International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 2 Issue 12, December-2013, pp: (107-114), Available online at: **www.erpublications.com**

Behaviour analysis of malware in android enabled devices using reverse engineering tools

Sunit Gupta[1], Jeberson[2]
[1]M.Tech. Student, [2]Associate Professor
Shepherd School of Engineering & Technology
Sam Higginbottom Institute of Agriculture, Technology and Sciences, Allahabad (U.P.), India

Abstract: Smartphones have gained tremendous popularity over the last few years. In this growing market of smartphones, Android, an open source platform of Google which has become one of the most popular Operating Systems. Android is mainly used in smartphones and tablets. A worrying trend that is emerging is the increasing sophistication of Android malware to evade detection by traditional signature-based scanners. As such, Android app marketplaces remain at risk of hosting malicious apps that could evade detection before being downloaded by unsuspecting users.Users store all their sensitive data and information on these devices. However, this ease of use comes at a very big price and comes with side-effects which most users are unaware of. With the increase use of mobile devices, malware is also enjoying unprecedented growth at the expense of unsuspecting and naïve users. Even though several mobile security solutions have been proposed, it is apparent that more effort is required to ensure the security of the data on these devices. The research presented in this paper is an attempt to analyze malware behavior by combining static analysis and dynamic analysis to collect and analyze data in an effort to suggest security techniques not currently found in Android-based devices. The results of this research will provide and insight into the targets and actions of malware as well as provide higher security if the techniques are coded into the Android OS.

Index Terms: Mobile security, Malware, Android, Wireshark

1.0 Introduction

Since the launch of Google's Android OS in 2008, the smartphone market grew stupendously and has never looked back. As of October 2012, according to estimates, out of the total number of smartphones shipped, seventy-five percent were Android devices (IDC 2012). Google claims that each day there are approximately 1.3 million activations and the total number of Android devices exceeds 5 million making it the most widely used mobile operating system today (Burns 2012). Due to its ever-increasing numbers and easy acceptance into Google's App Store, the malware market is also thriving and enjoying unprecedented growth. To counteract these threats, anti-virus and anti-malware companies are making considerable efforts. However, these efforts fail to keep up with the current security requirements. Thus, this research makes an effort to perform static and dynamic analysis to analyze the behavior of the malicious apps.
This paper begins with the introduction of the Android OS and follows with a description of the tools used to perform static and dynamic analysis. Then the paper leads to the research methodology used for analysis. Then, the results and description of the data will be discussed and finally conclusion with the findings.

2.0 Android and Need for Analysis

2.1 The Android Operating System

Android, the open source mobile operating system developed by the Open Handset Alliance led by Google, is gaining increasing popularity and market share among smartphones. Built on top of a Linux 2:6 kernel, and was released in 2008. Rigorous compartmentalization of installed applications is enforced through traditional Linux permissions **(Google, 2012).** Android applications are mainly implemented in Java, with the compiled class files further converted into Dalvik bytecode, running on the proprietary register based Dalvik VM. It is similar to the JVM, but designed for a resource-constrained environment with a higher code density and smaller footprint. Applications are tightly coupled with a large and function-rich Android framework library. Also, applications are free to include compiled native code as standalone Linux shared object. The interaction between an application's Java and native code is well defined by the Java Native Interface (JNI) specification and supported by Android's Native Development Kit **(Borstein 2012).**

The most interesting feature about Android is that the Kernel places each application in a sandbox when it executes.

2.2 Need for App Analysis

Since mobile devices have seen considerable innovation and creativity in terms of their features and functions like web browsers, task managers, games and email access etc. This increasing complexity of smartphones brings with it increasing vulnerabilities. Although Android is most widely used, there exists a dearth of applications in order to completely benefit from this operating system. Hence, third party application developers create new applications and launch them in the Android Market. This gives users access to thousands of applications; it is however important that the user needs to entirely trust the applications before installing them.

Users entrust more and more sensitive data like banking data, social networking identification to the security mechanisms embedded within these mobile devices and operating systems. It is apparent that the current security technologies are insufficient and there is a need to assess the Android OS and application software for malicious activity.

2.3 Existing Security Issues

Applications in Android devices may be installed through either the Google Play Store (formerly called Android Market) or through a variety of third party application stores. Some of the third party stores are Opera Mobile app store, GetJar, SlideME, etc. The fact that Android permits application installation from third party vendors means that Google has no control over the quality or safety of the applications provided in these stores. Several cases were encountered where legitimate apps from the Google Play Store were modified to inject malicious code and the modified apps were sold in these third party stores. It is difficult to determine whether the application is genuine or not. In these cases, the reliability of the application depends upon the security measures implemented by the application store. This makes it essential to provide a reliable means to verify the authenticity of the applications **(Google 2012).**

2.4 Currently Available Security Measures

Google's Play Store has a security enforcement known as the "Bouncer" which verifies the applications being uploaded for any suspicious behavior. It works on a blacklisting based approach and instances have occurred where-in the malicious application survived several hours or even days before it was detected and taken off. However, this app still does not protect the users from installing malicious apps from sources other than the Play Store. 4.2 version (Jelly Bean) of Android has shipped with a security feature to counter this problem. A new feature allows the user to verify the third party application being installed on the phone. However, research published by (Jiang 2012) suggests that the system has a malware detection rate of 15.32%. An anti-malware system must have a detection rate of at least 80% to be deemed acceptable.

3.0 Past and Present Mobile Application Analysis

The earliest research on the current generation of mobile devices dates back to 2007, when iOS, Blackberry and Android operating systems began to appear in the market. When security became a major concern, many researches followed. Cheng et al. developed SmartSiren: an intrusion detection system for the Symbian OS and Windows Mobile (Cheng 2007). A monitoring agent runs on the mobile device and it collects and records the system calls made by the applications running on the phone. This data is forwarded to the researcher's server over the Internet. Each device registers with the proxy and provide information so that the proxy can differentiate between the data obtained from the different devices. This data is then analyzed to find out if the device is infected with some malware that uses either SMS or Bluetooth to spread. Bose et al. also proposed a behavior-based malware detection system for Symbian OS in 2008 **(Bose 2008).** Known families of existing malware are used to collect data regarding system events and API calls and a logical flow diagram is constructed and captured as a signature for that family. Then, any new system or application is compared against this database of signatures. If a match is found, the application is considered malicious.

pBMDS was another system proposed by Xie et al. that compares inputs to applications instead of reading and creating logs of malware behavior **(Xie, 2010).** The basis of this system was that user input differs from the automated input. This difference was then used to identify abnormal behavior that may result from malware on the device.

Another monitoring system was developed by **(Houmansadr, 2011).** This system was cloud-based in which an emulated device would run in the cloud. The mobile device sends all its network traffic through a controlled proxy, which duplicates this device traffic on the emulator running on the cloud. A monitoring system connected to the emulator would analyze this traffic and would alert the monitoring agent running on the mobile device regarding the malicious activity to take protective action.

All the above-mentioned studies suggest that mobile security is increasingly incorporating real-time monitoring. A major component if this research is to analyze the source code to collect data, which will be a significant step in the field of malware analysis and provides critical information.

4.0 Methodology

The methodology to be followed for malware analysis using reverse engineering tools has two phases 1) Static analysis and 2) Dynamic Analysis. First, malware samples are retrieved from contagiodump.com repositories and quarantined and sorted into categories like Trojan, worm, spyware, etc. This classification helps in the analysis by categorizing malware, which may share similar behavior. Then we proceed to analyze the source code of the apk files to determine coding style, encryption, obfuscation and infer the behavior that the malware is expected to exhibit once installed in the device. In the next phase, the malware samples are run on the device emulator to observe the malware interaction with the device and the user. The results of both these phases are then compiled, interpreted and reported manually to verify effectiveness.

To complete the investigation of the malware samples, the researchers used the following tools:

Table 1: Malware Investigation Tools used for Analysis

Tools Name	**Purpose**	**Process Used**
JD-GUI	Java disassembly and analysis	Static Analysis
Dex2Jar	Dex to Java bytecode translation	Static Analysis
Android Emulator	Android device emulation	Dynamic Analysis
Wireshark	Network Protocol Analyzer	Dynamic Analysis

4.1. Static Analysis

This phase involves analyzing the source apk file and all its contents. The contents are made accessible by converting them into an interpretable form. The classes are converted from DEX to Java classes, and the binary XML file, androidmanifest.XML, is converted to readable XML. Once the conversion is successful, the code is reviewed for any suspicious behavior. This is done by going through the required permissions of the application in the actual code, etc. A log is then made for all behaviors noticed that may be deemed malicious.

The following tools are required for static analysis:

- DEX2JAR
- Java Decompiler

The static analysis begins with converting the DEX code to Java using DEX2JAR. The Java shows all the resources and class files that are used for the app. Then, the Java Decompiler converts the class files into readable format.

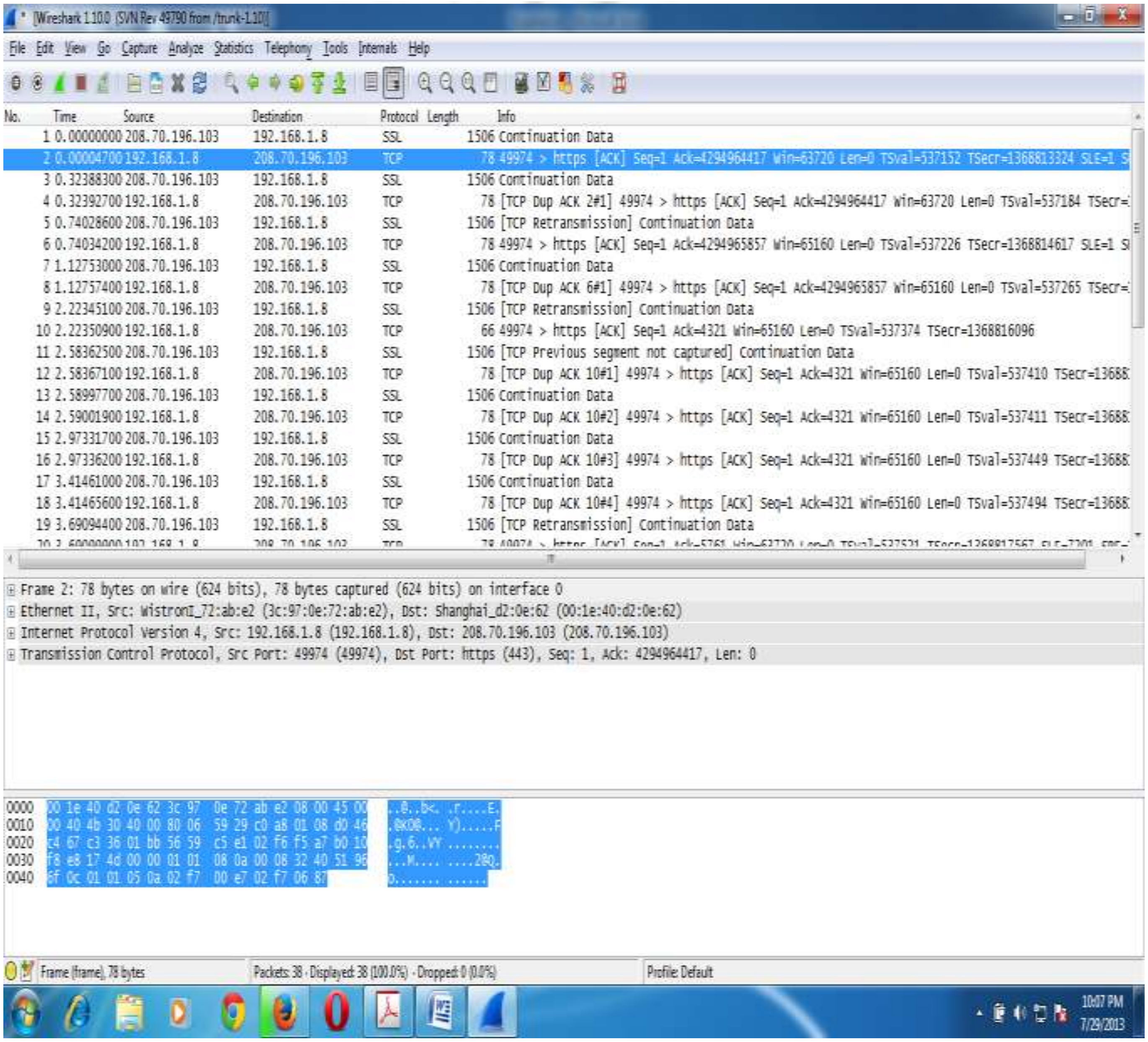
Once all the content is ready, the actual code is analyzed. The reviewer goes through the androidmanifest.XML file and extracts the permissions requested by the app. The activities within the app are also noted. The next stage is to review the source code extracted previously. Reviews look out for specific API calls like those to SMS Manager classes to find specific behavior (sending SMS in this case). A report is generated for all of these instances.

4.2 Dynamic Analysis

In this phase, the apk file is actually installed on an emulated device and the behavior is observed. A report is generated and behavior is compared with the expected results from the static analysis.

Dynamic analysis are performed by running the app on emulators. Various metrics, such as, wireshark summary, conversation, IO graph, Endpoints and Flow Graph are monitored. Nexus S is used for Dynamic analysis. This gives a more generalized view to the tests, which makes it applicable to a variety of devices. Tcpdump is used for capturing network traffic. A dynamic analysis tool is developed in order to assist in installing the application, accessing the shell and capturing metrics. This takes the effort of typing tedious commands in a terminal off the tester and hence improves efficiency.

The app is installed using the tool, and the network capture is started. The application is started and used as intended. Report from the static analysis is referred to possibly generate the conditions required for the malware to become active. The packet capture and other metrics are analyzed and a report is generated.


[Wireshark 1.10.0 (SVN Rev 49790 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
No.   Time       Source          Destination     Protocol Length  Info
    1 0.00000000 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
    2 0.00004700 192.168.1.8     208.70.196.103  TCP        78 49974 > https [ACK] Seq=1 Ack=4294964417 Win=63720 Len=0 TSval=537152 TSecr=1368813324 SLE=1 S
    3 0.32388300 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
    4 0.32392700 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 2#1] 49974 > https [ACK] Seq=1 Ack=4294964417 Win=63720 Len=0 TSval=537184 TSecr=
    5 0.74028600 208.70.196.103  192.168.1.8     SSL      1506 [TCP Retransmission] Continuation Data
    6 0.74034200 192.168.1.8     208.70.196.103  TCP        78 49974 > https [ACK] Seq=1 Ack=4294965857 Win=65160 Len=0 TSval=537226 TSecr=1368814617 SLE=1 S
    7 1.12753000 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
    8 1.12757400 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 6#1] 49974 > https [ACK] Seq=1 Ack=4294965857 Win=65160 Len=0 TSval=537265 TSecr=
    9 2.22345100 208.70.196.103  192.168.1.8     SSL      1506 [TCP Retransmission] Continuation Data
   10 2.22350900 192.168.1.8     208.70.196.103  TCP        66 49974 > https [ACK] Seq=1 Ack=4321 Win=65160 Len=0 TSval=537374 TSecr=1368816096
   11 2.58362500 208.70.196.103  192.168.1.8     SSL      1506 [TCP Previous segment not captured] Continuation Data
   12 2.58367100 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 10#1] 49974 > https [ACK] Seq=1 Ack=4321 Win=65160 Len=0 TSval=537410 TSecr=13688
   13 2.58997700 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
   14 2.59001900 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 10#2] 49974 > https [ACK] Seq=1 Ack=4321 Win=65160 Len=0 TSval=537411 TSecr=13688
   15 2.97331700 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
   16 2.97336200 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 10#3] 49974 > https [ACK] Seq=1 Ack=4321 Win=65160 Len=0 TSval=537449 TSecr=13688
   17 3.41461000 208.70.196.103  192.168.1.8     SSL      1506 Continuation Data
   18 3.41465600 192.168.1.8     208.70.196.103  TCP        78 [TCP Dup ACK 10#4] 49974 > https [ACK] Seq=1 Ack=4321 Win=65160 Len=0 TSval=537494 TSecr=13688
   19 3.69094400 208.70.196.103  192.168.1.8     SSL      1506 [TCP Retransmission] Continuation Data

⊞ Frame 2: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
⊞ Ethernet II, Src: WistronI_72:ab:e2 (3c:97:0e:72:ab:e2), Dst: Shanghai_d2:0e:62 (00:1e:40:d2:0e:62)
⊞ Internet Protocol Version 4, Src: 192.168.1.8 (192.168.1.8), Dst: 208.70.196.103 (208.70.196.103)
⊞ Transmission Control Protocol, Src Port: 49974 (49974), Dst Port: https (443), Seq: 1, Ack: 4294964417, Len: 0

0000  00 1e 40 d2 0e 62 3c 97  0e 72 ab e2 08 00 45 00   ..@..b<. .r....E.
0010  00 40 4b 30 40 00 80 06  59 29 c0 a8 01 08 d0 46   .@K0@... Y).....F
0020  c4 67 c3 36 01 bb 56 59  c5 e1 02 f6 f5 a7 b0 10   .g.6..VY ........
0030  f8 e8 17 4d 00 00 01 01  08 0a 00 08 32 40 51 96   ...M.... ....2@Q.
0040  6f 0c 01 01 05 0a 02 f7  00 e7 02 f7 06 87         o....... ......

Frame (frame), 78 bytes    Packets: 38 · Displayed: 38 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Fig. 1: Incoming and outgoing traffic monitoring

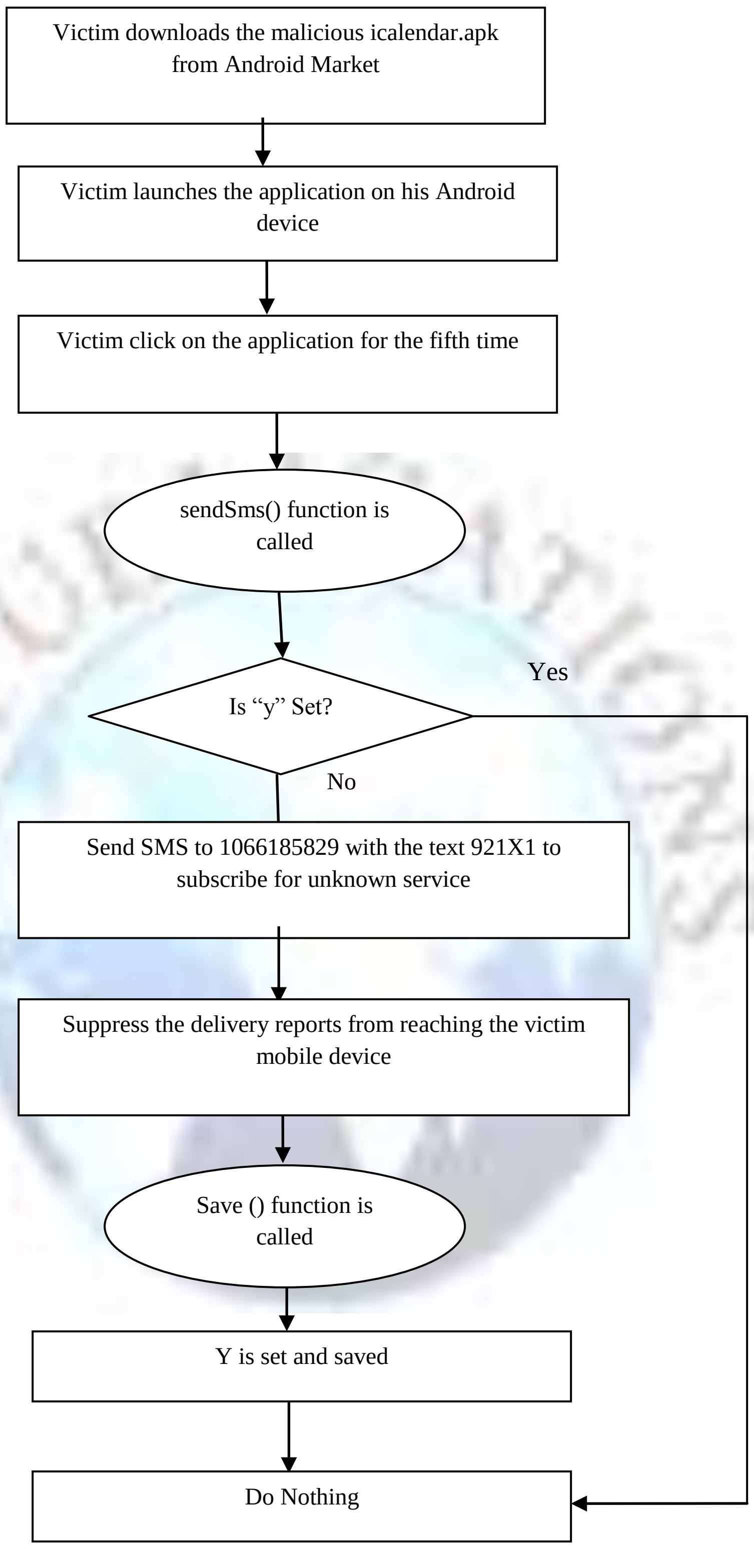

Victim downloads the malicious icalendar.apk from Android Market
Victim launches the application on his Android device
Victim click on the application for the fifth time
sendSms() function is called
Is “y” Set?
Yes
No
Send SMS to 1066185829 with the text 921X1 to subscribe for unknown service
Suppress the delivery reports from reaching the victim mobile device
Save () function is called
Y is set and saved
Do Nothing

Fig. 2 Flow Chart of Complete operation cycle of icalendar.apk Malware

In fig. 2 the flowchart explains how the icalendar.apk malware gets installed and perform malicious activities. First the victim downloads the icalendar.apk from android market. Then the victim launches the application on his android device. From the analysis we can confirm that if the user click for the 5th time then another method called send Sms() is invoked.

The sendSMS() assigns a value "y" to a variable and it also obtains the current value in the xml file. It then checks if they are equal then it is assumed that a message has already been sent and the application only displays the calendar. If it is not equal then a text message is sent using a method sendTextMessage() to a premium number 1066185829. This number belongs to a Chinese telecom company. The content of this message is 921X1. This message is a special code that indicates to the telecom company that a user wants a monthly subscription of an SMS service.

Once the message is sent a method save() is invoked. This method is the one that actually writes the value "Y" in the xml file present in the shared preferences folder.

The developer hides all incoming messages from certain numbers that pertain to the subscription messages that the user receives. The developer uses broadcast receivers in order to abort messages that arrive from certain numbers.

5.0 Aggregation of Findings

Once the static and dynamic analysis phases are done, the findings are aggregated and a final report is generated which contains information about the category of the malware, its behaviour, resources it targets, etc.

SMS related activity is the most commonly observed behavior. Most apps request permission to send/receive SMS, and most of these are from premium numbers or command and control servers. The other most common behavior was to send/ receive data using a http connection. Again this was done to contact some command and control server and to transfer the data from the device to the server. Obfuscation was also observed in the malware samples within the source code as most of the class names and variable names were changed to single letters. This though does not impact how the app interacts with the device, it makes the app difficult to reverse engineer.

5.1 Static Analysis

Source code of SmsReceiver.class: It was observed that the class file named 'SmsReceiver.class' seemed suspicious as this was a Calendar application and therefore SmsReceiver was not required. On further inspection of the source code of the 'SmsReceiver.class', it was found that it contains three numbers i.e. 1066185829, 1066133 and 106601412004, which looked rather suspicious and also looked like there was an attempt to block messages from these numbers coming to the Android mobile device, which had this application installed and running.

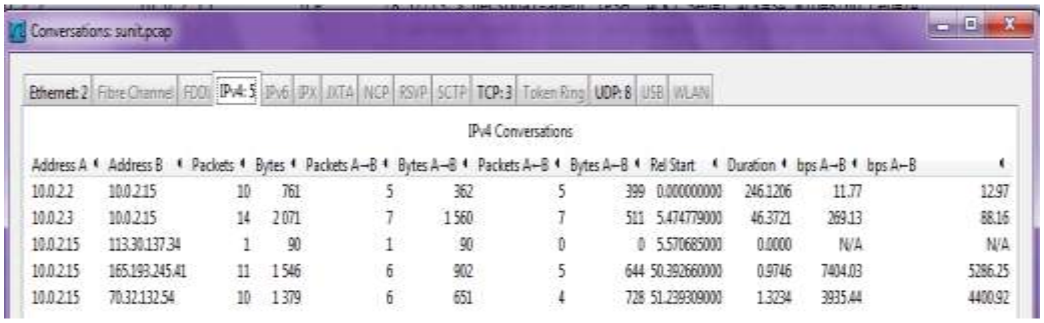
Source code of icalendar.class: The first most suspicious thing we noticed was that, in the showImg() function, after 5 clicks, there was a call to a sendSms() function. At the end of the sendSms() function, we noticed that there was a call to the save() function. So we looked for the save function in the code and found it to be just above the sendSms() function.

After proper analysis and understanding of the save() function, it was found that the string "y" is passed whenever the save() function is called. Also, it was concluded that sendSms() function can be called only once and never again due to the "if" loop that is set for the sendSms() function.

5.2 Dynamic Analysis

The Nexus S was selected for running the dynamic analysis; the main reason for this is that this device comes with stock Android, which provides a general view for the purpose of this research. The live testing is based on the results of the static analysis and effort is taken to trace the suspected behavior from the static analysis to the dynamic analysis. The behavior testing is done on the emulator.
Wireshark was used for traffic capture. This reveals the general statistics such as size, timestamp of first packet capture and last packet capture, duration for capture and average flow of traffic of the captured file during dynamic analysis of the malware execution.

A network Conversation captured by Wireshark is the traffic between two specific endpoints (IP Addresses). In My experiment, my host IP Address is 10.0.2.15, Gateway IP is 10.0.2.2 and remote C2 (Command and Control server) IP is 165.193.245.41. Conversation between 10.0.2.15 and 10.0.2.2 will be name resolution and later on infected client may contact remote C2. (Command and Control server) IP is 165.193.245.41 it reveals no of packets that have been exchanged.



IPv4 Conversations

Address A	Address B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	Rel Start	Duration	bps A→B	bps A←B
10.0.2.2	10.0.2.15	10	761	5	362	5	399	0.000000000	246.1206	11.77	12.97
10.0.2.3	10.0.2.15	14	2 071	7	1 560	7	511	5.474779000	46.3721	269.13	88.16
10.0.2.15	113.30.137.34	1	90	1	90	0	0	5.570685000	0.0000	N/A	N/A
10.0.2.15	165.193.245.41	11	1 546	6	902	5	644	50.392660000	0.9746	7404.03	5286.25
10.0.2.15	70.32.132.54	10	1 379	6	651	4	728	51.239309000	1.3234	3935.44	4400.92

6.0 Conclusion

In this paper, a method is presented by which Android applications can be assessed for malicious activity in two ways: through static and dynamic analysis. The assessment of known Android malware using this method yields sufficient knowledge to propose several technical solutions for Android that currently do not fully exist. The solutions aim to increase confidentiality, integrity, and availability of the system by improving the isolation of data, protecting access to data, and mitigating the threat of service interruption to data or applications. Thus, it is very apparent that other technical and conceptual solutions must be formulated which are more efficient in thwarting the attempts of the malware developers to access private and sensitive data and exploit the vulnerabilities of the mobile operating system.

7.0 Future Work

The research described in this paper is ongoing. Research will continue in investigating the source code of newly identified Android malware. To improve the quality and efficiency of this process, it is desirable to invest in the design, development, and implementation of a fully automated application static analysis system that is employed to triage arbitrary Android applications.

The several malwares that exist in the Android platform are a field of concern for both the users as well as Google. Therefore this project has discussed in great detail regarding the malwares, their effects and how they can be eradicated in order to provide malware/spyware free applications for users.

In the scope of this project, the most appropriate way of understanding malwares is by studying the working of an already existing malware. The next step is to try and alter a normally functioning application and craft it to behave as a malware. These two steps have been successfully achieved in this piece of work.

References

[1]. An Analysis of the AnserverBot Trojan. http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot Analysis.pdf.
[2]. Android SDK: http://developer.android.com/support.html.
[3]. Assem Nazar1;2, Mark M. Seeger1;3, and Harald Baier (2010) "Rooting Android Extending the ADB by an Auto-Connecting WiFi-Accessible Service".
[4]. B. Wissingh and T. Kruger, (2011) "Privacy Issues with the Android Market RP1 Project Paper" Available http://staff.science.uva.nl/~delaat/sne-2010-2011/p21/report.pdf.
[5]. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in Proceedings of the 6th international conference on mobile systems, applications, and services (Mobisys '08)., New York, NY, USA: ACM, 2008, pp. 225-238.
[6]. Contagiodump : http://contagiodump.blogspot.in/downloading.html .
[7]. Google,(2012). Android permissions Available: http://developer.android.com/reference/ android/Manifest.permission.
[8]. Google, (2012). Android source code Available: http://source.android.com/source/.
[9]. Houmansadr, S. A. Zonouz, and R. Berthier, "A cloud-based intrusion detection and response system for mobile phones," in Dependable systems and networks workshops (DSN-W), 2011 IEEE/FIP 41st international conference on, June 2011, pp. 31-32.
[10]. IDC, (2012 November). Android marks fourth anniversary since launch with 75% market share in third quarter, according to IDC [online]. Available http://www.idc.com/getdoc.jsp?containerId=prUS23771812.
[11]. J. Burns, (2009) "MOBILE APPLICATION SECURITY ON ANDROID", Available: http://www.blackhat.com/presentations/bh-usa-09/BURNS/BHUSA09-Burns-AndroidSurgery-PAPER.pdf.

[12]. J. Cheng, S. H. Y. Wong, H. Yang, and S. Lu. "SmartSiren: virus detection and alert for smartphones," in Proceedings of the 5th international conference on mobile systems, applications, and services (Mobisys '07)., New York, NY, USA: ACM, 2007, pp. 258-271.

[13]. K. Sharma, T, Dand, T.Oh and W. Stackpole(2013). "Malware Analysis for Android Operating". 8th Annual Symposium on Information Assurance (Asia'13), June 4-5, 2013, Albany, NY.

[14]. K.H. Khan and M.N. Tahir, (2010) "Android Security, A survey. So far so good". Available: http://imsciences.edu.pk/serg/2010/07/android-security-asurvey-so-far-so-good/.

[15]. L. Xie, X. Zhang, J. Seifert, and S. Zhu, "pBMDS: a behavior-based malware detection system for cellphone devices," in Proceedings of the third ACM conference on wireless network security (WiSec '10)., New York, NY, USA: ACM, 2010, pp. 37-48.

[16]. L.Jeter, M. Mani and T. Reinschmidt, (2010) "Smart Phone Malware: The danger and protective strategies" Available:https://csel.cs.colorado.edu/~luje3922/Guardian2010-05-05.pdf.

[17]. M. Burns, (2012 September 5). Eric Schmidt: "There are now 1.3 million Android device activations per day" [Online]. Available: http://techcrunch.com/2012/09/05/eric-schmidt-there-are-now-1-3-million-android-device-activations-per-day/.

[18]. M. Rahman (2011). Reversing Android Malware", The Honeynet Project 10th Annual Workshop, ESIEA, Paris, France.

[19]. Malicious QR Codes Pushing Android Malware. https://www.securelist.com/en/blog/ 208193145 /Its time formaliciousQR codes.

[20]. Malware Threats", Available: http://www.scribd.com/doc/27444254/Android-Malware-Whitepaper.

[21]. One Year Of Android Malware (Full List). http://paulsparrows.wordpress.com /2011/08/11/ one-year-ofandroid-malware-full-list/.

[22]. R. Sen, (2011). "Android Applications", Available: http://entityx.com/wp-content/uploads/2011/02/daedalus_focus_android.pdf.

[23]. R.Xu, H.Saidi and R. Andreson, (2011). Aurasium: Practical Policy Enforcement for Android Applications.

[24]. T. Vennon, (2010). "Android Malware: A Study of Known and Potential.

[25]. TrendMicro. http://www.virustotal.com/.

[26]. W. Enck, D.Octeau, P. McDaniel and S. Chaudari, (2011). "A Study Of Android Application Security". Available: http://www.enck.org/pubs/enck-sec11.pdf.

[27]. X. Jiang, "An Evaluation of the Application Verification Service in Android 4.2," 10-Dec-2012. [Online]. Available: http://www.cs.ncsu.edu/faculty/jiang/appverify/.

[28]. Y. Zhou and X.Jiang, (2012). "Dissecting Android Malware: Characterization and Evolution". Department of Computer Science, North Carolina State University.

[29]. Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh,(2011). "TamingInformation-Stealing Smartphone Applications (on Android)," in Proceeding of the 4th International Conference on Trust and Trustworthy Computing, 2011.Malicious Mobile Threats Report 2010/2011.

[30]. ZeuS-in-the-Mobile - Facts and Theories. http://www.securelist.com/en/analysis/204792194/ZeuS in the Mobile Facts and Theories.