Android Spyware Disease and Medication

Mustafa Hassan Saad,
Department Of Computer Engineering
MTC
Cairo, Egypt
eng.mustafa.h.saad@gmail.com

***Abstract*—Android-based smartphones are gaining significant advantages on its counterparts in terms of market share among users. The increasing usage of Android OS make it ideal target for attackers. There is an urgent need to develop solutions that guard the user's privacy and can monitor, detect and block these Eavesdropping applications. In this paper, two proposed paradigm are presented. The first proposed paradigm is a spy-ware application to highlight the security weaknesses "disease". The spy-ware application has been used to deeply understand the vulnerabilities in the Android operating system, and to study how the spy-ware can be developed to abuse these vulnerabilities for intercepting victim's privacy such as received SMS, incoming calls and outgoing calls. The spy-ware abuses the Internet service to transfer the intercepted information from victim's cell phone illegally to a cloud database. The Android OS permission subsystem and the broadcast receiver subsystem contribute to form a haven for the spy-ware by granting it absolute control to listen, intercept and track the victim's privacy. The second proposed paradigm is a new detection paradigm "medication" based on fuzz testing technique to mitigate known vulnerabilities. In this proposal, anti-spy-ware solution "DroidSmartFuzzer" has been designed. The implementation of the anti-spy-ware application has been used to mitigate the risks of the mentioned attacks.**

It should be noted that the proposed paradigm "DroidSmart-Fuzzer" and its fuzzing test cases are designed not only to catch the proposed spy-ware application but also to catch any similar malicious application designed to intercept one or more of the listed privacies.

Keywords*—Android spyware, fuzz testing, malware behavior analysis, android smart fuzzer, anti spy-ware.

I. Introduction

ANDROID is a Linux-based operating system designed primarily for mobile smartphones and tablet computers. Initially developed by Open Handset Alliance which Google backed financially and later purchased in 2005. Android has expanded beyond its roots as a mobile phone operating system, providing a consistent platform for application development across an increasingly wide range of hardware (smartphones, tablets, smart TVs, android wear).

Android offers new possibilities for mobile applications by offering an open development environment built on an open-source Linux kernel. As an application-neutral platform, Android gives you the opportunity to create applications that are as much a part of the phone as anything provided out-of-the-box through a series of API libraries, and this openness attract large number of attackers which use this platform to implement a malicious applications, but here comes the risk that user may download and use these malicious applications which cause privacy leakage allowing attackers to put man-in-middle (MiTM) visibility into every user transaction like the proposed spy-ware in this paper. The application layer has the largest attack surface where maximum damage to security occurs. In this paper an exploit to the broadcast receiver has been presented for intercepting Received SMS, Incoming, and Outgoing calls information and silently transfer these information to the developed cloud database using the mobile Internet capabilities. Hence to keep a check on the malwares and the authenticity of the application, a medication solution is needed to dynamically analyze the behavior of the already installed applications, and to work as a spy-ware alarm system. This alarm system will warn the user about any running application eavesdropping his privacy.

In this paper, The formulation to the problem of sensitive privacy leakage has been presented as a two side game problem. The first side presents an indication about how these sensitive information can be intercepted and transmitted out to a cloud database with the help of Broadcast receivers subsystem. A spy-ware application will be presented to illustrate the techniques of attackers in intercepting these privacies and will lead to stand on weakness points. The other side game introduces an automated fuzzing approach solution referenced here as DroidSmartFuzzer. Based on the deep understanding of the previous spy-ware behavior, DroidSmartFuzzer is a light-weight, yet effective, technique for fuzz-testing security protocols. The proposed technique is modular. It generates valid inputs, and mutate the inputs using a set of fuzz operators. A dynamic Internet usage analysis execution as a reaction of fuzz test cases will detect the vulnerabilities exposed by the Application Under Test (AUT).

The fuzzer provided with the necessary keys and algorithms in order to properly mutate SMS messages, Incoming Calls and Outgoing Calls. According to high rate installation of Commercial Spy-ware which has increased in 2014 as mentioned in Google Android Security report [1] , Lacoon research team report [2], ALCATEL-LUCENT mobile malware reports [3], [4], [5], [6], and Joshua Dalman and Valerie Hantke research on Black Hat USA 2015 [7]. DroidSmartFuzzer has been tested against the top 15 commercial spy-ware [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], two free spy applications on Google Play[23], [24], two free spy applications on Amazon store [25], [26] and the proposed spy-ware application.

An adaptive, modular, and real time design of the fuzzing

test protocol has been used for generating valid inputs to the AUT and then monitoring its malicious behavior dynamically. For example, to test the behavior for a doubtful spy application according to eavesdropping on Received SMS , we use the PDUs format to generate a valid SMS [27] and inject it dynamically from the DroidSmartFuzzer to the Android mobile. The evidence for the AUT Internet usage behavior according to SMS injector will be accurate as will be illustrated in the experimental evaluation section.

A. Contributions

To summarize, our contributions are fourfold. First, we developed a real android spy application to deeply understand the attacks that we will resist. Second, we proposed a dynamic modular fuzz-testing technique for monitoring and detecting the unexpected behavior for the installed applications . Third, we implemented a set of fuzz operators that are effectively constructed a real environment for finding malicious behavior for the AUT. Finally, we empirically evaluated the effectiveness of our proposed fuzzer against the proposed spy-ware application, and set of free and commercial spy applications.

B. Related Works

Researchers have already done a good efforts in malicious application analysis. Felt et al. [28] analyzed a total of 46 Android, iOS and Symbian malware samples in detail to provide us with surveys on mobile malware. Along with the processed information, the authors also provide us with dangerous permissions these apps used. ProfileDroid [29] is a behavior profiling system for Android applications which is also not suitable for analyzing internal behavior logic for the running applications. DroidScope [30] is an analysis platform designed for Android that belongs to static analysis techniques to cover Java semantics. However, the problem of analyzing Android application is not simple as how to capture behaviors from different language implementations. It is hard to achieve effective analysis without considering the Android security architecture. Permission Event Graph [31], represents the relationship between Android actions and permission requests, it is proposed to characterize unintended sensitive behaviors. However, this technique could not capture the internal logic of permission usage. Zhou et al. [32] proposed DroidRanger to detect malicious applications by doing pre-filtering step based on authenticated permissions by an application. It then analyze the application code structure, as well as other properties of applications. Finally, an heuristics based detection engine is run with the data gathered about applications. With this approach, the authors were able to find malware on the official Android market, and two zero-day malware. In addition, the authors of [33], captured and evaluated 1,260 Android malware samples. The main focus was the mechanisms through which the malware propagate, the activation procedures, the permissions required, and the events also known as the broadcasted intents that will be listened to by the malware and this is our concern. Results have shown that 21 malware families of the 49 captured listen for Received SMS messages, also find that malware are actively intercepting various information on the infected phones, including SMS messages, phone numbers as well as user accounts. In particular, there are 13 malware families (138 samples) in this dataset collect and upload SMS messages and 15 families (563 samples) gather incoming and outgoing phone numbers, and 3 families (43 samples) obtain and upload the information about user accounts.

This paper is organized as follows, Section II illustrates an overview of android OS and Android malware, Section III explains proposed spy-ware application model, Section IV explains our DroidSmartFuzzer approach, Section V discusses the results of various experiments and Section VI concludes the paper with future possible work.

II. Overview of Android OS & Malware

A. Android OS Architecture

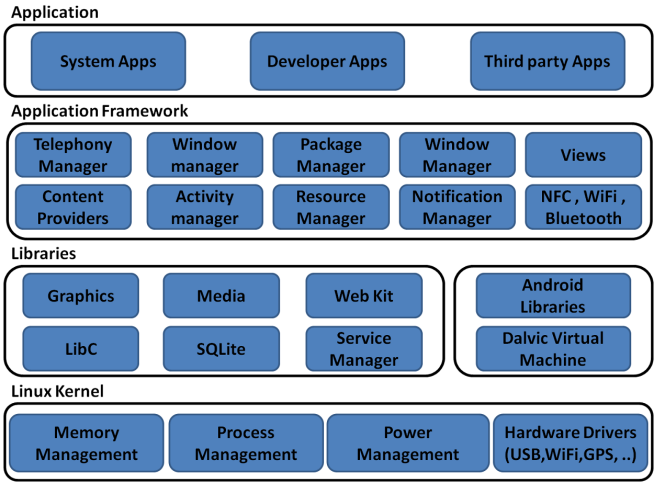

Fig. 1. Android Operating System Layers

Android operating system comprise of different software layers as shown in Figure 1:

- Linux kernel: Is the Bottom layer of android operating system which provides the basic system functionality such as process management, memory management, device management and device drivers which make our task easier while interfacing the android with other devices.
- Libraries and Android Run Time : Libraries are Java libraries build specific for android operating system. It provides the different libraries useful for well functioning of android operating system. Android Run Time placed in second layer from bottom and It provides most important part of the Android called Dalvik Virtual Machine which is similar to Java Virtual Machine (JVM) but only difference is that it is designed and optimized for Android. Dalvik Virtual machine uses core functions of Linux such as memory management and multithreading and enables each android application to run on its sandbox.

- Application Framework : This layer directly interacts with the Android applications and also manages the basic functions of android device such as location management, voice call management and resource management.
- Applications : This is the final destination for the developer journey where his application installed and run here. This layer communicate with the underlying layers through different form of Inter Process Communication (IPC) Endpoints.

B. IPC Endpoints

- Android Manifest: All Android Application Packages (APKs) must include the AndroidManifest.xml file which contains the core information about the application, such as:
  - Package and version name
  - Permissions (restriction limiting access to a part of the code or to data on the device)
  - Application info (such as launcher icon , title and the start point of the application)
  - Activities, Services and Broadcast Receivers definitions
- Activity: Is the UI component built on the base Activity class which takes care of creating a window for the developer in which he can place his UI. While activities are often presented to the user as full-screen windows, they can also be used in other ways as floating windows or embedded inside of another activity. Lower level managements of activities handled by the Activity Manager service which placed on application framework.
- Content Provider: Act as a programming interface to common shared data stores. Contacts provider for example manage centralized repositories of contact entries which can be accessed by other applications with a specific permissions. Applications may also design their own content providers and may optionally exposed them to other interested applications.
- Service: Is an application component without a UI that perform long running operation in the background. Another application component can start a service like activities or broadcast receivers and it will continue to run in the background even if the user switches to another application. For example, a service might handle network transactions, play music, or interact with a content provider, all from the background.
- Broadcast Receivers : These are commonly found where applications want to register for system or application events or actions. All registered receivers for an event are notified by the Android runtime once this event happens. For example, applications can register for the PROCESS_OUTGOING_CALL system event which is fired once the Android user try to dial a number. The registration for the broadcast receiver has two types, first is static and mostly used when you want to listen to an event all the time, second is dynamic and may be used when one of the screen of your application is open and unregister that receiver once application is closed.

C. What is Mobile Malware

Generally malware is a software designed to damage a computer system without the owner's knowledge or consent. Malware is short for malicious software and includes viruses, Trojan horses, spy-ware and or any other unwanted or malicious software. Malware is a crime and the most common path criminal's use is through the Internet. Malware is being produced at an increasingly alarming rate. Unfortunately, it is well known that computers already attacked by a malware so we are always looking forward to update our anti-virus application, but what about the malwares which attack our mobile devices? As we will discuss and implement our proposed spy-ware application in the next section. We will see by practical experiment that mobile malicious applications on the rise, and the intercepted information from our cell phones are horrible.

For instance, we could accidentally download a malicious application that hijacks our identity, personal photos, contacts, SMS,emails and our private information and sends them to a remote server. Or, we could download a dangerous application that sends SMSs or dialing services numbers from our phone, charging us with an expensive money on our mobile bill. Other malicious programs can potentially malfunction our cell phone basic functionality.

D. Types of Android Malware

The majority of Android malware can be classified in two types:

- Trojans and Viruses: Trojans and Viruses infect Android devices by attaching themselves to seemingly harmless or legitimate programs, which are installed as third parties with the application and then carry out malicious actions. Such malicious actions hijack the browser, cause the device to capture user login information from other applications such as social media and mobile banking and disastrously Android viruses can root the device and gain access to files and flash memory like DroidKungFu [37]. Trojans and viruses can become installed on the device any number of ways and cause effects that range from simply annoying to highly destructive and irreparable.
- Spyware and Botnet: Another observed type of Android malware is classified as spyware and has capabilities to forward private data to a remote server. On the other hand, the spy-ware could also receive orders from the server to intercept specific privacy in which case it is part of a botnet. spy-ware is likely to use some of the IPC endpoints described in Section II-B. Broadcast receivers are of particular interest as they can be used to secretly intercept and forward incoming SMS, and Call information to a remote server or to wait for BOOT COMPLETED intent to start a background service as soon as the device is started.

III. PROPOSED MODEL FOR SPY-WARE "DISEASE"

Before examining the attacks and the proposed countermeasures in more detail, we outline the threat model that our work

is based on. So in this section we will demonstrate our spy-ware application which we called it Chameleon, this name coming from the invisibility of the application icon on the launcher window like most of real detected spy-wares, now let's start with the Chameleon Application structure as shown in Figure 2.

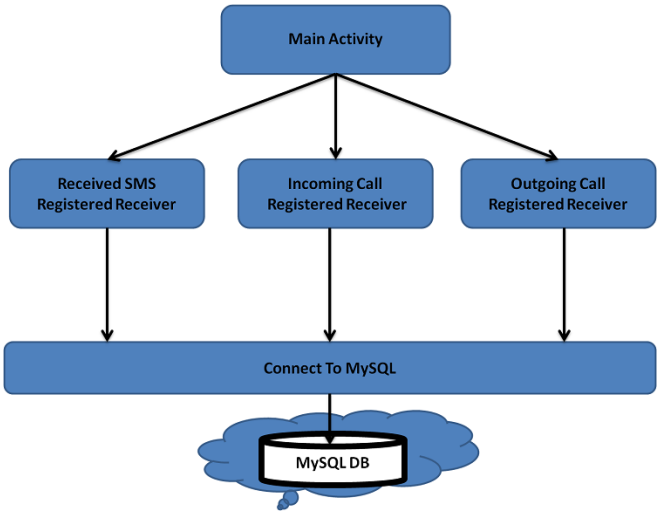

Fig. 2. Chameleon Application Structure

A. Chameleon Design

- Main Activity : This is the main class which will achieve the following functionalities:
  1) Get the victim cell phone name and IMSI of his installed SIM card.
  2) Run a background AsyncTask to insert the victim intercepted data in the cloud DB.
  3) Hide the application launcher icon by using the application framework layer component PackageManager as seen in Figure 3. This component has the methods ability to hide application icon without killing our application process.
  4) Hide the Main Activity view.
- Received SMS Registered Receiver : This class is a sub class of android.content.BroadcastReceiver. It is responsible for intercepting all received SMS through the next implementation scenario:
  1) Overriding the onReceive method which will be called automatically when the intent filter action android.provider.Telephony.SMS_RECEIVED fired by Android system instantaneously if any SMS received.
  2) Inside the onReceive method we intercept the SMS information by getting the PDU (protocol description unit) extra data loaded with the intent.
  3) Converting this PDU format to a Unicode format and then get out the sender number, the message body and the time stamp of this message.
  4) Finally we create a background AsyncTask to send this SMS information to our cloud DB.
- Incoming Call Registered Receiver :
- This class is also a sub class of android.content.BroadcastReceiver. It is responsible for intercepting the incoming calls information through the next implementation scenario:
  1) Overriding the onReceive method which will be called automatically when the intent filter action android.intent.action.PHONE_STATE fired by Android system instantaneously when any incoming calls received to the victim mobile.
  2) The receiving intent will have an extra string variable TelephonyManager.EXTRA_STATE which describes the phone state. If this state is TelephonyManager.EXTRA_STATE_RINGING then there will be another extra string variable TelephonyManager.EXTRA_INCOMING_NUMBER. This variable contains the incoming phone number.
  3) Finally we create a background AsyncTask to send this incoming call information to the cloud DB.
- Outgoing Call Registered Receiver : This class is a sub class of android.content.BroadcastReceiver. It is responsible for intercepting the outgoing calls information through the next implementation scenario:
  1) Overriding the onReceive method which will be called automatically when the intent filter action android.intent.action.NEW_OUTGOING_CALL fired by Android system instantaneously when the victim mobile receiving any outgoing calls.
  2) The receiving intent will have an extra string variable Intent.EXTRA_PHONE_NUMBER.This variable contains the outgoing phone number.
  3) Finally we create a background AsyncTask to send these outgoing call information to the cloud DB.
- Connect To MySql : This is the JDBC helper class which encapsulates the attributes and methods needed for the cloud DB connection and Data Manipulation Language (DML) commands.
- MySQL DB: We design a very simple relational cloud DB consists of four tables. cell phone information, SMS, outgoing call logs and incoming call logs using Google SQL cloud service to host our DB, and created it using razorSQL tool. It should be noted that AndroidManifest.xml already registered a static receivers for the incoming SMS, outgoing calls, and incoming calls which will listen to these events all the time as explained in Section II-B. This will give us the opportunity to intercept our preferred data even if the victim reboot his cell phone. Also inside the AndroidManifest we granted a permissions to access Internet, receive SMS, read phone state and process outgoing calls.

B. Conclusion of the proposed spy-ware model

The proposed Chameleon application has already intercepted and sent out received SMS messages, Outgoing

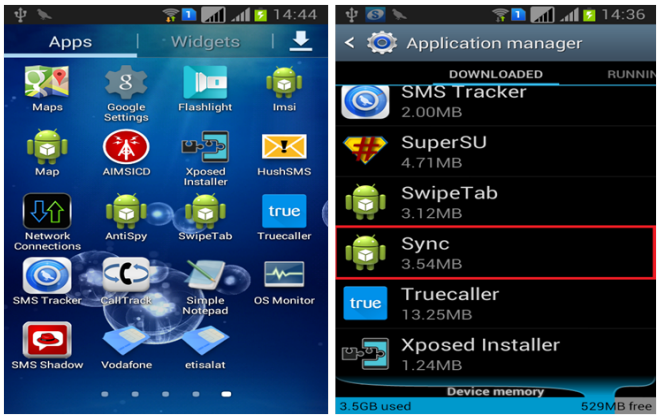

Fig. 3. Chameleon Application

and Incoming calls information from victim's Android mobile to our cloud database. We have registered a Broadcast Receivers which respond to broadcast messages (Received SMS, Incoming Call, Outgoing Call) from the system itself. These messages are sometimes called events or intents. Now whenever the Android device received SMS, started an out-call or received incoming call event, it will be intercepted by Broadcast Receiver classes. This system events will fire the implemented logic inside onReceive() method that will be executed to send these information via the device Internet capability out to our cloud database. This concept of Broadcast Receiver working mechanism will lead us to the medication process. Also will give us the idea to uncover the behavior mask of any application intended to eavesdrop our privacy.

IV. PROPOSED MODEL FOR DROIDSMARTFUZZER "MEDICATION"

In this section we will take into consideration our deep understanding of the Chameleon behavior and technique for intercepting the user's privacy, so we will think like a white hat vulnerability testers. The main target is to develop a full automated fuzz testing solution, write a good test scenario, design this scenario to be a real fuzz testing environment for the AUT, analyze the AUT behavior as a response to our fuzzing system, then create our pass fail report.

A. DroidSmartFuzzer Methodology

DroidSmartFuzzer methodology is based on Fuzz testing or fuzzing which is an effective technique for finding security vulnerabilities in software or computer systems. Traditionally, fuzz testing is a software testing technique, often automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program. The program is then monitored for exceptions such as crashes, or failing built-in code assertions or for finding unexpected behavior. Applications under fuzzing test fail when they behave in a way their developers did not intend or anticipate. In traditionally applied fuzzing, failure modes come in four categories:

- Crashes
- Endless loops
- Resource leaks or shortages
- Unexpected behavior

These failure modes vary based on type of the system or software being tested, the underlying operating system, and more. Our DroidSmartFuzzer has been designed and implemented according to the last failure mode category "Unexpected behavior". The consequences depend on the purpose and function of the software, where and when it is operated, and so on. In general, The key to catch the spy is to put it in an ideal environment and give it the information that it waits and then watching it's behavior. The job of the DroidSmartFuzzer is to detect Internet usage Unexpected behavior for the doubtful applications which authorized to access the next permissions:

- RECEIVE_SMS
- PROCESS_OUTGOING_CALLS
- READ_PHONE_STATE

And for sure these doubtful applications will be authorized to access INTERNET permission. DroidSmartFuzzer test cases scenario will be as follows:

- Preparing of SMS, Incoming Calls and Outgoing Calls fuzzing models that will be injected to the application layer.
- Reading all installed applications and their permissions.
- Filtering these applications by any or all of the authorized permissions that we listed previously.
- Fuzzing the filtered applications with our prepared injectors according to their authorized permissions.
- Monitoring the Internet usage for AUT during the injection life cycle.
- Reporting a Pass/Fail criteria for the applications under fuzzing test according to its Internet usage behavior.

B. DroidSmartFuzzer Design

Based on the methodology presented above, we implemented a new tool named DroidSmartFuzzer. The overview of DroidSmartFuzzer design is shown in Figure 4.

DroidSmartFuzzer can test any running application on Android real device dynamically once it was authorized to one of the interested permissions and report its Internet usage behavior instantaneously. To correctly capture the behaviors of using permissions inside our proposed spy-ware application or its similar spy-ware applications, we analyze the execution flow of the DroidSmartFuzzer fuzzing scenario into two related stages, the first stage is the manifest explorer and applications packages information engine that will read all Androidmanifest.xml attached to the installed applications of the victim's cell phone and then filter them according to the interested permissions: INTERNET, PROCESS_OUTGOING_CALLS, READ_PHONE_STATE, and RECEIVE_SMS. These selected permissions gave us a vulnerability warning of spying issue on our privacy like our proposed Chameleon application. Second stage will be achieved by fuzzing these filtered applications one by one by our prepared injectors for Received SMS, Outgoing Calls and Incoming Calls. For the Received SMS injector we implement a real SMS in PDU format.

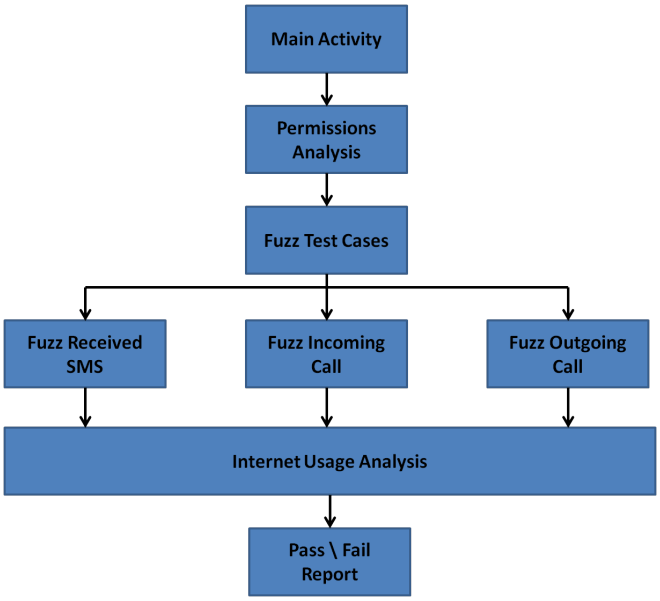

Fig. 4. DroidSmartFuzzer Application Structure

This PDU contains the message ”Wake Up”, and this SMS contains some meta-data about itself [34].

For the Incoming call injector we used a Cognalys Android Library [35] which used commercially in mobile number verification. Integrating Congalys library in our DroidSmartFuzzer project gave us a good opportunity to inject the AUT with a real Incoming call environment. The last injector was for Outgoing Call and we simply use ACTION_CALL action to trigger built-in phone call functionality available in Android device.

According to the filtered applications of stage 1. We will fuzzy them by the previous injectors, then monitoring and recording the Internet usage behavior according to this fuzz test scenario. The application that will pass stage one and gives a positive impact in consuming Internet usage from any of the fuzzy test cases will be reported to the user as a spy-ware application to take a decision of uninstall it or not.

According to Figure 4 the proposed model for DroidSmartFuzzer will consist of :

- Main Activity : This activity is the starting point and main view for DroidSmartFuzzer. Main Activity acts as a sensor to collect all Manifest permissions information from all installed applications. The main task for this module is to convert all installed applications to a list of modeled Java objects. Each object will encapsulate all needed information about this application and a modeled list of its authenticated permissions. For example the application model will have application name, package name, list of assigned permissions, aliases, and required fuzz actions.
- Permissions Analysis : This module acts as a filter layer before the starting of fuzzing test cases. Simply it collects all modeled application objects from the previous list according to authenticated permissions filter. The application model that contains permission name attribute ”Internet” and one or all of (”Received SMS”,”Incoming Calls”,”Outgoing Calls”) will go through our filter and be ready to the next stage.
- Fuzz Test Cases : Now we have a list of the doubtful applications, So we implement a Fuzz Test Cases class that will act as an abstract parent class for Fuzz Received SMS, Fuzz Incoming Calls, and Fuzz Outgoing Calls child classes. This class will encapsulate the intent attributes and the abstract methods needed for the child classes to perform their fuzzing functionalities.
- Fuzz Received SMS : The main functions of this child class is to prepare a real PDU SMS ”Wake Up” and its meta data, creating an intent which its action is ”android.provider.Telephony.SMS_RECEIVED”, adding the extra bundle data needed to this intent to perform as a real Received SMS injector, and then broadcast this injection to the android application layer. All doubtful applications will receive this SMS as a real one coming from the provider.
- Fuzz Incoming Calls : This child class implements its Incoming Call injector with the help of Cognalys. Cognalys provides a multi platform service which help application developers in verifying the mobile number in their applications. We integrate and implement Cognalys API in this child class, creating an intent to initialize Cognalys activity, adding the extra bundle data needed to this intent, adding the verified mobile phone number which is our target cell phone number, and then running this intent. Our target cell phone will receive a real incoming call from Cognalys provider within maximum 30 Seconds to verify its number.
- Fuzz Outgoing Calls : The main function of this child class is to implement an Outgoing Call injector. The easiest way to accomplish this call is to create an intent which its action is ”Intent.ACTION_CALL”, adding the extra bundle data needed to this action to perform as a real Outgoing call injector, and then start the activity of this intent. We put the dialed phone number the same as the target mobile phone number to process a call and then stop it automatically after few seconds because it will respond with busy number. So DroidSmartFuzzer will start and stop a real outgoing call within maximum 6 seconds.
- Internet Usage Analysis : Now coming to the Internet usage behavior analysis as a response to the fuzzing injectors. This module is responsible for measuring the Internet usage before and during the fuzzing process. We keep track of Internet consuming for the AUT by the help of android.net.TrafficStats built-in class. This class provides network traffic statistics. These statistics include bytes transmitted and received and network packets transmitted and received, over all interfaces, over the mobile interface, and on a per-UID basis.
- Pass Fail Report : The DroidSmartFuzzer testing results accomplished via this module. After reading all installed applications, filtering them to a list of doubtful applications, fuzzing this list, monitoring the Internet usage

during the fuzzing life cycle. Now this module will take a decision regarding to the AUT and notify the user if this AUT is PASS which means that no Internet usage behavior changes during the whole fuzzing process life cycle, or FAIL which means that the AUT Internet usage behavior has changed during the fuzzing life cycle. In the case of FAIL, this module will report in details which privacy or fuzz test case has consumed the Internet as shown in Figure 5. From this report the user can take action to uninstall this application or not.

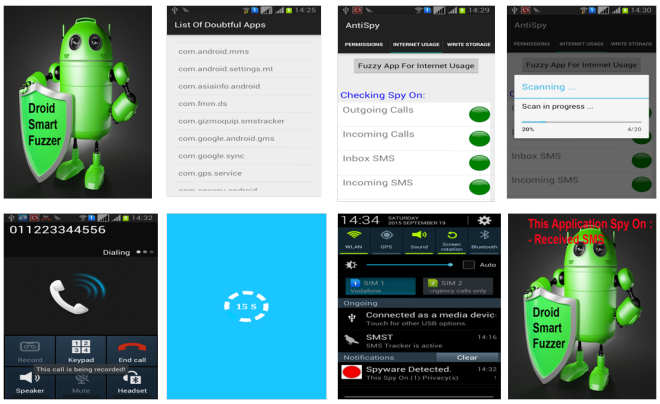

Fig. 5. DroidSmartFuzzer Application

V. EXPERIMENTAL EVALUATION

We run the DroidSmartFuzzer on Samsung Galaxy A3. DroidSmartFuzzer installed on Android 4.4 "Kitkat" because of its large distribution among the Android users [36]. The results of DroidSmartFuzzer against all the 20 spyware samples including the proposed spy-ware "Chameleon" is summarized in Table I, and the already installed applications are summarized in Tables II. These tables show the Internet usage behavior results of AUTs. Behavior chart which represents sample of AUTs Vs Transmitted bytes during the fuzzing process shown in Figure 6.

- Evaluation 1 : All the studied commercial spy-ware products Internet usage behaviors and also the free ones on Google play or Amazon store are changing during our fuzzing life cycle. They all are transmitting data packets using the available mobile Internet media (WiFi Or Mobile Data). All these spy-ware applications have been reported by DroidSmartFuzzer as spying on (Received SMS, Incoming Calls, and Outgoing Calls information). Table I, shows a sample of these results.
- Evaluation 2 : Most of the commercial spy-ware providers gave their customers the ability to try their spy products from 2 to 7 days, as a full functioning demo versions, But DroidSmartFuzzer detects that these demo versions still transmitting out the victim's privacy after the trial period is running out. This gave us a big question. Why these providers still spying on and sending out the privacy information although their applications are only demo versions?
- Evaluation 3 : As shown in Figure 6, the number of transmitted bytes during the fuzzing process are differ from application to application, this evaluation gave us a moral incentive to analyze the AUT transmitted packets in our future work.
- Evaluation 4 : As shown in Table II and behavior chart shown in Figure 6, there is another future study on a system application called "NetworkLocation_baidu.apk", and its package name is "com.baidu.map.location". This application interested in Phone state permission which tested by our DroidSmartFuzzer with the Incoming call injector. The application Internet usage behave unexpectedly during the incoming call fuzzing test, and it transmits 267 Bytes with every incoming call fuzz life cycle, so it has been reported as a spy-ware application, which means that this system application is spying on our Incoming call information.

TABLE I. SAMPLE OF THE 20 TESTED SPY-WARE APPLICATIONS

App Name	Installed Package	Store/ Provider	DroidSmartFuzzer		
			Rx SMS	In. Calls	Out. Calls
Truth Spy	com.systemservice	thetruthspy.com	✓	✓	✓
Ino Spy	com.inospy	inospy.com	✓	✓	✓
Hello Spy	com.hellospy.system	hellospy.com	✓	✓	✓
Ti Spy	com.sce.display	Amazon.com	✓	✓	✓
SMS Tracker	com.gizmoquip.smstracker	Google Play	✓	✓	✓
Chameleon	com.google.sync	Propsed Application	✓	✓	✓

TABLE II. SAMPLE OF INSTALLED ANDROID SYSTEM APPLICATIONS

App Name	Installed Package	Store/ Provider	DroidSmartFuzzer		
			Rx SMS	In. Calls	Out. Calls
Sys. App.	com.baidu.map. location	Samsung	■	✓	■
Sys. App.	com.android.browser	Samsung	■	×	■
Sys. App.	com.android.email	Samsung	×	×	■
Sys. App.	com.android.exchange	Samsung	×	×	■
Sys. App.	com.android.mms	Samsung	×	×	■

VI. CONCLUSION AND FUTURE WORK

In this paper, we studied the Android spy-ware attacks that led to user's privacy leakage. We have proposed a spy-ware application to understand the attacker's strategy and techniques with the help of the Android system permissions, broadcast receivers and intents mechanism to intercept our Received SMS, Incoming and Outgoing calls information. According to this study, we propose a medication solution which we called DroidSmartFuzzer, it based on the fuzz testing concept to analyze the AUT behavior against the Internet usage during the fuzzing life cycle. DroidSmartFuzzer achieved the full automation fuzz testing techniques by driving test on those spy-ware applications, cataloging the results and warning the user immediately to take a decision against this AUT. DroidSmartFuzzer has the ability to do fuzz testing on all installed applications after filtering them relative to their permissions. DroidSmartFuzzer efficiently detect 4 free and

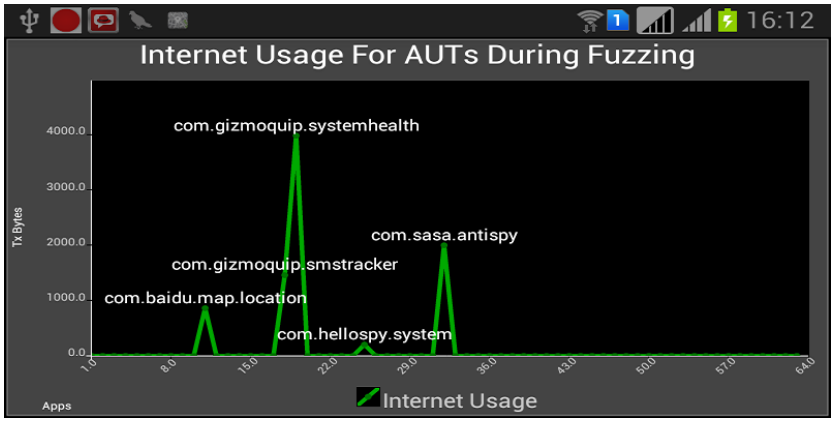

Fig. 6. Internet Usage of AUTs During Fuzzing

top 15 commercial spy-ware applications sold in the market, also DroidSmartFuzzer can actually give us an indication of the Zero Day Attack like our Chameleon spy-ware detection.

By the end of 2015 Google announced that a new Android 6 Marshmallow, will be applicable with a pair of Nexues phones. Marshmallow users will be able to agree to application permissions as they are needed rather than as a long list when software is installed. The catch is that as we illustrated before the commercial spy-ware and our chameleon spy-ware are hidden applications, also most of commercial spy-ware applications installed physically to the target victim's mobile. DroidSmartFuzzer also will help Marshmallow users to take a decisions of stopping one or more of the permissions after fuzzing these Marshmallow's applications. Our future work will be designing a cloud database that will be connected to our DroidSmartFuzzer for storing all spy-ware detected applications and their attached information to be an applicable reference to all researchers in this field.

Implementing a new injectors for fuzzing more important privacies like location, send SMS, call recording ... etc.

REFERENCES

[1] Google, "Android Security 2014 Year in Review Report" Oct. 2014.

[2] Lacoon Mobile Security & Check Point, "targeted attacks on enterprise mobile Threat Research" Feb. 2015.

[3] Alcatel.lucent, "Kindsight Security Labs Malware Report H1 2015" Sep. 2015.

[4] Alcatel.lucent, "Kindsight Security Labs Malware Report H2 2014" Feb. 2015.

[5] Alcatel.lucent, "Kindsight Security Labs Malware Report H1 2014" Sep. 2014.

[6] Alcatel.lucent, "Kindsight Security Labs Malware Report Q4 2013" Jan. 2014.

[7] J. Dalman, V. Hantke "Commercial Spyware-Detecting the Undetectable," Black Hat USA 2015, July. 2015.

[8] iKeyMonitor, Commercial Spy, http://ikeymonitor.com, 30-10-2015

[9] spy1dollar, Commercial Spy, http://spy1dollar.com, 30-10-2015

[10] spy-phone, Commercial Spy, http://spy-phone-app.com, 30-10-2015

[11] MxSpy, Commercial Spy, http://mxspy.com, 30-10-2015

[12] ExactSpy, Commercial Spy, http://exactspy.com, 30-10-2015

[13] Spyera, Commercial Spy, http://spyera.com, 30-10-2015

[14] Snoop, Commercial Spy, http://www.snoopvip.com, 30-10-2015

[15] Mobikids, Commercial Spy, http://mobikids.net, 30-10-2015

[16] MobileSpy, Commercial Spy, http://immobilespy.com, 30-10-2015

[17] HelloSpy, Commercial Spy, http://hellospy.com, 30-10-2015

[18] InoSpy, Commercial Spy, http://inospy.com, 30-10-2015

[19] MobileSpy, Commercial Spy, http://mobile-spy.com, 30-10-2015

[20] SMS Tracker, Commercial Spy, https://smstracker.com, 30-10-2015

[21] MaxxSpy, Commercial Spy, http://maxxspy.com, 30-10-2015

[22] TheTruthSpy, Commercial Spy, http://thetruthspy.com, 30-10-2015

[23] SMS Tracker, Google Play, http://play.google.com, 30-10-2015

[24] SMS Tracker Plus, Google Play, http://play.google.com, 30-10-2015

[25] Monitor Call SMS, Amazon Store, http://amazon.com, 30-10-2015

[26] Mobile Monitor, Amazon Store, http://amazon.com/, 30-10-2015

[27] PDU Format, Smart Position

[28] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner,"A survey of mobile malware in the wild," in SPSM 11. ACM, 2011, pp. 314.

[29] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Profiledroid: multi-layer profiling of android applications. In Proc. of Mobicom12, 2012.

[30] L. K. Yan and H. Yin. Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In Proc. of USENIX Security12, 2012.

[31] K. Z. Chen, N. Johnson, V. DSilva, S. Dai, K. MacNamara, T. Magrino, E. X. Wu, M. Rinard, and D. Song. Contextual policy enforcement in android applications with permission event graphs. In Proc. of NDSS13, February 2013.

[32] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets, in NDSS12, 2012.

[33] Y. Zhou, X. Jiang, Dissecting Android Malware: Characterization and Evolution, in IEEE Symposium on Security and Privacy, 2012.

[34] SMS and PDU format, http://www.smartposition.nl/resources/sms_pdu.html.

[35] Cognalys, Inc., https://www.cognalys.com.

[36] Android Developers, Dashboards, https://developer.android.com/about/dashboards/ , Oct 2015.

[37] F-Secure Lab, Threat Description, DROIDKUNGFU.C , Nov 2012.