Android Malware And its Analysis Techniques

Olawale Surajudeen Adebayo
[1]Computer Science Department, International Islamic University Malaysia, [2]CSS Department, Federal University of Technology Minna, Nigeria
[1]adebayo.olawale@live.iium.edu.my,
[2]waleadebayo@futminna.edu.ng

Normaziah Abdul Aziz
Computer Science Department, International Islamic University Malaysia
naa@iium.edu.my

Abstract— **The more android operating system on the smartphones is becoming more acceptable due to its market openness and easier accessibility and operability, the more it is increasingly targeted by malware. This research examines several attack vectors on an android smartphone and its analysis in order to identify and obtain useful features for analysis and classification and to understand the attack vectors and offer possible solutions. This work recommends appropriate practices to ensure the security of information on an android smartphone.**

Keywords*— *Android Malware; Malware Analysis; Static Analysis; Android Smartphones; Malware Writer

I. Introduction

Malwares are malicious applications or software targeted at operating systems or internet. Mobile malwares are malicious software specifically designed to target the mobile operating system. Android malwares are malicious executable program designed to hamper the normal operation of android operating system on smartphones. It is for various reasons like financial gain, challenges or system testing and information stealing. Efforts have been made by several researchers to develop malware detection system to address the attack and effects of malware on smartphones. Detection systems include antimalware algorithm, intrusion detection system, among others to address the malware attack. However, old malware detection bank based on the signatures, IP addresses and anomaly behaviours to detect malware may not necessarily work for new malware that are developed using new techniques.

Android operating system being an open source OS can be obtained through its official market Google play [31] and other open market i.e. Amazon Appstore [29], GetJar [30]. The most popular operating system (OS) that allow smartphones to perform most of the functions available on the internet and desktop computer is Android by Google. This paper examined the trend of malware on the Android phone due to its acceptability and ubiquity. As anti-malware writers are successfully developing various algorithms against existing malware, malware writers also continue to change their stealthy and obfuscation techniques to hide their malicious codes. The task of preventing mobile facilities therefore lies on the device's security mechanism, and the stakeholders' knowledge. For example, some malwares rely on user interaction before their execution, while others exploit the bugs in the operating system of important facilities. Analysis of malware has to do with identifying it by examining the program semantics of malware code statically [32] or dynamically [28] and using the attributes of known malware characteristics for interpretation [1].

The major contribution of this research is analyzing android malware to identify and obtain useful features through static code analysis for classification and detection. The experiment is also to examine several attack vectors targeting android operating system to proffer appropriate solutions and recommendation. The remaining sections of the paper are organized as follows: Section II discusses the existing related researches; Android framework and exploit were discussed in section III and IV respectively; Section VI and VII discussed the malware analysis techniques and android malware experiment. Section VIII, IX and X discussed research benefit, future works, and recommendations respectively.

II. Related works

Malicious programs over time present an incessant threat to the privacy and security of sensitive data and the availability of critical services at crucial point in time [1]. The first observable feature adopted by malware most detector at the outset of smartphone is battery power consumption [9, 23, 24]. The technique was basically to observe the mobile phone power consumption and compares it with the normal power consumption in order to detect occurrence of anomaly. The first malware specifically written for Symbian OS platform was discovered in 2004 [6]. After the infection successfully carried out by Cabir malware and its variants [8], researchers proposed several approaches and developed different mechanisms in order to detect malware on a mobile phone. With the advent of smartphones availability, malware has consistently double on the mobile phone due to its ubiquity.

The F-Secure's Threat report [29] stated that the number of Android malware has been doubling yearly since 2011 up to the first quarter of 2013. In a bid to curtail the effect of malware on android smartphone, Mohammad Karami et.al [17] in a paper titled Behavioral Analysis of Android Applications Using Automated Instrumentation, presented effective security inspection mechanisms for identification of malicious applications for Android mobile applications, where they developed a comprehensive software inspection framework for android smartphone. This system only works on android phone. Another work by Abhijit, B. et al. [3] proposed a behavioural detection on mobile handsets using a support vector machine (SVM) algorithm to build a behavioural malware detection system that work on a platform dependent operating system like Symbian phones. In the same vein, the behavioural-based Android malware detection system by Abela, Kevin Joshua L. et al. [2] was designed to categorize different Android applications in the market.

T. Bläsing et al. [24] also developed an Android Application Sandbox system for suspicious software detection using dynamic, single API, clustering and fake API injection techniques. This application only works on android platform. Thomas Eder et al. [25] developed an expandable and modular framework for analyzing Android applications called ANANAS. The application takes care of common needs for dynamic malware analysis and provides an interface for the development of plugins a framework for analysing Android applications. Suhas Holla and Mahima M Katti [22] discussed Android mobile platform for the mobile application development, layered approach and the details of its security information. Andrew Walenstein et al. [4] proposes an approach for selecting features of mobile malware by using knowledge of malicious program structure to heuristically identify malicious portions of any applications.

Dinesh Shetty [7] also analysed a malware called iCalender.apk using static analysis and found out that this malware was designed to send suspicious messages to a designated number for a malicious purpose without the knowledge of the victim. According to the static and dynamic analysis carried out on DroidKungFu2-A using a reverse engineering technique, [28] was able to identify DroidKungFu2-A as a malware designated to send an unauthorized message from legitimate user's mobile to a certain number. Geinimi also according to [28] was a bot that connect with a remote server and delete and steal sms, and make an unauthorised call. Hieu Le Thanh et al. [10] based on the samples classified malware into existing families or addition of a new family using collection of 58 malware families and 1485 malware samples and introduced three different techniques to analyse the samples.

Michael Grace et al. [15] proposed a proactive scheme to spot zero-day Android malware by relying on malware samples and their signatures, by specifically developed an automated system called RiskRanker to scalably analyze whether a particular app exhibits dangerous behaviour (e.g., launching a root exploit or sending background SMS mes- sages). Radek Vala et al. [19] examined the recent advances in reverse engineering popular mobile operating systems, namely iOS and Android. Iker Burguera et al. [11] developed a framework for collection of traces from an unlimited number of real users based on crowdsourcing that was used to analyze the data collected in the central server using two types of data sets: those from artificial malware created for test purposes, and those from real malware found in the wild (malware contained in Steamy Window and Monkey Jump 2 applications). This method shows to be an effective means of isolating the malware and alerting the users of a downloaded malware.

Steven Meyer [21] examines the development phases (submission, update, usage, development, validation, download and ranking), and the misbehaviour that a developer could do like sandbox escape, developer impersonation, multiple download and ranking, social engineering etc. Qiang Yan et al. [18] study the state of-the-art of mobile malware, as well as the progress of academic research and industrial effort against mobile malware. This research also analyze three potential directions for effective malware detection and prevention on mobile phones

William Enck et al. [27] developed a Taintdroid to provide users with adequate control over and visibility into how third-party applications use their private data. The framework analyses the behaviour of application using sufficient contextual information about the data that leaves a device and its destination. Anubis [5] is the Windows operating system malware analysis version that analyse applications on a windows for malware. A modified version of Anubis for Androids applications is called Codename Andrubis [14]. This malware detector on the Androids however lacked in public expository of its inner architecture for further research analysis. Droidscope [12] is another malware analysis tool built on top of QEMU and relies on introspection of an emulated system running the application in question. This detector uses dynamic analysis techniques to detect malware and exports APIs. Min Zhao et al. [16] uses artificial immunology to develop a framework for malware detection based on dynamic behaviours of smartphone applications.

III. ANDROID APPLICATION FRAMEWORK

Android software stack has four layer components [20]: Linux kernel, Libraries, Application framework, and Applications. The Android operating system permits users to decide whether an application is malicious or not [28], which means that users must analyze and confirm its identity and if found malicious need to report and thus will ensure its removal from the Android Market. For example, a malware application named droid09 [26] was released to allow users to carry out banking activities. In order to use this application, user requires to provide the account information details and it would direct the communication to the bank. However, the credentials that meant for the bank were actually sent to the malware writer. In order to avoid this type of attack, an Android application needs to show the permissions during the installation so that user can decide whether to install it or not.

Android operating system is developed using Java code. The Java code is compiled into class files by android's compiler, which later converted into dex files. Dex files are bytecode for Dalvik virtual machine (VM), a non-standard JVM that runs Android applications. The dex file format keeps the Dalvik bytecode and specifies the organization of the various sections and items in the file. The dex files, binary

XML files, and other applications are put together in the android application packages with the extension .apk. Android applications' components namely activities, services, broadcast receivers, and content providers are implemented as classes in application code and are declared in the AndroidManifest (.xml file).

The AndroidManifest consist of important information associated to android application such as package name, names of the components declared, the permissions declared and requested, and so on. AndroidManifest is written in human readable XML and is transformed to binary XML during application build.

IV. ANDROID EXPLOIT

The two layers of android application basically targeted by malware are kernel layer and application layer. The kernel layer contains important information within the smartphone system inner operation while the application layer contains browser application. The features that enhance the effective operation of Android application but also increase its exploit chances include Near Filed Communication (NFC), SMS Stack, Android browser, Universal Serial Bus (USB), and Drivers i.e. buggy driver. Some of these features might be exploited by malware writer to install malware on the mobile device or crashing the system to make it unavailable through malfunction SMS [13].

V. ANDROID MALWARE METHOD OF PROPAGATION

1. User Interaction: This is a method whereby malware lure its host to interact with it in order to cause execution.

2. Social engineering techniques: These techniques include embedding malware in online games, system patches, pretentious messages to authorize user to lure them to run malware on a smartphone.

3. Software download: malware may hide inside pdf files or other files and infect the phone during downloading.

VI. MALWARE ANALYSIS TECHNIQUES

A) *Static Analysis*

This is the analysis of malware behaviour without executing the code of the mobile application. In this analysis malware behaviour is examined statistically by studying their interaction with the environment, the data they captured, the files tampered with, network and port disruptions, and operation activities among others. In static analysis, the code of the program is disassembled and examined for possible pattern change. Some of the tools for static analysis include debugger and decompiler: Winzip, dexjar, JD-GUI, Wireshark, TCPView, PEiD, Process Monitor, Process explorer, and Strings.

B) *Dynamic Analysis*

Dynamic analysis is the analysis of malware by examining its activity during the code runtime. Malware dynamic analysis is usually carried out through reverse engineering process, where malicious code embedded in the original code of software are being decoded, examined, analysed and decompiled. In this analysis, several tools are employed like debuggers, which analyse the equivalent software code and decompilers, which convert malware to its binary equivalent code. Facilities for malware analysis on an android smartphone includes Winzip (tool to unpack .apk file), dex2jar (tool to convert .dex to jar file), JD-GUI (decompiler for decompiling java code), and sample android malware (Baksmali), Ollydbg.

C) *Hybrid Approaches*

This approach combines the features of static and dynamic analysis in order to dissect the activity of malware.

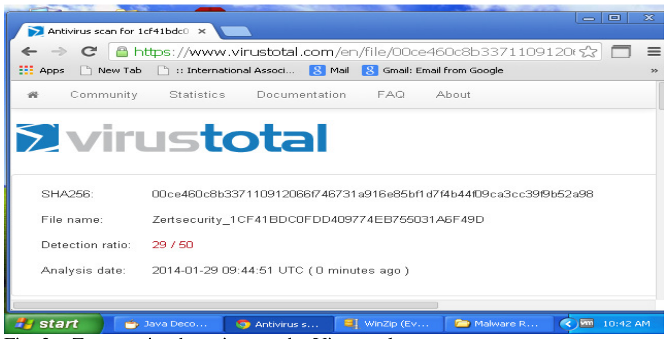
VII. MALWARE ANALYSIS EXPERIMENT

In this research, we analyzed Zertsecurity Android malwares to obtain android malware features and demonstrate the analysis of android malware statically.

1. *Obtain samples of android malwares from www.contagiomobile.com and www.contagiominidump.com*
2. *Test the malware for the rate of detection using virus total engine and Andrubis.*
3. *Unpack the .apk file and extracted the file into a component.*
4. *Convert the .dex file (dalvik code) to .jar file (java file) equivalent*
5. *Decompiled java code using jD-GUI to view its content of source code.*
6. *Analyze the source code of the malware for features identification.*

A) *Step 1:*

After acquiring Zertsecurity from www.contagiomobile.com, we tested it using virustotal engine and found the detection rate of 29/50 as shown below in the virustotal report of figure 2.



SHA256: 00ce460c8b3371109120 66f746731a916e85bf1d7f4b44f09ca3cc39f9b52a98
File name: Zertsecurity_1CF41BDC0FDD409774EB755031A6F49D
Detection ratio: 29 / 50
Analysis date: 2014-01-29 09:44:51 UTC (0 minutes ago)

Fig. 2. : Zertsecurity detection rate by Virustotal

B) Step II:

We extracted the zertsecurity.apk file using winzip in order to explore the content of .apk file. Winzip also displays the .dex file as wel as .xml files. The .dex files are the dalvik code that is lightweight in nature and suitable for mobile application while .xml files are contain in AndroidManifest file, which store important attributes of android application. This is shown in the figure 3.

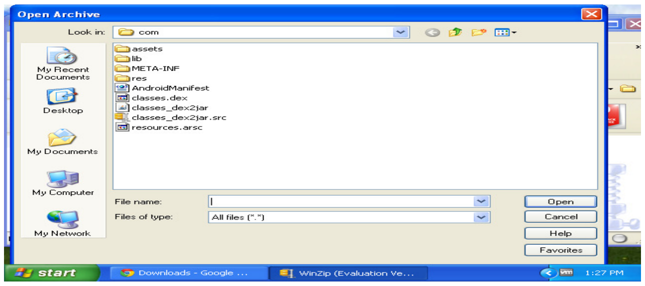
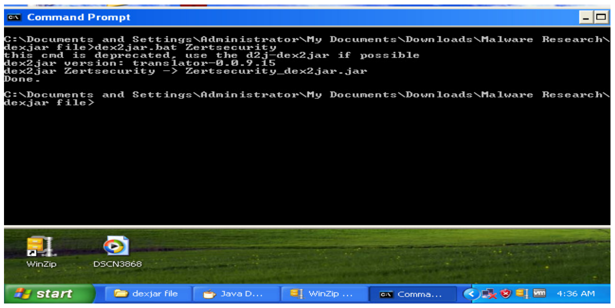


Fig. 3. Extraction of Zertsecurity.apk using winzip.

C) Step III:

In this stage, we used dex2jar tool to render better view of the dalvik code. This tool converts the dalvik executable code into java .classes files equivalent . We dropped the classes.dex file contains in the zertsecurity application into the dex2jar's directory and converted it into java .jar file using the command **dex2jar.bat classes.dex** as shown in the figure 4 below:

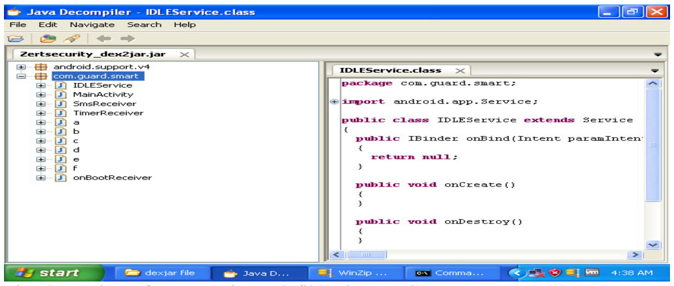


```
C:\Documents and Settings\Administrator\My Documents\Downloads\Malware Research\
dexjar file>dex2jar.bat Zertsecurity
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar Zertsecurity -> Zertsecurity_dex2jar.jar
Done.

C:\Documents and Settings\Administrator\My Documents\Downloads\Malware Research\
dexjar file>
```

Fig. 4: Dropping classes.dex file in the dex2jar tool for conversion

D) Step IV:

The **classes_dex2jar.jar** file is not yet readable and in order to make it readable, we used JD-GUI to open this file into a redable format.



```
package com.guard.smart;

import android.app.Service;

public class IDLEService extends Service
{
  public IBinder onBind(Intent paramInten
  {
    return null;
  }

  public void onCreate()
  {
  }

  public void onDestroy()
  {
  }
```

Fig. 5: Preview of Zertsecurity .apk file using JD-GUI

E) Step V:

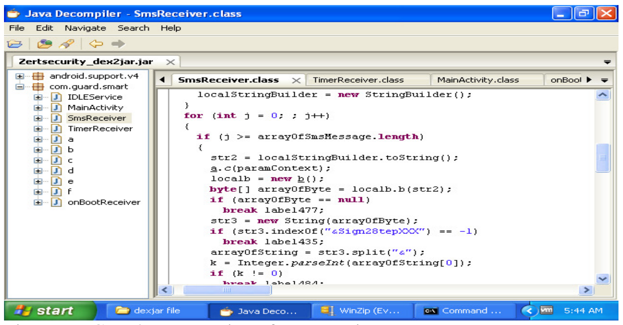
In this stage, we check the entire source code to determine the insertion or substitution of malicious code injected into the original code or build from scratch by malware writer. Zertsecurity Analysis is as follows:

1. Its malicious code hides under four modules namely; MainActivity, SmsReceiver, TimeReceiver, and onBootReceiver

2. This malware application uses masquerading technique and pretends as a German certificate installer application.
3. In the MainActivity module, it uses two hard-coded domains http://app-smartsystem.com/sms/d_m009.php and http://app-smartsystem.net/sms/d_m009.php as multiple Command and Control Servers as shown in figure 6.
4. It uses a fake login pin on top of phone number and IMEI to identify different phone uniquely.
5. It updates C & C universal resouce locators by either communicating the exisiting C & C servers, or through sent SMS by the user as contains in the string "&Sign28tepXXX" as shown in fig. 7.
6. It encrypts the Command and Control communication with AES-ECB, using "0523850789a8cfed" as static key (Fig. 7)
7. Zertsecurity uses onBootReceive module to monitor the boot start time so as to launch itself using BOOT_COMPLETED and query the server to detect updated C & C universal resource locators within fiften minutes (Fig. 9).

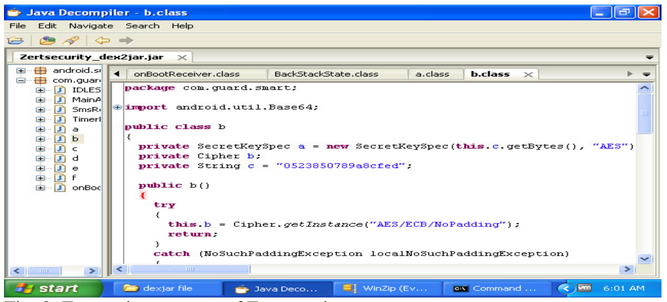
```
package com.guard.smart;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.Button;

public class MainActivity extends Activity
{
  public static int a = 900000;
  public static String[] b = { "http://app-smartsystem.com/sms/d_m009.php", "http://app-
smartsystem.net/sms/d_m009.php", "", "", "", "" };
  public static boolean c = false;
  public static String d = "";
  public static String e = "";
  public static String f = "";
  public static int g = 0;
  public static int h = 1;
  public static String i = "";
  public static Context j;

  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    j = getApplicationContext();
    a.c(j);
    if (!b[0].isEmpty())
      a.a(j);
    if (f.isEmpty())
    {
      setContentView(2130903040);
      ((Button)findViewById(2131165190)).setOnClickListener(new c(this));
```

Fig. 6: Code snippet showing Zertsecurity multiple C&C servers



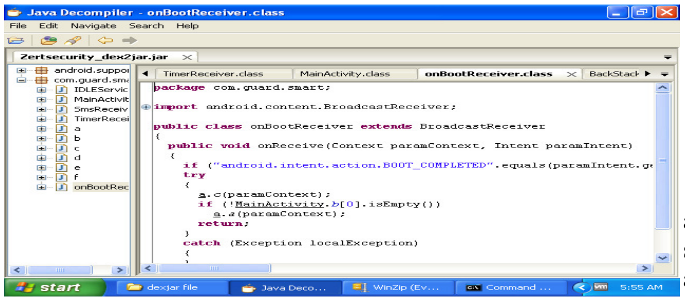
```
    localStringBuilder = new StringBuilder();
  }
  for (int j = 0; ; j++)
  {
    if (j >= arrayOfSmsMessage.length)
    {
      str2 = localStringBuilder.toString();
      a.c(paramContext);
      localb = new b();
      byte[] arrayOfByte = localb.b(str2);
      if (arrayOfByte == null)
        break label477;
      str3 = new String(arrayOfByte);
      if (str3.indexOf("&Sign28tepXXX") == -1)
        break label435;
      arrayOfString = str3.split("&");
      k = Integer.parseInt(arrayOfString[0]);
      if (k != 0)
```

Fig. 7: JD-GUI decompression of Zertsecurity



```
package com.guard.smart;

import android.util.Base64;

public class b
{
  private SecretKeySpec a = new SecretKeySpec(this.c.getBytes(), "AES")
  private Cipher b;
  private String c = "0523850789a8cfed";

  public b()
  {
    try
    {
      this.b = Cipher.getInstance("AES/ECB/NoPadding");
      return;
    }
    catch (NoSuchPaddingException localNoSuchPaddingException)
```

Fig. 8: Encryption strategy of Zertsecurity



```
package com.guard.smart;

import android.content.BroadcastReceiver;

public class onBootReceiver extends BroadcastReceiver
{
  public void onReceive(Context paramContext, Intent paramIntent)
  {
    if ("android.intent.action.BOOT_COMPLETED".equals(paramIntent.ge
    try
    {
      a.c(paramContext);
      if (!MainActivity.b[0].isEmpty())
        a.a(paramContext);
      return;
    }
    catch (Exception localException)
    {
```

Fig. 9: Booting action of Zertsecurity

F) Analysis Conclusion

The research identified several android malware features which include .apk file features, .xml file features, .dex file features. The .apk file consists of the size of apk, the zip number, related folders that store files, quantity of files for file type, among others. The .xml file consists of element names, attribute type, used permission, xml elements, attribute names etc., while .dex file features include strings elements, types, methods used, static values, classes, prototypes, fields etc.

It is observed that Zertsecurity is a Trojan malware that steal user's login information on system boot-up and send it to a remote servers through command and control universal resource locator (C&C urls) . It uses onBootReceive module to monitor the boot start time so as to launch itself using BOOT_COMPLETED and query the server to detect updated C & C universal resource locators within fiften minutes. It also uses encryption techniques to cover its dastard activities. Prevention stage includes alert raised, report generation, threat intelligence gathering and forwarding, and malware profile creation.

VIII. Future Work

The future direction of this research is to carry out comprehensive assessment of security on the Android mobile framework, hybrid analysis and use the identified features for effective classification in order to develop a detection system that can help in the detection and containment of malware on the same platform. The researchers aim to improve apriori association rule for extracting and selecting features using particle swarm optimization for purpose of effective detection.

IX. Research Benefits and Recommendations

This research is beneficial for us to: a) emphasize on the need to ensure the security of vita information on android smartphone; b) understand the attack vectors on android phones and better ways to address it. The followings are some of the recommendations for the effective use of smartphone for the protection against malware:

- *User education:* Since many malware can only request for the user permission before installing themselves on the phone, hence a need to educate the users to be aware of strange or unfamiliar application before installing it on their phones.

- *Download and Install software and application from approved and trusted sources.*

- *Use of mobile anti-malware software*

X. Conclusion

This research examined the vulnerabilities, exploits and attacks associated with android operating system security on a smartphone. The researchers identified several features associated to android malware through the analysis, which could be used in further research. The research studied several existing approaches in the analysis, detection and classification of malware on an android phone in a bid to identify weaknesses and propose a consolidated approach. The researchers share analysis of a sample android malware to obtain features, determine its attacks strategy and code semantics. The research finally highlighted the benefit of this research to the global society and recommends better approaches to ensure security of vita information on android smartphones.

References

[1] O. S. Adebayo, M. A. Mabayoje, A. Mishra, O. Osho "Malware Detection, Supportive Software Agents and Its Classification Schemes", International Journal of Network Security & Its Applications (IJNSA), Vol.4 (6), Pp. 33 – 49, 2012.

[2] K. J. Abela, L. Angeles, Don Kristopher E., D. Alas, J. Raynier P., Tolentino, R. Joseph, Gomez, A. N. Miguel "An Automated Malware Detection System for Android using Behavior-based Analysis, AMDA" International Journal of Cyber-Security and Digital Forensics (IJCSDF), 2 (2): The Society of Digital Information and Wireless Communications, ISSN: 2305-0012, 2013.

[3] B. Abhijit, H. Xin, G. S. Kang and P. Taejoon" Behavioral detection of Malware on Mobile Handsets", June 17–20, 2008, Breckenridge, Colorado, USA. ACM 978-1- 60558-139-2/08/06, 2008.

[4] A. Walenstein, L. Deshotels, and A. Lakhotia "Program Structure-Based Feature Selection for Android Malware Analysis" MOBISEC 2012, LNICST 107, Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 51–52, 2012.

[5] Anubis: Analyzing unknown binaries. [Online]. Available: http://anubis.iseclab.org/

[6] Cabir, Smartphone Malware. Available at http://www.f-secure.com/v-descs/cabir.shtml.

[7] D. Shetty, "Demystifying the Android Malware", Retrieved From: http://packetstormsecurity.org/files/view/104458/demystifying-android.pdf, Last Accessed: 25 October, 2013.

[8] F-secure. Cabir. Access from http://www.f-secure.com/v-descs/cabir.shtml, 29-10-2011.

[9] H. Kim, J. Smith, and K. G. Shin "Detecting energy-greedy anomalies and mobile malware variants". In Proceeding of the 6th international conference on Mobile systems, applications, and services, MobiSys '08, ACM , pages 239-252, New York, NY, USA, 2008.

[10] H. Le Thanh "Analysis of Malware Families on Android Mobiles: Detection Characteristics Recognizable by Ordinary Phone Users and How to Fix It" Journal of Information Security, 4 (4), 213-224, 2013.

[11] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani "Crowdroid: Behavior-Based Malware Detection System for Android". In Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and mobile devices (October 2011), Pp.15-26, 2011.

[12] L. K. Yan and H. Yin, "Droidscope: seamlessly reconstructing, the OS and dalvik semantic views for dynamic android malware analysis," in Proceedings of the 21st USENIX conference on Security symposium, ser. Security'12. Berkeley, CA, USA: USENIX Association, pp. 29–29, 2012. Available: http://dl.acm.org/citation.cfm?id=2362793.2362822.

[13] L. Delosieres and D. Garcia, "Infrastructure for detecting Android malware," in Proc. 28th Int. Symp. on Computer and Information Sciences (ISCIS'13), 2013.

[14] M. Lindorfer, "Andrubis: A tool for analyzing unknown android applications". Available at: http://blog.iseclab.org/2012/06/04/andrubis-a-tool-for-analyzing-unknown-android-applications-2/

[15] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, "RiskRanker: Scalable and Accurate Zero-day Android Malware Detection" Proceedings of the 10th International Conference on Mobile Systems, Applications and Services (MobiSys 2012), 2012.

[16] M. Christodorescu, S. Jha, D. Maughan, D. Song, C. Wang, "Malware Detection: Advance Information Security", ISBN-10: 0-387-32720-7, ISBN-13: 978-0-387-32720-4,e-I SBN-10: 0-387-44599- 4, e-ISBN-13: 978-0-387-44599-1.

[17] M. Zhao, T. Zhang, J. Wang, Z. Yuan, "A Smartphone Malware Detection Framework based on Artificial Immunology". Journal of Networks, VOL. 8 (2), 469-472, 2013.

[18] M. Karami, E. Mohamed, N. Parnian, and S. Angelos, "Behavioral Analysis of Android Applications Using Automated Instrumentation". In Proceedings of the 7th International Conference on Software Security and Reliability (SERE), Washington DC, USA, 18-20 June 2013.

[19] Q. Yan, Y. Li, T. Li, and R. Deng, "Insights into malware detection and prevention on mobile phones," Security Technology, pp. 242– 249, 2009.

[20] R. Vala, L. Sarga, and R. Benda, "Security Reverse Engineering of Mobile Operating Systems: A Summary". Recent Advances in Computer Science, ISBN: 978-960-474-311-7, 2013.

[21] S. Meyer, J. Freudiger, J. Hubaux "Misbehavior in Mobile Application Markets: Security and Cooperation in Wireless Networks Mini-project" EPFL, 2011.

[22] S. Holla, and M. Katti "Android based Mobile Application and Its Security". International Journal of Computer Trends and Technology, 3 (3)Page 486 – 490, ISSN 2231 – 2801, 2013. Available at http://www.internationaljournalssrg.org

[23] T. K. Buennemeyer, M. N. Theresa, M. C. Lee, P. D. John, C. M. Randy, and G. T. Joseph, "Mobile device profiling and intrusion detection using smart batteries." In Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, pp. 296-296. IEEE, 2008.

[24] T. Blasing, A. Schmidt, L. Batyuk, A. C. Seyit, and S. Albayrak "An android application sandbox system for suspicious software detection". In 5th International Conference on Malicious and Unwanted Software (Malware 2010), Nancy France, 2010.

[25] T. Eder, M. Rodler, D. Vymazal, M. Zeilinger "A Framework For Analyzing Android Applications". Workshop on Emerging Cyberthreats and Countermeasures ECTCM., 2013. Available at http://arxiv.org/find/all/1/all:+ANANAS/0/1/0/all/0/1

[26] T. Vennon, "Android Market Threat Analysis", Retrieved From: http://www.scribd.com/doc/33454935/Android-Market-Threat-Analysis-6-22-10-v1#download, Last Accessed: 04 October, 2013

[27] E. William, G. Peter, C. Byung-Gon, P. C. Landon, J. Jung, P. McDaniel, A.l N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones". 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10), 2010.

[28] V. J. Varghese and S. Walker, "Dissecting Andro Malware". SAN Institute, School of Computer and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK, 2011.

[29] Amazon appstore for android. URL http://www.amazon.com/mobile-apps/b?ie=UTF8\&node=2350149011, 2013.

[30] Getjar, 2013. URL http://www.getjar.com/

[31] Google play, 2013. URL https://play.google.com/store

[32] A. Shabtai, Y. Fledel, & Y. Elovici, "Automated static code analysis for classifying Android applications using machine learning." International Conference on Computational Intelligence and Security (CIS), 2010.