Digital Investigation 13 (2015) 1–14

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

APK Auditor: Permission-based Android malware detection system

Kabakus Abdullah Talha [a, *], Dogru Ibrahim Alper [b], Cetin Aydin [b]

[a] *IT Center, Abant Izzet Baysal University, Bolu 14280, Turkey*
[b] *Department of Computer Engineering, Gazi University, Ankara 06500, Turkey*

A R T I C L E I N F O

Article history:
Received 24 July 2014
Received in revised form 11 January 2015
Accepted 15 January 2015
Available online

Keywords:
Android
Smartphone
Mobile security
Malware
Static analysis

A B S T R A C T

Android operating system has the highest market share in 2014; making it the most widely used mobile operating system in the world. This fact makes Android users the biggest target group for malware developers. Trend analyses show large increase in mobile malware targeting the Android platform. Android's security mechanism is based on an instrument that informs users about which permissions the application needs to be granted before installing them. This permission system provides an overview of the application and may help gain awareness about the risks. However, we do not have enough information to conclude that standard users read or digital investigators understand these permissions and their implications. Digital investigators need to be on the alert for the presence of malware when examining Android devices, and can benefit from supporting tools that help them understand the capabilities of such malicious code. This paper presents a permission-based Android malware detection system, *APK Auditor* that uses static analysis to characterize and classify Android applications as benign or malicious. *APK Auditor* consists of three components: (1) A signature database to store extracted information about applications and analysis results, (2) an Android client which is used by end-users to grant application analysis requests, and (3) a central server responsible for communicating with both signature database and smartphone client and managing whole analysis process. To test system performance, 8762 applications in total, 1853 benign applications from Google's Play Store and 6909 malicious applications from different sources were collected and analyzed by the system developed. The results show that *APK Auditor* is able to detect most well-known malwares and highlights the ones with a potential in approximately 88% accuracy with a 0.925 specificity.

Introduction

Today, Android is the most widely used mobile operating system in the world. The reasons for being favored by users can be listed as: (1) Being an open source software, (2) being supported by Google, (3) being an advanced programmable software framework used to develop mobile applications in popular programming language Java (4) being open to customization. According to the International Data Corporation's latest report, Android dominates the market with 84.7% share with more than 300 million shipments in Q2 2014.[1] Android's senior vice president Sundar Pichai recently revealed that Android has reached over 1 billion devices.[2] Hugo Barra, Android product development vice president, declares that Google Play Store has more than 1 million applications.[3] Lookout

* Corresponding author. Tel.: +90 5353787341.
E-mail address: talha.kabakus@ibu.edu.tr (K.A. Talha).

[1] http://www.idc.com/getdoc.jsp?containerId=prUS25037214.
[2] https://twitter.com/sundarpichai/status/374933465998708736.
[3] http://readwrite.com/2013/07/24/google-play-hits-one-million-android-apps.

http://dx.doi.org/10.1016/j.diin.2015.01.001
1742-2876/© 2015 Elsevier Ltd. All rights reserved.

reports that Play Store grows three times faster than its biggest rival Apple Store (Lookout, 2011). Parallel to this popularity, the number of malicious applications is increasing continually (Felt et al., 2011a). Sophos Mobile Security Threat Report reveals that number of Android malware has increased by 600% in last 12 months (Mobile Security Threat Report, 2014). Kaspersky Labs reports that Android is the main target of malicious software, attracting a whopping 98.05% of known malware.[4] Similarly, Symantec's latest Internet Security Threat Report reviews that "the number of mobile threats that track users doubled in 2013 and mobile malware seemed almost solely focused on the Android platform" (Internet Security Threat Report, 2014). Hence, mobile security applications can only protect systems from certain kinds of malicious programs. Most of these security applications compare newly installed applications against a repository that stores malware signatures (Guido et al., 2013). This "blacklisting" technique has known weaknesses that can be exploited by malware distributors (Vidas et al., 2011b). This blacklisting approach never suffices to ascertain whether an application is malicious or not since applications get updated over time and their purposes may change during these updates. A piece of Android malware can surreptitiously escalate privileges by modifying system files, send short messaging service (SMS) messages, make calls, share location information through global positioning system (GPS) or internet and collect excessive amounts of personal information. The Android security mechanism does not set a limit on how much of the system resources an application can use such as central processing unit (CPU), memory and local drive, etc. This is a critical vulnerability point for malicious applications.

Android's security model significantly relies on permission-based mechanisms (Barrera et al., 2010; Felt et al., 2011b). Permissions are requested by applications and they are required to access application's interfaces/data that are defined in its manifest file (Enck et al., 2011). They are used to inform users about application's capabilities ("Android Security Overview," n.d.). Android security framework does not block or review applications through their permissions. Instead of that, Android informs users about which permissions an application wants to be granted when user initiates installation process. Evaluation process is handled by user and s/he can continue on the installation or cancel it if s/he is not comfortable with the permissions. These permissions are not shown to users at any time other than installation (Felt et al., 2012). According to the research by Felt et al., only 17% of participants are interested in these permissions; 42% of them are unaware about them (Felt et al., 2012). Some researchers on the other hand speculated that most of the time these permissions are ignored and not understood by users (Enck et al., 2009; Felt et al., 2011b; Kelley et al., 2012; King et al., 2011). Past studies on smartphone users' privacy concerns have primarily focused on location tracking and sharing (Barkuus et al., 2003; Consolvo et al., 2005; Kelley et al., 2011; Lindqvist et al., 2011; Sadeh et al., 2009). They are just 2 of 145 permissions that Android 4.4 (KitKat) defines. At this point, users need to be informed and guided by the application about these permissions before installation.

Android applications are packaged as Android application (APK) files which contain manifest file (`AndroidManifest.xml`), compiled Java classes and application resources. We have developed an Android malware detection system based on permission analysis through APK files, named *APK Auditor*. The system developed uses static analysis techniques based on permissions in order to characterize and extract profiles for Android applications. *APK Auditor* contains three main components: (1) An Android client, (2) a signature database that contains extracted analysis, (3) a central server that communicates with both the signature database server and the clients.

This paper is structured as follows: Related works section presents the related works. APK Auditor section discusses the materials and methods, *APK Auditor*'s architecture with its components. Results and discussion section presents the experiments and Conclusions section presents the conclusions and the future works.

Related works

DroidMat (Wu et al., 2012) detects Android malware through static analyst paradigm. It takes into consideration some static information including permissions, deployment of components, intent message passing and Application Programming Interface (API) calls for characterizing Android applications' behaviors. After application characteristics are extracted, K-means algorithm is used to enhance malware modeling capability. Numbers of clusters are decided by Singular Value Decomposition (SVD) method on the low rank approximation. At last, the system classifies applications as benign or malicious using k-Nearest Neighbors (kNN) algorithm.

MADAM (a Multi-level Anomaly Detector for Android Malware) (Dini et al., 2012) is a multi-level Android malware detector that concurrently monitors Android at the kernel-level and user-level to detect real malware infections using machine learning techniques to distinguish between standard behaviors and malicious ones. At kernel-level, *MADAM* evaluates system calls, running processes, free random access memory (RAM) and CPU usages. At user/application level, it evaluates idle/active status, keystrokes, dialed numbers, sent/received SMS and Bluetooth/Wi-Fi analysis.

Guido et al. (2013) presents a real time malware detection system called *Periodic Mobile Forensics (PMF)* that contains a smartphone client, a central server, a database and an analysis framework. The smartphone client sends changed file system data to central server in order to allow expensive forensic processing and offline application of traditional tools and techniques rarely applied to mobile environment. The analysis framework identifies changes to important system partitions, recognizes the changes in the file system, including file deletions and finds persistent and triggering mechanisms in newly installed applications. Unlike *APK Auditor*, the analysis performed by PMF does not focus on application permissions.

[4] http://www.kaspersky.com/about/news/virus/2013/number-of-the-year.

Zhou and Jiang (2012) present a systematic characterization of Android malware using a huge dataset that was also utilized to develop *APK Auditor*. The dataset contains 1260 Android malware samples from 49 different families. Unlike *APK Auditor* they characterize malwares from various aspects such as installation, activation and payloads which are categorized as dynamic analysis. The development of *APK Auditor* included use of this dataset in order to enrich Android malware sample variety. In reference to the evaluation procedure they conducted on four mobile antivirus software existed, in the best case scenario, it detects 79.6% of them which is far less than *APK Auditor*'s true detection ratio, despite the fact that their dataset is smaller and contains 1260 samples.

Vidas et al. (2011a) present a tool based on popular Java programming, integrated development environment (IDE) Eclipse[5] to analyze application's requested permissions. The tool highlights necessary and unnecessary permissions through source code analysis. Unlike this tool *APK Auditor* inspects permission usages in order to highlight possible malicious intentions.

Vidas and Christin (2013) present an end-to-end verification tool named *AppIntegrity* to facilitate cryptographic verification between software developers and end users. They collected 41,507 applications from 194 alternative Android markets in addition to 35,423 applications from the official Android market “Play Store”. This tool aims to strengthen authentication mechanisms offered in application marketplaces, thereby making application repackaging – *which is another popular way to distribute malicious software* – more difficult. *AppIntegrity* has an application download limitation (i.e. 100 application download per hour) that can trigger temporary blocking and blacklisting when too many download attempts are requested. *APK Auditor* was designed to not have a download limit.

Rastogi et al. (2013) present a tool named *AppsPlayground* that offers automatic dynamic analysis of smartphone applications. *AppsPlayground*, in order to offer an effective analysis environment, benefits from dynamic analyses such as difference detection, exploration and disguise techniques – e.g. kernel-level monitoring, API call traces – event triggering. Unlike *AppsPlayground*, *APK Auditor*, in order to protect Android devices from malwares that are designed to take control of the target system, uses static analyses that provide a pro-active approach.

Burguera et al. (2011) present *Crowdroid*, a framework to detect anomalous behavior of Android applications. In order to categorize Android applications as benign or malicious, *Crowdroid* uses a Linux based tool, *Strace*[6] that collects Linux Kernel system calls containing device information, installed application list and monitoring results, as *APK Auditor* does *Crowdroid* also adapts uses machine learning techniques to evaluate collected data. While *APK Auditor* focuses on permissions, *Crowdroid* focuses on behaviors. A significant difference between *APK Auditor* and *Crowdroid* is that *APK Auditor* detects malware before installation owing to the fact that it benefits from static analyses.

Arp et al. (2014) present a tool named *Drebin*, a lightweight method that combines concepts from static analysis and machine learning to detect Android malware. *Drebin* gathers application features from application's source code and manifest file. These features can be ordered as permissions, API calls and network addresses inside source code. Features are embedded in a joint vector space in order to detect malwares using typical patterns. These features are extracted by their weights. During the development of *APK Auditor*, in order to improve *APK Auditor*'s malware detection accuracy, Arp et al.'s dataset of 5600 malwares from 179 different malware families was utilized. Unlike *APK Auditor*, *Drebin* implements a method to match API calls and permissions.

Paranoid Android (Portokalidis et al., 2010) applies security checks on remote security servers that host exact replicas of smartphones in virtual environments in order to decrease costs of both file storage and processing. Due to the fact that it uses a number of security servers in parallel, *Paranoid Android* is able to apply multiple detection techniques simultaneously.

CopperDroid (Reina et al., 2013), an approach built on top of QEMU[7] – an open source machine emulator – provides dynamic behavior analysis of Android malware by characterizing low-level operating system (OS) specific and high-level Android specific behaviors. *CopperDroid* is able to characterize behavior of Android malware whether it is initiated from Java, Java Native Interface (JNI) or native code execution.

Liang and Du (2014) present a similar approach to *APK Auditor* except they combine permissions in sets. This approach brings a difficulty for application classification when the number of permissions in a set increases. As the number of permissions in a set is increasing, more permissions are needed to declare an application as malicious. If one of the permissions in a set is missed, even the analyzed application contains malicious content, the system does not classify the application correctly.

APK Auditor

APK Auditor is a permission-based Android malware assessment system. *APK Auditor* consists of three main components: (1) An Android client, (2) a signature database, (3) a central server that communicates with both the Android client and the signature database and handles the analysis process. Fig. 1 represents an overview of *APK Auditor*'s software architecture. *APK Auditor* central server is responsible for accessing signature database and manages the whole malware analysis process. Android client serves as an application that is installed into system and it communicates with central server through a web service. Analysis results retrieved from this web service are shown to users via the client they use which can be an Android client or web-based portal. *APK Auditor* signature database

[5] http://eclipse.org.

[6] http://linux.die.net/man/1/strace.

[7] http://wiki.qemu.org/Main_Page.

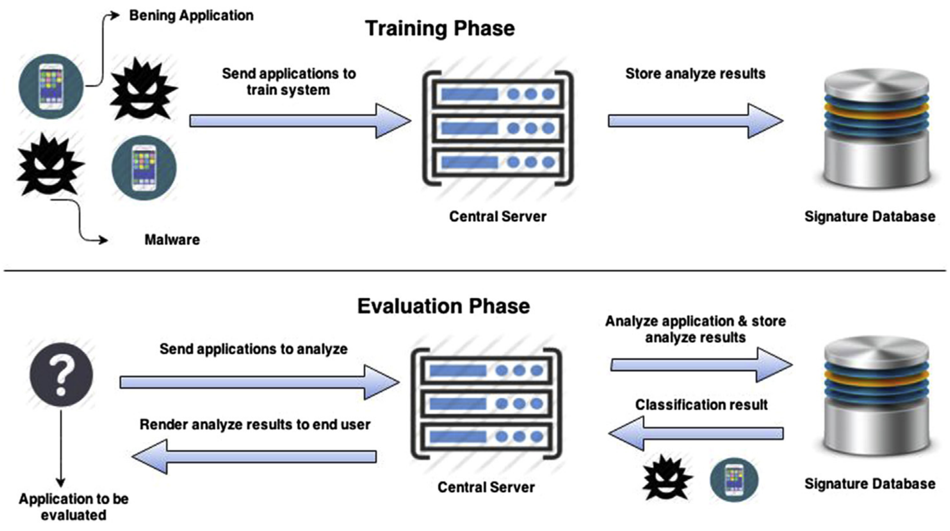

Training Phase
Bening Application
Malware
Send applications to train system
Central Server
Store analyze results
Signature Database
Evaluation Phase
Send applications to analyze
Render analyze results to end user
Application to be evaluated
Central Server
Analyze application & store analyze results
Classification result
Signature Database

Fig. 1. *APK Auditor* software architecture.

is a relational database management system that stores the results of the Android applications analyzed.

To give deeper insight to digital investigators about the system, *APK Auditor* components are explained in detail in the following subsections.

APK Auditor Android client

APK Auditor client simply offers an analysis request, showing whether the application can be trusted or not. This client application lets users analyze both local applications on an Android device and remote applications on Play Store. Remote applications are analyzed through HTTPS POST requests by their unique package names ("Android Developers API Guide Manifest," 2014). Because Play Store maintains the latest version of each application only, the *APK Auditor* central server only downloads the current version of a specified application, does not obtain older versions of the applications[8,9]. Analyzed results (*HTTPS Response*) are sent back to the *APK Auditor* client application and shown to the user. While conducting an analysis on a centralized server, applications do not have to be downloaded or installed on the Android devices and this procedure simply do not use memory of mobile devices to perform the analysis. This pro-active approach protects the Android device from malicious content as some kinds of malwares can get the device under control and mark themselves as "benign" as soon as they are installed. A screenshot of *APK Auditor* smartphone client main user interface is shown in Fig. 2. *APK Auditor* client does not require root privileges. Permissions that are required to install the *APK Auditor* client are listed in Table 1 with their intentions.

For digital investigators to have more information about the applications, the *APK Auditor* client also provides a user interface to monitor protection levels and permissions of currently installed applications. This gives an overall idea to users about their smartphones status. Android itself does not set system resource usage limits for its applications. Therefore a piece of malware can use 100% of the system resources such as CPU, RAM to make smartphone unusable; this is known as a Denial of Service (DoS) attacks (Carl et al., 2006; Leiwo et al., 2000; Park and Lee, 2001) in literature. To protect Android operating systems against such possible attacks, *APK Auditor* client lets users set a system resource usage (*memory usage in MB*) limit and kills applications that exceed this limit. A resource usage interface, Fig. 3, is added to system to let the users check the status of resources. Applications that exceed the limit are shown in this user interface. The selected applications can be killed by selection after confirming the prompt dialogs. This process is not performed automatically since some benign applications can also consume high level of system resources while they are actively working. Current memory status which is taken from operating system itself is shown in this user interface. Users can benefit from this information before taking action. There are no specific rules to calculate resource usage limit for an Android application because it can change depending on the application's capability & device's capacity. Due to these reasons, authors think that it is best to leave application termination to users.

According to the Android developers guide ("Android Developer Guide Permission," 2014), all permissions have a protection level that characterizes the potential risk implied in the permission and indicates that the procedure the system should follow when determining whether or not to grant the permission to an application requesting it. *Normal* permissions which are the default value for permissions are those granted automatically without asking user's explicit approval. *Dangerous* permissions give requested application to access user's private data or

[8] http://stackoverflow.com/questions/20924393/get-older-apk-from-google-play-developer-console.

[9] http://stackoverflow.com/questions/11278387/is-it-possible-to-download-an-old-apk-for-my-app-from-google-play.

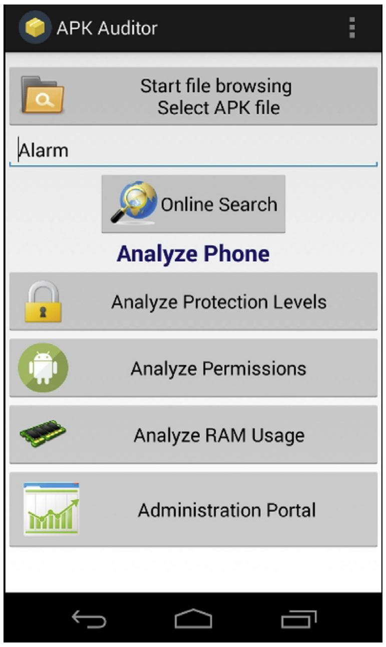
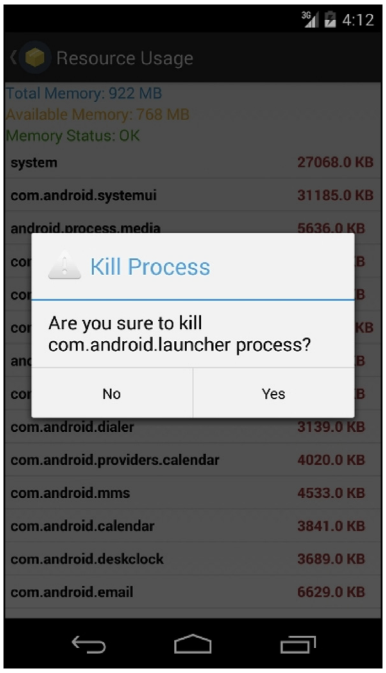


Fig. 2. *APK Auditor* Android client main user interface.



Total Memory: 922 MB
Available Memory: 768 MB
Memory Status: OK

system	27068.0 KB
com.android.systemui	31185.0 KB
android.process.media	5636.0 KB
com.android.dialer	3139.0 KB
com.android.providers.calendar	4020.0 KB
com.android.mms	4533.0 KB
com.android.calendar	3841.0 KB
com.android.deskclock	3689.0 KB
com.android.email	6629.0 KB

Are you sure to kill com.android.launcher process?

Fig. 3. *APK Auditor* Android client resource usage interface.

control the device. Because of these permissions can introduce potential risk, they are not automatically granted by the system. If this type of permission is not granted by user, installation process does not continue. *Signature* permissions are only granted if the requested application is signed with the same certificate as the application that declared the permission. *SignatureOrSystem* permissions are like the *signature* one; but also used for pre-installed applications that are installed into*/system/app* folder where others are installed into*/data/app* folder in the system. It is strongly recommended to developers to avoid using this type of permission since this protection level is defined and used for certain special situations such as built-in applications and operating system tools.

APK Auditor Signature Database

Applications are stored in the *APK Auditor* signature database server together with the results of the analysis. The following information are extracted and stored in signature database as they are represented in E-R (Entity-Relation) diagram presented in Fig. 4:

Table 1
APK Auditor Android client requested permissions and their usage intentions.

Permission	Intention
`INTERNET`	Online application search
`WRITE_EXTERNAL_STORAGE`	Extract local `.apk` archives for malware analysis
`READ_EXTERNAL_STORAGE`	Read local `.apk` archives
`KILL_BACKGROUND_PROCESSES`	Kill applications that exceeds resource usage limit

- Application Permissions
- Application Services
- Application Receivers

The latest version of Android, Android 4.4, defines 145 permissions and all of them are stored in `Permission` table as a constant data. Because each application can contain as many permissions as it needs, a third table `(android_app_permission)` is created to store application's permissions. *APK Auditor* central server classifies applications through these permissions, based on their existence in malwares. Service and receiver information is ignored due to their application specific definitions. Stored analysis results for each application are displayed through *APK Auditor* Administration Portal.

APK Auditor central server

The *APK Auditor* central server manages analysis process and works as a link between signature database and Android client while analyzing requested applications. The central server regularly *(every midnight)* downloads top rated 100 applications from Play Store and analyzes them. The number of applications that will be analyzed per day can be set dynamically. Play Store, the official Android market, offers applications to be downloaded by only real Android devices. This official market requires authentication and a device identifier in order to download an application. *APK Auditor* uses a web service named *APIfy* that provides `.apk` archives of Android applications by the names of their packages. This service is free and just requires a free registration. Then it creates an API key *(access token)*, which is unique and belongs to the registered user.

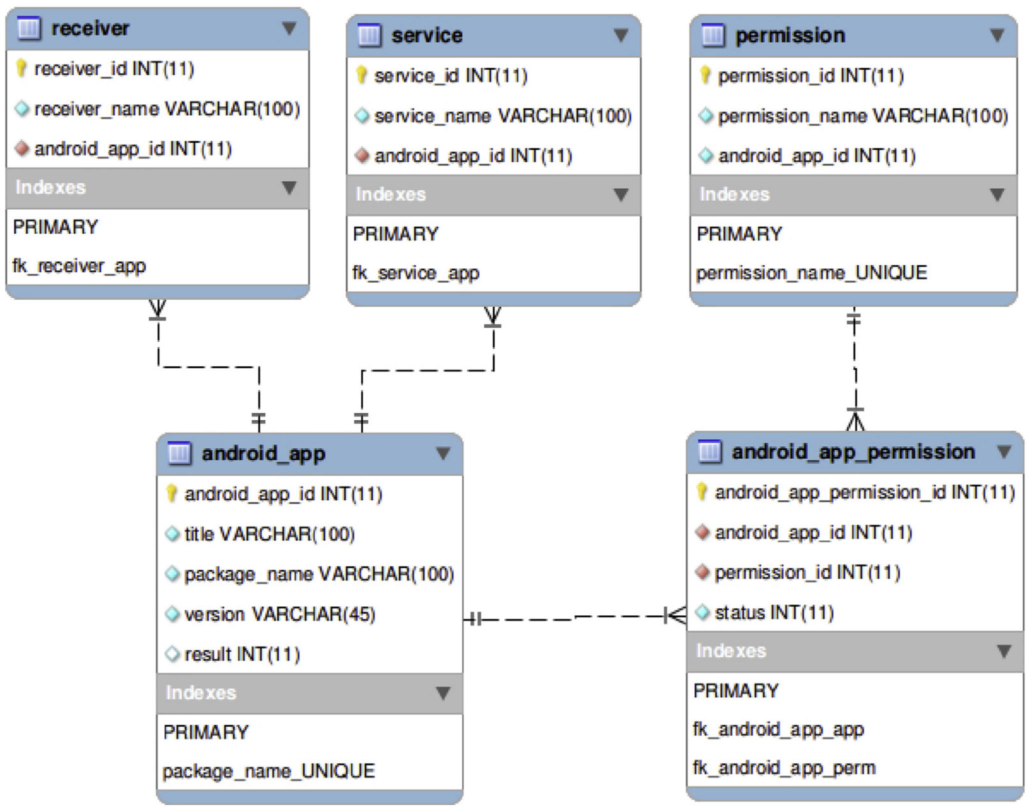


Fig. 4. *APK Auditor* Signature Database E-R diagram.

An APK file is basically an archive that contains application's resources (such as images, XML layouts, and bundles), compiled Java classes (in Dalvik DEX format) and `AndroidManifest.xml` file that defines application metadata. *APK Auditor* identifies applications by their package names which are necessarily unique. This necessity is defined by Play Store ("Android Developers API Guide Manifest," 2014). Therefore, there is no application duplication in *APK Auditor*. Once the central server gets the list of applications to be analyzed, it checks each application to make sure it has not been analyzed before, through querying the signature database with application's unique package name (`package_name` column of `android_app` table in Fig. 4) and version (`version` column of `android_app` table in Fig. 4). As it has been discussed earlier, Play Store only provides the latest versions of applications. But *APK Auditor* stores each version of applications in the signature database as soon as analyzing process is concluded. If the application with the exact version has not been analyzed before, *APK Auditor* downloads the application and starts analyzing.

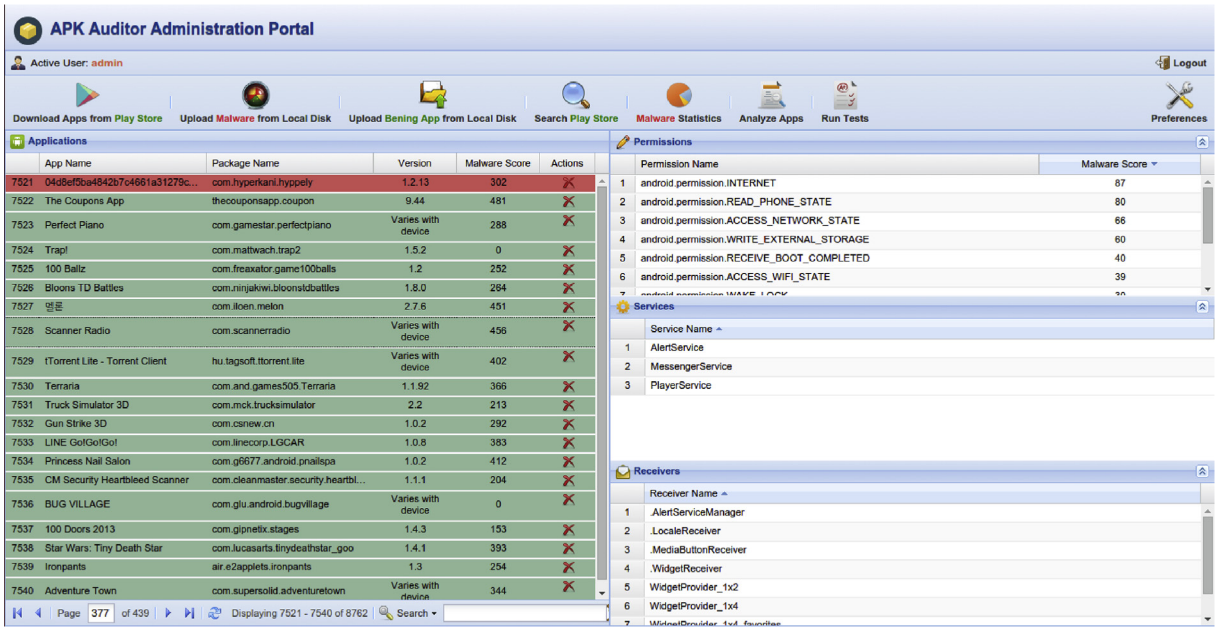
The analysis process is continuous; each analyzed application effects the success of malware detection. Because *APK Auditor* regularly downloads and analyzes applications from Play Store, learning curve of system never ends. The central server extracts application permissions from manifest file and stores them in signature database. Fig. 5 represents malware score calculations for applications and permissions. A permission malware score *(PMS)* is calculated for each permission through its existence in all malwares. Then application's malware score *(AMS)* is calculated by summing its permissions malware scores. This score states the application's maliciousness mark. If this score is more than the limit calculated by the analysis framework, then the application is regarded as malicious. For example, the `com.keji.danti` application requests 18 permissions including the most dangerous ones like `INTERNET`, `READ_PHONE_STATE`, `ACCESS_NETWORK_STATE`, `SEND_SMS`. *APK Auditor* sums each permission's malware score *(PMS)* to calculate an overall *Application Malware Score (AMS)*. For this example, the *AMS* is calculated as 520. At this time, malware threshold limit is set to 195 using logistic regression. Because this application's malware score exceeds this limit, this application is classified as 'malicious' and which is also reported as malware[10].

$$PMS = \textit{Number of malwares that uses permission} / \textit{Number of all malwares}$$

$$AMS = \sum PMS$$

Fig. 5. *APK Auditor* malware score calculations.

[10] http://foresafe.com/report/ED8621092D77E747F26B92B6FFDCF095.


APK Auditor Administration Portal
Active User: admin
Logout
Download Apps from Play Store
Upload Malware from Local Disk
Upload Bening App from Local Disk
Search Play Store
Malware Statistics
Analyze Apps
Run Tests
Preferences
Applications
Permissions
Services
Receivers
Page 377 of 439
Displaying 7521 - 7540 of 8762
Search

	App Name	Package Name	Version	Malware Score	Actions
7521	04d8ef5ba4842b7c4661a31279c...	com.hyperkani.hyppely	1.2.13	302	✗
7522	The Coupons App	thecouponsapp.coupon	9.44	481	✗
7523	Perfect Piano	com.gamestar.perfectpiano	Varies with device	288	✗
7524	Trap!	com.mattwach.trap2	1.5.2	0	✗
7525	100 Ballz	com.freaxator.game100balls	1.2	252	✗
7526	Bloons TD Battles	com.ninjakiwi.bloonstdbattles	1.8.0	264	✗
7527	멜론	com.iloen.melon	2.7.6	451	✗
7528	Scanner Radio	com.scannerradio	Varies with device	456	✗
7529	tTorrent Lite - Torrent Client	hu.tagsoft.ttorrent.lite	Varies with device	402	✗
7530	Terraria	com.and.games505.Terraria	1.1.92	366	✗
7531	Truck Simulator 3D	com.mck.trucksimulator	2.2	213	✗
7532	Gun Strike 3D	com.csnew.cn	1.0.2	292	✗
7533	LINE Go!Go!Go!	com.linecorp.LGCAR	1.0.8	383	✗
7534	Princess Nail Salon	com.g6677.android.pnailspa	1.0.2	412	✗
7535	CM Security Heartbleed Scanner	com.cleanmaster.security.heartbl...	1.1.1	204	✗
7536	BUG VILLAGE	com.glu.android.bugvillage	Varies with device	0	✗
7537	100 Doors 2013	com.gipnetix.stages	1.4.3	153	✗
7538	Star Wars: Tiny Death Star	com.lucasarts.tinydeathstar_goo	1.4.1	393	✗
7539	Ironpants	air.e2applets.ironpants	1.3	254	✗
7540	Adventure Town	com.supersolid.adventuretown	Varies with device	344	✗

	Permission Name	Malware Score
1	android.permission.INTERNET	87
2	android.permission.READ_PHONE_STATE	80
3	android.permission.ACCESS_NETWORK_STATE	66
4	android.permission.WRITE_EXTERNAL_STORAGE	60
5	android.permission.RECEIVE_BOOT_COMPLETED	40
6	android.permission.ACCESS_WIFI_STATE	39

	Service Name
1	AlertService
2	MessengerService
3	PlayerService

	Receiver Name
1	.AlertServiceManager
2	.LocaleReceiver
3	.MediaButtonReceiver
4	.WidgetReceiver
5	WidgetProvider_1x2
6	WidgetProvider_1x4

Fig. 6. Administration portal main user interface.

APK Auditor administration portal

A web based administration portal is developed to manage systems and to see the details of the analyses. A view of *APK Auditor* administration portal main user interface is given in Fig. 6.

Main functionalities/user interfaces this portal provides can be listed as:

- Upload applications from local hard drive to analyze
- Search specific application from Play Store to analyze
- Download & analyze most popular applications from Play Store with one-click
- Regularly download & analyze most popular applications from Play Store
- Top 10 permissions used by inspected malwares (Fig. 7 and Table 1)
- Application list by permission (Fig. 7)
- Permission statistics limited by a malware score *(share)* (Fig. 8 and Table 2)
- System training using logistic regression – malware score threshold identification
- Running and evaluating system tests including their calculated malware scores and result of system detection success (Fig. 10)

Applications can be directly searched by name from Play Store and analyzed through *APK Auditor* Administration Portal. *42matters* provides a web service[11] to query applications by name, language, location (a specific country or worldwide), and category such as game, finance, education through the service's internal indexes. Application results can be ordered by different criteria like ratings, download counts, price, size and release dates. Similar to *APIfy*, this service is open for access by an API key that is delivered after registration. Applications can be sorted by their popularity ranks in the last day or week. This service is used by *APK Auditor* to query Android applications by name and also to fetch most popular *n* (*n* is a dynamic value and can be set through web user interface) applications.

Administration Portal provides two user authorization levels; *administrators* and *guests*. *Administrators* are users who have all privileges to manage the system. *Guests* can use the portal in read-only mode which means they cannot change/affect system's logic. This authorization level is provided for digital investigators who want to see deeper details of malware analyses that come from Android client.

Results and discussion

To develop *APK Auditor*'s application permission assessment capabilities, 6909 malwares were collected from a malware repository named contagio mobile,[12] Drebin[13] dataset and Android Malware Genome Project[14] (Zhou and Jiang, 2012) which are generally accepted as Android malware sources in literature. 1853 benign applications were downloaded from Play Store. In total, 8762 applications were collected and analyzed to develop *APK Auditor*. Contagio mobile blog shares `.apk` archives of malware with the reports showing the malicious content analyzed by VirusTotal.[15] VirusTotal is a free service that

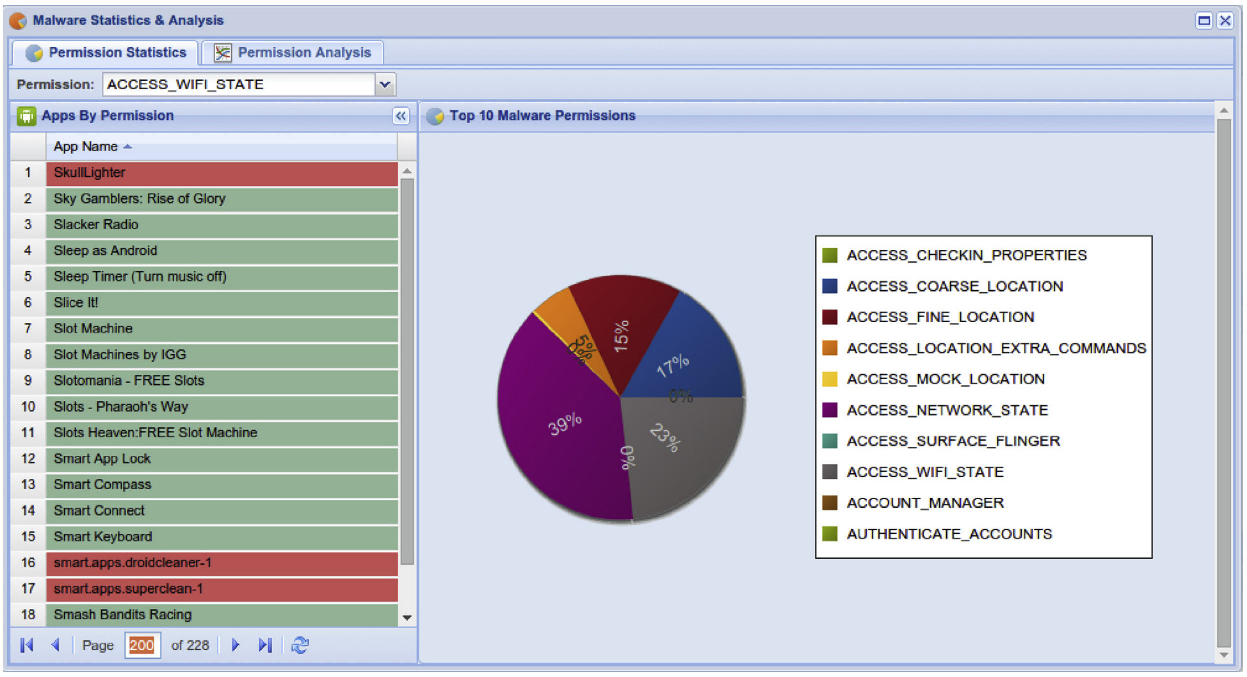
[11] https://42matters.com/api/.

[12] http://contagiodump.blogspot.com.

[13] http://user.informatik.uni-goettingen.de/~darp/drebin/.

[14] http://www.malgenomeproject.org.

[15] https://www.virustotal.com.


Malware Statistics & Analysis

Permission Statistics | Permission Analysis

Permission: ACCESS_WIFI_STATE

Apps By Permission

	App Name
1	SkullLighter
2	Sky Gamblers: Rise of Glory
3	Slacker Radio
4	Sleep as Android
5	Sleep Timer (Turn music off)
6	Slice It!
7	Slot Machine
8	Slot Machines by IGG
9	Slotomania - FREE Slots
10	Slots - Pharaoh's Way
11	Slots Heaven:FREE Slot Machine
12	Smart App Lock
13	Smart Compass
14	Smart Connect
15	Smart Keyboard
16	smart.apps.droidcleaner-1
17	smart.apps.superclean-1
18	Smash Bandits Racing

Page 200 of 228

Top 10 Malware Permissions

Fig. 7. Administration portal user interface for permission statistics.

analyzes suspicious files and URLs and facilitates quick definition of all kinds of malware through 42 antivirus products by vendors such as Symantec, McAfee, Kaspersky and TrendMicro (Vidas and Christin, 2013). Top 10 permissions used by the malwares analyzed are listed in Table 2, which was concluded after these collected applications have been analyzed. The results are consistent with the ones Zhou and Jiang (Zhou and Jiang, 2012) — owners of Android Malware Genome Project that *APK Auditor* uses — published.

The results of the analyses show that permissions to access network state and location are highly requested by

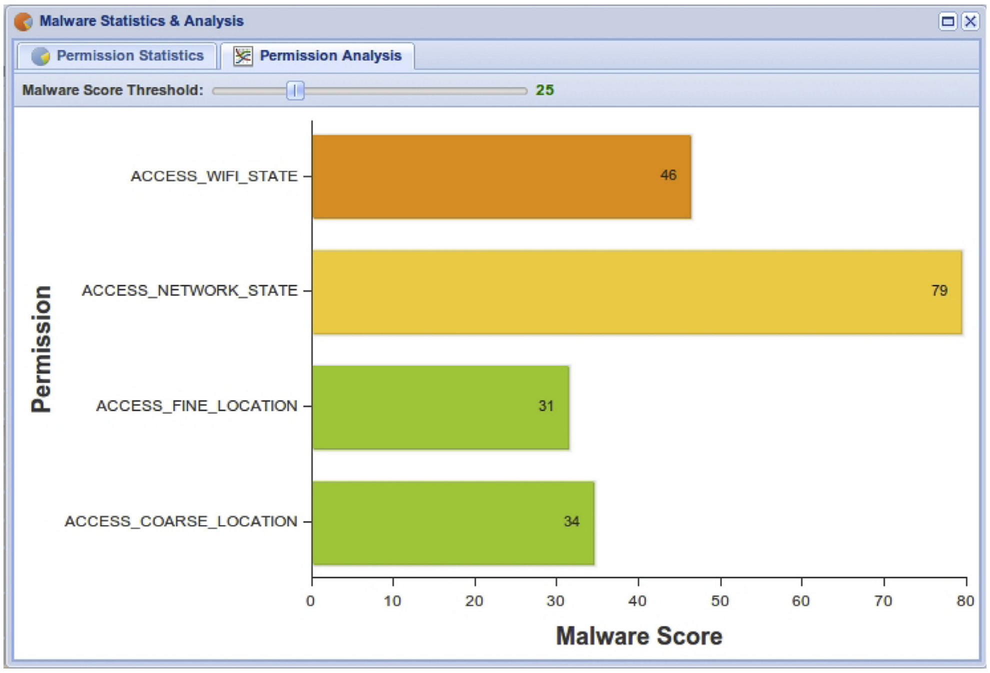


Fig. 8. Administration portal user interface for permission analysis.

malwares. Similarly, permissions related with account management (ACCOUNT_MANAGER, AUTHENTICATE_ACCOUNTS) can be accepted as "risky" and give hints about the trend of malwares. This emphasizes that users must be more careful while granting these permissions during application installation. Table 3 lists permissions that are requested by more than 25% of inspected 6909 malwares.

The best malware score threshold value for classifying applications as benign or malicious is calculated through logistic regression using training dataset. Logistic regression is a method to determine the relationship between a scalar dependent variable and one factor. A generic logistic regression calculation formula is shown in Fig. 9[16] where X represents an independent variable which calculates application's malware score *(AMS)* for *APK Auditor*. The analysis result factor is the final score for the system which is a binary value used to categorize applications as benign or malicious. There is no specific rule to separate training and test dataset but there are some papers (Arguelles-Cruz et al., 2008; Byström, 2013; Mokhtar and Elsayad, 2013; Zhang et al., 2010) that recommend to use 70% of whole dataset for training and the rest 30% for test. Hence 2629 different Android applications are used to evaluate system's malware detection ability *(test phase)*. *APK Auditor*'s malware detection success ratio refers to the rate of the number of correct predictions, and to the number of all predictions which is also known as *accuracy*. It is calculated to be 88.28%, as it is shown in Fig. 10 which equals to 2321 correct detections. "Malware score" data shows application's calculated threat score (AMS) as it has been described earlier. Test results for each application are shown in "Evaluation result" column; each correct prediction is labeled as "Correct" and each false prediction is labeled as "Mismatch". A receiver operating characteristics (ROC) analysis is a technique used to visualize, organize and select classifiers based on their performance (Fawcett, 2006). Table 4 presents ROC analysis of test results on 2629 unique Android applications. 'Positive' refers to benign and 'Negative' refers to malicious applications. *Specificity* or *true negative rate* is calculated as it is shown in Fig. 11. *APK Auditor*'s *specificity* has been calculated to be 0.925.

False Positive (FP) applications can be considered as potential malicious applications or malwares that have not been reported yet. Authors think that *APK Auditor* highlights many potential malwares from Play Store. At least, these applications must be analyzed in depth in order to highlight their real intentions and necessity of permissions they demand. Some permission requests such as geolocation and WIFI accesses are commonly demanded by malicious applications as it is shown in Table 3. If a benign application contains these permissions then it might be qualified as 'malicious'. This can be accepted as another reason for FP results. False Negative (FN) applications are those that need only a few permissions for their malicious actions. Despite the fact that most malwares demand many permissions, there are some malwares that are in need only of a few permissions for their malicious actions. *APK Auditor* is able to detect well-known malwares which are confirmed by different security reports including malware clones of the popular game Flappy Birds,[17] com.keji.danti,[18] com.gp.solitaire,[19] com.zwv.flyapp,[20] com.app.lotte.auth,[21] solution.newsadroid,[22] com.soft.android.installer,[23] com.km.launcher,[24] com.stephbriggs5.batteryimprove,[25] com.convertoman.proin,[26] Jk7H.PwcD,[27] com.bratolubzet.stlstart.[28]

Despite the fact that there are some tools such as VirusTotal,[29] NVISO ApkScan[30], and CopperDroid[31] that offer malware analysis, most of these services have application size limit (i.e. VirusTotal does not scan applications larger than 20 MB (Vidas and Christin, 2013)). According to the developers, Anubis/Andrubis[32] is a service for analyzing malware with remote upload support up to 8 MB file size limit per file. Even though we have submitted many sample tasks both using the script[33] this system provides and the web user interface, we did not achieve any helpful results. During of an analysis for a sample is shown in days and interestingly it increased day after day. After two weeks, a notification mail was received from Anubis stating the uploaded file – which was a valid .apk file downloaded from Google Play Store – could not be executed. Our unsuccessful attempts are presented in Appendix A. Since our system is capable of working with large datasets and most of them have larger size than defined limitations of other services, we are not able to compare our results with another malware detection/analysis tool.

APK Auditor Administration Portal's user interfaces show important details of analyses such as; top 10 permissions used by malwares, and the permissions that are requested by a user-defined percentage of malwares. Because it is a web-based system, digital investigators who

16 http://luna.cas.usf.edu/~mbrannic/files/regression/Logistic.html.

17 https://www.virustotal.com/en/file/a0010710501c32d728c44f8ffddc63fbe57f11d8d1810f07652341c99bd54964/analysis/.

18 http://foresafe.com/report/ED8621092D77E747F26B92B6FFDCF095.

19 https://www.trustlook.com/report/AE0FC35D759DD515D8AD72D688D20019/com.gp.solitaire/.

20 http://www.avgthreatlabs.com/android-app-reports/app/com.zwx.flyapp.

21 https://www.virustotal.com/en/file/d95d7d2b6e3f73e83d93ec4df4afb681db93697e100011f6486fdfc44fbead34/analysis/1378988957/.

22 http://www.avgthreatlabs.com/android-app-reports/app/solution.newsandroid.

23 http://www.avgthreatlabs.com/android-app-reports/app/com.soft.android.appinstaller/.

24 http://www.avgthreatlabs.com/android-app-reports/app/com.km.launcher/.

25 http://www.avgthreatlabs.com/android-app-reports/app/com.stephbriggs5.batteryimprove.

26 https://www.virustotal.com/en/file/b381def002304f62dcdbefdd1165c857ebabd60d7f3c8db10f690c2b8a05d8c7/analysis/.

27 http://www.avgthreatlabs.com/android-app-reports/app/jk7h.pwcd/.

28 http://www.avgthreatlabs.com/android-app-reports/app/com.bratolubzet.stlstart/.

29 https://www.virustotal.com.

30 http://apkscan.nviso.be.

31 http://copperdroid.isg.rhul.ac.uk/copperdroid/.

32 https://anubis.iseclab.org.

33 https://anubis.iseclab.org/Resources/submit_to_anubis.py.

Table 2
Top 10 permissions used by inspected malwares and their existence ratio.

Permission	Ratio
ACCESS_CHECKIN_PROPERTIES	1%
ACCESS_COARSE_LOCATION	17%
ACCESS_FINE_LOCATION	15%
ACCESS_LOCATION_EXTRA_COMMANDS	5%
ACCESS_MOCK_LOCATION	1%
ACCESS_NETWORK_STATE	39%
ACCESS_SURFACE_FLINGER	1%
ACCESS_WIFI_STATE	23%
ACCOUNT_MANAGER	1%
AUTHENTICATE_ACCOUNTS	1%

Table 3
Permissions that are requested by more than 25% of inspected 6909 malwares.

Permission
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$

Fig. 9. Logistic regression generic formula.

Table 4
ROC analysis of test results on 2629 unique Android applications.

	Positive (P)	Negative (N)	Total
True (T)	21	2300	2321
False (F)	123	185	308
Total	144	2485	2629

want to get further details of malware analyses can use this open access portal from http://app.ibu.edu.tr:8080/apkinspectoradmin with by using "*guest*/*Guest1*" login credentials. A user interface has been implemented into *APK Auditor* Android client in order to direct these users to administration portal.

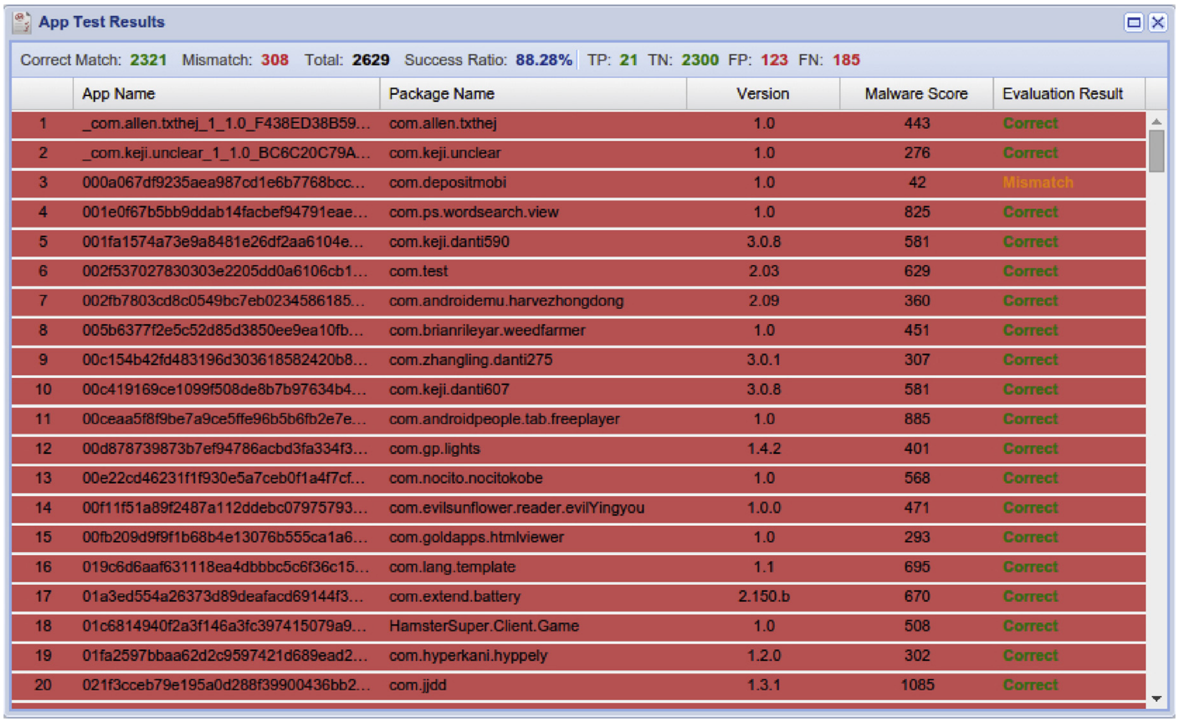
Conclusions

APK Auditor is a learning-based, extensible and lightweight system that provides a new approach for Android malware detection. The purpose of this system is to help digital investigators, and likely Android users check whether or not applications are malicious. Contributions of this study can be summarized as:

- This work provides a new approach to assessing potential maliciousness of Android applications by

$$Specificity = TN/N = TN/(FP+TN)$$

Fig. 11. Specificity calculation formula.



App Test Results

Correct Match: 2321 Mismatch: 308 Total: 2629 Success Ratio: 88.28% TP: 21 TN: 2300 FP: 123 FN: 185

	App Name	Package Name	Version	Malware Score	Evaluation Result
1	_com.allen.txthej_1_1.0_F438ED38B59...	com.allen.txthej	1.0	443	Correct
2	_com.keji.unclear_1_1.0_BC6C20C79A...	com.keji.unclear	1.0	276	Correct
3	000a067df9235aea987cd1e6b7768bcc...	com.depositmobi	1.0	42	Mismatch
4	001e0f67b5bb9ddab14facbef94791eae...	com.ps.wordsearch.view	1.0	825	Correct
5	001fa1574a73e9a8481e26df2aa6104e...	com.keji.danti590	3.0.8	581	Correct
6	002f537027830303e2205dd0a6106cb1...	com.test	2.03	629	Correct
7	002fb7803cd8c0549bc7eb0234586185...	com.androidemu.harvezhongdong	2.09	360	Correct
8	005b6377f2e5c52d85d3850ee9ea10fb...	com.brianrileyar.weedfarmer	1.0	451	Correct
9	00c154b42fd483196d303618582420b8...	com.zhangling.danti275	3.0.1	307	Correct
10	00c419169ce1099f508de8b7b97634b4...	com.keji.danti607	3.0.8	581	Correct
11	00ceaa5f8f9be7a9ce5ffe96b5b6fb2e7e...	com.androidpeople.tab.freeplayer	1.0	885	Correct
12	00d878739873b7ef94786acbd3fa334f3...	com.gp.lights	1.4.2	401	Correct
13	00e22cd46231f1f930e5a7ceb0f1a4f7cf...	com.nocito.nocitokobe	1.0	568	Correct
14	00f11f51a89f2487a112ddebc07975793...	com.evilsunflower.reader.evilYingyou	1.0.0	471	Correct
15	00fb209d9f9f1b68b4e13076b555ca1a6...	com.goldapps.htmlviewer	1.0	293	Correct
16	019c6d6aaf631118ea4dbbbc5c6f36c15...	com.lang.template	1.1	695	Correct
17	01a3ed554a26373d89deafacd69144f3...	com.extend.battery	2.150.b	670	Correct
18	01c6814940f2a3f146a3fc397415079a9...	HamsterSuper.Client.Game	1.0	508	Correct
19	01fa2597bbaa62d2c9597421d689ead2...	com.hyperkani.hyppely	1.2.0	302	Correct
20	021f3cceb79e195a0d288f39900436bb2...	com.jjdd	1.3.1	1085	Correct

Fig. 10. Administration portal user interface to monitor system test results.

calculating a statistical score through the requested permissions.

- Application analysis is completely carried out on a central server and the results are retrieved by a web service. This approach avoids downloading malware on the Android device, and does not decrease the overall system performance during the analysis since hardware resources of smartphones are not consumed.
- The web-based portal provides a broader analysis of Android malwares such as top 10 permissions used by malwares, permissions limited by a threat score to give further idea to users.
- *APK Auditor* keeps track of new malware trends without major modifications in the future since *APK Auditor*'s signature database is being enriched with new Android applications on a daily base. Thus, users of *APK Auditor* will be aware of the most potentially dangerous permissions and their implications.
- *APK Auditor* Android client provides a simple and user-friendly interface to Android users to give a clear idea about the application before installation.
- *APK Auditor* Android client provides two different ways of analyzing an Android application: (a) search through Play Store, and (b) application (`.apk` file) upload to analyze applications from other Android markets/sources.
- *APK Auditor* Android client provides a user-interface to limit memory usage per application. This process is not performed automatically because some benign Android applications may also use high level of memory for their complex activities and the capacity of memory varies on different devices.

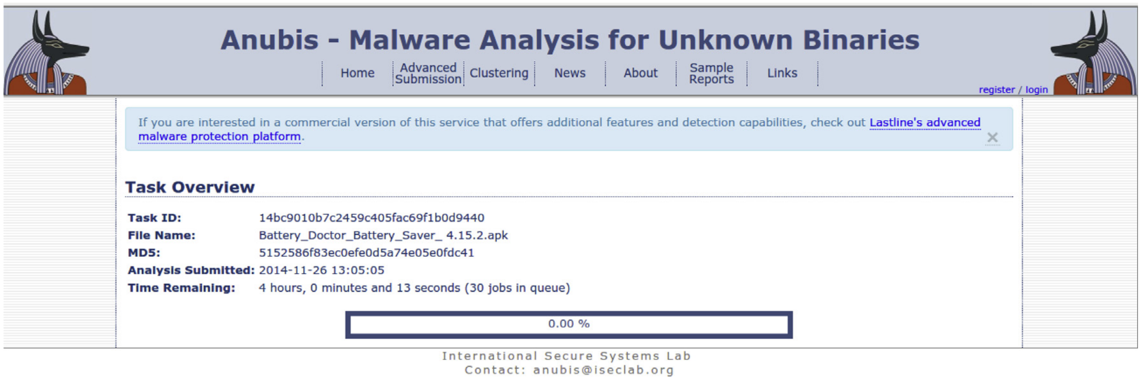
APK Auditor is designed to be a lightweight application that can be used by as many smartphones as possible. The only requirements for using the system is to install the *APK Auditor* client on an Android device that has access to the applications to be analyzed. The administration portal is deployed on a remote server and provides web based user interfaces through which the *APK Auditor* client directs advanced users for further analysis. Standard users may benefit from *APK Auditor* indirectly through the reports that are generated by the system. Authors think that these user interfaces improve users' understanding in applications by analyzing their permissions. During this study, 8762 Android applications have been collected and analyzed. The *APK Auditor* success ratio has been calculated to be 88.28% with a 0.925 specificity after the tests on 2629 unique Android applications. *APK Auditor* regularly downloads and analyzes applications from Play Store to have an up-to-date signature database. Because it is a learning-based mechanism, each application analysis affects malware detection process positively and updates the signature database.

One of the most important future enhancements of *APK Auditor* will be decreasing the number of FP and FN detections. With further analysis including source code inspection, *APK Auditor* will be able to detect permissions' real intentions and applications will be evaluated in a more solid way. On the other hand, the developed system has some external dependencies such as *APIfy* and *42matter App Market API* to query and download Android applications. External system dependencies can make the whole system unavailable when these dependencies are out of service. These dependencies could be considered as a drawback and alternatively can be replaced by local services offering a standalone application that does not depend on any other systems. This study will continue with use of different datasets to check whether the system's success ratio converges to 100%.

Acknowledgments

Authors would like to thank Computer Security Group – University of Göttingen for sharing the Drebin dataset and Zhou Y., Jiang X. for sharing the dataset of Android Malware Genome Project.

Appendix A



If you are interested in a commercial version of this service that offers additional features and detection capabilities, check out Lastline's advanced malware protection platform.

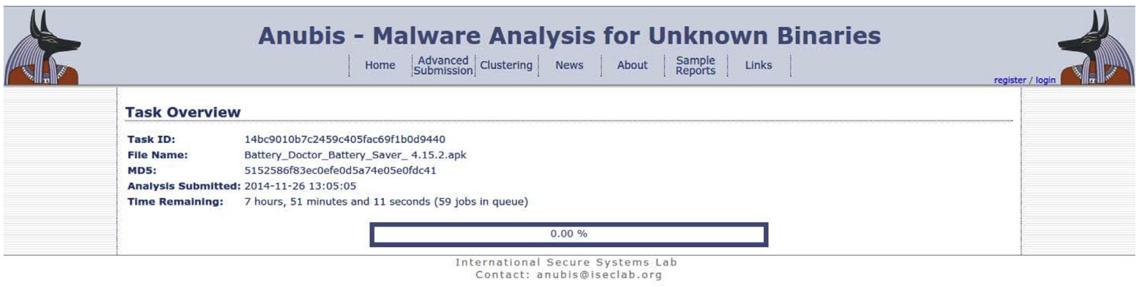
Task Overview

Task ID:	14bc9010b7c2459c405fac69f1b0d9440
File Name:	Battery_Doctor_Battery_Saver_ 4.15.2.apk
MD5:	5152586f83ec0efe0d5a74e05e0fdc41
Analysis Submitted:	2014-11-26 13:05:05
Time Remaining:	4 hours, 0 minutes and 13 seconds (30 jobs in queue)

0.00 %

International Secure Systems Lab
Contact: anubis@iseclab.org

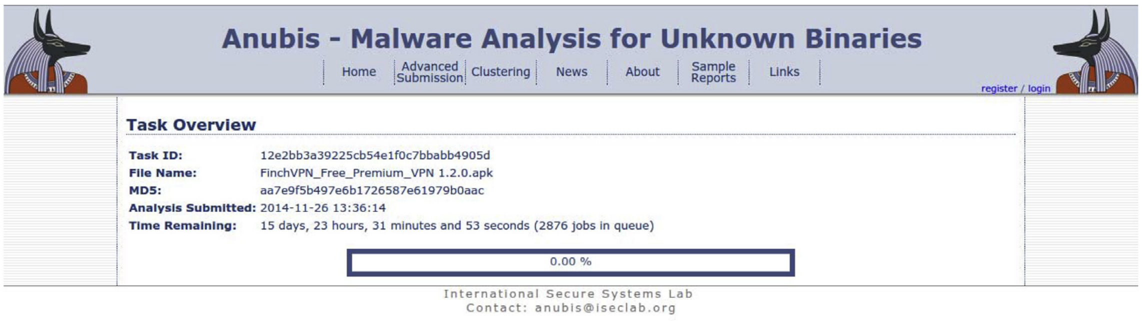
Appendix A.1. Anubis web upload initial expected completion time.


Anubis - Malware Analysis for Unknown Binaries
Home
Advanced Submission
Clustering
News
About
Sample Reports
Links
register / login
0.00 %
International Secure Systems Lab
Contact: anubis@iseclab.org

Task Overview

Task ID:	14bc9010b7c2459c405fac69f1b0d9440
File Name:	Battery_Doctor_Battery_Saver_ 4.15.2.apk
MD5:	5152586f83ec0efe0d5a74e05e0fdc41
Analysis Submitted:	2014-11-26 13:05:05
Time Remaining:	7 hours, 51 minutes and 11 seconds (59 jobs in queue)

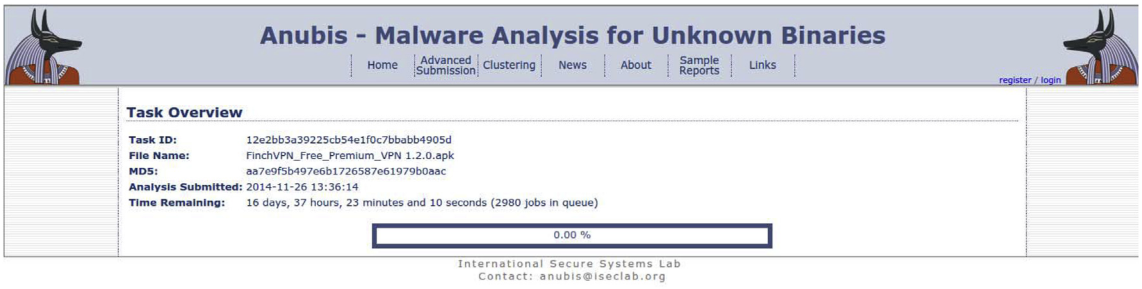
Appendix A.2. Anubis web upload initial expected completion time at the following day of upload.


Anubis - Malware Analysis for Unknown Binaries
Home
Advanced Submission
Clustering
News
About
Sample Reports
Links
register / login
0.00 %
International Secure Systems Lab
Contact: anubis@iseclab.org

Task Overview

Task ID:	12e2bb3a39225cb54e1f0c7bbabb4905d
File Name:	FinchVPN_Free_Premium_VPN 1.2.0.apk
MD5:	aa7e9f5b497e6b1726587e61979b0aac
Analysis Submitted:	2014-11-26 13:36:14
Time Remaining:	15 days, 23 hours, 31 minutes and 53 seconds (2876 jobs in queue)

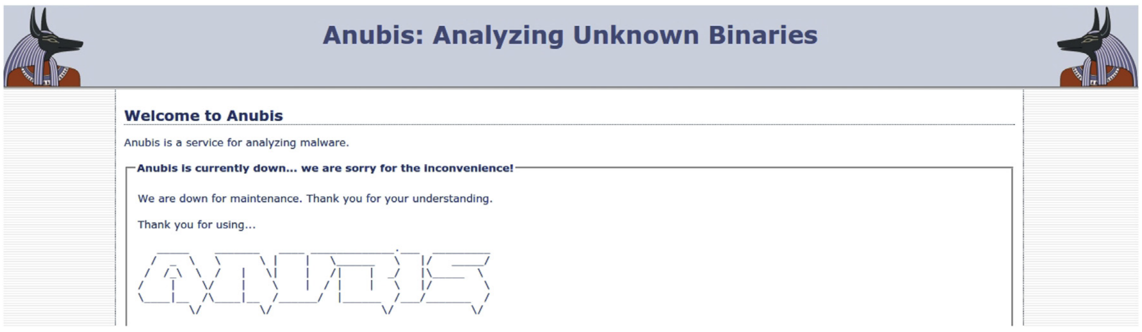
Appendix A.3. Anubis script upload initial expected completion time.


Anubis - Malware Analysis for Unknown Binaries
Home
Advanced Submission
Clustering
News
About
Sample Reports
Links
register / login
0.00 %
International Secure Systems Lab
Contact: anubis@iseclab.org

Task Overview

Task ID:	12e2bb3a39225cb54e1f0c7bbabb4905d
File Name:	FinchVPN_Free_Premium_VPN 1.2.0.apk
MD5:	aa7e9f5b497e6b1726587e61979b0aac
Analysis Submitted:	2014-11-26 13:36:14
Time Remaining:	16 days, 37 hours, 23 minutes and 10 seconds (2980 jobs in queue)

Appendix A.4. Anubis script upload initial expected completion time at the following day of upload.



Welcome to Anubis

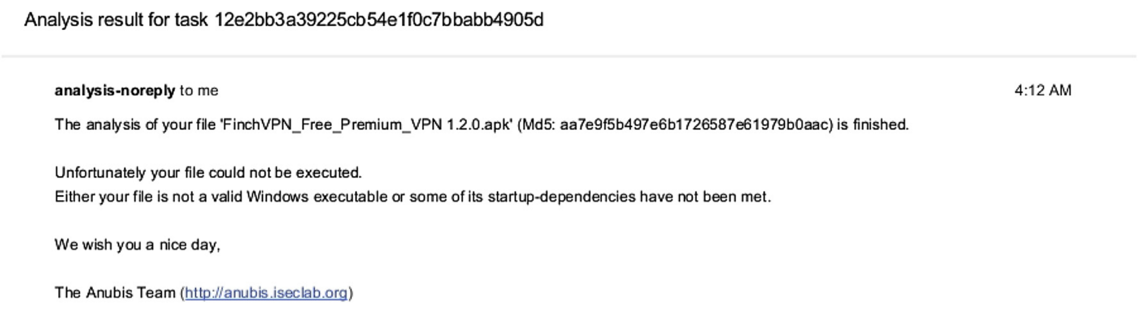
Anubis is a service for analyzing malware.

Anubis is currently down... we are sorry for the inconvenience!

We are down for maintenance. Thank you for your understanding.

Thank you for using...

Appendix A.5. Anubis analysis service is down.


Analysis result for task 12e2bb3a39225cb54e1f0c7bbabb4905d

analysis-noreply to me 4:12 AM

The analysis of your file 'FinchVPN_Free_Premium_VPN 1.2.0.apk' (Md5: aa7e9f5b497e6b1726587e61979b0aac) is finished.

Unfortunately your file could not be executed.
Either your file is not a valid Windows executable or some of its startup-dependencies have not been met.

We wish you a nice day,

The Anubis Team (http://anubis.iseclab.org)

Appendix A.6. Anubis analysis is failed.

References

Android developer guide permission [WWW document]. Google. URL, http://developer.android.com/guide/topics/manifest/permission-element.html; 2014 [accessed 25.12.14].

Android security overview [WWW document]. Google. URL, http://source.android.com/devices/tech/security/ [accessed 25.12.14].

Android developers API guide manifest [WWW document]. Google. URL, http://developer.android.com/guide/topics/manifest/manifest-element.html#package; 2014 [accessed 25.12.14].

Arguelles-Cruz AJ, López-Yáñez I, Aldape-Pérez M, Conde-Gaxiola N. Alpha-beta weightless neural networks. Lect Notes Comput Sci Incl Subser Lect Notes Artif Intell Lect Notes Bioinform 2008;5197: 496–503. http://dx.doi.org/10.1007/978-3-540-85920-8_61.

Arp D, Spreitzenbarth M, Malte H, Gascon H, Rieck K. Drebin: effective and explainable detection of Android malware in your pocket. In: Symposium on network and distributed system security (NDSS); 2014. p. 23–6.

Barkuus L, Dey A, Barkhuus L. Location-based services for mobile telephony: a study of users' privacy concerns. In: Proceedings of the INTERACT 2003, 9TH IFIP TC13 international conference on human–computer interaction. Zurich, Switzerland; 2003. p. 709–12.

Barrera D, Oorschot PC Van, Somayaji A. A methodology for empirical analysis of permission-based security models and its application to Android categories and subject descriptors. In: Proceedings of 17th ACM conference on computer and communications security. New York, NY, USA: ACM; 2010. p. 73–84. http://dx.doi.org/10.1145/1866307.1866317.

Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for Android. Science 2011;80:15–25. http://dx.doi.org/10.1145/2046614.2046619.

Byström H. Movie recommendations from user ratings. 2013.

Carl G, Kesidis G, Brooks RR, Rai SRS. Denial-of-service attack-detection techniques. IEEE Internet Comput 2006;10:82–9. http://dx.doi.org/10.1109/MIC.2006.5.

Consolvo S, Smith IE, Matthews T, LaMarca A, Tabert J, Powledge P. Location disclosure to social relations: why, when, & what people want to share. In: CHI 2005 conference on human factors in computing systems. Portland, OR, USA; 2005. p. 81–90. http://dx.doi.org/10.1145/1054972.1054985.

Dini G, Martinelli F, Saracino A, Sgandurra D. MADAM: a multi-level anomaly detector for Android malware. In: Kotenko I, Skormin V, editors. Computer network security. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 240–53. http://dx.doi.org/10.1007/978-3-642-33704-8.

Enck W, Ongtang M, Mcdaniel P. On lightweight mobile phone application certification. In: ACM conference on computer and communications security. Chicago, IL, USA; 2009. p. 235–45. http://dx.doi.org/10.1145/1653662.1653691.

Enck W, Octeau D, McDaniel P, Chaudhuri S. A study of Android application security. In: SEC'11 proceedings of the 20th USENIX conference on security. San Francisco, CA, USA; 2011. p. 21. http://dx.doi.org/10.1007/s00256-010-0882-8.

Fawcett T. An introduction to ROC analysis. Pattern Recognit Lett 2006;27: 861–74. http://dx.doi.org/10.1016/j.patrec.2005.10.010.

Felt AP, Finifter M, Chin E, Hanna S, Wagner D. A survey of mobile malware in the wild. In: SPSM '11 proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices. Chicago, IL, USA; 2011. p. 3–14. http://dx.doi.org/10.1145/2046614.2046618.

Felt AP, Greenwood K, Wagner D. The effectiveness of application permissions. In: Proceeding of the WebApps'11 proceedings of the 2nd USENIX conference on web application development. Berkeley, CA, USA: USENIX Association; 2011b. p. 7.

Felt AP, Ha E, Egelman S, Haney A, Chin E, Wagner D. Android permissions: user attention, comprehension, and behavior. In: Proceedings of the eighth symposium on usable privacy and security – SOUPS '12. Washington, DC, USA; 2012. p. 1. http://dx.doi.org/10.1145/2335356.2335360.

Guido M, Ondricek J, Grover J, Wilburn D, Nguyen T, Hunt A. Automated identification of installed malicious Android applications. Digit Investig 2013;10:96–104. http://dx.doi.org/10.1016/j.diin.2013.06.011.

Internet security threat report. 2014.

Kelley PG, Benisch M, Cranor LF, Sadeh N. When are users comfortable sharing locations with advertisers? Hum Factors 2011:2449–52. http://dx.doi.org/10.1145/1978942.1979299.

Kelley PG, Consolvo S, Cranor LF, Jung J, Sadeh N, Wetherall D. A conundrum of permissions: installing applications on an Android smartphone. In: Blyth J, Dietrich S, Camp LJ, editors. Financial cryptography and data security. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 68–79. http://dx.doi.org/10.1007/978-3-642-34638-5.

King J, Lampinen A, Smolen A. Privacy: is there an app for that?. In: Proceedings of the seventh symposium on usable privacy and security – SOUPS '11. New York, NY, USA: ACM Press; 2011. p. 1. http://dx.doi.org/10.1145/2078827.2078843.

Leiwo J, Aura T, Nikander P. Towards network denial of service resistant protocols. Inf Secur Glob Inf Infrastruct 2000;47:301–10.

Liang S, Du X. Permission-combination-based scheme for Android mobile malware detection. In: 2014 IEEE international conference on communications (ICC). Sydney, Australia: IEEE; 2014. p. 2301–6. http://dx.doi.org/10.1109/ICC.2014.6883666.

Lindqvist J, Cranshaw J, Wiese J, Hong J, Zimmerman J. I'm the mayor of my house. In: Proceedings of the 2011 annual conference on human factors in computing systems CHI 11. New York, NY, USA; 2011. p. 2409. http://dx.doi.org/10.1145/1978942.1979295.

Lookout. App genome report. 2011 [WWW document]. URL, https://www.lookout.com/resources/reports/appgenome [accessed 25.12.14].

Mobile Security Threat Report. Sophos. 2014.

Mokhtar SA, Elsayad AM. Predicting the severity of breast masses with data mining methods. Int J Comput Sci 2013;10.

Park KPK, Lee HLH. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In: Proceedings IEEE INFOCOM 2001. Conference on computer communications. Twentieth annual joint conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213). Anchorage, Alaska, USA; 2001. http://dx.doi.org/10.1109/INFCOM.2001.916716.

Portokalidis G, Homburg P, Anagnostakis K, Bos H. Paranoid Android: versatile protection for smartphones. In: Annual computer security applications conference (ACSAC). Austin, Texas, USA; 2010. p. 347–56. http://dx.doi.org/10.1145/1920261.1920313.

Rastogi V, Chen Y, Enck W. AppsPlayground: automatic security analysis of smartphone applications. In: CODASPY '13 proceedings of the third ACM conference on data and application security and privacy; 2013. p. 209–20. http://dx.doi.org/10.1145/2435349.2435379.

Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors. In: Proceedings of the 6th European workshop on system security (EuroSec). Prague: Czech Republic; 2013.

Sadeh N, Hong J, Cranor L, Fette I, Kelley P. Understanding and capturing people's privacy policies in a mobile social networking application. Pers Ubiquitous Comput 2009;13:401–12. http://dx.doi.org/10.1007/s00779-008-0214-3.

Vidas T, Christin N. Sweetening Android lemon markets: measuring and combating malware in application marketplaces. In: Proceedings of the third ACM conference on data and application security and privacy – CODASPY '13. New York, New York, USA: ACM Press; 2013. p. 197. http://dx.doi.org/10.1145/2435349.2435378.

Vidas T, Christin N, Cranor LF. Curbing Android permission creep. In: Proceedings of the 2011 web 2.0 security and privacy workshop (W2SP 2011). Oakland, CA; 2011.

Vidas T, Cylab ECE, Votipka D, Cylab INI, Christin N. All your droid are belong to us: a survey of current Android attacks. In: Proceedings of the 5th USENIX conference on offensive technologies. San Francisco, CA, USA; 2011. p. 10.

Wu D-J, Mao C-H, Wei T-E, Lee H-M, Wu K-P. DroidMat: Android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia joint conference on information security. Minato, Tokyo, Japan; 2012. p. 62–9. http://dx.doi.org/10.1109/AsiaJCIS.2012.18.

Zhang Y, Orgun MA, Baxter R, Lin W. An application of element oriented analysis based credit scoring. Lect Notes Comput Sci Incl Subser Lect Notes Artif Intell Lect Notes Bioinform 2010;6171:544–57. http://dx.doi.org/10.1007/978-3-642-14400-4_42.

Zhou Y, Jiang X. Dissecting Android malware: characterization and evolution. In: Proceedings of the 33rd IEEE symposium on security and privacy (Oakland 2012). San Francisco, CA, USA: IEEE; 2012. p. 95–109. http://dx.doi.org/10.1109/SP.2012.16.