Oct 12th, 11:55 AM - 12:20 PM

An Exploratory Analysis of Mobile Security Tools

Hossain Shahriar
Kennesaw State University, hshahria@kennesaw.edu

Md Arabin Talukder
Kennesaw State University, mtalukd1@students.kennesaw.edu

Md Saiful Islam
Kennesaw State University, mislam16@students.kennesaw.edu

Abstract
The growing market of the mobile application is overtaking the web application. Mobile application development environment is open source, which attracts new inexperienced developers to gain hands on experience with applicationn development. However, the security of data and vulnerable coding practice is an issue. Among all mobile Operating systems such as, iOS (by Apple), Android (by Google) and Blackberry (RIM), Android dominates the market. The majority of malicious mobile attacks take advantage of vulnerabilities in mobile applications, such as sensitive data leakage via the inadvertent or side channel, unsecured sensitive data storage, data transition and many others. Most of these vulnerabilities can be detected during mobile application analysis phase. In this paper, we explore vulnerability detection for static and dynamic analysis tools. We also suggest limitations of the tools and future directions such as the development of new plugins.

Location
KSU Center Rm 400

Disciplines
Information Security | Management Information Systems | Technology and Innovation

INTRODUCTION

In 2018, a report by Statcounter Global (Statcounter, 2018) suggested that Android dominates the smartphone market with 76.6% share with over two billion users active monthly. Consequently, this gigantic market of potential victims has not remained unnoticed by cybercriminals and online malicious users. Over the years, several third party websites and application stores have emerged on the internet that allow android phone users to download and install android application which are otherwise deemed malicious or dangerous by Google's Play store. These websites and stores often contain and advertise several malicious versions of various otherwise pay-to-use android applications on the Google Play store.

The attackers exploit typical human nature, by advertising these applications as free to use, which often results in large download traffic of these malicious applications through these websites and stores. To make matters worse, these websites are often unmonitored and ungoverned, which allows virtually anyone to upload malicious software online without the supervision of any centralized or decentralized authority. This malicious software are generally packed with several forms of malware payloads including but not limited to, trojans, botnets, and spyware. These applications can easily help in theft of valuable personal information regarding an android phone user such as, usernames, passwords, Social Security Numbers (SSN), Health history, location history and much more.

Recent reports have suggested that the number of overall malware attacks over the internet is still growing exponentially. Moreover, the majority of these attacks are targeted on the Android platform. According to a Kaspersky Labs report, more than 291,800 malware programs had surfaced over the second quarter of the year 2015. This increase was over 2.8 times more than the number of malwares surfacing the first quarter of the same year. Additionally, the number of such malware applications installed from various untrusted third-party websites and stores increased by over one million in shear numbers over the same time period.

With this explosive rise in the number of Android malware applications, analysis, detection and prevention against such applications has become a critical research topic. Several techniques, frameworks, tools and software have been proposed in literature to prevent and detect such applications. Consequently, the research in Android security and analysis has transitioned into a wide domain both in academic and enterprise communities. Despite many tools are available, there are limited resources in the literature emphasizing hands-on application of the tools for analyzing malware.

In this work, we explore two analysis tools (static and dynamic) to analyze the source code and behavior of android applications. Static analysis is the analysis without actually running the source code of the software or application under test, whereas, dynamic analysis tends to be much more comprehensive and informative, since such type of analysis is performed under actual code execution.

LITERATURE REVIEW

The growth of mobile market and its development is increasing. In accordance with the security and privacy of users data in the mobile application is also an issue with higher priority. Most of the mobile application gets access to secret information of the user. To review the security hole in

the application, the number tools that we have in the market is not enough. Most of the useful tools has been developed a long time ago which has almost no effectiveness to current features. In this paper we reviewed coupe of renowned security tools that were effectively used in the past to detect security problems in the Android Application. Interested readers can see the extensive survey by Kong et al. (2019) for other related work.

CuckooDroid

Cuckoo is a premier malware analyzing software. It is capable of methodically examining multiple variants of malware through the use of virtual machines that monitor the behavior in a protected and isolated atmosphere (Cuckoo, 2019). It is written in the python programming language and facilitates its analysis in both the static and dynamic dimension. Cuckoo is an open source programmed malware analysis system. It's used to run and investigate files and gather inclusive analysis results that framework what the malware is and does even though running inside a secluded Windows OS. It can generate the below results:

- Files created, deleted, and downloaded by the malware
- Memory dumps of the malware processes.
- Traces of win32 API calls accomplished by all processes produced by the malware.
- Network traffic trace in PCAP format.
- Full memory dumps of the machines.

The sandbox of Cuckoo started back in 2010, it started as a Google summer of code project within the honeynet project (Cuckoo, 2019). The first beta release of Cuckoo was released on Feb 2011 and then in March 2011, it was selected again as a supported project. After many versions of the sandbox, Cuckoo sandbox was released in April 2014.

It comprises of a host that is responsible for the sample execution and the analysis in which the guests run. So, when the host has to launch a new analysis, it chooses the guests and uploads that sample as well as the other components that are required by the guest to function (Li, 2016). Cuckoo initializes modules when it first starts up and when there is a new task sent to Cuckoo, it identifies the file by using the machinery modules which is used to interrelate with the diverse possible virtualization systems, and its config, it installs what is known as the analyzer inside one of the available virtual machines.

Once the analysis has completed, the analyzer refers the results of the analysis to the ResultServer, which in turn will implement whichever processing modules are configured (the modules used to populate the product of the analysis, the report) and produce the report. The analysis took place in the virtualized machine, it has the monitoring system components The proxy is a python script that intervals listening to a port in the guest machine. When a new inquiry is launched in the machine, the host sends the equivalent analyzer (the component in charge of managing the analysis inside the machine) and the package module used to accomplish the sample sent, which depends on the type of sample.

For instance, the array used to implement an exe file will be altered from that used to open a PDF sample, or a ZIP file. When the mockup completes its implementation, or a break is reached, the analyzer stops the analysis, collects the results from the monitor, and sends them back to the result server.

FlowDroid

With increasing data leaks happening, privacy is a very important concept to maintain. Most privacy leaks are due to flaws in the code that could have been prevented if it noticed and fixed in a timely manner. At a high level view, FlowDroid is an open source Java based tool that can be used to analyze Android applications for potential data leakage. FlowDroid is the first full context, object sensitive, field, flow, and static taint analysis tool that specifically models the full Android lifecycle with high precision and recall (Artz et al., 2013). The tool can detect and analyze data flows, specifically an Android application's bytecode, and configuration files, to find any possible privacy vulnerabilities, also known as data leakage. It is not meant to analyze malware (Fratantonio, 2016). FlowDroid does different types of taint analysis: objective sensitive, flow, context, field, and lifecycle aware (Reaves et al., 2016). In regards to reflective calls, FlowDroid can only fix reflective calls who has constant strings as parameters (Qiu, 2018).

This tool can be built using Maven, a build automation tool for Java projects, or Eclipse, an IDE used for software development. The data tracker can used from the command line (Flowdroid, 2019). This tool cannot be used in Android Studio since it is not an Android application. These additional tools need to be downloaded in order for FlowDroid to run properly. Those tools are: Jasmin, Soot, Heros, and GitHub repositories soot-infoflow and soot-infoflow-android (Flowdroid, 2019). Jasmin is an open source tool that can convert Java classes ACII descriptions into binary Java class files that can be loaded into a Java interpreter (Jasmin, 2019). Soot is an open source Java optimization framework that has four different types of representation of analyzing and changing Java bytecode (Soot, 2008). Heros is a general implementation of an IFDS and IDE framework solution that can be integrated into an existing Java program analysis framework (Heroes, 2019).

Based on Reaves et al. (2016) experience with setting up FlowDroid, it took 1.45 hours for them to fully and properly set up the tool. They had to download missing SDK files that were necessary to run DroidBench, as well as, Flow Droid .jar files. 2 minutes were spent changing configuration settings of the analyzed mobile applications.

FlowDroid uses an analysis technique based on an analysis framework that does not rely on every program path (Artz et al., 2013). This means that every program path does not need to be analyzed. Android applications does not contain a main method in their code. Instead, they have methods that are indirectly invoked upon by the Android framework. This leads to a problem where Android analyzing tools cannot start the analyzing process by evaluating the main method of the program. FlowDroid solves this problem by generating and analyzing a fake main method where there is every possible life cycle arrangement of separate application components and callbacks. It is not necessary to go through all the possible paths, because the technique previously stated solves this problem, and it would also be expensive to implement.

FlowDroid uses a call graph technique to accurately map components to callbacks, which leads to minimal false positives and lowered taint analysis running time. The tool generates one call

graph for each application component. The call graph is used in the process of scanning calls to Android system methods that has a popular callback interface as a parameter. The call graph gets extended until all callbacks are found. FlowDroid was tested for malware analysis along with three other analysis tools. The other three analysis tools were Kirin, TriggerScope, DroidAPIMiner. Among the other analysis tools, FlowDroid had the highest false positive percentage, and second lowest false negative percentage. This tool is not great at malware detection, specifically logic bombs, because that was not the intended purpose of this tool. A logic bomb is an unauthorized software that changes the output of the Android application or does applications actions that are not intended (Fratantonio, 2016).

DroidBox

DroidBox is a dynamic malware analysis tool for Android applications. DroidBox v4.1.1 is a framework for analyzing automatically Android applications. It uses a modified version of the Android emulator 4.1.1_rc6 enabling to track Android applications' activity, i.e., tainted data leaked out, SMS sent, network communications, etc. It is composed of two folds: one fold on the guest machine (Android emulator) that tracks the Android application's activity and sends the corresponding DroidBox logs through ADB to the host machine and the other fold on the host machine that parses the ADB log to extract the log of DroidBox (2019). The release has only been tested on Linux and Mac OS. If you do not have the Android SDK, download it from http://developer.android.com/sdk/index.html. The following libraries are required: pylab and matplotlib to provide visualization of the analysis result.

Export the path for the SDK tools.

```
$ export PATH=$PATH:/path/to/android-sdk/tools/
$ export PATH=$PATH:/path/to/android-sdk/platform-tools/
```

Download necessary files and decompress it anywhere.

```
wget
https://github.com/pjlantz/droidbox/releases/download/v4.1.1/DroidBox411RC.tar.gz
```

Setup a new AVD targeting Android 4.1.2 and choose Nexus 4 as device as well as ARM as CPU type by running:
Start the emulator with the new AVD:

```
$ ./startemu.sh <AVD name>
```

When emulator has booted up, start analyzing samples (please use the absolute path to the apk):

```
$ ./droidbox.sh <file.apk> <duration in secs (optional)>
```

The analysis is currently not automated except for installing and starting packages. Ending the analysis is simply done by pressing Ctrl-C.

DroidBox analyses incoming and outgoing network activity in an application. It also records and analyses all file read and or write activity of an application. All initialized services and loaded

classes are recorded and analyzed through DexClassLoader component. It also reports on any information leakages either through network activity, file operations, and or SMS. It also monitors security permission protocols and returns warnings in case a protocol is circumvented. If there are any broadcast activity, DroidBox monitors and lists all receivers and listeners. As default, DroidBox monitors all calls and SMS activity, analyses each one and returns results. Additionally, two graphs are generated visualizing the behavior of the package. One showing the temporal order of the operations and the other one being a treemap that can be used to check similarity between analyzed packages (Droidbox, 2019). In the graph showing the temporal order of application behavior, DroidBox maps each activity to a specific timestamps which provides an overview of the linear behavior of the application based on Android system events. These events could be sending SMS, making a call, read/write or other internal system activity.

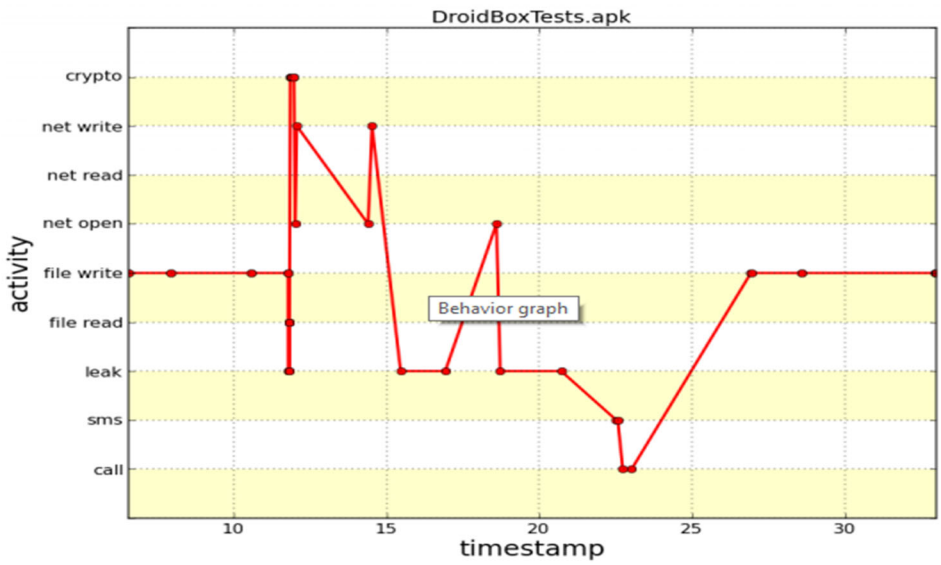

Figure 1. Apk Test result by DroidBox

The second graph presents a simple image that shows the similarity between packages and related operations carried out on them. This enables the user to easily locate related packages that were affected as part of a specific operation.

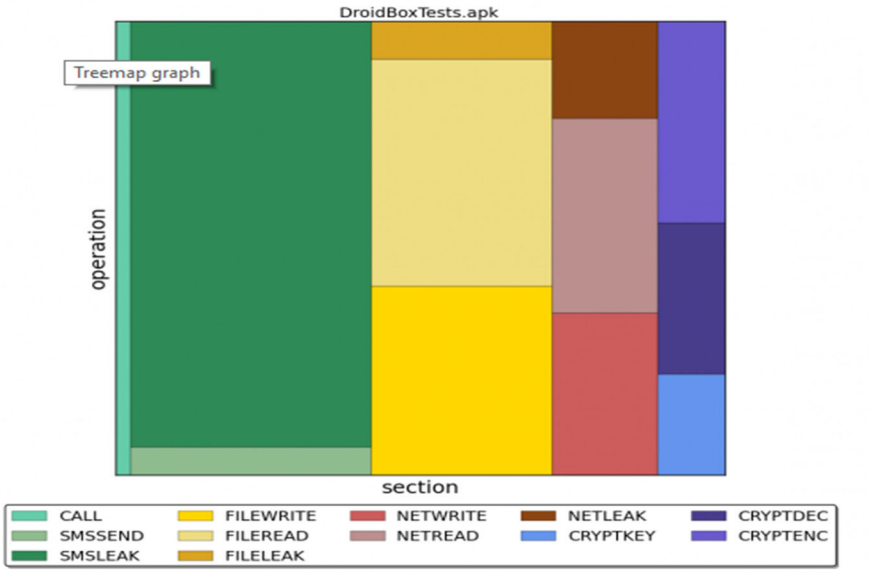

Figure 2. Comparison between packages and related operation

Unlike MobSF that provides output targeting the specific source of a behavior, DroidBox provides and overview of the related packages and type of behavior such as SMS, network leakages, etc. that occurred. Although DroidBox indicates the general behavior and possible suspect packages, it does not pinpoint the source code. Other tools such as MobSF provides a indicate to what source is related to the behavior and captures timeline of behavior. However, it must noted that DroidBox provides relevant information regarding multiple behavior of packages and application. DroidBox analysis may be used to have an overview of malware behavior and can provide suggestions as to which specific packages and resources in which security testing could begin. DroidBox is a good analysis tool that can be used in the early states of dynamic malware analysis in Android applications. Although it provides some helpful output to identify application behavior and localize affected packages, it does not give detailed information about what the underlying code is nor does it provide any output to what specific modules in the code that is responsible for the output behavior and what steps must be taken to remove and or quarantine the affected packages. Droidbox is a purely dynamic analyses tool. It does not perform any static analyses. Hence, for a user who wants to do both dynamic and static analyses, they would need to install another tool to perform static analysis.

EMPIRICAL ANALYSIS

In this section we are going to discuss about static and dynamic analysis of malware application. Static analysis approach analyzes the source apk file of android application, and identifies possible security risks without actually running on a mobile device or emulator. On the other hand, a dynamic analysis approach runs an apk file into a virtual emulator and analyzes activities performed to identify malicious activities.

We collected 1260 malware applications from different third party marketplaces. For example, Contagiodump (2019). Later we examined couple of applications using static and dynamic analysis

tool. We tested these malware samples in Mobile Security Framework (MobSF, 2019) and Flowdroid (2019). However, there are a couple of more analysis tools available for the analysis such as DroidBox, CuckooDroid, and DroidSafe. At first, we tried all the mentioned tools here. However, we found MobSF and Flowdroid are the most effective and updated tool among all. Table 1 shows the list of tables and their coverage in terms of static and dynamic analysis of apk files.

Table 1: Analysis coverage of example tools

Tool	Static analysis	Dynamic analysis
Cuckoodroid	No	Yes
Flowdroid	Yes	No
Droidbox	No	Yes
MobiSF	Yes	Yes

Static Analysis by MobSF

Android is a combined form of java and xml. It uses Linux operating system which is developed for embedded systems and mobile devices. The upper layers of Android language is written by itself. For Graphical User Interface (GUI) android uses XML layout files, it also uses event-based library. Static analysis refers to decompiling the APK of an application to its corresponding Java and XML files. To perform static analysis a static analyzer must have features to examine XML with correctness and precision. Java files can be extracted by DEXtoJar decompiler (Skylot, 2019). A Static analysis tool de-compiles the code of an APK to human readable format. So that an analyzer can read the code and identify the vulnerability that an application could have. "Mobile Security Framework (MobSF)" is a combined tool which does the static and dynamic analysis of an APK. MobSF is developed based on Python 3.7. To generate the report MobSF uses html. It runs the local server via command line in the host computer. All the existing static and dynamic analysis tool do the analysis using DroidMon-Dalvik Monitoring Framework (Idan, 2019) and Xposed Module Repository (Rovo, 2019). Xposed is a framework that can change the behavior of the system and apps without touching the APK (Spreitzenbarth et al., 2014). MobSF (2019) works fine in Windows 7 and 10, macOs(El capitan, high Sierra), Linux(Kali, Ubuntu 16.04). We used MacOS high sierra to perform the analysis. To start the development server the following command has to be executed on the terminal.

Figure 3. Terminal Command to start server MobSF

Figure 3, shows the terminal command for unix operating system to start development server. The command will initiate the development server at "http://127.0.0.1:8000/". In this development server we can upload the APK file for analysis. The dynamic environment is also available in the development server. See the image below.

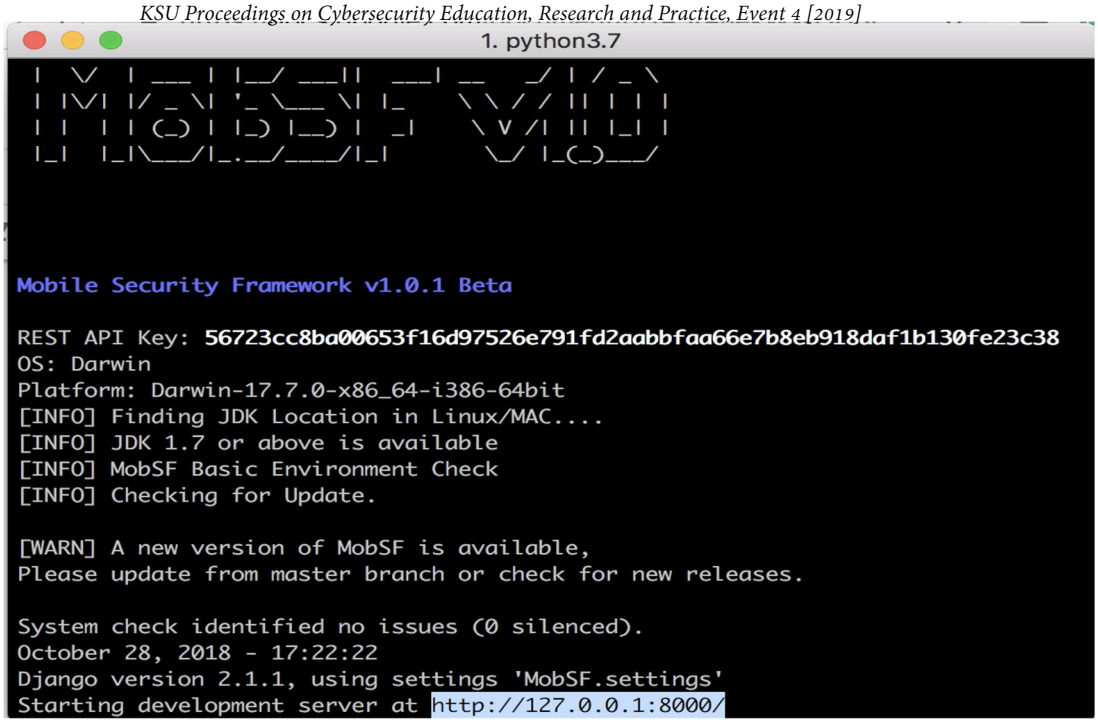

Figure 4. Development server running by MobSF

Figure 4, show the development server running at "http://127.0.0.1:8000/". We have tested couple of APK files using MobSF. A vulnerable APK has been tested by MobSF that contains a SQL injective query in it. The application finds data from the database based on user input. It performs a raw query without sanitizing the user input. Which dumps all the content from the database. Later it tries to send the data to a static number that is hardcoded in the application. Figure 5, shows SMS API permission used in the manifest

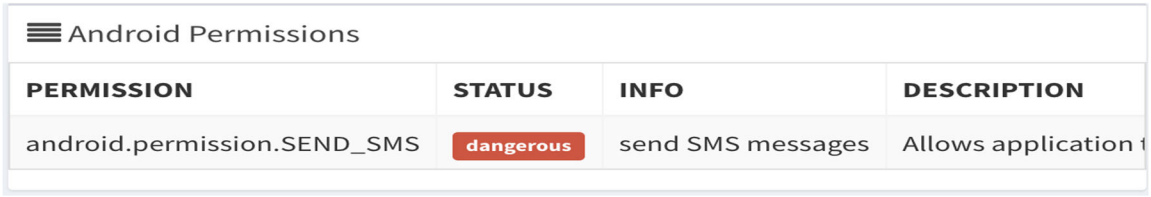

Figure 5. Permission Analysis

MobSF finds the Android API call in an application that also finds the corresponding java file for example, in this application SMS API has been called in the MainActivity.java file. See the image below.

Android API

API	FILES
Inter Process Communication	sqlinjection/sqliexample/sqlinjection0717/MainActivity.java
Send SMS	sqlinjection/sqliexample/sqlinjection0717/MainActivity.java
Java Reflection	android/arch/lifecycle/Lifecycling.java android/arch/lifecycle/ReflectiveGenericLifecycleObserver.java
Crypto	com/h3xstream/findsecbugs/crypto/InsufficientKeySizeBlowfishDetector.java com/h3xstream/findsecbugs/crypto/StaticIvDetector.java
Message Digest	com/h3xstream/findsecbugs/crypto/CustomMessageDigestDetector.java com/h3xstream/findsecbugs/crypto/BadHexadecimalConversionDetector.java
URL Connection supports file,http,https,ftp and jar	com/h3xstream/findsecbugs/injection/custom/CustomInjectionDetector.java
Content Provider	com/h3xstream/findsecbugs/injection/sql/AndroidSqlInjectionDetector.java

Figure 6. API call

Dynamic Analysis by MobSF

Dynamic analysis refers to analyzing the functionality of an application in an isolated device or emulator. In terms of emulator it should be launched on a virtual machine in the host computer that keep our host computer safe from being affected. We did dynamic analysis using MobSF's pre configured Android virtual machine (MobiSF, 2019). We used a sample malware apk which apparently look like a movie player. To start analysis virtual environment has to be created first. See the image below to create a virtual environment in a virtual box.

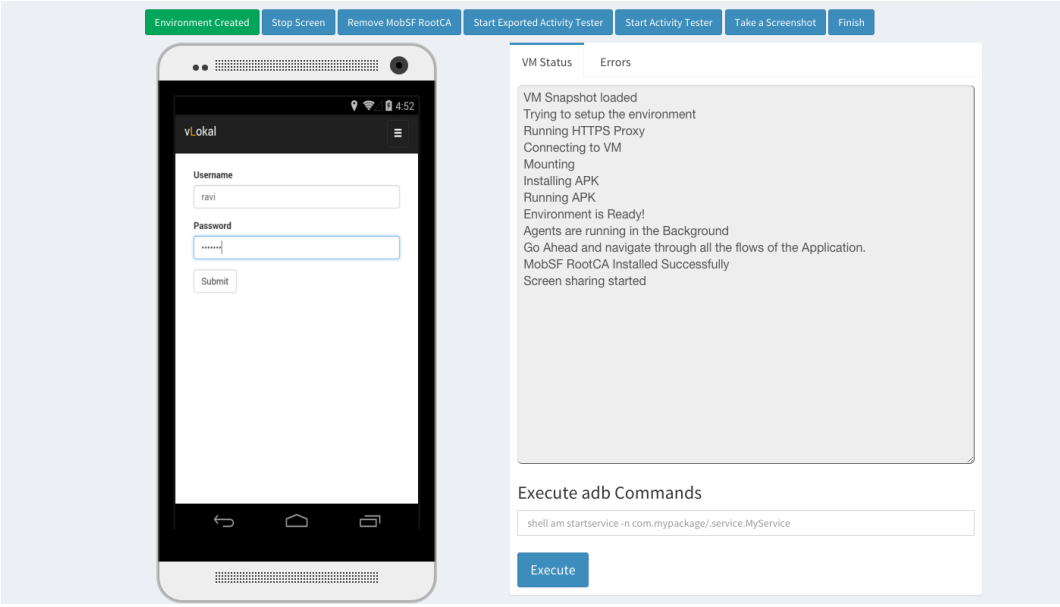

Figure 7. Virtual Environment MobSF

Figure 7, show the virtual environment created to perform the dynamic analysis of an application. Here we found a couple of options for analysis such as Activity Tester, Screenshot of

each screen, Adb command execution field. After successful installation of the apk, we got a POP up window that show a notification for sending a text message to a static number. See Figure 8, below.

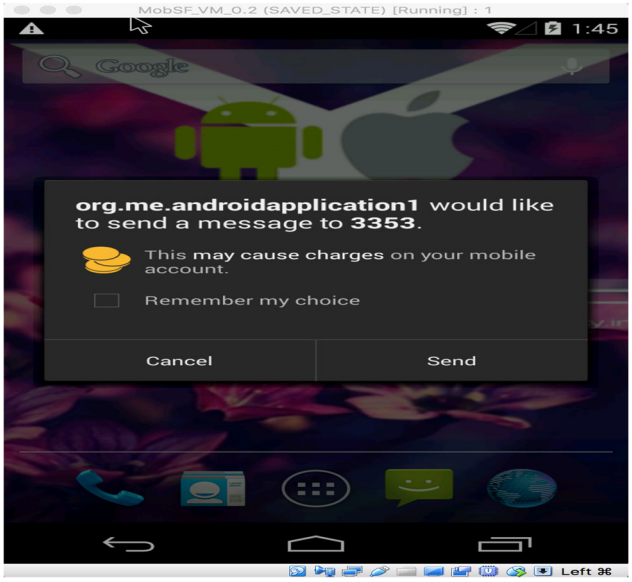

Figure 8. Dynamic analysis of android application in MobSF

MobSF also generates the dynamic analysis report. After finishing the analysis process it produces an analysis report in HTML format. See images below for dynamic analysis report.

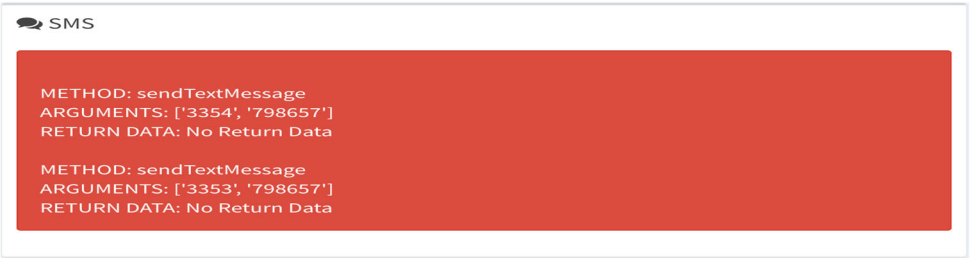

Figure 9. Dynamic analysis report

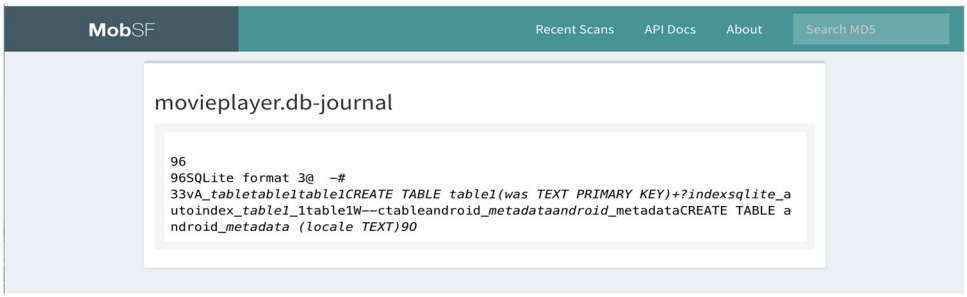

Figure 10. Dynamic analysis report

Figures 9 and 10 show the dynamic reports which clarify that the application sends text message to hard coded numbers. In Figure 10, we also found the database journal that has been created in the device through this application.

Tainted Data Flow Analysis

Android has multiple data sources as well as sinks. Most of the sinks are built in API's for example, SMS API, CALL API etc. An application get takes data through these sources for processing. Soot and Heroes are FlowAnalysis tool built to analyze java application (Soot, 2008; Heroes, 2019). Soot uses worklist algorithm to do static data flow analysis. It has FlowAnalyis classes for fixed-point computation of static data flow analysis. There are classes as well for ForwardAnalysis and BackwardAnalysis (Soot, 2008). Heros (2019) is an extension of soot, it requires its implementation as a class that extends SceneTransformer. FlowDroid is a data flow analysis tool which is built over soot and heros to do static data flow analysis from source to sink of an Android Application (Talukder, Shahriar and Haddad, 2019). FlowDroid does context, flow, field and object-sensitive analyses. To increase recall, it creates a complete model of Android's app lifecycle (Artz, 2017). Figure 11, show a model of source to sink data flow design that used in FlowDroid.

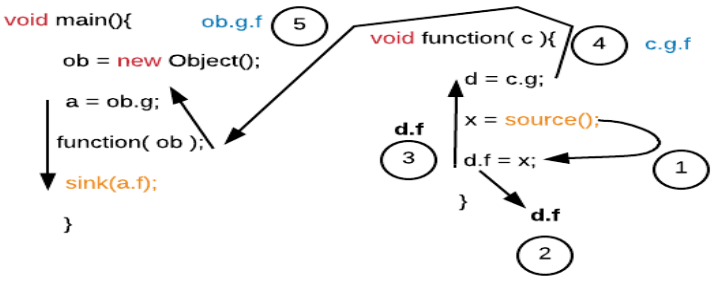

Figure 11. Data flow from source to sink

In Figure 11, in main the function a source encountered. The value of the source is attached with the object that has been passed as a formal parameter to the function. In this case when the object has been initialized in main ultimately receives the source data and finally passed to the sink.

In this project, we performed data flow analysis using FlowDroid. In Android, SQLinjection is possible because android has the feature SQLite Database. We did our analysis on an application that has fetch username and password from the database and sends it to number via SMS API. Following command will start the analysis.

```
java -jar soot-infoflow-cmd/target/soot-infoflow-cmd-jar-with-dependencies.jar \
  -a <APK File> \
```

-p <Android JAR folder> \

-s <SourcesSinks file>

java -jar soot-infoflow-cmd/target/soot-infoflow-cmd-jar-with-dependencies.jar \

-a <APK File> \

-p <Android JAR folder> \

-s <SourcesSinks file> \

-o <Xml file for output>

Here '-a' is the path of the APK file, '-p' is the path of the platforms folder that could be found under sdk folder of the android directory, '-s' is a text file where we can declare the possible sources and sinks for the analysis, '-o' is an additional parameter for the analysis that is used to produce an output file as a XML file.

SourcesAndSinks (1).txt

```
<android.database.Cursor: java.lang.String getString(int)> -> _SOURCE_


<android.telephony.SmsManager: void
sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingInte
nt,android.app.PendingIntent)> android.permission.SEND_SMS -> _SINK_
```

Figure 12. SourcesSinks.txt file

Figure 12, is sample SourceSinks.txt file that has all the possible sources and sinks declared in it. Figure 13 shows only one data leak in this application.

```
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - The sink virtualin
voke $r5.<android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String
,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)>("123456", null, $r1, n
ull, null) in method <sqlinjection.sqliexample.sqlinjection0717.MainActivity: void showResult(
java.lang.String)> was called with values from the following sources:
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - - $r1 = interfacei
nvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(1) in method <sqlinjectio
n.sqliexample.sqlinjection0717.MainActivity: void showResult(java.lang.String)>
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - - $r1 = interfacei
nvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(2) in method <sqlinjectio
n.sqliexample.sqlinjection0717.MainActivity: void showResult(java.lang.String)>
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Data flow solver t
ook 0.338522 seconds. Maximum memory consumption: 104.322368 MB
[main] INFO soot.jimple.infoflow.android.SetupApplication - Found 1 leaks
Arabins-MacBook-Pro:FlowDroid-2.6.1 arabin$
```

Figure 13. Data flow analysis by FlowDroid

Here, we find one sink inside the *onCreate()* method of the MainActivity.java file that is sendTextMessage() method in the SMS manager API. Where possible sources for this sink are getString() methods that has been found in showResult() method in the MainActivity.java class (Talukder et al., 2019).

CONCLUSIONS

This paper presents the analysis report of existing mobile security tools we have in the market. We found that Flowdroid was the best analysis tools for detecting malware in malicious applications. In addition, we have also provided a review of all the state-of-the-art research in both academic and enterprise communities in this report. Based on the results obtained from this empirical study, we found an extremely trivial, but critical flaw in the Android application development environments, i.e., Android Studio, which is the most widely used development environment for building Android applications. It does not have any comprehensive in-built security analysis module. We propose that the community to look into building a security analysis plugin based on open source tools such as Flowdroid to improve the native security testing ability of Android Studio. Most of the Android Application is data driven app now a days. It is high time to develop a plugin for Android Studio that will be capable of detecting the tainted data flow in the application detecting the source and sinks of data.

In continuation of this research we already started our project towards development of a plugin for Android Studio that will be capable of doing tainted data flow analysis during the development phase of the application. Since Android Studio is extremely resource intensive, especially on system Random Access Memory (RAM), the challenge to developers will be making such a plugin lightweight so that it integrates seamlessly with Android Studio without intensively consuming more system resources.

A step in this direction could greatly improve Android Studio as new and immature developers will have the ability to test their Android applications before deploying them on the open Google Store market for the general public to download and take advantage of, thus, possibly reducing the risk of privacy invasion, and data leakage and stealth with the use of malwares and spywares. For the future, we propose the development of a plugin or a native module for Android Studio which allows both experienced and inexperienced application developers to easily test their applications before deploying them on the open Google Play Store application market. This would make the consumers much more immune to malicious attacks based on malwares and spywares. We hope that as a consequence of the results and analysis obtained from our work, both the academic and enterprise communities could come up with innovative ideas, approaches, tools and plugins for the advancement and improvement of the current state of the domain.

Acknowledgement

The work is partially supported by National Science Foundation (Award #1723578); KSU Office of Vice President Research (OVPR 2018-2019) Award, and University System Georgia Affordable Learning Grant (ALG) (#M54, #M87, #422, #429).

References

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Mcdaniel, P. (2013). FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-Aware Taint Analysis for Android Apps. *Proc. of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 259-269.

Arzt, S. (2017). Static Data Flow Analysis for Android Applications, https://bodden.de/pubs/phd-arzt.pdf

Contagio Dump. (2019). http://contagiodump.blogspot.com.

Cuckoo. (2019). Installation - CuckooDroid v1.0 Book. (n.d.). Retrieved from https://cuckoo-droid.readthedocs.io/en/latest/installation/

Detecting Logic Bombs in Android Applications. *Proc. of IEEE Symposium on Security and Privacy*.

Droidbox. (2019). https://github.com/pjlantz/droidbox/tree/master/droidbox4.1.1.

Flowdroid. (2019). https://github.com/secure-software-engineering/FlowDroid

Fratantonio, Y., Bianchi, A., Robertson, W., Kirda, E., Kruegel, C., & Vigna, G. (2016). TriggerScope: Towards

Heroes. (2019). https://github.com/Sable/heros/wiki/Example%3A-Using-Heros-with-Soot

Idan. (2019). Dalvik Monitoring Framework for CuckooDroid, https://github.com/idanr1986/droidmon

Jasmin. (2019). https://github.com/Sable/jasmin

Kong, P., Li, L., Gao, J., Liu, K., Bissyandé, T., and Klein, J. (2019). Automated Testing of Android Apps: A Systematic Literature Review. IEEE Transactions on Reliability, Vol. 68, No. 1, pp. 45-66.

Li, L. (2016). Boosting static analysis of android apps through code instrumentation. *Proceedings of the 38th International Conference on Software Engineering*, pp. 819–822.

MobSF. (2019). MobSF Android x86 4.4.2 VM (v0.3) ova file, https://goo.gl/QxgHZa

Qiu, L. (2018). Analyzing the Analyzers: FlowDroid/IccTA, AmanDroid, and DroidSafe, *Proc. of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, July 2018. Amsterdam, The Netherlands.

Skylot. (2019). https://github.com/skylot/jadx

Soot. (2008). http://www.brics.dk/SootGuide

Rovo. (2019). Xposed Module Repository, Accessed from https://repo.xposed.info/module/de.robv.android.xposed.installer

Spreitzenbarth, M., et al. (2014). Mobile-sandbox: combining static and dynamic analysis with machine learning techniques. *International Journal of Information Security*, 14(2):141–153.

Statcounter. (2018). http://gs.statcounter.com/os-market-share/mobile/worldwide

Talukder, Md., Shahriar, S., Qian, K., Rahman, M., Ahmed, S., Agu, E. (2019). DroidPatrol: A Static Analysis Plugin For Secure Mobile Software Development, Proc. of 43rd IEEE Annual Computer Software and Applications Conference (COMPSAC), pp. 565-569.

Talukder, Md., Shahriar, S., and Haddad, H. (2019). Point-of-Sale Device Attacks and Mitigation Approaches for Cyber-Physical Systems, *Cybersecurity and Privacy in Cyber Physical Systems*, CRC Press. pp. 368-383.