# Security Analysis of Mobile Money Applications on Android

Hesham Darwish
California State Polytechnic University, Pomona
hidarwish@cpp.edu

Mohammad Husain
California State Polytechnic University, Pomona
mihusain@cpp.edu

## ABSTRACT
Mobile Money Applications are thriving mainly due to the ease and convenience it brings to people, where it offers transferring money between people's bank accounts/cards with a few taps on a smartphone either in the form of Mobile Banking or Mobile Payment Services. However, a key challenge with gaining user adoption of mobile banking and payments is the customer's lack of confidence in security of the services, and that makes a lot of sense because whenever people grants a service access to their debit/credit cards or bank accounts that automatically opens the door for identity thefts, fraudulent transactions and stolen money. Add to that, the fact that already people and developers are not giving much attention to the security aspect of the applications. This paper consists of two parts, an intensive security analysis on a selection of different mobile banking and mobile payment applications on Android platform where 80% of the selected applications were found not following the best security measures, and also a thorough step-step Android security testing guide in the form of this paper to ease the process of security testing any android application to be used by developers, ethical hackers, and anyone interested in testing the security of any application.

## CCS Concepts
**• Security and Privacy→ Systems security → Operating systems security → Mobile platform security.**

## Keywords
Mobile Security; Mobile Applications Security; Android; Android Security; Android Applications; Mobile Money Applications; Security of Mobile Money Applications.

## 1. INTRODUCTION
Mobile Money Applications has been a hot topic in the last few years and is expected to continue like that as all the tech is moving towards mobile. However, unfortunately, In the face of accelerating user demand, businesses aren't validating that their apps are safe and secure and they are more dragged by are the speed-to-market pressure as well as the need to maintain a high level of user experience. According to a Security Intelligence study shown in [1] that was done on 400 well known organizations (40% Fortune 500 companies), 40% of them aren't even scanning their code for security vulnerabilities. On top of all these applications are the Mobile money applications who are basically dealing with high-level private financial and confidential information, where the security has to be top notch as any vulnerabilities or threats can't be accepted as this is were the ultimate security is needed. So by targeting the mobile money applications, this paper is indirectly targeting all the other mobile apps. Security in general has been a growing concern lately especially with the evolution of mobile, cloud, and Internet of things and what makes it more challenging is the almost daily vulnerabilities found in operating systems, applications, and websites. In this paper, a detailed security analysis was done on 25+ mobile money applications by using various tools and frameworks which made it easier to do the second part of the paper which is a thorough step-by-step guide for Android security testing that can be used on any android application not necessarily mobile money applications. This paper focuses on Mobile Money Applications for multiple reasons that will be addressed in this chapter, and Android OS was used as our target platform in this paper due to it being the most used OS in the world and the one with more vulnerability history as of July 2016. This paper categorizes mobile money applications into mobile payment services and mobile banking. And categorizes the security analysis into static and dynamic vulnerability analysis.

### 1.1 Why Android?
According to the Q2 2015 statistics done by IDC, Android is dominating the smartphone market with a share of 82% as of December 2015 while iOS comes second with a share of 13.9% [2]. And that on its own is an enough reason why this paper is performing the analysis on Android but it's not the only reason. Android ecosystem has become massively fragmented in the recent years with uprising number of manufacturers, each having its own Android OS flavor, which means each having potential different vulnerabilities on top of any core Android Issues which is the complete opposite in the iOS side where Apple is making sure to have a more compact closed ecosystem. However, iOS is still suffering from mobile security issues, the android suffering is much deeper. Moreover, Android being an open source based on Linux makes things much easier to work with, so extracting the source code and searching the files and even having insecure code in apps is far easier and common in Android.

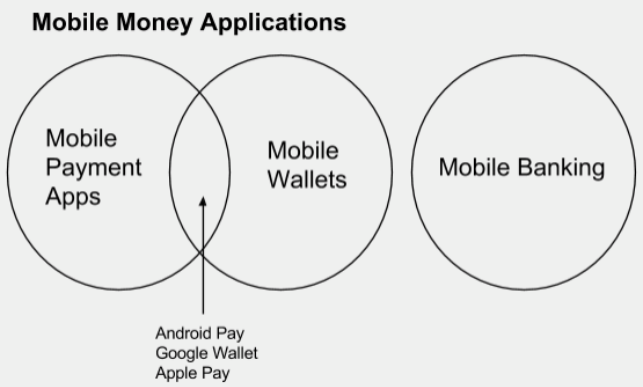


**Figure 1. Mobile Money Applications**

### 1.2 Why Mobile Money Applications?
As shown in figure 1, Mobile Money Applications is the big umbrella that contains mobile payment applications, mobile wallets, mobile banking applications. And there are some

applications that lay between the mobile payment and mobile wallets like Android Pay, Google Wallet and Apple Pay. The mobile money applications in general have became ubiquitous and essence, and it is on the rise due to the convenience and ease it brings to people, as with one press on their phone; people can pay their bills, transfer money to a friend, transfer money to a bank account or even buy groceries and do shopping. It will probably come a time soon where most people won't carry wallets, and do all their transactions through their phone. Currently, providing payments through apps has been dramatically adopted by a lot of people especially younger generations. Mobile payment services accounted for $8.7 billion worth of U.S. transactions last year, and expected to grow by 210% in 2016 according to [3]. The downside of all that is that people are not thinking about the other half of the cup, where they are actually putting themselves in risk for identity theft, fraudulent transactions and stolen money if these apps are not secure enough to hold their data and to perform these financial transactions. And as mentioned in [4] by Jalaluddin Khan et.al Android users are not aware that their smartphones are also as vulnerable as any computer, and I'm sure that it goes the same for iOS users.

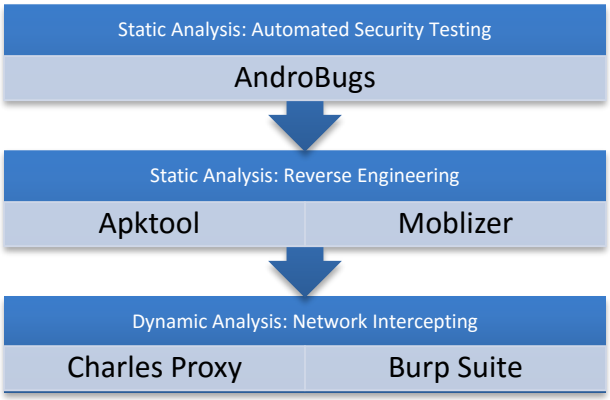


**Figure 2. Analysis Process**

## 1.3 Analysis Process

Figure 2 shows the analysis process created and followed in this paper, the process started by the static part; beginning with the automated security testing which produces an overview and a bigger picture of the potential vulnerabilities. Then the next step is to validate some of these potential vulnerabilities and identify any other vulnerabilities found by reverse engineering which was done by the Apktool that converts the application to smali byte code which is a human readable byte code and produces the android manifest xml file which has all the permissions granted by the application. Then the Moblizer framework was used which searches for vulnerabilities in the extracted source code. After that, the dynamic analysis phase take place to validate how secure is the app while it is running and how is it protected from potential MITM (man in the middle) attacks, and to monitor how the app communicates with the server. All the results from the static and dynamic testing are then analyzed and collected together to produce the final full security testing result for all of the selected apps.

## 1.4 Device Used

The device used in the analysis is a Nexus 7 tablet that has android version 5.1, and it's important to note that Nexus devices are probably the most secure devices on Android with less vulnerabilities on the OS level than any of the other Android devices, as the android OS is raw and not customized like in most of the other manufacturers.

## 1.5 Selected Applications

The criteria of the choosing the applications as following:

1. Choosing some market leading mobile banking and mobile payment apps.
2. Choosing apps with low and high download rates.
3. Choosing some old apps in the market that haven't been updated recently that we expect to find some vulnerabilities in.

After digging into the Android Play Store, the following list of 26 apps were chosen to be included in the analysis paper. Table 1 shows all the selected apps with the version used in the analysis, the date where that version was added, the download count, and the sector that the app fits into either banking, payment or money wallet.

| App Name | Version | Download Count | Date Added |
|---|---|---|---|
| Money Fellows | 1.1.2 | 100+ | 07/10/2016 |
| Venmo | 6.31.1 | 1,000,000+ | 07/12/2016 |
| Square Cash | 2.12.3 | 1,000,000+ | 07/13/2016 |
| Transfer Wise | 2.7.6 | 500,000+ | 07/10/2016 |
| PayPal | 6.4.2 | 10,000,000+ | 06/30/2016 |
| Dunia Mobile | 1.1.0 | 10,000+ | 12/06/2015 |
| Pocket Moni | 2.3 | 5,000+ | 09/02/2015 |
| Phone Cash | 1.2.3 | 1000+ | 07/12/2016 |
| Xoom | 4.2 | 1,000,000+ | 05/12/2016 |
| Mobile Money | 1.3.0 | 10,000+ | 05/09/2016 |
| Easy Pay | 3.2.4 | 50,000+ | 10/13/2015 |
| EPE Mobile Pay | 1.1.2 | 10,000+ | 05/15/2016 |
| Wells Fargo | 5.1.0.74 | 10,000,000+ | 06/13/2016 |
| CIB Smart Wallet | 1.0.52 | 10,000+ | 03/06/2016 |
| QNB | 2.3 | 50,000+ | 05/05/2016 |
| HSBC | 1.5.14.0 | 1,000,000+ | 07/07/2016 |
| Chase | 3.29 | 10,000,000+ | 07/06/2016 |
| SAIB | 2.9.2 | 50,000+ | 04/27/2016 |
| Arabi Bank | 2.7 | 100,000+ | 06/30/2016 |
| AGB | 1.1 | 10,000+ | 11/12/2014 |
| AlAhli Bank | 2.4.0 | 500,000+ | 03/10/2016 |
| eBLOM Bank | 1.0.19 | 10,000+ | 01/18/2016 |
| ENBD | 1.0.4 | 10,000+ | 02/10/2016 |
| Ma7fazty | 1.0.61 | 500+ | 06/22/2016 |
| Google Wallet | 15.0 | 1,000,000+ | 06/08/2016 |

| | | | |
|---|---|---|---|
| Android Pay | 1.4 | 10,000,000+ | 06/11/2016 |

**Table 1. Selected Applications**

# 2. RELATED WORK

In this section, some related work in the android applications security field are presented that this paper was impacted by them.

Diana Gabriela et.al [6] highlighted how smartphones are used in daily activities like mobile banking and mobile payment services and then went on to describe some mobile threats and attack vectors in android, outlining the variety of sources of intentional threats, and different motivations. Then after that the paper digs deep into the structure of an android application and explains in detail the different components of an apk. Moreover, the authors presented an analysis of one sample of a remote access Trojan in a form of an application, and they proved how a simple application can collect private information from a phone and access passwords, etc. [6]. This paper main motive and idea was to show the important of permissions in apps and how some applications will request all the permission although it's not needed, and how it can affect the security of the information on all devices including other applications on the phone. Adrienne Porter Felt et.al in [7] went deeper to analyze the issue of android permissions, as they built a static testing tool to detect over privileged applications by determining the set of APIs the application is using then matching the API calls to permissions [7]. They applied the tool that they created on a set of 940 applications and they found out that one-third are over privileged.

Erika Chin et.al. [8] saw the importance of securing the inter-application communication so in this paper they talk about the intent-based attack surfaces, and how sending an intent to the wrong application can leak user information. In this paper they created a tool that detects application communication vulnerabilities and then they tested on a set of 20 applications, 60% of these applications appeared to be vulnerable as they found 34 vulnerabilities in 12 of these applications [8].

William Enck et. al [9] is one of the few papers found that has the same direction like this paper. The authors in [9] had a target of better understanding smartphone application security so they studied 1100 free android applications. They designed and executed a horizontal study of smartphone applications based on a static analysis of 21 million lines or code [9]. They retrieved the 21 m line of code by implementing a Dalvik decompilier "ded" which performs DVM to JVM bytecode retargeting, and translating class and method structures [9]. The authors analyzed the 21 million lines of code retrieved using automated tests as well as manual inspection. The results of this study showed a wide misuse of privacy sensitive information, like IMEI, IMSI that were used in account numbers or even in session authentication token or as device fingerprints [9]. They also found out that ads and analytic network libraries are available in 51% of the applications included in the study. The study found out some android-specific vulnerabilities which include leaking information to logs, leaking information via IPC, unprotected broadcast receivers, intent injection attacks, delegating control where apps unsafely delegate actions to other applications (found in 116 applications, 10.5%), null checks on IPC input, SD card use which means that the app can read or write any other application's data on the SD card (found in 251 applications, 22.8%) [9]. The unique thing about this paper that it did not only showed the existence of dangerous functionality but it showed how it occurred within the context of the application. They indicated that they had some challenges because of the implementation of code obfuscation by some of the apps that made them unable to test these applications.

In all the prior presented work, a lot of findings were found about mobile applications and android applications in general, random applications were tested and their security measures were analyzed and documented. But this paper is most similar to the work done by Bradley Reaves et. al. [10], where that paper was the first detailed comprehensive security analysis of mobile money applications or branchless banking applications as called in the paper [10]. The focus of this paper was to analyze the security of mobile applications in the developing world so after running some automated tests, they picked 7 android applications from Brazil, India, Indonesia, Thailand, and the Philippines [10]. The authors applied reverse engineering on the applications in order to have a deep insight into the application behavior and client/server protocols. The results they got in their research was one of the main motives of this paper, as they found 28 significant vulnerabilities across the seven applications [10]. They also found out that the automated analysis tool that they used "Mallodroid" failed to detect accurate results compared to their manual inspection. Moreover, The work of Bradley Reaves et. al. also investigated the security guarantees and the severe consequences of smart phone application compromise in mobile money applications, so they went through all the terms of service "ToS" of all the 7 investigated applications and found out that most apps hold the customer solely responsible for most forms of fraudulent activity [10].

This paper focuses on Mobile Money Applications can be considered an extension of the work done by Bradley Reaves et al. [10]. But it differs a lot in the whole analysis process starting from the size of the targeted applications in the analysis growing from 7 applications to 26 applications, this paper is more concerned about the most used mobile money applications in the market. Apktool is the only tool used in both in papers while this paper uses an automated static analysis tool that was built on top of the core of Mallodroid but yet it is much more advanced and it gives more accurate results. This paper includes a dynamic analysis phase that wasn't included in most of the prior research where all the selected applications where tested while running on MITM attacks, and that gave more accurate results about the SSL/TLS implementation of the applications. Moreover, this paper includes a step by step easy to follow guide to test any android application.

# 3. AUTOMATED SECURITY TESTING

This part will present the tools, process and results of the automated security testing that was done on the selected applications. The automated security testing is a part of the static vulnerability analysis covered throughout the paper, and it is importance comes from that it gives an overview on where to look for vulnerabilities in the app. AndroBugs Framework was used as the automated security testing tool, and it was chosen despite that it doesn't have a 100% accuracy because of how it works, as it basically searches the code for the known common problems by following a sequence of search on compromised methods or functions that it is stated that it's insecure to use in the android developer documentation. This chapter will start with giving a brief on AndroBugs then showing the process of using it, then presenting the results of the automated security testing.

## 3.1 Automated Security Testing Tool

AndroBugs framework is an open source android application security vulnerability scanner, it is developed by Yu-Cheng Lin and was released in November 2015[11]. What makes AndroBugs

special compared with other vulnerability scanner tools is that AndroBugs tries to emulate the operation of an app, and considers the attack vectors through which those weaknesses would be exploited, instead of just scanning the code for weak spots. AndroBugs is written fully in Python and it's considered a static analysis tool that checks for a number of known common vulnerabilities in the android apps, it also checks if the code is missing best practices and checks dangerous shell commands (su) [11]. The AndroBugs tool provides a severity level for each issue found starting from critical, warning, notice and info [11].

## 3.2 Automated Security Testing Process

One of the main advantages of AndroBugs is that it doesn't need any installation in Unix, you just download the folder and start using it if you have python 2.7 or later installed. It's probably the easiest tool in the market that you can use for android security testing. The first and only step needed before starting to use it is to have the .apk of the application, and this can be retrieved easily by downloading an app called "Apk Extractor" on your android device and then transferring the .apk to your device through USB or email. Then placing it inside the same folder of AndroBugs. In order to run the AndroBugs Framework all you need to do is:

*python androbugs.py -f [APK file]*

The AndroBugs produces a full report with the application name in a folder created called *Reports.* Below are some screenshots of how the report looks like and what kind of vulnerabilities can find. AndroBugs starts by giving a brief on the application that is being tested; name, version, etc. Then it goes on to find vulnerabilities and give a brief on them. One of the best features about AndroBugs is that it gives you all the URLs that is found not being under SSL, which might be a vulnerability if any of these URLs have any kind of sensitive information.

## 3.3 Automated Security Testing Results

All the selected applications have been security tested by AndroBugs tool to get an initial overview about where to look for vulnerabilities, while it appeared that the results given by AndroBugs gives in most cases a good idea about what how the application is built and where you can have potential vulnerabilities. It was really helpful as a static vulnerability analysis tool and in order to know where to look when going to the next static vulnerability analysis tool; Reverse Engineering.

The Vulnerabilities found in all apps were the following:

**Application Sandbox:** It's mainly because AndroBugs found that the application being tested is probably using "MODE_WO-RLD_READABLE"or "MODE_WORLD_WRITEABLE", which makes the application vulnerable under M2 – Insecure data storage.
**Runtime Command:** This is because AndroBugs found in the code a critical function "Runtime.getRuntime().exec("…")".In this function a user can provide an input that will cause a shell to run and then change commands inside it.
**Fragment Vulnerability:** This is because AndroBugs found in the code that a 'Fragment' or 'Fragment for ActionbarSherlock' used which has a severe vulnerability on devices with android version prior to 4.4. The use of this application on an old android device will be vulnerable for any attacker to execute some code that can break the Android Sandbox which means accessing sensitive information that shouldn't be accessible by the application itself.
**SSL Certificate Verification:** This means that this application doesn't check the validation of SSL Certificate, that means it allows self-signed, expired or mismatch CN certificates for SSL. This is definitely a critical vulnerability because it allows attackers to do MITM attacks.
**SSL Implementation**: This means that this application allows self-defined '*HOSTNAME VERIFIER'* to accept all common names. This means that any attacker with a valid certificate will be able to perform MITM attacks.
**Implicit Service:** This means that this application is using an implicit intent to start a service, which is really risky because the responding service can't be identified and the user can't see which service.
**WebView Vulnerability:** This means that AndroBugs found in the code of the application the method "addJavaScriptInterface" which is a vulnerability that can be used to allow JavaScript to control the application in devices running Android versions prior to 4.2.
**Android Manifest:** This indicates that AndroBugs found that this app has very high privileges, and that AndroBugs found that the android permission "Mount_Unmount_FileSystems" is used by this application which is not justified as this permission allows mounting and un mounting file systems for removable storage and it is indicated in the android developer's website that it's not for the use of third party apps.
**KeyStore Protection:** This means that AndroBugs found that this application is not protecting it's KeyStore properly as it seems that it is using byte array or a hard-coded certificate info to do SSL pinning.
During the process of running all selected applications on AndroBugs, it was found that the applications owned by Google (Android Pay and Google Wallet) had an additional layer of protection that doesn't exist in any other application, where the .apk requires a string argument of length 4 to be unzipped, which can be brute forced for sure but it makes it harder to reverse engineer this application. Table 2, shows a full summary of the vulnerabilities found in all the selected applications after performing the security testing using AndroBugs.

# 4. REVERSE ENGINEERING

Reverse Engineering is defined as the process of extracting knowledge or design information from anything man-made. The idea behind it is ancient, it has been known for ages that disassembling anything will make you understand it more, analyze it, and you could even tweak it to perform a different task. Reverse Engineering is used heavily in the computer security industry for virus and malware analysis, vulnerability analysis, binary code auditing, and exploit development. Reverse engineering was implemented in this paper as a second part of the static vulnerability analysis after automated security tested, the sequence followed to reverse engineer an application is shown in figure 3.

Table 2. Automated Security Testing Results

| App Name | Sandbox | Fragment | Runtime Cmd | SSL Verif | SSL Implemen | Implicit Serv | SSLurls | WebView | KeyStore | AndroidManife |
|---|---|---|---|---|---|---|---|---|---|---|
| MoneyFellow | Y | N | N | N | N | N | Y | Y | N | N |
| Venmo | N | N | N | Y | N | N | Y | Y | Y | Y |
| SquareCash | Y | N | N | N | N | N | Y | N | N | N |
| TransferWise | Y | N | Y | Y | Y | Y | Y | Y | N | N |
| PayPal | Y | N | N | N | Y | N | Y | Y | Y | N |
| Dunia | N | N | N | Y | N | N | Y | N | N | N |
| PhoneCash | N | Y | N | N | Y | N | Y | N | N | N |
| Xoom | Y | Y | N | N | N | N | Y | N | N | N |
| MobileMoney | Y | N | N | N | N | N | Y | N | N | N |
| EasyPay | N | Y | N | Y | Y | N | Y | Y | Y | N |
| EPE | Y | Y | N | Y | N | N | Y | Y | Y | Y |
| WellsFargo | Y | N | N | Y | Y | N | Y | Y | Y | N |
| CIB | N | Y | N | Y | Y | N | Y | N | Y | N |
| QNB | Y | N | N | Y | Y | N | Y | Y | Y | N |
| HSBC | Y | N | Y | Y | Y | N | Y | Y | Y | Y |
| Chase | N | N | N | N | Y | N | Y | N | N | N |
| SAIB | N | N | N | N | N | N | Y | N | N | N |
| Arabi | N | N | N | N | N | N | N | N | N | N |
| AGB | N | N | N | Y | Y | N | Y | N | Y | N |
| AlAhli | N | N | N | N | N | N | Y | Y | N | N |
| eBLOM | Y | N | N | Y | Y | N | Y | Y | N | N |
| ENBD | N | N | Y | Y | Y | N | Y | N | Y | N |
| Ma7fazty | N | Y | N | N | Y | N | Y | N | N | N |
| GWallet | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| AndroidPay | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |

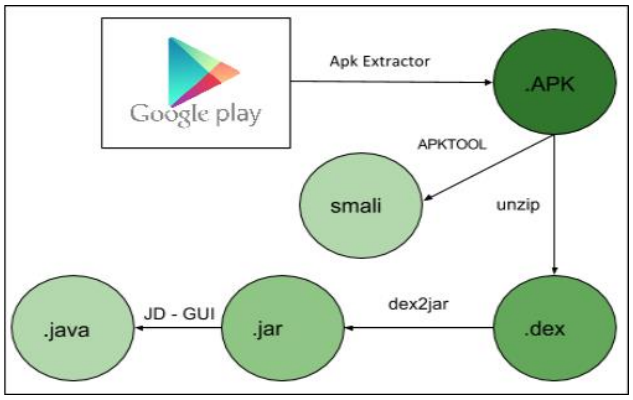


**Figure 3. Reverse Engineering Process**

While reverse engineering can be done on its own as a static vulnerability analysis method, I used reverse engineering in this paper after performing the automated security test so I would have an idea on where I should be looking and for what. In general, reverse engineering is used to see how the developers built that specific application and how they dealt with the crucial security tasks and requirements. These are some examples on what we should be looking for when using reverse engineering in security tests on Android:

Database Connection, db name, or db password, any hardcoded usernames or passwords that can be used to access the database or the application, the APIs used by the application to see if any of them are compromise, or the API key, and to search on a known vulnerable method if it's used in the application.

This part will address the tools used in Reverse Engineering, and then the process followed to find vulnerabilities in the selected applications after retrieving the source code, and finally will go over the results obtained from the reverse engineering and comparing them with the results achieved by the automated security testing.

## 4.1 Reverse Engineering Tools

### *4.1.1 Apktool*

It is a static security analysis tool for reverse engineering 3rd party, closed, binary Android apps. It is written in java and It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the paper like file structure and automation of some repetitive tasks. The powerful thing about Apktool that it can not only used for reverse engineering, it can be used in re-engineering and this means that an application can be reverse engineered and the code can be edited in a malicious way and then repacked again in a new .apk that can be placed on third party application stores and now everyone downloads it will be in danger [12].

### *4.1.2 Moblizer*

It is a static security analysis framework that is built on top of Apktool, it's simply a 70 lines python script that just can help in saving time and effort. It basically searches in the code produced by the apktool for certain words that can indicate a database connection, db-name, password, FTP, API, etc. I personally edited this framework by adding some keywords that gives it a better functionality, and was helpful in my analysis.

### *4.1.3 Dex2jar*

Dex2jar is a tool that converts Dalvik bytecode (DEX) to java bytecode (JAR), so it takes as an input the .apk and produces a .jar file that can de decompiled by any java decompiler.

### *4.1.4 JD-GUI*

This is the java decompiler used to convert the java bytecode (JAR) into the readable java source.

## 4.2 Reverse Engineering Process

The process created and followed during the reverse engineering part of the paper was initially solely built on Apktool which basically takes an apk file and delivers the source code in Smali which is an easy to understand language.

The apktool is a powerful reverse engineering tool that needs no kind of installation, after downloading the apktool jar file, and making sure that java is installed on the computer; the following command is used to retrieve the source folders and code.

*java -jar apktool.jar d TransferWise.apk*

In the initial process, I performed manually checks on the code of each and every application tested which is definitely time consuming so I thought about building a script to search for keywords in the smali code but then I came across the Mobilizer framework.

The moblizer framework requires any python 3.x version, and it's used by downloading the Moblizer.py to the working directory along with the apktool and the apk file that is needed to be tested. The idea of mobiizer is that it searches in all the files for any sensitive keyword such email, IP, username, db-name, password, etc. And it also provides a Manifest permission details at the end of each output

*Python3 /path/to/moblizer.py /path/to/your. Apk*

I edited the Moblizer script by adding keywords to its search list and making it executable in the way indicated above. The output is usually about 2 pages long but it saves a lot of time since you don't need to open a lot of files to find the potential vulnerabilities.

The process I followed was to double check all the results that was achieved by AndroBugs by adding all the keywords needed to the Mobilizer list.

Dex2jar and JD-GUI were also used to extract the java source code from the apps. It appeared that some of the apps have a layer of protection on their code (code obfuscation) which makes the code real hard to understand, it makes it unreadable, and that's why I used the apktool to reverse engineer the selected applications to smali which is a readable byte code and that was done by using the advantage that most applications do not do byte code obfuscation.

## 4.3 Reverse Engineering Results

All the selected applications have been reverse engineered by Apktool and Moblizer and the full source code was extracted. All the results got from AndroBugs were double checked in this second part of static vulnerability analysis. Every application was first reverse engineered by Apktool and the code was checked in all the places identified by AndroBugs.

Table 3 Shows the reverse engineering results that was achieved by manually checking the source code and by using Moblizer framework. The SSL verification, URLs not under SSL, and SSL checking wasn't checked in Reverse Engineering as that will be double checked in the dynamic analysis part.

These results indicated that the results achieved from AndroBugs were more than 70% correct, which is reasonably good since it's just a quick automated test. Any vulnerability found by AndroBugs but couldn't be found in the code then it was deleted from the vulnerability.

| App Name | Sandbox | Fragment | Runtime Cmd | Implicit Service | WebView | KeyStore | Android Manifes |
|---|---|---|---|---|---|---|---|
| MoneyFellow | Y | N | N | N | Y | N | N |
| Venmo | N | N | N | N | N | N | Y |
| SquareCash | N | N | N | N | N | N | N |
| TransferWise | N | N | Y | N | Y | N | N |
| PayPal | N | N | N | N | N | Y | N |
| Dunia | Y | N | N | N | N | N | N |
| PhoneCash | N | Y | N | N | N | N | N |
| Xoom | Y | Y | N | N | N | N | N |
| MobileMoney | N | N | N | N | N | N | N |
| EasyPay | N | Y | N | N | Y | Y | N |
| EPE | Y | Y | N | N | N | Y | Y |
| WellsFargo | N | N | N | N | N | N | N |
| CIB | N | N | N | N | N | N | N |
| QNB | N | N | N | N | Y | N | N |
| HSBC | Y | N | Y | N | Y | Y | Y |
| Chase | N | N | N | N | N | N | N |
| SAIB | N | N | N | N | N | N | N |
| Arabi | N | N | N | N | N | N | N |
| AGB | Y | N | N | N | N | Y | N |
| AlAhli | N | N | N | N | Y | N | N |
| eBLOM | N | N | N | N | Y | N | N |
| ENBD | N | N | Y | N | N | Y | N |
| Ma7fazty | N | N | N | N | N | N | N |

**Table 3. Reverse Engineering Results**

# 5. DYNAMIC ANALYSIS: NETWORK INTERCEPTING

Network Intercepting is the dynamic vulnerability analysis part in this paper and I believe that it's crucial for any application to undergo a dynamic vulnerability analysis as it differentiates between a threat and a vulnerability. Being able to intercept the network by having a self signed certificate or an expired certificate is a big vulnerability that the static analysis showed that it is really common in the selected applications. In this chapter, the tools used in intercepting the network are addressed, then the process that was followed to perform a MITM attack and observe vulnerabilities in the applications, then the results obtained from the dynamic analysis.

## 5.1 Dynamic Analysis Tools

### *5.1.1 Charles Proxy*

Charles is an HTTP proxy / HTTP monitor / Reverse Proxy that enables a developer to view all of the HTTP and SSL/ HTTPS traffic between the device and the internet [13]. This includes requests, responses and the HTTP headers (which contain the cookies and caching information). I used Charles Proxy (Trial Mac version), to perform MITM attack on the selected applications mainly in order to test the SSL certificate validation, SSL certificate checking, and that all important URLs are SSL protected [13].

### *5.1.2 Burp Suite*

Burp Suite is an integrated platform for performing security testing on web and mobile applications. It is a rich platform that has various tools that can work together for a complete testing process [14]. Burp Suite contains a lot of key components but those 2 components are the only ones that was used in this dynamic analysis.

1. An Intercepting Proxy which gives the flexibility of inspecting and modifying traffic between the target application and its server.
2. An Intruder tool for performing powerful customized attacks to find and exploit vulnerabilities.

## 5.2 Dynamic Analysis Process

The process created and followed for the dynamic vulnerability analysis part in the paper was initially solely built on network intercepting which is done by the use of the tool, Charles Proxy. But after that Burp Suite was added to the analysis due to it's interesting features that can help in producing more vulnerabilities.

Charles proxy is an easy to download and install software but a few steps that should be done on the android device are needed before being able to intercept any packages.

**On the Android Device:**

1. Download the Charles proxy certificate and install it on the device.
2. Open the device settings and connect to the same WIFI that the computer is connected to.
3. Press on the connected network and modify the host to the IP address of the computer and the port as the proxy port set in Charles proxy (default: 8080).

Now the setup is done and intercepting packets can be start. The process starts by opening the target application and logging in and trying different functionalities on the application, specially those have sensitive information. And at the same time intercepting the packets on Charles proxy to identify if there are any vulnerabilities. Basically, you shouldn't be able to see any sensitive information as it should be encrypted during the transfer (SSL/TLS).

## 5.3 Dynamic Analysis Results

The results of this part was shocking as only 5 apps out of the total of 26 apps implemented SSL pinning which prevents the kind of attack we were trying to do, the man in the middle attack (MITM) as it rejects the certificate of the tools used (Charles proxy CA and Burp CA) as they aren't identified as the right certificates.

Table 4 shows the overall results achieved from the dynamic security testing of all the selected applications. The results indicate that only 5 applications were performing proper SSL pinning, and 17 apps from the selected 26 apps had SSL vulnerabilities which sounds shocking as MITM attack can be performed easily by faking a WIFI host and forcing users to install the certificate as part of the connection process, and by that any access to these applications will mean that the important financial data of the user can be compromised.

# 6. CONCLUSION

Mobile money applications are really convenient and they are the future of banking and transactions but the problem is that most of the mobile applications in general are not secure and that goes for the money applications too. Companies and developers has to take a break from adding features and following user demand and invest some time in securing their applications. And as mentioned in this paper, securing the application is actually not an impossible task although nothing will be 100% secure but at least developers can make it way harder for the hackers to attack the applications. From the results of the security analysis done, we found out that only 4 applications were secure against all the attacks that was part of the analysis, and two of these apps are the google applications (Android Pay, Google Wallet) which means that by following the android developer guidelines and keeping your code up to date, you can save the private information of a person or prevent an identity theft or a theft in general.

| Application Name | SSL Verifi Vuln | SSL Implem Vuln | SSL urls | Web view | SSL Pinning |
|---|---|---|---|---|---|
| Money Fellows | Yes | Yes | No | Yes | No |
| Venmo | Yes | Yes | No | No | No |
| SquareCash | No | No | No | No | Yes |
| TransferWise | Yes | Yes | No | No | No |
| PayPal | Yes | Yes | No | No | No |
| Dunia Mobile | Yes | Yes | Yes | Yes | No |
| Phone Cash | Yes | Yes | No | No | No |
| Xoom | Yes | Yes | No | Yes | No |
| Mobile Money | No | No | No | No | No |
| EasyPay | Yes | Yes | No | No | No |
| EPE MobilePay | Yes | Yes | No | Yes | No |
| Wells Fargo | No | No | No | No | No |
| CIB Smart Wallet | Yes | Yes | No | No | No |
| QNB | Yes | Yes | No | No | No |
| HSBC | Yes | Yes | No | No | No |
| Chase | No | No | No | No | Yes |
| SAIB | No | No | Yes | No | No |
| Arabi Bank | Yes | Yes | No | No | No |
| AGB | Yes | Yes | Yes | No | No |
| AlAhli Bank | No | No | No | No | No |
| eBLOM Bank | Yes | Yes | No | Yes | Yes |
| ENBD | Yes | Yes | No | No | No |
| Ma7fazty | Yes | Yes | No | No | No |
| Google Wallet | No | No | No | No | Yes |
| Android Pay | No | No | No | No | Yes |

**Table 4. Dynamic Analysis Results**

We believe that a lot of the vulnerabilities that was found during this paper are critical. Utilizing the moblizer framework is something also to be considered as this will help in finding more vulnerabilities through the reverse engineering process. Also more work can be done to test and analyze more apps outside the sector of mobile money as other sectors applications will probably have less security measures than most of the mobile money applications. With vulnerabilities arising everyday, updating the process as well as updating the keywords for the moblizer framework will be crucial to make sure that the vulnerabilities and threats are found through following the process. Moreover, iOS devices make up to 20% of the market share of the mobile devices and it has a higher percentage of devices accessing these mobile money applications, so extending this paper by building a similar easy to follow process to test iOS applications will be really helpful for businesses as to test their iOS applications as well.

## 7. REFERENCES


[1] IBM Security (2015, March 19). IBM Sponsored Study Finds Mobile App Developers Not Investing in Security. Retrieved from https://www-03.ibm.com/press/us/en/pressrelease/46360.wss

[2] IDC (2015, August 1). Smartphone OS Market Share, 2015 Q2. Retrieved from http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[3] EMarketer (2015, October 26). Mobile Payments Will Triple in the US in 2016. Retrieved from http://www.emarketer.com/Article/Mobile-Payments-Will-Triple-US-2016/1013147

[4] Khan, J., Abbas, H., & Al-Muhtadi, J. (2015). Survey on Mobile User's Data Privacy Threats and Defense Mechanisms. *Procedia Computer Science*, *56*, 376-383.

[5] OWASP Mobile Security Paper. (2015). Top 10 mobile risks. Retrieved from https://www.owasp.org/index.php/OWASP_Mobile_Security_Paper# Top_Ten_Mobile_Risks.

[6] Benítez-Mejía, D. G. N., Sánchez-Pérez, G., & Toscano-Medina, L. K. (2016, July). Android applications and security breach. In *2016 Third International Conference on Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC)* (pp. 164-169). IEEE.

[7] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011, October). Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 627-638). ACM.

[8] Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011, June). Analyzing inter-application communication in Android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (pp. 239-252). ACM.

[9] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011, August). A Study of Android Application Security. In *USENIX security symposium* (Vol. 2, p. 2).

[10] Reaves, B., Scaife, N., Bates, A., Traynor, P., & Butler, K. R. (2015). Mo(bile) money, mo(bile) problems: analysis of branchless banking applications in the developing world. In *24th USENIX Security Symposium (USENIX Security 15)* (pp. 17-32).

[11] AndroBugs Framework. Retrieved from https://github.com/AndroBugs/AndroBugs_Framework

[12] Apktool: A tool for reverse engineering Android apk files. Retrieved from https://ibotpeaches.github.io/Apktool/

[13] Charles Proxy: Getting started https://www.charlesproxy.com/documentation/getting-started/

[14] Burp Suite: Getting started https://portswigger.net/burp/help/suite_gettingstarted.html