Crowdroid: Behavior-Based Malware Detection System for Android

Iker Burguera and Urko Zurutuza
Electronics and Computing Department
Mondragon University
20500 Mondragon, Spain
iker.burguera@alumni.eps.mondragon.edu,
uzurutuza@mondragon.edu

Simin Nadjm-Tehrani
Dept. of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
simin.nadjm-tehrani@liu.se

ABSTRACT

The sharp increase in the number of smartphones on the market, with the Android platform posed to becoming a market leader makes the need for malware analysis on this platform an urgent issue.

In this paper we capitalize on earlier approaches for dynamic analysis of application behavior as a means for detecting malware in the Android platform. The detector is embedded in a overall framework for collection of traces from an unlimited number of real users based on crowdsourcing. Our framework has been demonstrated by analyzing 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. The method is shown to be an effective means of isolating the malware and alerting the users of a downloaded malware. This shows the potential for avoiding the spreading of a detected malware to a larger community.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software (e.g., viruses, worms, Trojan horses); H.2.8 [**Database Applications**]: Data Mining)

General Terms

Security, Experimentation

Keywords

smartphone security, malware detection, anomaly detection, dynamic analysis, crowdsourcing, intrusion detection, data mining

1. INTRODUCTION

Malware has threatened PCs for many years. Due to the high growth of smartphone sales, it was only a question of time when malware developers would get interested in smartphone platforms to perform attacks. According to a study made by International Data Corporation, smartphone vendors will ship more than 450 million devices in 2011, compared to the 303.4 million units shipped in 2010 [24]. Moreover, the smartphone market will grow four times faster than mobile phone market and the demand of smartphones will rise considerably reaching the point where customers will replace their old mobile phones with smartphones.

The sales growth of mobile phone companies like Samsung and HTC between 2009-2010, has revolutionized the smartphone market. According to this, IDC predicted that Android OS would pass Nokia's Symbian OS in 2011 and would continue leading the smartphone OS Market in the upcoming years. Furthermore, they predicted that Android OS and Windows Mobile would grow almost 50% between 2010-2014, having a high probability of becoming leaders of smartphones Operating Systems vendors in the future.

Google's Android Market [19] is the official online mechanism for delivering software to an Android based smartphone. Unfortunately Android application developers can upload their applications without any check of their trustworthiness. The applications are self signed by developers themselves, without the intervention of any certification authority. Unofficial repositories also exist, where developers can upload applications, including cracked applications or trojan horses. This has allowed malicious attackers to upload malware to the Market [1] and also to spread malware through unofficial repositories.

According to Juniper Networks, their Global Threat Center found a 400% increase in Android malware since summer 2010 [20]. “Fake Player”, “Geinimi”, “PJApps” and “HongToutou” are some known examples. A number of applications have been modified and the malware have been binded, packed and spread through unofficial repositories. Android's Market has also been targeted, where more than 50 infected applications were found in March 2011, all of them infected with “DroidDream” trojan [1] application. Recently John Oberheide, made a proof of concept malware application as an Angry Birds bonus to show the weakness of security of the Android Marketplace [27].

In this paper we propose a new approach to analyze the behavior of Android applications, providing a framework to distinguish between applications that, having the same name

SPSM'11, October 17, 2011, Chicago, Illinois, USA.

Operating System	2011 Market Share	2015 Market Share	2015/2011 Evolution
Android	39.5%	45.4%	23.8%
BlackBerry OS	14.9%	13.7%	17.1%
iOS	15.7%	15.3%	18.8%
Symbian	20.9%	0.2%	-65.0%
Windows mobile 7	5.5%	20.9%	67.9%
Others	3.5%	4.6%	28.0%
Total	100%	100%	19.6%

Table 1: Worldwide Smartphone Operating System Market Share 2011 and 2015

and version, behave differently. The aim is to detect anomalously behaving applications, thus detecting malware in the form of trojan horses.

The main contribution of this work is the use of a crowdsourcing system to obtain the traces of applications' behavior, which helps researchers to collect different samples of application execution traces. These traces can then be used into two different groups, leading to clear differentiation between the benign applications from those containing malware.

Our experimental results show that our system was capable of detecting every malware execution in self-written malware, giving a 100% of detection rate for this particular malware. We also provide results for the analysis and detection of real malware that can be found in the wild.

This work is organized as follows. Section 2 describes related work. In Section 3 we explain the behavior-based malware detection system framework, detailing the process of building a crowdsourcing application to collect and give information about malware detection system internals. In Section 4, we present the results of the malware detection system performed with a set of self-written malware applications as well as applications containing real malware. In Section 5 we conclude and give possible future work to reduce limitations of the system proposed.

2. RELATED WORK

So far two approaches have been proposed for the analysis and detection of malware: static analysis [10, 39] and dynamic analysis [21, 11, 30]. Static analysis, mostly used by antivirus companies, is based on source code or binaries inspection looking at suspicious patterns. Although some approaches have been successful, the malware authors have developed various obfuscation techniques especially effective against static analysis [26]. On the other hand, dynamic analysis or behavior-based detection involves running the sample in a controlled and isolated environment in order to analyze its execution traces. Egele [14] provides a complete overview of automated dynamic malware analysis techniques.

David Dagon et al. alerted the community in 2004 predicting the feasibility of malware in mobile phones [12]. Even if wi-fi and bluetooth were considered as the most probable infection paths, the growth of smartphone sales with continuous Internet connectivity made the prediction come true. Concretely, in June of the same year, the first malware specifically written for Symbian OS platform was discovered [7]. After the infection success carried out by Cabir malware and its variants [8], researchers proposed approaches and developed different mechanisms in order to detect malware in smartphones.

Due to the lack of smartphone malware patterns by that time, most of anomaly detection techniques used the battery power consumption as the main malware detection system feature [22, 6, 23]. These techniques were based on checking and monitoring mobile phones power consumption and comparing them with the normal power consumption pattern to detect anomalies. These techniques are specifically designed to detect attacks targeting battery lifes.

Resource limitations of smartphones have lead researchers to propose collaborative analysis techniques, where the analysis is made by a network of devices. Both static [31] and dynamic analysis [9, 40] have been proposed using these techniques.

Static analysis works have also been proposed for malware detection in individual smartphones. Antivirus companies have adapted their signature-based detection systems to smartphones, but considering the level of resources needed by antivirus techniques and the power and memory constraints of mobile devices, in-phone analysis is not a preferred solution to apply in smartphones. Schmidt et al. proposed the analysis of static function calls from binaries applying a clustering algorithm in [33]. This technique was used to detect Symbian OS malware depending on mobile phones requirements, such as device efficiency, speed and limited resource usage. Anomaly detection was also proposed to detect malware on Symbian devices [5, 34].

Regarding Android Operating System, some authors provide overviews of its security model [17, 36, 16]. One of the most important security measures of Android devices is the permission-based security model. Each application specifies which resources of the device needs to be used, and the user grants or denies it's installation regarding the permissions needed. Analyzing and enforcing Android's permission security model has been proposed by various authors [28, 41, 42, 43, 2, 13]. Even if a user can be warned about the risk of having accepted suspicious permissions, the spreading of real malware has demonstrated that users directly trust any application request and install them on their phones.

As recently proposed by Antivirus companies, static analysis can be deployed for malware detection in Android devices [31]. But due to the limited resources of smartphones, most of the recent proposals for malware detection on Android devices are based on behavior analysis for anomaly detection.

Schmidt et al. proposed a solution based on monitoring events occurring on Linux-kernel level [35]. They reviewed Linux based tools for enhancing security, and extracting features such as system calls, modified files, etc. from the Linux kernel. These features were then used to create a normal model for the smartphone behavior. At that time there were still no real Android devices available, so they

Author	Approach	Detection Method	Platform	Description
Schmidt et al.(2008)[35]	HIDS, NIDS	Anomaly Detection	Android OS	Analyzes the security on Android smartphones from Linux-kernel view. Uses Network traffic, Kernel system calls, File system logs and Event detection modules to detect anomalies in the system.
Schmidt et al.(2009)[32]	HIDS	Signature-Based Detection	Android OS	Performs static analysis on the executables to extract function calls in Android OS using the command readelf. Function calls are compared with malware executables for classification.
Bläsing et al.(2010)[3]	HIDS	Signature-Based Detection	AndroidOS	Uses an Android Application Sandbox to perform Static and Dynamic analysis on Android applications. Static analysis scans Android source code to detect Malware patterns. Dynamic analysis executes and monitors Android applications in a totally secure environment.
Enck et al.(2010)[15]	HIDS,NIDS	Anomaly Detection	Android OS	TaintDroid is a real time monitoring system for Android OS. TaintDroid monitors Android applications and alerts the user whenever a sensitive data of the user is compromised. Uses "taint tracking" analysis to monitor privacy sensitive information.
Portolakidis et al.(2010)[29]	HIDS,NIDS	Anomaly Detection	Android OS	A remote security server in the cloud performs the Malware detection analysis. Virtual environments will be used to analyze Android mobile phone replicas.
Shabtai et al.(2010)[37]	HIDS	Anomaly Detection	Android OS	Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method (KBTA) methodology. Detects suspicious temporal patterns and to issues an alert if an intrusion is found. These patterns are compatible with a set of predefined classes of malware as defined by a security expert.
Shabtai et al.(2011)[38]	HIDS	Anomaly Detection	Android OS	Host-based malware detection system that continuously monitors smartphone features and events and applies machine learning to classify the collected data as normal (benign) or abnormal (malicious) based on a already known malware and behavior.

Table 2: Android-based malware detection systems

could not test their system properly. Same authors proposed in [3] an Android application sandbox. First they perform static analysis dissembling Android APK files in order to detect Malware patterns. Then, dynamic analysis is carried out, executing and monitoring Android applications in a totally secure environment, also known as Sandbox. During dynamic analysis, all the events occurring in the device (opened files, accessed files, battery consumption, etc.) were monitored. The main drawback of their system is the use of an application that simulates user interaction (known as ADB Monkey), which will never be as real as a user.

Enck et al. presented TaintDroid in [15]. Their system used dynamic taint analysis techniques to monitor sensitive information on smartphones. Thus, they can track a suspicious third-party application that uses sensitive data as GPS location information or address book information. An application using sensitive data does not necessarily correspond to malware, though.

In Paranoid Android [29] Portolakidis et al. proposed a system where researchers can perform a complete malware analysis in the cloud using mobile phone replicas. Their approach needs running those replicas in a secure virtual environment, limiting their system to no more than 105 replicas running concurrently. Then different malware detection techniques can be applied.

Finally, Shabtai et al. presented in [37] a methodology to detect suspicious temporal patterns as malicious behavior, known as knowledge-based temporal abstraction. Later in [38] same authors presented Andromaly, as a a framework for anomaly detection on Android smartphones. Both works use knowledge-based analysis while our system is behavior based. These can be complementary techniques. Even though, their approach is recommended for detecting continuous attacks (e.g., DoS, worm infection), and our framework detects trojan horses, the most frequently seen attacks nowadays. Once more, authors could not find real malware to test their proposal.

Table 2 shows a summarized view of Android-based malware detection works.

3. BEHAVIOR-BASED MALWARE DETECTION SYSTEM FRAMEWORK

The implementation of malware detection systems in mobile devices is a relatively new concept. Security tools and mechanisms used in computers are not feasible for applying on smartphones due to the excessive resource consumption and battery depletion. Hence, we decided to perform the whole analysis process on a dedicated remote server. This server will be used exclusively to collect information and detect malicious and suspicious applications in the Android platform.

Our framework is composed of several components which provide enough resources and mechanisms to detect malware on the Android platform. First, we have developed a lightweight client called Crowdroid, which can be downloaded and installed from Google's Market. This application is in charge of monitoring Linux Kernel system calls and sending them preprocessed to a centralized server. According to a crowdsourcing philosophy, users will help with sending non-personal, but behavior-related data of each application they use. These applications could have been downloaded both from the official Market and also from unofficial repositories, as shown in figure 1.

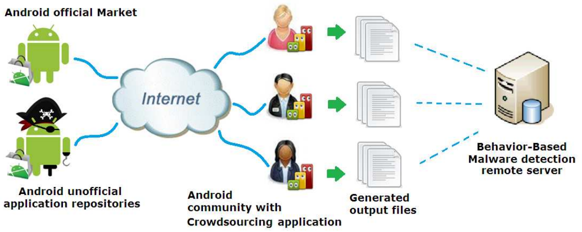
Android official Market
Internet
Android unofficial application repositories
Android community with Crowdsourcing application
Generated output files
Behavior-Based Malware detection remote server

Figure 1: Behavior-Based Malware Detection Framework

Then, the remote server will be in charge of parsing data, and creating a system call vector per each interaction of the users within their applications. Thus, a dataset of behavior data will be created for every application used. The more users using our Crowdroid application, the more complete and accurate will be our system. Finally, we cluster each dataset using a partitional clustering algorithm. This way we can differentiate between benign applications that demonstrate very similar system call patterns, and malicious trojan applications that, even if having the same name and identifier, have a different behavior in terms of distance between example vectors. Partitional clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Each cluster may be represented by a centroid or a cluster representative. Partitioning algorithms either try to discover clusters by iteratively relocating points between subsets (probabilistic clustering, medoids methods, k-means methods), or try to identify clusters as areas highly populated with data (density-based clustering). We chose k-means algorithm [25] due to it's simplicity, efficiency, speed, and a known number of $k = 2$ clusters as an input parameter: we know that an application will be benign, or malicious.

In Linux, a system call is how a program requests a service from the operating system's kernel. Linux kernel 2.6.23 has more than 250 system calls and each one is identified by a unique number that is written in the kernel's system call table. System calls provide useful functions to application programs like network, file, or process related operations. As shown in figure 2, when an application from user space makes a request to the Operating System, the petition goes through *glibc* library, System Call Interface, Kernel and finally to Hardware. *glibc* library interprets the petition and CPU switches to kernel mode to execute the appropriate kernel function looking into system call table. The kernel will be responsible for understanding the petition and making the request to the hardware platform. Afterwards the user gets the information requested by the application in the user space in an inverse process. Functions like *getpid()*, *open()*, *read()* and *socket()* are some of the functions that *glibc* can provide applications to invoke a system call.

Linux kernel is executed in the lowest layer of Android architecture. This means that all requests made from upper layers pass through the kernel using system call interface before they're executed in hardware. Capturing and analyzing the system calls that pass through system call interface, will provide accurate information about the behavior of the application. Crowdroid will use a tool available in Linux called *Strace* to collect the system calls. The aim of hijacking these system calls, is to generate an output file with all events generated by the Android application. This file will provide useful information, like opened and accessed files, execution time stamps and the count of each system call number executed by the application. We will use this last feature to represent the behavior of each Android application execution.

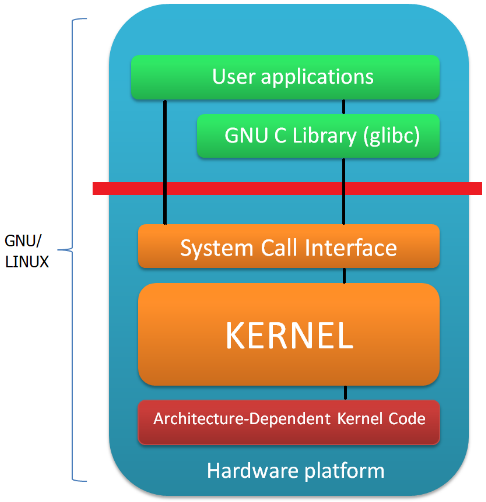

Figure 2: Linux User and Kernel space

Next we see an example of an Android application behavior system call feature vector. Each element represents a count of the specific system call requested. Note that the complete list of Android system calls is too large to show in this paper, but the reader can find them on the Linux kernel manual pages.

```
0 ,0 ,0 ,25 ,47 ,4 ,34 ,0 ,0 ,0 ,0 ,0 ,0 ,12 ,0 ,0 ,0 ,0 ,0 ,260 ,9 ,0 ,0 ,0 ,0 ,
1649 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,10 ,0 ,0 ,0 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,22 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,3466 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,12 ,0 ,0 ,0 ,0 ,
132 ,0 ,0 ,0 ,0 ,0 ,0 ,40 ,41 ,0 ,0 ,0 ,0 ,0 ,76 ,0 ,0 ,00 ,0 ,0 ,4 ,0 ,87 ,17 ,0 ,...
```

Each number separated by commas, represents how many request/executions have been made by a specific Android application during the monitoring process. For instance, the system call *open()* is used 25 times and *kill()* 47 times. This means that the monitored application opened files or system libraries 25 times and killed processes 47 times, and so on.

The framework creates as many datasets as application identifiers, so we need as many Crowdroid users as possible to enrich the database and provide enough information including benign and malicious application traces until the system can discover anomalies. Figure 3 shows the architecture of the whole framework. Summarizing, the framework relies on three components: data acquisition, data manipulation, and the malware analysis and detection system.

- **Data acquisition:** This component is responsible for obtaining application data from users, using the Crow-

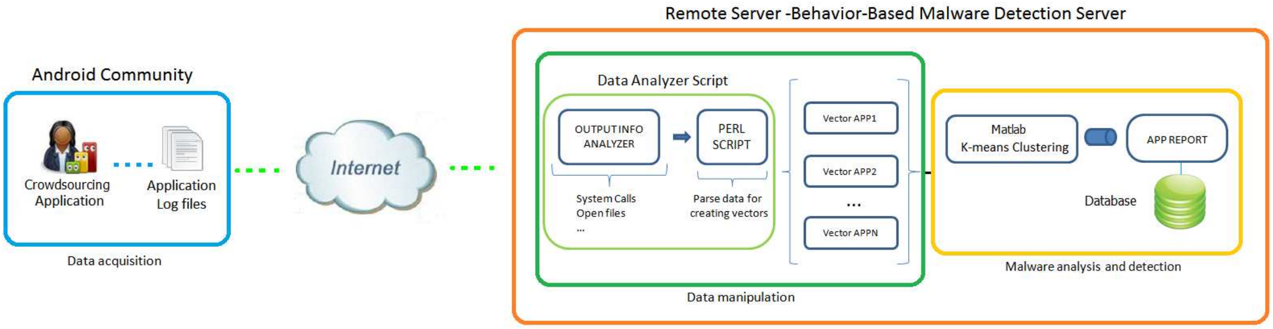

Figure 3: Android Malware Detection process

droid application. Collected data is composed by basic device information, installed applications list and the result of monitoring applications with *Strace* tool system calls log file.

- **Data manipulation:** This component is responsible for managing and parsing all the information collected from Android users. It collects, extracts and analyzes received information from *Strace* output files. Devices' basic information are stored in a central database, and system call traces are processed to produce the feature vectors that will be used for clustering.
- **Malware analysis and detection:** This component is responsible for analyzing and clustering the feature vectors obtained from the previous phase in order to create the normality model and detect anomalous behavior in Android applications, using K-means clustering over system call count feature vectors.

4. EXPERIMENTAL RESULTS

In this section we provide the detailed results of the experiments carried out using the proposed framework. We test our system with different types of malware. First, self-written malware is used, giving us a 100% of detection rate. In order to create the normality model of self-written programs, a calculator, a countdown application, and a money converter have been used (Calculator_G, Countdown_G, MoneyConverter_G). We have developed modified versions of those applications in order to simulate trojan malware.

Next, we have tested the whole framework using two real malware specimens: PJApps [45] contained in Steamy Window application, and HongTouTou [4] trojan binded in Monkey Jump 2 application. We used Virustotal Malware Intelligence Service [44] to obtain the infected applications, and manually downloaded the benign applications from the Market. In this case, we obtained a 100% detection accuracy for PJApps, and an 85% for the HongToutou trojan.

The set up for the experiments can be summarized as follows:

- We have used 20 clients running Crowdroid application. The idea is that in the future more users will contribute to the system, and thus a richer dataset will be in our system.
- We consider 60 interactions of users with each application enough for malware discovery in self-written applications, while real malware have been tested with our prototype using 6 and 20 interactions. This was due to the lack of available crowd at the time of the experiments.
- The data transfer/communication between the client application and the server uses FTP protocol.
- Our system will create models to differentiate between benign and malicious applications, like trojans. An unknown malware with no respective goodware will not be discovered.
- We assume that benign applications are those which have been executed more times. A trojan will be uploaded to an unofficial repository later than the original application, as malware writers need the original applications in order to create such programs.

4.1 Self-Written Malware

The results of our behavior-based Android malware detection system on self written applications are shown in table 3.

In order to test the system, we obtained 60 execution traces from each of the self-written applications. 50 traces of benign applications and 10 traces of malicious ones. The 50 benign interactions will represent the normality model of the application. The system collects all generated output files from every interaction and creates 3 files, one per application. We finally obtain 60 feature vectors one per application (calculator, countdown, money converter) including goodware and malware application behavior feature vectors.

App.	**Interactions**		**Clustering result**		**Detection rate**
	Good App	Malware App	Good Clustered	Malware Clustered	
Calculator	50	10	50	10	100%
Countdown	50	10	50	10	100%
Money Converter	50	10	50	10	100%

Table 3: Self-written malware result

As shown in the table above, the system was able to classify the feature vectors in two different clusters, grouping the benign applications interactions and malicious ones correctly.

4.2 Real Malware

As the results obtained with self-written malware in the system were successful, we decided to make a deeper analysis for malware contained in Steamy Window and Monkey Jump 2 applications, using the Crowdroid client.

4.2.1 *Steamy Window application with PJApps malware*

Steamy Window is a free application available at the Android Market that covers the screen of the smartphone with steam and lets the user to wipe it off with the fingers. The malicious version of the application containing PJApps malware, which was discovered in unofficial repositories, sends sensitive information containing the IMEI, Device ID, Line Number and Subscriber ID to a web server. Then the infected smartphone gets registered in a Command and Control botnet waiting for instructions. It has the ability to send text messages to premium-rate numbers, SMS-spamming, install more applications, navigate, and even bookmark web sites.

For Steamy Window, six interactions were performed to test the system, 4 using the original Steamy Window application and 2 with malicious code of PJApps malware attached. System call feature vectors where collected and clustered with k-means algorithm. Figure 4 shows how the system obtained the applications from different sources. Some users installed Android's Official Steamy Window application and others downloaded the application from Android unofficial repositories.

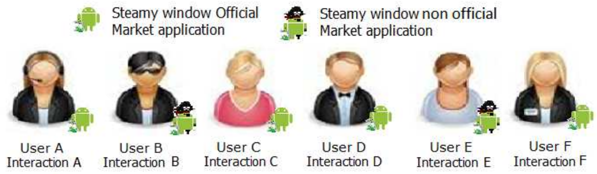

Figure 4: Six different users running the experiment

Next, we show the feature vectors of processed system calls collected during the different interactions of users with the application, each one consisting in the interactions made by the users of figure 4:

```
Interaction_A= 0 ,0 ,0 ,3 ,7 ,7 ,7 ,0 ,0 ,1 ,1 ,0 ,0 ,11 ,0 ,1 ,0 ,0 ,0 ,3 ,
438 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,0 ,0 ,0 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,3 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,12 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,12 ,7 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,8 ,1 ,3 ,4065 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,2 ,0 ,0 ,0 ,0 ,2 ,2 ,0 ,
0 ,0 ,0 ,0 ,14011 ,0 ,0 ,0 ,0 ,0 ,648 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,6 ,0 ,0 ,0 ,0 ,0 ,
0 ,12 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,


Interaction_B =0 ,0 ,0 ,34 ,43 ,45 ,87 ,0 ,0 ,5 ,5 ,0 ,0 ,47 ,0 ,5 ,0 ,
0 ,0 ,31 ,2695 ,0 ,0 ,0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,22 ,0 ,0 ,0 ,5 ,5 ,0 ,0 ,27 ,
0 ,0 ,0 ,46 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,48 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,16 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,132 ,88 ,0 ,0 ,0 ,0 ,
2 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,60 ,5 ,27 ,13717 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,16 ,0 ,0 ,0 ,0 ,68 ,262 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,2328 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,38 ,0 ,0 ,0 ,0 ,0 ,0 ,132 ,0 ,0 ,0 ,0 ,0 ,2 ,0 ,0 ,0 ,1 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,

Interaction_C =0 ,0 ,0 ,19 ,12 ,28 ,29 ,0 ,0 ,1 ,1 ,0 ,0 ,22 ,0 ,1 ,0 ,
0 ,0 ,19 ,1718 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,11 ,0 ,0 ,0 ,3 ,1 ,0 ,0 ,
4 ,0 ,0 ,0 ,8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,36 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,41 ,21 ,0 ,0 ,0 ,0 ,2 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,24 ,1 ,19 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,6 ,
0 ,0 ,0 ,0 ,27 ,15 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1855 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
11 ,0 ,0 ,0 ,0 ,0 ,0 ,41 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,

Interaction_D =0 ,0 ,0 ,16 ,12 ,27 ,28 ,0 ,0 ,1 ,1 ,0 ,0 ,19 ,0 ,1 ,0 ,
0 ,0 ,16 ,1214 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 ,0 ,0 ,0 ,2 ,1 ,0 ,0 ,4 ,0 ,
0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,24 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,40 ,20 ,0 ,0 ,0 ,0 ,2 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,21 ,1 ,16 ,8597 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
6 ,0 ,0 ,0 ,0 ,27 ,15 ,0 ,0 ,0 ,0 ,0 ,29712 ,0 ,0 ,0 ,0 ,0 ,1549 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,11 ,0 ,0 ,0 ,0 ,0 ,0 ,40 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,

Interaction_E =0 ,0 ,0 ,48 ,73 ,67 ,139 ,0 ,0 ,8 ,8 ,0 ,0 ,56 ,0 ,8 ,
0 ,0 ,0 ,38 ,2964 ,0 ,0 ,0 ,8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,28 ,0 ,0 ,0 ,6 ,8 ,0 ,0 ,
45 ,0 ,0 ,0 ,78 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,48 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,24 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,210 ,151 ,0 ,0 ,
0 ,0 ,3 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,93 ,8 ,37 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,21 ,0 ,0 ,0 ,0 ,108 ,501 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,2328 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,65 ,0 ,0 ,0 ,0 ,0 ,0 ,210 ,0 ,0 ,0 ,0 ,0 ,2 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,

Interaction_F =0 ,0 ,0 ,22 ,13 ,29 ,30 ,0 ,0 ,1 ,1 ,0 ,0 ,32 ,0 ,1 ,0 ,
0 ,0 ,22 ,2512 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,14 ,0 ,0 ,0 ,4 ,1 ,0 ,0 ,4 ,
0 ,0 ,0 ,12 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,48 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,44 ,22 ,0 ,0 ,0 ,0 ,2 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,27 ,1 ,22 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,
0 ,0 ,0 ,0 ,28 ,15 ,0 ,0 ,0 ,0 ,0 ,48565 ,0 ,0 ,0 ,0 ,0 ,2328 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,12 ,0 ,0 ,0 ,0 ,0 ,0 ,44 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,
```

Table 4 shows the distance matrix between each interaction of Steamy Window applications with Euclidean distance as similarity metric:

Interaction	A	B	C	D	E	F
A	0	0.1818	0.1414	0.1414	0.1818	0.1414
B	0.1818	0	0.1768	0.1768	0.1616	0.1667
C	0.1414	0.1768	0	0.1010	0.1818	0.1212
D	0.1414	0.1768	0.1010	0	0.1818	0.1212
E	0.1818	0.1616	0.1818	0.1818	0	0.1717
F	0.1414	0.1667	0.1212	0.1212	0.1717	0

Table 4: Steamy Window system call vectors distance matrix table

Distances close to 0, are identical or similar vectors. Distances with a value far from 0, are non-similar vectors. The distance matrix shows that values for interactions B and E compared to the benign application interactions are higher than others.

Finally, k-means algorithm has been used to cluster the interactions. Results are shown in table 5. First row shows the cluster number that each interaction is given as a result of applying k-means. This row contains the number of the cluster to which the data belongs. Second row shows which of the interactions were benign (✓) and malware (✗). The system is able to correctly identify the two malicious applications, B and E, which is an indication that the behavior-based Android malware detection system is able to detect malicious executions of the Steamy Window application.

Interaction	A	B	C	D	E	F
Cluster	1	2	1	1	2	1
Application	✓	✗	✓	✓	✗	✓

Table 5: Steamy window clustering result

Another way of representing the system call vectors obtained, is using bar graphs as shown in figure 5. Blue colored bars, represent the behavior of Steamy Window benign application executions, and the red colored bars, represent the Steamy Window behavior infected with PJApps trojan. Every system call has its own number, and the X axis represents the designated number for the executed system call. On the other hand, Y axis represents the number of times that such system call has been executed. We have removed from the graphs those system calls with very high invocation ratio in all of the interactions including malware (*ioctls*,

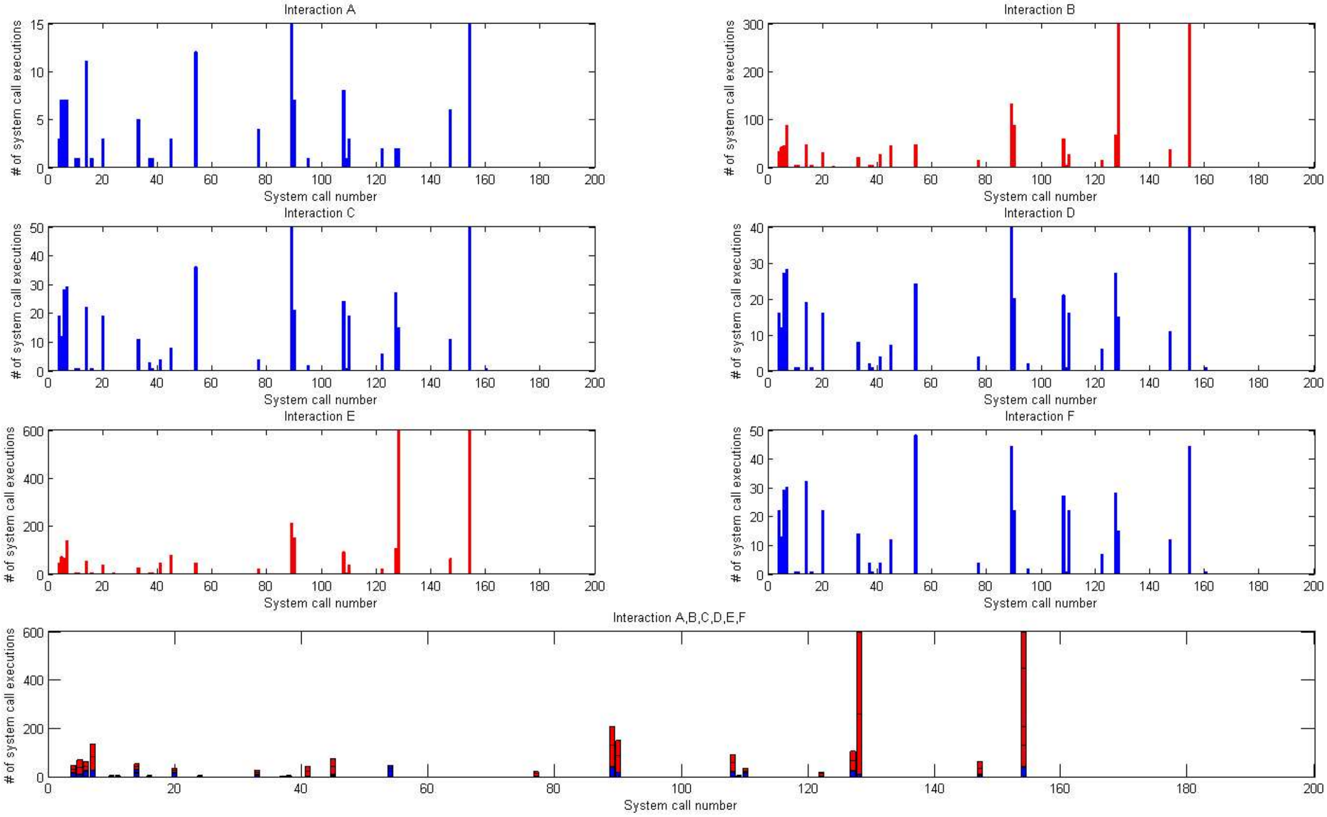

Figure 5: Steamy Window Application Interactions bar plot

time, *recv* and *ptrace*) in order to focus on relevant system calls. Considering that blue bars represent the normal behavior of Steamy Window application, we can see that the trojanized version is executing more and additional system calls in the device. Taking into account that both applications have the same version, we can assume that the Steamy Window application downloaded from unofficial repositories, executed in interactions B and E, are anomalous Android applications. The last graph shows a mixture of the rest of the bars. There we can easily identify which system calls are responsible for such a behavior. Specifically, system calls *read(), open(), access(), chmod()* and *chown()* have been the most relevant. The original application also uses the first two calls, but with a lower frequency. *access(), chmod()* and *chown()* are invoked by the malware, allowing to access and change permissions and ownerships of a set of files and directories.

4.2.2 Monkey Jump 2 application with HongTouTou Malware

Second experiment with real malware is done using a game called Monkey Jump 2. Even if this application is free and can be installed via Android Market, HongTouTou is included in repackaged apps made available through a variety of alternative app markets and forums targeting Chinese-speaking users. When Monkey Jump 2 infected with HongTouTou is executed, it sends device IMEI and IMSI data to a remote host. Then it receives instructions to click on web search result sites depending on received keywords. It also has the ability to download an application with the ability to monitor SMS conversations, and insert spam contents on them.

Using the Crowdroid framework we have obtained 20 feature vectors of both benign and malicious applications, corresponding to 15 interactions with the benign Monkey Jump 2, and the rest to the same application and version containing the HongTouTou malware. In this case we found three false positives, as k-means algorithm classified three benign interactions as malicious. The five malicious applications were correctly classified though. Table 6 summarises the clustering results for both Steamy Window and Monkey Jump 2. Detection rate is the proportion of correctly identified interaction among all application interactions.

App.	**Interactions**		**Clustering result**		**Detection rate**
	Good App	Malware App	Good Clustered	Malware Clustered	
Steamy Window	6	2	6	2	100%
Monkey Jump 2	15	5	12	8	85%

Table 6: Result for Monkey Jump 2 Application Behavior Clustering

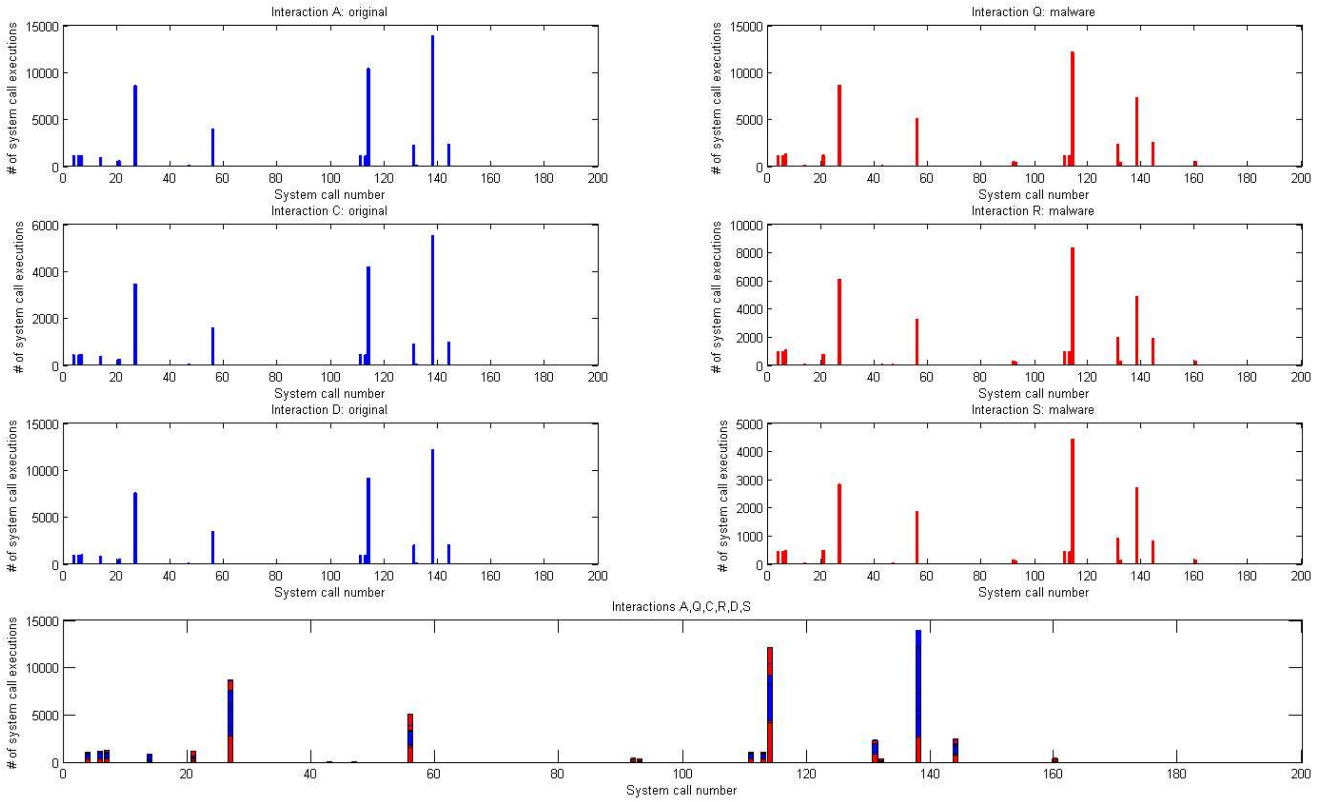

Figure 6: Monkey Jump 2 Application Interactions bar plot

The reason for obtaining a worse detection rate on Monkey Jump 2 is the simpler nature of the actions performed by the malware. As we have explained, HongTouTou sends information and browses the Internet, while PJApps starts a background application and has more functions programmed. Bar plots for Monkey Jump 2 are shown in figure 6. There, we have plotted 6 interactions, 3 of them are benign and correctly clustered interactions (blue bars), while the rest corresponds to malware. The different patterns can be clearly distinguished. In comparison with figure 5 for the Steamy Windows application, Monkey Jump 2 makes use of less system calls, which makes it more difficult to obtain a normal behavior model, and thus false positives are more likely. As with the previous malware, the most relevant system calls executed are *read(), open(), chmod()* and *chown()*. In this case files are not accessed, but permissions and ownerships of files and directories are changed.

5. CONCLUSIONS AND FURTHER WORK

All market indicators foresee a massive increase in the number of smartphones purchased in the next 5 years. This will create a potential for a massive increase in malware generation, and in particular in the sector dominated by the market leader, potentially the Android platform.

In this paper we have proposed a new framework to obtain and analyze smartphone application activity. In collaboration with the Android users community, it will be capable of distinguishing between benign and malicious applications of the same name and version, detecting anomalous behavior of known applications. Furthermore, by deploying our platform on a number of test smartphones, we have created a proof of concept for this mechanism, as a means of analyzing emerging threats.

We have indicated that monitoring system calls is a feasible way for detecting malware. This analysis technique has been widely used in the literature [18]. According to the brief survey in section 2, we have seen that there're many different approaches to detect malware. We considered that monitoring system calls is one of the most accurate techniques to determine the behavior of Android applications, since they provide detailed low level information. We do realize that API call analysis, information flow tracking or network monitoring techniques can contribute to a deeper analysis of the malware, providing more useful information about malware behavior and more accurate results. On the other hand, more monitoring capability will place a higher demand on the amount of resources consumed in the device.

We have seen that *open(), read(), access(), chmod()* and *chown()* are the most used system calls by malware. A benign application could make moderate or heavy use of those system calls and thus trigger false positives, but authors trust that slight differences would make the system classify trojans correctly. We have seen that trojanized applications made more system call executions and invoke different system calls to the Kernel.

The most important contribution of this work is the mechanism we propose for obtaining real traces of application behavior. We have seen in previous works that it is possible to obtain behavior information using artificially created user actions, or creating replicas of smartphones, but crowdsourcing helps the community to obtain real application traces of hundreds or even thousands of applications.

Next step is to deploy the Crowdroid lightweight client on Google's Market and distribute it to as many users as possible. Users running our application will have the opportunity to see their own smartphone behavior. We could even alert the users when one of their applications shows an abnormal trace. The system can also act as an early warning system, being capable of detecting malicious or abnormally behaving applications in early stages of propagation.

By implementing a set of tools we have demonstrated that one can obtain behavior-based information and get it processed and clustered on a central server. Clustering results have been flawless for self written malware, and promising in real malware. Whether the performance of a central server would suffice for a large scale deployment is an interesting topic for further study. A configuration with multiple cooperating servers each with lower load and faster response would be a direction to explore.

We have chosen a simple 2-means clustering algorithm to distinguish between benign applications and their correspondent malware version. The results have been encouraging, although we need to address some open issues. First, the system would always separate the system call data vectors in two clusters even if there is no malware on it. The cluster mapping would change drastically whenever a malicious execution vector enters into the dataset. These issue requires some manual check or further automatic analysis. Second, one could intentionally submit incorrect data to the system leaving the dataset corrupt. One next step is to authenticate the submitting application so we can ensure that nobody is directly sending wrong data to the system. Regarding the communication mechanism between the Crowdroid client and our server, it is made using the FTP protocol in this first version, without focusing on protecting the privacy of transferred data. If an attacker sniffs and manipulates the traffic in the communication process, it can lead to misclassification errors. In order to avoid this, we are introducing encryption mechanisms to provide integrity of data and authenticity of the sender. We have to take into account that applying this technique in the mobile device, it might have an extra overhead in the processor, resulting in a fast battery drain.

Finally, we have the challenge of convincing the Android user community to install the Crowdroid application. We need to manage the perception of loss of privacy when supporting research community with their behavior information, against the benefit of having access to up-to-date behavioral-based detected malware statistics.

6. ACKNOWLEDGEMENTS

This work is partially supported by the Embedded Systems Group supported by the Department of Education, Universities and Research of the Basque Government.

7. REFERENCES

[1] 50 Malware applications found on Android Official Market. `http://m.guardian.co.uk/technology/blog/2011/mar/02/android-market-apps-malware?cat=technology&type=article`.

[2] Guangdong Bai, Liang Gu, Tao Feng, Yao Guo, and Xiangqun Chen. Context-aware usage control for android. In *Security and Privacy in Communication Networks*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 326–343. Springer Berlin Heidelberg, 2010.

[3] Thomas Bläsing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A. Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *5th International Conference on Malicious and Unwanted Software (Malware 2010) (MALWARE'2010)*, Nancy, France, France, 2010.

[4] Mylookout blog. Hongtoutou. `http://bit.ly/iOu5AA`.

[5] Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. Behavioral detection of malware on mobile handsets. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 225–238, New York, NY, USA, 2008. ACM.

[6] Timothy K. Buennemeyer, Theresa M. Nelson, Lee M. Clagett, John P. Dunning, Randy C. Marchany, and Joseph G. Tront. Mobile device profiling and intrusion detection using smart batteries. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, pages 296–, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Cabir, Smartphone Malware. `http://www.f-secure.com/v-descs/cabir.shtml`.

[8] Cabir Malware variants. `http://www.f-secure.com/weblog/archives/00000414.html`.

[9] Jerry Cheng, Starsky H.Y. Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 258–271, New York, NY, USA, 2007. ACM.

[10] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, pages 12–12, Berkeley, CA, USA, 2003. USENIX Association.

[11] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 5–14, New York, NY, USA, 2007. ACM.

[12] David Dagon, Tom Martin, and Thad Starner. Mobile phones as computing devices: The viruses are coming! *IEEE Pervasive Computing*, 3:11–15, October 2004.

[13] Francesco Di Cerbo, Andrea Girardello, Florian Michahelles, and Svetlana Voronkova. Detection of malicious applications on android os. In *Proceedings of the 4th international conference on Computational forensics*, IWCF'10, pages 138–149, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] Manuel Egele. A survey on automated dynamic malware analysis techniques and tools. *ACM Computing Surveys*, to appear.

[15] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[16] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX Security Symposium*. USENIX Association, August 2011.

[17] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE Security and Privacy*, 7:50–57, January 2009.

[18] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *J. Comput. Secur.*, 6:151–180, August 1998.

[19] Google Inc. Android market. `https://market.android.com/`.

[20] Juniper Networks Inc. Malicious mobile threats report 2010/2011. Technical report, Juniper Networks, Inc., 2011.

[21] T. J Lee and J.J. Mody. Behavioral classification. In *Proceedings of EICAR 2006*, April 2006.

[22] G A Jacoby and Nathaniel J Davis Iv. Battery-based intrusion detection. In *Global Telecommunications Conference, 2004. GLOBECOM '04*, pages 2250 – 2255. IEEE, 2004.

[23] Hahnsang Kim, Joshua Smith, and Kang 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, pages 239–252, New York, NY, USA, 2008. ACM.

[24] Ramon T. Llamas, William Stofega, Stephen D. Drake, and Stacy K. Crook. Worldwide smartphone 2011–2015 forecast and analysis. Technical report, International Data Corporation, 2011.

[25] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[26] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference, ACSAC'07*, pages 421–430, Los Alamitos, CA, USA, December 2007. IEEE Computer Society.

[27] Jon Oberheide and Zach Lanier. Team joch vs android: The ultimate showdown. ShmooCon 2011, January 2011.

[28] Machigar Ongtang, Stephen McLaughlin, William Enck, and Patrick McDaniel. Semantically rich application-centric security in android. In *Proceedings of the 25th Annual Computer Security Applications Conference, ACSAC'09*, pages 340–349, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[29] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC'10*, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM.

[30] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag.

[31] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz, Kamer Y§ksel, Seyit Camtepe, and Albayrak Sahin. Static analysis of executables for collaborative malware detection on android. In *ICC 2009 Communication and Information Systems Security Symposium*, Dresden, Germany, Germany, 6 2009.

[32] Aubrey-Derrick Schmidt, Ahmet Camtepe, and Sahin Albayrak. Static smartphone malware detection. In *proceedings of the 5th Security Research Conference (Future Security 2010), ISBN: 978-3-8396-0159-4*, page 146, 2010.

[33] Aubrey-Derrick Schmidt, Jan Hendrik Clausen, Ahmet Camtepe, and Sahin Albayrak. Detecting symbian os malware through static function call analysis. In *4th International Conference on Malicious and Unwanted Software (MALWARE'09)*, pages 15–22. IEEE, 2009.

[34] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, Christian Scheel, Seyit Ahmet Camtepe, and Sahin Albayrak. Monitoring smartphones for anomaly detection. *Mobile Networks and Applications (MONET) – SPECIAL ISSUE on Mobility of Systems, Users, Data and Computing*, November 2008.

[35] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Jan Clausen, Kamer Ali Yüksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak. Enhancing security of linux-based android devices. In *in Proceedings of 15th International Linux Kongress*. Lehmann, October 2008.

[36] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, and Chanan Glezer. Google android: A comprehensive security assessment. *IEEE Security and Privacy*, 8:35–44, 2010.

[37] Asaf Shabtai, Uri Kanonov, and Yuval Elovici. Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method. *J. Syst. Softw.*, 83:1524–1537, August 2010.

[38] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan

Glezer, and Yael Weiss. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, pages 1–30, 2011. 10.1007/s10844-010-0148-x.

[39] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Inf. Secur. Tech. Rep.*, 14:16–29, February 2009.

[40] Ashkan Sharifi Shamili, Christian Bauckhage, and Tansu Alpcan. Malware detection on mobile devices using distributed machine learning. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ICPR '10, pages 4348–4351, Washington, DC, USA, 2010. IEEE Computer Society.

[41] Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. Towards formal analysis of the permission-based security model for android. In *Proceedings of the 2009 Fifth International Conference on Wireless and Mobile Communications*, ICWMC '09, pages 87–92, Washington, DC, USA, 2009. IEEE Computer Society.

[42] Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A formal model to analyze the permission authorization and enforcement in the android framework. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, pages 944–951, Washington, DC, USA, 2010. IEEE Computer Society.

[43] Wook Shin, Sanghoon Kwak, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A small but non-negligible flaw in the android permission scheme. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '10, pages 107–110, Washington, DC, USA, 2010. IEEE Computer Society.

[44] Hispasec Sistemas. Virustotal malware intelligence service.
`http://bit.ly/mytpXt`.

[45] Symantec. Pjapps.
`http://bit.ly/juL7Rh`.