findings in its 'Info' category, while MobSF classifies most them as 'Dangerous'. These mixed results are not surprising considering that the Ghera Benchmark developers also came to a similar mixed conclusions about different security analyzers [21].

c) Performance on the Live Applications: As mentioned before, we operated MARA and MobSF separately on 20 live Android applications obtained from APKPure.com.

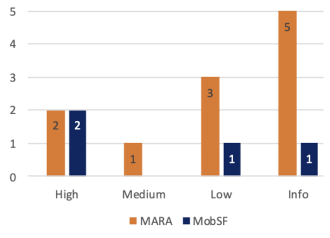


(a) BlockCipher-ECB

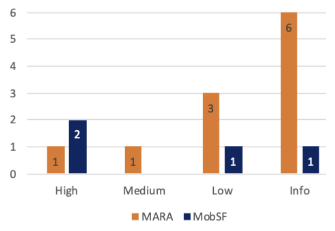


(b) BlockCipher-NonRandomIV

Fig. 2: Vulnerabilities in the Ghera Crypto (benign) Benchmark

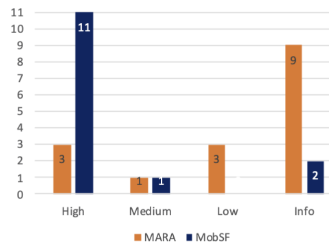


(a) BlockCipher-ECB

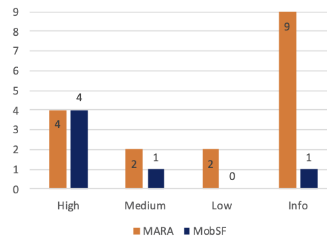


(b) BlockCipher-NonRandomIV

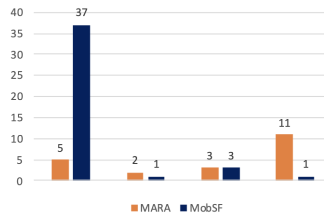
Fig. 3: False Positives for the Ghera Crypto (secure) Benchmark



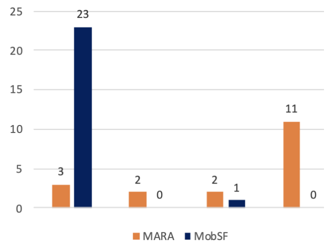
(a) My Leviton v2.0.75



(b) Color Flashlight v3.8.8



(c) Messenger for Messages Text and Video Chat v2.95



(d) New Orleans Jazz Fest v17.1

Fig. 4: Vulnerabilities Detected in Live Android Applications

TABLE III: Vulnerabilities Detected in Live Applications

App	Scanner	H	M	L	I	App	Scanner	H	M	L	I
1	MARA	5	2	2	11	9	MARA	4	2	2	9
	MobSF	40	2	0	2		MobSF	3	1	1	1
2	MARA	1	1	1	10	10	MARA	4	2	3	8
	MobSF	20	1	1	2		MobSF	13	1	0	1
3	MARA	4	0	2	11	11	MARA	3	1	3	9
	MobSF	28	1	1	1		MobSF	21	1	1	1
4	MARA	4	2	3	9	12	MARA	7	3	4	12
	MobSF	15	1	2	1		MobSF	65	2	3	2
5	MARA	4	2	3	8	**13**	MARA	0	0	0	0
	MobSF	2	1	0	1		MobSF	31	1	3	1
6	MARA	4	1	2	10	14	MARA	4	2	3	8
	MobSF	8	1	2	1		MobSF	8	1	1	2
7	MARA	5	3	2	10	15	MARA	1	1	3	9
	MobSF	14	1	1	1		MobSF	18	1	1	1
8	MARA	2	1	3	6	**16**	MARA	0	0	0	0
	MobSF	6	1	0	1		MobSF	194	0	1	1

Here, H = high, M = medium, L = low, I = info

Figure 4a presents the number of vulnerabilities at difference severity levels detected by MARA and MobSF in My Leviton v2.0.75 (App-17). As seen in the figure, many vulnerabilities, which MARA categorized as 'high' severity are characterized as 'info' by MobSF. A slightly different scenario is found in the results for Color Flashlight v3.8.8 (App-18). As presented in Figure 4b, the number of vulnerabilities detected and classified in each category are by both MARA and MobSF are in agreement. The only exception is that MARA identified higher number of vulnerabilities classified as 'info'. These additional vulnerabilities are legitimate informational findings such as text strings and URLs, while MobSF does not give a categorization score to these types of findings in its report.

Figure 4c presents the results for Messenger for Messages Text and Video Chat v2.95 (App-19). Compared to MARA's result, MobSF reports a large number of vulnerabilities classified as 'high' severity. This appears surprising. In most cases, the MobSF-reported 'high' severity vulnerabilities would be counter balanced by MARA-reported 'info'-level vulnerabilities. But, this is not the case here.

MobSF gave the application a general Security Score 15/100, which is the lowest among all the 20 live applications in this study. Further investigation reveal that MobSF reports a vulnerability to the application requesting root privileges and a decent percentage of the source code was not fully de-obfuscated. MARA was also not able to fully de-obfuscate the application's source code. MobSF also found 16 trackers associated with the application. For each tracker, MobSF cross-references the information with a privacy organization called Exodus-Privacy [2]. This allows MobSF users to quickly find information on the trackers used in applications. CallDorado, one of the trackers found in this Messenger clone, is able to collect vast amounts of private data from users' phones (call logs, contact lists, and location). Worst yet, this data is shared with unspecified third parties [20]. Such an application seems to be bordering on the definition of spyware.

A similar result is found for New Orleans Jazz Fest v17.1 (App-20), as presented in Figure 4d. The number of vulnerabilities at different severity levels detected by MARA and MobSF in the other 16 live applications (App-1 through App-16) are presented in Table III. As seen in the table, for App-13 (Instasweet Retro Vintage Camera Filter Camera v1.0.1) and App-16 (WeChat v7.0.5), MARA reports zero vulnerabilities. This is because MARA fails to reverse engineer these two applications.

VI. Threats to Validity

Many of the assertions in this study comes down to the overall usability of the two analyzers in question. This is the main threat to validity. It can be argued that MARA and MobSF are capable of many of the same findings. Those arguments are largely correct, yet a tool is meant to be used. MARA catalogs all of the same findings that are presented in MobSF's report, but the information is presented in such a difficult way. Only one report, Trueseeing, was dependably generated when using MARA across the benchmarks and live applications. Qark, another sub-tool of MARA, also produces reports. These Qark reports were not always generated dependably. When MARA fails, it does so silently. A developer or security researcher can comb through all of MARA's text file dumps to piece together usable information. Or they can get the same results in one easy to read MobSF report.

Another threat to validity was the methods of comparison. MARA and MobSF do not have the same severity level categorizations. Some levels had to be fused for both security analyzers. Also, MobSF has some specialized severity level categorizations that specific to sub-sections within its own report. If severity level combinations are done incorrectly, one tool's perceived effectiveness may be affected.

VII. Discussion

In addition to the two Android application analyzers (i.e., MobSF and MARA), we also attempted to include a third analyzer names StaCoAn. The Static Code Analyzer (StaCoAn) is a cross platform tool perform static code analysis on Android .apk files. It is meant to provide a way for developers and security researchers to find major coding mistakes, hard coded credentials, API keys, and decryption keys to name a few. The open-source project is written in Python 3.7 and is licensed under The MIT License. StaCoAn provides a graphical user interface through using a web browser, similarly to MobSF. Users can upload an .apk file to go through the reverse engineering process and analysis phases. Source code de-obfuscation support is not included in StaCoAn's reverse engineering phases. Indeed, code de-obfuscation could be used maliciously by bad actors to reveal security vulnerabilities and proprietary source code [7].

StaCoAn's analysis phase is limited, it relies on regular expressions for pattern and keyword recognition. Other than potential keywords or patterns. StaCoAn can be useful during internal development of an application, when source code is not obfuscated. Operating StaCoAn v0.80 on the android applications in the early phases of this study produced unreadable reports with difficult to understand flagged code. Thus, it is dropped from this study.

VIII. Conclusion

This study compared two Android app security analyzers to investigate if their use could benefit developers and security researchers in assessing application security. Either scanner, MARA or MobSF, could be used to assess the security posture of an Android application. Both identify security vulnerabilities within the context of Android applications and reverse engineer fully packaged APKs. The major difference between the two is how the information is presented post analysis.

MobSF has an easy to follow report that logically shows information in subsections. Each individual finding that may pose a security risk is given some sort of vulnerability score. MARA's sub-tools are haphazardly strung together by a bash script that dumps their findings in directories. It may not always produce the same results if one of the sub-tools fails. If a sub- tool fails, it does so silently without alerting the user. MARA's most dependable report is Trueseeing, which often categorizes high risk findings within its Info category. MARA needs its own reporting format that collects information from its disparate sub-tools in one place. In its current state, MARA is a time-consuming venture; it is possibly useful but arguably not usable. MobSF can be preferable over MARA.

References

[1] *APKPure*. https://apkpure.com, verified: Apr 2021.

[2] *Exodus Privacy*. https://reports.exodus-privacy.eu.org/en/info/organization/, verified: Apr 2021.

[3] Ajin Abraham. *MobSF: Mobile Security Framework*. https://github.com/MobSF/Mobile-Security-Framework-MobSF, verified: Apr 2021.

[4] Axelle Apvrille and Tim Strazzere. Reducing the window of opportunity for android malware gotta catch 'em all. *Journal in Computer Virology*, 8(1):61–71, 2012.

[5] Pew Research Center. *Demographics of Mobile Device Ownership and Adoption in the United States*. https://www.pewresearch.org/internet/fact-sheet/mobile/, verified: Apr 2021.

[6] N. Chiluka, A. K. Singh, and R. Eswarawaka. Privacy and security issues due to permissions glut in android system. In *International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 406–411, 2018.

[7] Vincent Cox. *StaCoAn*. https://github.com/vincentcox/StaCoAn, verified: Apr 2021.

[8] Eugenio Delfa. *Smali-CFGs: Smali Control Flow Graph*. https://github.com/EugenioDelfa/Smali-CFGs, verified: Apr 2021.

[9] Omer Levi Hevroni. *How To: (Continuously) Hacking Your App*. https://medium.com/@omerlh/how-to-continuously-hacking-your-app-c8b32d1633ad, verified: Apr 2021.

[10] M. Islam and M. Zibran. A comparative study on vulnerabilities in categories of clones and non-cloned code. In *Proceedings of the 10th IEEE International Workshop on Software Clones*, pages 8–14, 2016.

[11] M. Islam and M. Zibran. On the characteristics of buggy code clones: A code quality perspective. In *Proceedings of the 12th IEEE International Workshop on Software Clones*, pages 23 – 29, 2018.

[12] M. Islam and M. Zibran. How bugs are fixed: Exposing bug-fix patterns with edits and nesting levels. In *Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing*, pages 1523–1531, 2020.

[13] M. Islam and M. Zibran. What changes in where? an empirical study of bug-fixing change patterns. *ACM Applied Computing Review*, 20(4):18–34, 2021.

[14] M. Islam, M. Zibran, and A. Nagpal. Security vulnerabilities in categories of clones and non-cloned code: An empirical study. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 20–29, 2017.

[15] Aseem Jakhar. *DIVA Android*. https://github.com/payatu/diva-android, verified: Apr 2021.

[16] J. Joshi and C. Parekh. Android smartphone vulnerabilities: A survey. In *International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring)*, pages 1–5, 2016.

[17] Christian Kisutsa. *MARA framework*. https://github.com/xtiankisutsa/MARA_Framework, verified: Apr 2021.

[18] Joydeep Mitra, Venkatesh-Prasad Ranganath, Aditya Narkar, Nasik Nafi, and Catherine Mansfield. *Ghera*. https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/src/master/, verified: Apr 2021.

[19] Daniel T. Murphy, Minhaz F. Zibran, and Farjana Z. Eishita. Plugins to detect vulnerable plugins: An empirical assessment of the security scanner plugins for wordpress. In *Proceedings of the International Conference on Software Engineering, Management and Applications (SERA 2021)*, pages 1–6 (to appear), 2021.

[20] Exodus Privacy. *CallDorado*. https://reports.exodus-privacy.eu.org/en/trackers/252, verified: Apr 2021.

[21] Venkatesh-Prasad Ranganath and Joydeep Mitra. Are free android app security analysis tools effective in detecting known vulnerabilities? *Empirical Software Engineering*, 25(1):178–219, 2020.

[22] StatCounter. *Mobile Operating System Market Share Worldwide*. from https://gs.statcounter.com/os-market-share/mobile/worldwide, verified: Apr 2021.

[23] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik. A new android malware detection approach using bayesian classification. In *27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 121–128, 2013.