2013 27th International Conference on Advanced Information Networking and Applications Workshops

Android SMS Malware: Vulnerability and Mitigation

Khodor Hamandi Ali Chehab Imad H. Elhajj Ayman Kayssi
Department of Electrical and Computer Engineering
American University of Beirut
Beirut 1107 2020, Lebanon
{kmh09, chehab, imad.elhajj, ayman}@aub.edu.lb

Abstract— **In this paper, we study some messaging design decisions which resulted in a set of vulnerabilities in the Android operating system, and we demonstrate how a malware application can be built to abuse these vulnerabilities. The application presents itself as a regular SMS messaging application and uses its basic permissions to send/receive short messages. Since many operators worldwide provide services that allow users to transfer credits/units through SMS, the application abuses this service to transfer credits from users illegally. The "permission" subsystem, the "broadcast receiver" subsystem, and the message-sending mechanism contribute to forming a haven for SMS malware by granting them absolute control over sending, receiving, and hiding SMS messages. Accordingly, the malicious application hides any acknowledgments from the telecom operator that might appear after a credit transfer transaction. This enables malware to drain the balance of the attacked user and has the potential to cause damage to a large number of users as well as telecom operators. The application was demonstrated on a local operator and it successfully passed standard screening procedures that claim to catch malware. A set of possible solutions are also presented in order to mitigate the risks of such attacks.**

Keywords - Android; Vulnerability; Malware; SMS; credit transfer; Permission; Broadcast Receiver

I. INTRODUCTION

The digital mobile telecommunication is now in its third decade and is steadily progressing. The advancement in telecom has touched all its components including the mobile stations. These mobile stations are cellphones that were meant originally to do phone calls over Circuit Switched (CS) networks and to provide basic text services like the Short Message Service (SMS). These limited-features devices evolved to become smartphones with enhanced capabilities and this led to their wide proliferation across the globe. In 2011, reports estimated that close to 500 million smartphones were shipped, which represents an increase of more than 60% over 2010 and at the same time representing over 30% of all shipped mobile phones [1, 2]. Among these smartphones, over 48% were based on the Android operating system (OS) [3]. Moreover, by 2011, three billion mobile applications were downloaded [4], while in 2012, 1.5 billion applications are downloaded each month [5]. Android is, by design, an *open* OS; users are able to access its source code and can use the publicly-available application programming interfaces (API) to build applications and publish them on the Android application market [5]. This philosophy has enriched the Android market with over 675,000 applications by September 2012 [6], but has opened the door for a large variety of malware. In fact, eight million new mobile applications were identified by McAfee as malware between April and June of 2012 [7]. The report [7] added that the newly-discovered problems emanated from: SMS-based malware, mobile botnets, spyware, and destructive Trojans [8]. Trojan!SMSZombie, an SMS android Trojan, discovered in July 2012, was able to infect 500,000 phones in China. Many financial transactions and payments are processed using SMS in China; the ability to intercept SMS payloads and to have the privileges to send SMS messages has granted this Trojan the capability to execute numerous attacks [9]. In September 2012, a malicious Android game application (or simply app) was detected and its developers were fined 77,500 USD. This app used SMS in order to make users subscribe to a non-free service and was estimated to have collected 397,000 USD [10]. Also in the same month, FakeInst was discovered. This Trojan masquerades as a basic text exchange app while secretly subscribing to premium rate services by sending SMS messages silently. This Trojan was reported to have stolen 10 billion USD [11].

In this paper, we discuss the main features and vulnerabilities of the Android OS that allow the development and infection of SMS malware. In addition, we demonstrate how Android-based smartphones can be exploited by deploying a malware that uses the SMS service as its medium of operation. Typically, this type of malware relies on vulnerabilities from two parties: the first is telecom-operator dependent, while the other is Android-specific. For the former, a good number of mobile operators use SMS text messaging to transfer units/credits between two mobile users without requiring any form of validation/authorization beyond the message being sent from the phone. The units/credits refer to the user balance or airtime that can be used to make phone calls, to send/receive SMS messages, or to access the Internet. For example, a user that wants to transfer units builds a structure-defined text by entering two elements: the amount they want to transfer from their balance and the mobile number of the beneficiary. After drafting the message, the user sends it to a specific number and the transaction is completed by the operator through balance transfer. This can be extended to other potential financial transactions that are made through SMS as well. As for the Android-specific vulnerabilities, the problem consists of two design features relating to the way SMS messages are sent and received on Android. To demonstrate these vulnerabilities, we were able to build and test a proof-of-

978-0-7695-4952-1/13 $26.00 © 2013 IEEE
DOI 10.1109/WAINA.2013.134

concept malicious mobile app. This malware masquerades as a normal messaging application while in reality it would be covertly conducting credit transfers without user knowledge, while at the same time suppressing any confirmation messages received from the operator.

Based on a survey we performed, we identified more than 20 mobile operators that use such a service or a variation of it for unit/credit transfer between users. These operators are spread geographically in 28 countries (some operators have presence in more than one country). As such, the number of vulnerable users and operators at risk is significant.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 presents some Android background relevant to the work done, and section 4 presents our application design. We detail the application testing and analysis in sections 5 and 6, respectively. Finally, we present our proposed solutions in section 7, and we conclude the paper in section 8.

II. Related Work

Golde was able to find a number of vulnerabilities in SMS implementations that are used by the majority of the feature phones on the market despite the closed-source nature of these phones' operating systems [12]. He showed how SMS vulnerabilities can be used to disconnect the phone from the network, end calls, crash, and reboot. In addition, Denial-of-Service cases were also seen because of some tests that made the phone crash before the SMS is acknowledged, so the network was under the impression that the message did not get delivered and hence continuously re-transmitted it. Moreover, he showed how a SIM Data Download (a management tool used by operators to remotely manage SIM cards), through which SMS is directly sent to SIM (or USIM), can be manipulated so the attacked phone will send SMS from the phone to any number the attacker specifies, a process through which user units/credits can be drained slowly. Furthermore, he showed how this last feature can be used to carry out a Denial-of-Service on a specific mobile number. Traynor et al. demonstrated how SMS messaging can be malicious and harmful to the network [13]. Many mobile operators provide an Internet-based SMS service through which users can send SMS messages directly from the web to a mobile phone connected to the operator network. This service, if exploited, can lead to Denial-of-Service and thus prevent mobile users from making phone calls in a targeted city. Mulliner et al. presented a general framework that can be used specifically with smartphones for testing and monitoring of SMS messages [14]. Although many smartphones were investigated in [14], our main interest is the Android-based ones. In their work, they introduced a way to inject messages and monitor telephony by modifying the serial line that the Radio Interface Layer uses to communicate with the modem.

From a different perspective, the Android permission system has been under study and proved to be vulnerable to privilege escalation attacks. Davi et al. presented a method whereby applications can place phone calls without having such a privilege [15]. Although they added that this problem was solved, they showed a proof-of-concept scenario through which much more harm can be done whereby a "non-privileged vulnerable application" was used to execute Tcl commands, which ended up sending some 50 SMS messages to any number the attacker specified. As a result, not understanding the difference between different Android permissions can put users at risk. This was also confirmed in [16] where an online survey and interviews were conducted with a group of Android users and the results showed that only a minority of these users were able to understand the reason for and the difference between the various permissions required by applications. For SMS in particular, it is very important to understand the difference between the four available permissions: "SEND_SMS", "RECEIVE_SMS", "READ_SMS", and "WRITE_SMS".

In [17], 1,260 Android malware samples were captured and evaluated. 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. Results have shown that 21 malware families of the 49 captured listen for incoming SMS messages, the second most used broadcast after boot broadcast. Even more, 45.3% of the malware samples "tend to subscribe to premium-rate services with background SMS messages." In addition, some were found to do some filtering of SMS messages and even some were discovered to reply to the received messages. Furthermore, the antiviruses were not able to demonstrate a full ability to detect malware beyond a best case detection rate of only 79.6%.

In addition, the authors of [18] were able to develop a malware in the form of a native "Linux binary" that can be stored in an image, for example, and that can be used to bypass Android permissions. Bypassing a specific permission allows an intruder to do any operation, including sending SMS messages.

III. Android Background

Android provides developers with a Software Developer Kit (SDK) that exposes the API needed to build applications. A comprehensive documentation along with the SDK is provided on a dedicated website [19]. In what follows, we will describe selected topics from the SDK that are relevant to our work.

A. Activities

In Android, an activity is an essential component for an application. Every application should at least have one activity, the "main" activity. Usually, an application has many activities and it can start another activity, where each activity holds a state (e.g. stopped or paused). An activity is the component that provides a user with a graphical user interface (GUI) [19].

B. Services

Services are components in Android that do not provide any user interface, and usually run in the background conducting long running operations. Services are started by other application components, so an activity or a service can

start a service, and a service life is independent of the component that started that service [19].

C. Permissions

For security reasons, some subsets of the API are not accessible by an application without acquiring permission for it. Usually, this permission-granting must be statically declared in a "Manifest.xml" file when developing the application. These permissions can be used for many reasons such as filtering in the Android store, notifying the user (at installation time only), and guarding these critical APIs from malicious use [19].

D. Broadcast Receiver

Broadcast receiver is a mechanism that defines how Android forwards data to applications. The main usage of these broadcast receivers is inter-process communication and tracking of specific events (e.g. arrival of an SMS message to the phone). Applications declare statically or dynamically their interest in receiving a certain type of information and accordingly the OS will try to deliver the requested data when available. For the procedure of sending information, Android uses "Intents" which are data structures that should be passed to "sendBroadcast", for example. Android defines two types of Broadcast Receivers: normal and ordered. The normal ones are asynchronous and there is no defined order according to which users registered to a broadcast would receive the data. As for the ordered ones, a priority can be set to require from the system to deliver the information to each app in a certain sequence, and as such, some apps will get the information before others. This feature allows developers to capture and possibly modify the carried data before it reaches lower-priority consumers. In this case, an app can prevent other apps from getting specific data by aborting the received data [19]. For a given broadcast, a set is maintained of all the registered apps:

$$\Re = \{a_0, a_1, a_2, a_3, \dots\}$$

where the a_i are the apps.

If the broadcast is normal, the assumption is that all the set elements will obtain the exact unmodified copy of the information. On the other hand, if the broadcast is ordered there is no guarantee that more than a *single* app will get the original information. According to our experiments, for an app to ensure that it will obtain an ordered broadcast, it has to be the first application to register to the desired intent with the highest priority. It is worth noting that an app can register for an intent with any priority it specifies with no constraints or limitations after being given permission.

E. SMS Manager

The SMS manager, part of the Android telephony stack, provides developers with the necessary functions to send messages. In order to send a text message, and apart from getting the right permissions, an app can send a text message at any time by a simple function call. The main function to send SMS messages is "sendTextMessage" [19]. Calling the send function displays no notifications on the phone; the sending process is seamless and transparent to mobile users.

F. Logcat

Android has a special logging system through which the OS stores the logs in several circular buffers (for radio, events, and main). Developers can benefit from these buffers to get information from the system and debug their apps. For that purpose, a special command (*logcat*) can be used in a tool called the Android Debug Bridge (ADB) to extract data from the targeted buffer [19].

IV. Application Design

In this section, we describe the proof-of-concept malware application. As stated previously, the application is designed to look like a normal SMS application that has the basic required ability to send and receive SMS. There are in fact many such applications for Android users, and many of them are popular due to the fact that users can replace the native plain SMS application with more sophisticated and user-friendly SMS applications. The popularity of these applications demonstrates the feasibility and ease with which a malicious messaging application can be deployed simply by masquerading as a user-friendly SMS application.

Malicious code was added to the application in order to specifically target the unit/credit transfer through the SMS service. For that purpose, the application needs at least a single activity and three service components as shown in Figure 1.

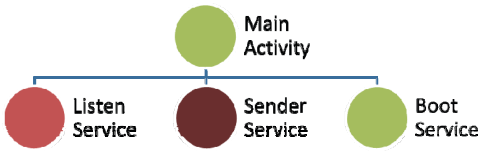

Figure 1: The components of the application

A. Main Activity

The Main Activity component has the entire graphical user interface to read and send SMS messages. On Android phones, SMS messages are inserted and read using database queries to the Content Provider *content://sms/*. In addition, this component is the base component that will launch the Listen Service and the Sender Service, at least for the first time. On the other hand, the Boot Service runs on its own after the first launch of the main activity.

B. Listen Service

This component listens for incoming SMS messages and takes actions according to pre-defined criteria. Since this service needs to listen for incoming messages, it has to be registered as a broadcast receiver.

Upon receipt of an SMS message from the radio, this component gets notified. Incoming messages are parsed and checked. If the newly received message is not an acknowledgment related to the "illegal" unit/credit transfer, the component allows the message to pass unmodified, or it

can directly insert it in the messaging database. If, on the other hand, this message is related to the malicious activity, it will be suppressed and will never reach the database or any other application. It is important to note that the broadcast that handles this transaction is an ordered broadcast. Accordingly, the priority option available for a registered broadcast receiver makes the suppression very efficient. This feature might be useful in filtering spam SMS messages, but based on its proven potential to cause harm, it tends to introduce a vulnerability in the Android OS. It is worth noting that the "Listen" service is made a sticky service; it will rerun, even if the user intentionally terminates it.

C. Sender Service

This component handles the unauthorized sending process of the SMS application. This service works in a silent manner. This is guaranteed by conducting a malicious action, such as a credit transfer, at random widely separated time instants in order to make the attack non-deterministic and undetectable by a simple observer of the phone activity from the operator side. This component will prevent its messages from being stored in the database; the user will not be able to track when the transfer was made by looking at the messaging database. In addition, this service is also sticky, similar to the Listen Service, and will rerun on its own even if it is intentionally terminated by the user. Several refinements can be added to this service to make it stealthier; for example, it can monitor the activity level of the user and then execute the malicious transfers during the busiest periods when the user is actually making phone calls and/or sending SMS. This will lower the likelihood of the user noticing the reduction in credit.

D. Boot Service

This component is needed to make the previously-mentioned services run at the launching of the OS. This is achieved by registering as broadcast receiver to BOOT_COMPLETED event.

E. Permissions

The minimum permissions needed to carry out the malicious activities are the "RECEIVE_SMS" and "SEND_SMS" which are requested by SMS applications. The most popular SMS applications surveyed on the Android market, at the time of writing of this work, additionally use the "READ_SMS" and the "WRITE_SMS" permissions. Therefore, the request for these permissions is not unusual and would not alert the user to the malicious behavior.

V. TESTING

In what follows, we demonstrate how we tested the application, the types of security checks that were performed, and the results that were obtained.

A. Implementation

Testing was conducted using two mobile phones. Since the application needs an Android-based smartphone to run, it is not a must to use two smartphones. We used a Samsung Galaxy SII smartphone and a Sony Ericsson K770i feature phone, as shown in Figure 2.

The application was installed on the "victim" phone, a Samsung Galaxy SII. The Android OS version pre-installed on this phone is 2.3.5 (Gingerbread), and the kernel version is 2.6.35. On the other hand, the Sony Ericsson holds the SIM card of the attacker which will get all the transferred credits. The credit transfer operation in the experiment is done by building a message that has the following format:

$$ReceiverNumber \parallel T \parallel Amount$$

This format is operator specific, and corresponds to one of the operators where the experiment was conducted. The message is usually sent to a special dedicated 4-digit number. Once the message is sent, the credits (*Amount* in US Dollars) are removed from the sender balance and added to the receiver balance. Finally, both sender and receiver get a message informing them that the credit transfer transaction was completed. It is worth noting that many operators charge transaction fees for this transfer. Therefore, we suspect that most operators would not, for financial purposes, suspend this service despite all its security concerns.

By design, the victim must not be informed of the transaction, accordingly the sent and received messages related to this operation are suppressed. We relied on the *logcat* tool to check that the operation was accomplished correctly. The output from the "main" buffer showed that the transfer is being carried out. A sample output is shown in Figure 3. Additional output from the "radio" buffer having the same timestamp confirmed the operation and a sample is shown in Figure 4.

At the time of writing this paper, the most downloaded Android SMS applications have millions of users. If any of these applications is designed to exploit the SMS credit transfer, or any other type of financial transactions, it can cause severe damage in a very short period of time. For example, the impact of 10 million users of a malware exploiting less than 1% of its users, could raise around $100,000 per month, by transferring only $1 per month per user. Such a small amount has a very low impact on a single user and hence it has a good chance of going unnoticed.

Figure 2: The two used phones in our test, left: Samsung Galaxy SII, right: Sony Ericsson K770i [20]

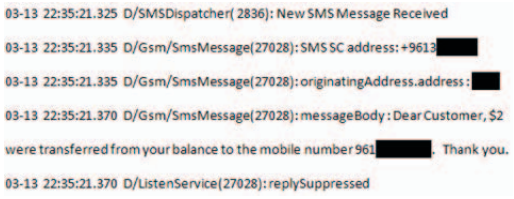

Figure 3: Main buffer showing logs of the receipt and suppression of SMS

```
03-13 22:35:15.210 D/RILJ   ( 2836): [2502]< SEND_SMS { messageRef = 6, errorCode = 0, ackPdu = null}
03-13 22:35:15.215 D/RILJ   ( 2836): [UNSL]< UNSOL_STK_SEND_SMS_RESULT {6}
03-13 22:35:15.220 D/STK    ( 2836): StkService: handleMsg : MSG_ID_SEND_SMS_RESULT
03-13 22:35:21.325 D/RILJ   ( 2836): [UNSL]< UNSOL_RESPONSE_NEW_SMS
03-13 22:35:21.325 D/SMS    ( 2836): android.telephony.SmsMessage.java - newFromCMT
03-13 22:35:21.375 D/RILJ   ( 2836): [2508]> SMS_ACKNOWLEDGE true 0
03-13 22:35:21.720 D/RILJ   ( 2836): [2508]< SMS_ACKNOWLEDGE
```

Figure 4: Radio buffer showing logs of the sending and receiving of SMS

B. *Antimalware*

In order to check if this malware is detectable, a freely available service called "Virus Total" was used. This service provides an online scanning for URLs and for files, including Android application files, using a set of the major antimalware products currently available on the market [21]. The report that was generated confirms that none of the 43 antimalware packages was able to detect the fact that this application is malicious. This is expected since most of these tools are signature-based.

C. *Android Market (Play Store)*

The final test was to check the response of the Android Market (Play Store). The test goal is to check whether the store can detect that the application executes malicious activities. For that purpose and with a modification to guarantee that the risk of accidental release of the application is minimal, the application was published successfully for a very short period of time; we then quickly unpublished it in order to make sure that it does not get downloaded by any user.

VI. Analysis

Three Android SMS Trojans were previously detected in China and the UK. Such malware was estimated to have stolen millions of dollars [9-11]. In previous sections, we demonstrated that at least 28 countries are at risk of a different type of SMS attack targeting a protocol for inter-mobiles credit transfer, resulting in stealing credits of subscribers if exploited. This leads to questioning the root cause of the attack. As can be remarked, this attack is multifaceted and many factors contribute to its feasibility.

The permission system incorporated in Android is a main contributor to the risk. Each SMS application has to be granted the permission to send/receive SMS messages but this decision is permanent. In this view, the OS does not work on guaranteeing the safe arrival or the approved sending of SMS messages. Instead, by using the permission system, these functions are delegated to users' downloaded applications. Accordingly, it is assumed that by downloading and using an application, the user has given a level of trust to this application and is fully aware of the granted permissions and their implications. Based on already-discovered vulnerabilities, it can be implied that these assumptions require a high level of user awareness to potential threats; a knowledge typical users might not always possess.

In addition, as we demonstrated above, the sending operation is not visible to the mobile subscriber. The user is never notified if an SMS is sent, and this can happen seamlessly once permissions are granted at install time. It is worth comparing this to other mobile operating systems, where an SMS cannot be sent without the user intervention by taking action such as clicking in a notification box.

The other root cause is the handling of the received messages, which is done using ordered broadcasts. As discussed previously, this gives a malicious application the ability to suppress or modify a payload for other applications installed on the phone or for the user.

In fact, our app used the permissions that most downloaded SMS applications acquire, which is considered minimal, and yet the app was capable of performing very harmful transactions because of the above Android design decisions that eventually led to serious vulnerabilities.

VII. Proposed Solutions

In this section, we propose practical permanent or temporary solutions to SMS-based malware. The solutions address the following issues:

- The permissions that give an application an absolute control over the time, the destination, and the decision of sending and receiving processes of SMS messages.
- The unsafe application-dependent sending of SMS.
- The ability to hide or modify the SMS payloads, accomplished through the use of ordered broadcasts.

Accordingly, the following are recommendations for in order to address these vulnerabilities:

- The user must always be notified/interrupted of the receipt of an SMS message.
- Applications must be prevented from controlling the receiving of SMS (and pre-empting another application from receiving messages) by setting its own priority.
- The user must grant explicit permission for every SMS send transaction, not only permission for the application at installation time.

Based on [14], a monitoring system can be added to monitor incoming and outgoing SMS messages by filtering the modem AT commands. For example, the +CMT command means that an SMS message has just arrived and sent an application-level notification. This solution would address the uncontrolled hidden sending and receiving of SMS, but would not stop them. In addition, it requires that the Android phones be rooted.

As for receiving SMS messages, a solution would be to install a trusted application before any SMS message application is installed, and assigning to this trusted app the highest priority for the broadcast receiver. Being the first app installed and having the highest priority, this app will get the notification of arrival of SMS messages before all other

applications, and will be able to notify the user of the arrival of SMS messages. This solution should work on all Android versions. It can be further enhanced by implementing the logic of the app as middleware that resides between the OS and all the other applications. This middleware would listen to all installed and removed applications and then checks for apps that request a permission to receive SMS messages. Starting with Android version 4, a feature was added in the OS to allow an application to send a broadcast to a particular application. Hence, the middleware gets the permission to broadcast SMS. Once an SMS is received, the middleware receives the message first and aborts the broadcast. Then it sends the SMS message to each of the *n* Apps, one by one. This would guarantee that none of the apps can hide, modify, or monopolize the receipt of SMS messages.

Finally, to mitigate the vulnerability of sending unauthorized SMS messages, Android can be patched in order to request user approval at each sending attempt.

VIII. CONCLUSION

In this paper, we studied the Android design features that led to the widespread proliferation of SMS malware. Our analysis highlighted three features that contribute to the vulnerability: the permission system, the broadcast receiver system and the sending process. The permissions granted to an app were demonstrated to give the app absolute ability to perform actions while no further checks are made. As for the broadcast receiver, the problem is in the ordered types of broadcasts which allow an app to drop, hide, or modify payload. The sending process used in Android is a function call that does not get approval for the sending operation. This allows apps to send unauthorized SMS messages. These design decisions, which can lead to vulnerabilities in the Android OS, were highlighted by presenting the design, development, and testing of a malicious prototype application that can be deployed on Android-based smartphones. This application appears as a regular SMS application while in the background it is using the SMS-based credit transfer service to carry out illegal transfers. We implemented and tested our application on an actual Android phone and the tests confirmed the objectives. The application was also checked by antimalware tools that were unable to detect any infection. In addition, the malicious application was successfully published on the official Android market. In summary, such an application would be difficult to detect and would cause substantial losses. Finally, we presented a number of possible practical solutions in order to mitigate the effects of the SMS vulnerabilities.

ACKNOWLEDGEMENT

This research is funded by TELUS Corporation. The authors would like to acknowledge the help of Mr. Gerard Touma who conducted the survey on mobile operators.

REFERENCES

[1] IDC. (2012). Smartphone Market Hits All-Time Quarterly High Due To Seasonal Strength and Wider Variety of Offerings, According to IDC. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23299912

[2] Strategy Analytics. (2012). Apple Becomes World's Largest Smartphone Vendor in Q4 2011. [Online]. Available: http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5170

[3] Canalys. (2012). Smart phones overtake client PCs in 2011. [Online]. Available: http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011.

[4] Daniel Ionescu. (2011). Google Brags About Android Market Stats. [Online]. Available: http://www.pcworld.com/article/225299/google_brags_about_android_market_stats.html

[5] Android. (2012, Sept). Android, the world's most popular mobile platform. [Online]. Available: http://developer.android.com/about/index.html

[6] Scott Lowe. (2012). Google Play celebrates 25 billion downloads with 25 cent apps, discounted books, music, and movies. [Online]. Available: http://www.theverge.com/2012/9/26/3409446/google-play-25-billion-downloads-sale

[7] Jeff Drew. (2012, Sept). Malware growth maintains rapid pace as mobile threats surge. [Online]. Available: http://www.journalofaccountancy.com/News/20126400.htm

[8] Hindustan Times (2012, Sept). Android users prime target of malware: McAfee. [Online]. Available: http://www.hindustantimes.com/technology/IndustryTrends/Android-users-prime-target-of-malware-McAfee/SP-Article1-925986.aspx

[9] Jon Russel. (2012). SMS Payment Virus Identified in China, 500,000 Android Device Infected. [Online]. Available: http://thenextweb.com/asia/2012/08/19/stealth-sms-payment-malware-identified-chinese-app-stores-500000-android-devices-infected/

[10] Charlie Osborne. (2012, Sept). SMS malware firm ordered to compensate victims. [Online]. Availability: http://www.zdnet.com/sms-malware-firm-ordered-to-compensate-victims-7000003639/

[11] Sara Yin. (2012, Sept). Will Your Android Device Catch Malware? Depends on Where You Live. [Online]. Available: http://securitywatch.pcmag.com/none/302362-will-your-android-device-catch-malware-depends-on-where-you-live

[12] N. Golde, "SMS Vulnerability on Feature Phones," Master Thesis, Berlin Institute of Technology, 2011.

[13] P. Traynor, W. Enck, P. McDaniel, and T. L. Porta, "Exploiting Open Functionality in SMSCapable Cellular Networks," in *Journal of Computer Security (JCS)*, 2008.

[14] C. Mulliner, C. Miller, "Injecting SMS Messages into Smart Phones for Security Analysis," in *Proceedings of the 3rd USENIX Workshop on Offensive Technologies (WOOT)*, 2009.

[15] L. Davi, A. Dmitrienko, A. Sadeghi, M. Winandy, Privilege escalation attacks on Android. In Information Security, 2011.

[16] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, (2012, Feb.) "Android Permissions: User Attention, Comprehension, and Behavior," Not published. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-26.pdf

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

[18] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli, "Smartphone malware evolution revisited: Android next target?" In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)*, 2009.

[19] Android Developers, (2012, Sept). Android Developers. [Online]. Available: http://developer.android.com/index.html

[20] GSMArena. (2012). GSMArena.com - GSM phone reviews, news, opinions, votes, manuals and more. [Online]. Available: http://www.gsmarena.com

[21] Virus Total. (2012). VirusTotal - Free Online Virus, Malware and URL Scanner. [Online]. Available: https://www.virustotal.com/