Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 24 (2016) 1436 – 1444

Procedia Technology

International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015)

Measuring Android App Repackaging Prevalence based on the Permissions of App

Sajal Rastogi, Kriti Bhushan, B. B. Gupta*

National Institute of Technology Kurukshetra, India

Abstract

Google Play is the official market of Android apps. The app publishers make money by selling apps, through in-app billing, and through advertisements. The apps, especially the popular ones, are disassembled by adversaries, who then add/replace ads in the apps, and/or add some malicious code to the apps, and then release it to app markets. This is called app repackaging. Any revenue these repacked-apps make on these ads go to the adversaries. Also, if the repackaged apps have malwares then the malwares now spread more swiftly because of the popularity of the apps. In this paper, we present our study on some Android apps released to unofficial markets which were originally released to Google Play to find how prevalent the repackaging of Android apps is. Moreover, we proposed a mechanism for the detection of repackaging based on the permissions of the apps. To evaluate the performance of proposed approach, we downloaded 50 apps, each with well over a hundred million downloads from the official Android market, and tried to find their repackaged versions on unofficial markets based on extra permissions. We found repackaged versions of 6 out of these 50 apps without such a naive approach. This just goes to demonstrate how widely available the repackaged versions of some of the most popular Android apps are. It also proves that, in many cases, it is possible to detect repackaging only by comparing the permissions of an app with its original version. To a wide extent, there is no need of complex code analysis, or adding some authentication entity such as a watermark to the app for deterring repackaging.

Peer-review under responsibility of the organizing committee of ICETEST – 2015

Keywords: Android; security; repackaging; permissions; mobile device; authentication; malware.

* Corresponding author. E-mail address: gupta.brij@gmail.com (Tel. +91-1744-233488)

Peer-review under responsibility of the organizing committee of ICETEST – 2015
doi:10.1016/j.protcy.2016.05.172

1. Introduction

Mostly, Android operating system is the used in today's smartphone devices [1]. An android app can be distributed through any medium the developer likes. Users obtain apps from unofficial markets as well as from the official one. A market may have some policy regarding the quality of the apps it releases, or it may not. Android provides no guarantee against the harms a third-party app may cause to the user. If a market provides no guarantee either, which is the case with many existing markets, then the user is installing apps from the market completely at his/her own risk. Also, if the market does not ensure that the app is original then someone may actually steal the revenue deserved by the original developer. The developers, therefore, are as much affected by app repackaging as the users. There are many techniques which focus on detection of the malwares in Android apps. Mostly, malwares use repackaging for spreading [2-3] and it might be more effective to directly detect repackaging.

There are many techniques exist which can detect repackaged apps. Most existing techniques do not take advantage of the fact that the majority of repackaged (trojanized) apps ask for more permissions than their original versions [4]. When adversaries repackage an app with different advertisements or with malicious code, they usually add some permissions also to the app. Detecting these extra permissions can be one of the most straightforward ways of detecting repackaging. In this paper, we proposed a mechanism for the detection of repackaging based on the permissions of the apps. To evaluate the performance of proposed approach, we downloaded 50 apps, each with well over a hundred million downloads from the official Android market, and tried to find their repackaged versions on unofficial markets based on extra permissions. We found repackaged versions of 6 out of these 50 apps without such a naive approach.

The paper is composed of the following sections. Section 1 highlights how popular Android is, and introduces the Android app repackaging issue. Section 2 briefs the security aspects of Android OS, examines some Android malwares, and then presents some basic knowledge about Android markets. Section 3 presents literature survey. Our study of prevalence of app repackaging is presented in detail in section 4. Section 5 has discussions on the experiments. Finally section 6, concludes the paper.

2. Android Security, Android malware, and Android markets

The Android operating system provides security to the user through a permission-based model. When the user installs an app, package installer prompts the user to grant the permissions asked by the app. A user has to grant all the permissions, or the user may choose not installing the app. There is also a signature in every Android app but it does not serve any security related purpose. Further, Android protects the data of one app from another by applying the technique of sandboxing. Every Android app has a manifest file, AndroidManifest.xml. It holds the permissions needed by the app among other things. If an app needs a permission from the user, it must be listed in the manifest file. These are exactly the permissions which are displayed to the user at the time of installation. When repackaging an app, if the adversary adds permission to the app then the manifest file of the repackaged app would certainly be different from that of original app.

2.1. Android Malware

There are many Android malwares that have been known to employ repackaging to spread. If they use a popular app, the spread is very quick. Following are some such malwares [12-32]: DroidKungFu,DroidKungFu*variants*,Endofday,Fakeangry,Fjcon,Geinimi,GeoFeeBot,GingerMaster,GoldDream,HipoSMS,Jifake,JSMSHider, Mania,Moghava,Pjapps, Plankton,Premiumtext,PushBot, YZHCSMS,Zitmo,Zsone.

2.2. Android App Markets

The default setting of Android is that it does not allow its users to install apps from any source other than the official market, Google Play. The user has to enable the "Allow installation of apps from unknown sources" option from the Security settings screen to be able to install apps from unofficial sources. There are many successful third-party Android markets such as SlideME, 1Mobile Market, Amazon App Store, Mobile9, F-Droid, Samsung Galaxy

Apps, Opera Mobile Store, Mobango, Soc.io Mall, GoAPK, GetJar, and Mobogenie. The benefits to the developer and to the users differ from market to market. Some markets are more user-friendly than others. They provide intelligent recommendations, advanced search features, etc. Some markets are more developer-friendly than others. They provide higher percentage of profit to the developer, better app promotion, etc.

3. Related Work

Zhou et al. [2] present their study of different traits of Android malwares including the modes of propagation and installation used by Android malwares. They observe that the most used mode is app repackaging. Vidas et al. [3] observe the same thing in their study. They observe that the majority of malware families in their dataset employed repackaging. There are several techniques which detect app repackaging by detecting how similar the source codes of the app is to some existing apps [4] [5]. Wu et al. [6] and Ren et al. [7] propose detection of app repackaging through a watermarking-based authentication mechanism. Some existing techniques [8] [9] [10] detect app repackaging based on similarity in data or control dependencies in the source code of the apps.

4. Measuring the prevalence of Android App Repackaging

Each market has a different set of policies it follows and due to differences in policies, different kinds of apps end up in markets. Some existing markets have much more repackaged apps then others [3]. Google Play scans apps and additionally, scrutinizes apps with negative user feedbacks. Other markets follow their own specific policies. Whether each and every app on a market has been tested as per its policies is never a guarantee.

We performed a very simple experiment. We downloaded 50 apps from the official Android market such that all of the apps have more than one hundred million downloads. After downloading these app from Google Plays, we searched the same app names on Mobogenie market (an unofficial market we randomly picked) and downloaded the apps with the same/similar names to obtain 50 more APKs. The resultant dataset has a total of 100 APKs. We then compared the permissions asked by each pair of APKs. The results and related observations are presented henceforth.

4.1. Dataset Used

The dataset consists of the APKs of the following apps downloaded from Google Play (the official Android app market), and Mobogenie markets: Adobe AIR(Adobe Systems); Adobe Flash Player(Adobe Systems); Adobe Reader(Adobe Systems); Amazon Kindle (Amazon.com); Angry Birds(Rovio Entertainment Ltd.); Angry Birds Rio(Rovio Entertainment Ltd.); Angry Birds Seasons(Rovio Entertainment Ltd.); AntiVirus Security – FREE(AVG Mobile); Avast Mobile Security & Antivirus(Avast); Barcode Scanner(ZXing Team); Blurb Checkout(Blurb Inc.); Candy Crush Saga(King); Clash of Clans(Supercell); Clean Master(Cheetah Mobile); Drag Racing(Creative Mobile); Dropbox(Dropbox, Inc.); DU Battery Saver(DU APPS STUDIO); eBay(eBay); ES File Explorer(ES APP Group); Facebook(Facebook); Facebook Messenger(Facebook); Farm Heroes Saga(King); Flipboard: Your News Magazine(Flipboard); Fruit Ninja Free (Halfbrick Studios); Glow Hockey(NatenaiAriyatrakool); GO Launcher EX(GO Launcher Dev Team); Hill Climb Racing(Fingersoft); Instagram(Instagram); Jetpack Joyride(Halfbrick Studios); KakaoTalk: Free Calls & Text(DaumKakao); LINE: Free Calls & Messages(LINE Corporation); Minion Rush(Gameloft); My Talking Tom(Outfit7); MX Player(J2 Interactive); Netflix(Netflix Inc.); Opera Mini web browser(Opera Software ASA); PicsArt Photo Studio(PicsArt); Pool Billiards Pro(TerranDroid); Pou(Zakeh); Shazam(Shazam); Shoot Bubble Deluxe(City Games LLC); Snapchat(Snapchat Inc.); Subway Surfers(Kiloo Games); Super-Bright LED Flashlight(Surpax Technology Inc.); Tango - Free Video Call & Chat(Tango); Temple Run(Imangi Studios); Temple Run 2(Imangi Studios);Viber: Free Calls & Messages(Viber Media, Ltd); WeChat(WeChatTencent); WhatsApp Messenger(WhatsApp Inc.).All of these apps are free on Google Play and have more than one hundred million downloads.

4.2. Results

We assume that the permissions asked by APKs from Google Play are legitimate. Their permissions are the base permissions against which permissions asked by APKs downloaded from other sources are compared in our experiment. This assumption has been further discussed in Section 5.

When the 50 apps listed in the dataset were analyzed for any extra permissions, differences were found among the permissions asked by APKs of 6 apps downloaded from Google Play and Mobogenie. One of the two APKs of the same app asked for extra permissions for the same functionality − for the six apps. Not surprisingly, for all the six apps, the APK from Mobogenie was the one asking for extra permissions. Table 1 mentions the six apps on Google Play which have apps with the same name on Mobogenie market asking for extra permissions. It is highly likely that all these APKs from Mobogenie market either have malicious code, or have some added or replaced advertisements, or have modified in-app billing parameters.

Pou is one of the six apps whose Mobogenie version asks for more permissions than its Google Play version. This comparison of permissions from two APKs can be automated. We used APKTool to decompress the APK files and obtained the manifest files (AndroidManifest.xml) from the APKs. Figure 1 shows part of the manifest file of Pou's APK downloaded from Google Play with all the uses-permissions tags. Figure 2 shows part of the manifest file of Pou's APK downloaded from Mobogenie with all the uses-permissions tags. Obviously, there are many more permissions in figure 2 than in figure 1.

Table 1.Apps with 100,000,000+ downloads on Google Play which also have an APK available on Mobogenie that ask extra permissions.

App (Developer)	Permissions asked by APK of the app downloaded from Google Play	Permissions asked by APK of the app downloaded from Mobogenie
Glow Hockey (NatenaiAriyatrakool)	VIBRATE INTERNET ACCESS_NETWORK_STATE WRITE_EXTERNAL_STORAGE	WAKE_LOCK READ_PHONE_STATE SET_ORIENTATION INTERNET ACCESS_NETWORK_STATE RECEIVE_BOOT_COMPLETED ACCESS_WIFI_STATE ACCESS_COARSE_LOCATION ACCESS_LOCATION_EXTRA_COMMANDS GET_ACCOUNTS com.android.launcher.permission.READ_SETTINGS
Pou (Zakeh)	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE WAKE_LOCK WRITE_EXTERNAL_STORAGE RECORD_AUDIO BLUETOOTH BLUETOOTH_ADMIN com.android.vending.BILLING	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE WAKE_LOCK WRITE_EXTERNAL_STORAGE RECORD_AUDIO BLUETOOTH BLUETOOTH_ADMIN com.android.vending.BILLING READ_PHONE_STATE READ_EXTERNAL_STORAGE com.fede.launcher.permission.READ_SETTINGS RECEIVE_BOOT_COMPLETED WRITE_SETTINGS

Angry Birds Seasons (Rovio Entertainment Ltd.)	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE READ_PHONE_STATE WRITE_EXTERNAL_STORAGE WAKE_LOCK GET_ACCOUNTS com.android.vending.BILLING com.google.android.c2dm.permission.RECEIVE com.google.android.c2dm.permission. REGISTRATION	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE READ_PHONE_STATE WRITE_EXTERNAL_STORAGE WAKE_LOCK GET_ACCOUNTS com.android.vending.BILLING com.google.android.c2dm.permission. RECEIVE com.google.android.c2dm.permission. REGISTRATION READ_EXTERNAL_STORAGE com.fede.launcher.permission.READ _SETTINGS RECEIVE_BOOT_COMPLETED WRITE_SETTINGS SYSTEM_ALERT_WINDOW GET_TASKS
Angry Birds Rio (Rovio Entertainment Ltd.)	(same as that for Angry Birds Seasons)	(same as that for Angry Birds Seasons)
Super-Bright LED Flashlight (Surpax Technology Inc.)	CAMERA FLASHLIGHT CHANGE_CONFIGURATION WRITE_SETTINGS WAKE_LOCK ACCESS_NETWORK_STATE INTERNET com.google.android.c2dm.permission.RECEIVE	ACCESS_NETWORK_STATE ACCESS_WIFI_STATE READ_PHONE_STATE WRITE_EXTERNAL_STORAGE READ_EXTERNAL_STORAGE INTERNET RECEIVE_BOOT_COMPLETED WRITE_SETTINGS RESTART_PACKAGES CAMERA FLASHLIGHT CHANGE_CONFIGURATION WAKE_LOCK com.google.android.c2dm.permission. RECEIVE
Pool Billiards Pro (TerranDroid)	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE READ_PHONE_STATE WRITE_EXTERNAL_STORAGE com.android.vending.BILLING	INTERNET ACCESS_NETWORK_STATE ACCESS_WIFI_STATE READ_PHONE_STATE WRITE_EXTERNAL_STORAGE READ_EXTERNAL_STORAGE RECEIVE_BOOT_COMPLETED WRITE_SETTINGS GET_TASKS com.android.vending.BILLING com.fede.launcher.permission.READ

		_SETTINGS
Glow Hockey (NatenaiAriyatrakool)	VIBRATE INTERNET ACCESS_NETWORK_STATE WRITE_EXTERNAL_STORAGE	WAKE_LOCK READ_PHONE_STATE SET_ORIENTATION INTERNET ACCESS_NETWORK_STATE RECEIVE_BOOT_COMPLETED ACCESS_WIFI_STATE ACCESS_COARSE_LOCATION ACCESS_LOCATION_EXTRA_COMMANDS GET_ACCOUNTS com.android.launcher.permission.READ_SETTINGS

```
<manifest android:installLocation="auto" package="me.pou.app" platformBuildVersionCode
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="com.android.vending.BILLING"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO"/>
  <uses-feature android:name="android.hardware.microphone" android:required="false"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.BLUETOOTH"/>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
  <uses-feature android:name="android.hardware.bluetooth" android:required="false"/>
```

Fig. 1. Part of manifest file of Pou gaming app downloaded from Google Play showing permissions required by the app

```
<manifest android:installLocation="auto" package="me.pou.app" platformBuildVersionCode='
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="com.fede.launcher.permission.READ_SETTINGS"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="com.fede.launcher.permission.READ_SETTINGS"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="com.fede.launcher.permission.READ_SETTINGS"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="com.android.vending.BILLING"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO"/>
  <uses-feature android:name="android.hardware.microphone" android:required="false"/>
```

Fig. 2. Part of manifest file of Pou gaming app downloaded from Mobogenie showing permissions required by the app

5. Discussion

First of all, the authenticity of the apps available on Google Play, that their permissions are legitimate, is an assumption in our experiment. However, this assumption is good to work with because Google takes various measures to protect the Android users. It removes any app which is found too similar to an existing app. Also, Google reviews the apps it receives complains about. Mostly, the real problem is caused by many Android app markets other than Google Play.

Ideal Condition: The ideal condition for a permission-based repackaging detection effort would be that developers provide the original permissions needed by their apps. We claim that the developers would go the extra mile to provide the permissions and prevent the repackaging of their apps. The fact that the developer has monetary incentive if the users of his/her app do not use a repackaged version is enough to ensure that the developer would provide the manifest information of their apps. Let us examine the following four exhaustive scenarios. [Scenario 1] The app is a paid app: If the users do not install the repackaged version for their own safety that it is likely that they would look for the original app and would, eventually, find it. So, the real developer makes money not some app plagiarist. [Scenario 2] The app is a free app with advertisements inside: Again, if the users find the original app, the revenue from advertisements would go to the real developer. In repackaged apps, plagiarists redirect the ad revenue. This is the most common motivation of the plagiarists [4] [10] [11]. [Scenario 3] The app is free with in-app billing: Inside most apps (particularly gaming apps), users can buy virtual commodities by paying real money. This revenue

again can be redirected by the plagiarists. If repackaging is hampered, the real developer makes the money. [Scenario 4] The app is free and the developer makes no money through it: Even in this case the developer would not like his/her app to be repackaged because the reputation of the developer is at stake. If an app associated with a developer/publisher is known to have many malicious variants, it is likely that the users are going to refrain from using any apps released later by that developer/publisher.

6. Conclusion

There is no doubt that the dataset could be larger. Larger dataset supports the claim more strongly. Though, the experimentation on the dataset used has satisfactorily proved all the points we set out to prove. Firstly, the repackaging is rampant in alternative Android markets. Out of just 50 apps from Google Play, repackaged versions of 6 apps were found on Mobogenie market. Secondly, the higher the popularity of the app, the higher the likelihood that an alternative market will have a repackaged version of it. Note that apps with more than one hundred million downloads were included in the dataset. Thirdly, most repackaged versions ask for more permissions. Lastly, the most important point that in many cases it may be possible to detect repackaging only on the basis of the permissions of apps.

Acknowledgements

The authors extend their thanks to the anonymous reviewers for their valuable feedbacks and suggestions.

References

[1] International Data Corporation, Smartphone OS Market Share, Q2 2014. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Last Accessed: February 10, 2015]

[2] Zhou, Y., and Jiang, X. Dissecting Android Malware: Characterization and Evolution. In Proceedings of the 33rd IEEE Symposium on Security and Privacy (SP), Pages: 95-109, San Francisco, CA, May 2012.

[3] T. Vidas, and N. Christin.Sweetening Android Lemon Markets: Measuring and Combating Malware in Application Marketplaces in the proceedings of the third ACM conference on Data and Application Security and Privacy, CODASPY '13, pp. 197-207, New York, NY, USA, February 2013.

[4] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. in the proceedings of the second ACM Conference on Data and Application Security and Privacy, pp. 317-326, New York, NY, USA, February 2012.

[5] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang. Plagiarizing Smartphone Applications: Attack Strategies and Defense Techniques.in the proceedings of the 4th International Conference on Engineering Secure Software and Systems, ESSoS'12, pages 106–120, Berlin, Heidelberg, 2012. Springer-Verlag.

[6] Z. Wu, X. Zhang, and X. Jiang.AppInk: Watermarking Android Apps for Repackaging Deterrence. in the proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, pp. 1-12, New York, NY, USA, May 2013.

[7] C. Ren, K. Chen, and P. Liu. Droidmarking: Resilient Software Watermarking for Impeding Android Application Repackaging. in the proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, pp. 635-646, New York, NY, USA, September 2014.

[8] J. Crussel, C. Gibler, and H. Chen. AnDarwin: Scalable Detection of Semantically Similar Android Applications.in the proceedings of the European Symposium on Research in Computer Security − ESORICS 2013, pp. 182-199, 2012. Springer.

[9] K. Chen, P. Liu, and Y. Zhang. Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets.in the proceedings of the 36th International Conference on Software Engineering, ICSE '14, pp. 175-186, New York, NY, USA, May 2014.

[10] J. Crussell, C. Gibler, and H. Chen. Attack of the Clones: Detecting Cloned Applications on Android Markets. in the proceedings of the European Symposium on Research in Computer Security Computer Security – ESORICS 2012, pp. 37–54, 2012. Springer.

[11]W. Zhou, Y. Zhou, M. Grace, X. Jiang, and ShihongZou. Fast, Scalable Detection of "Piggybacked" Mobile Applications. in the proceedings of the third ACM conference on Data and Application Security and Privacy, CODASPY '13, pp. 185-196, New York, NY, USA, February 2013.

[12]Security Alert: New Sophisticated Android Malware DroidKungFu Found in Alternative Chinese App Markets. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html. [Last Accessed: September 12, 2014].

[13]Security Alert: New DroidKungFu Variants Found in Alternative Chinese Android Markets. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu2/. [Last Accessed: September 12, 2014].

[14]Android Threat Set to Trigger On the End of Days, or the Day's End. [Online]. Available: http://www.symantec.com/connect/blogs/android-threat-set-trigger-end-days-or-day-s-end. [Last Accessed: September 12, 2014].

[15]Android.Fakeangry. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-022823-4233-99&tabid=2. [Last Accessed: September 12, 2014].
[16]Security Alert: New Android Malware Found Targeting Custom ROMs (Fjcon). [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/Fjcon/. [Last Accessed: September 12, 2014].
[17]Virus: Android/Geinimi.A!tr. [Online]. Available: http://www.fortiguard.com/encyclopedia/virus/#id=2374726. [Last Accessed: September 12, 2014].
[18]Security Alert: GeoFeeBot–The Biggest Malware Threat in 2012?. [Online]. Available: http://research.nq.com/?p=340. [Last Accessed: September 12, 2014].
[19]GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3 (Gingerbread). [Online]. Available: http://www.cs.ncsu.edu/faculty/jiang/GingerMaster/. [Last Accessed: September 12, 2014].
[20]Security Alert: New Android Malware -- GoldDream -- Found in Alternative App Markets. [Online]. Available: http://www.cs.ncsu.edu/faculty/jiang/GoldDream/. [Last Accessed: September 12, 2014].
[21]Security Alert: New Android Malware -- HippoSMS -- Found in Alternative Android Markets. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/HippoSMS/. [Last Accessed: September 12, 2014].
[22]Android.Jifake. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-073021-4247-99. [Last Accessed: September 12, 2014].
[23]Security Alert: Malware Found Targeting Custom ROMs (jSMSHider). [Online]. Available: https://blog.lookout.com/blog/2011/06/15/security-alert-malware-found-targeting-custom-roms-jsmshider/. [Last Accessed: September 12, 2014].
[24]Tracking Android/Foncy. [Online]. Available: http://blog.fortinet.com/post/tracking-android-foncy. [Last Accessed: September 12, 2014].
[25]Android.Moghava: A Recipe for Mayhem. [Online]. Available: http://www.symantec.com/connect/blogs/androidmoghava-recipe-mayhem. [Last Accessed: September 12, 2014].
[26]Android Threats Getting Steamy. [Online]. Available: http://www.symantec.com/connect/blogs/android-threats-getting-steamy. [Last Accessed: September 12, 2014].
[27]Security Alert: New Stealthy Android Spyware -- Plankton -- Found in Official Android Market. [Online]. Available: https://www.csc.ncsu.edu/faculty/jiang/Plankton/. [Last Accessed: September 12, 2014].
[28] Symantec, Android.Premiumtext. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-080213-5308-99. [Last Accessed: September 12, 2014].
[29]Security Alert: PushBot -- A Push-Based App Delivery Model Identified in the Wild. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/PushBot/. [Last Accessed: September 12, 2014].
[30]Security Alert: New Android SMS Trojan -- YZHCSMS -- Found in Official Android Market and Alternative Markets. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/YZHCSMS/. [Last Accessed: September 12, 2014].
[31]Teamwork: How the ZitMo Trojan Bypasses Online Banking Security. [Online]. Available: http://www.kaspersky.co.in/about/news/virus/2011/Teamwork_How_the_ZitMo_Trojan_Bypasses_Online_Banking_Security.
[32]Security Alert: Zsone Trojan found in Android Market. [Online]. Available: https://blog.lookout.com/2011/05/security-alert-zsone-trojan-found-in-android-market/. [Last Accessed: September 12, 2014].