A cloud-based architecture to crowdsource mobile app privacy leaks

Dimitrios Papamartzivanos
University of the Aegean
Karlovasi, Samos, Greece
dpapamartz@aegean.gr

Dimitrios Damopoulos
Stevens Institute of Technology
Hoboken, New Jersey
ddamopou@stevens.edu

Georgios Kambourakis
University of the Aegean
Karlovasi, Samos, Greece
gkamb@aegean.gr

ABSTRACT

Most would agree that modern app-markets have been flooded with applications that not only threaten the security of the OS superficially, but also in their majority, trample on user's privacy through the exposure of sensitive information not necessarily needed for their operation. In this context, the current work revolves around 3 key questions: Is there a way for the end-user to easily track - the many times - hidden privacy leaks occurring due to the way mobile apps operate? Can crowdsourcing provide the end-user with a quantitative assessment per app in terms of privacy exposure level? And if yes, in which way a cloud-based crowdsourcing mechanism can detect and alert for changes in the apps' behavior? Motivated by the aforementioned questions, we design a cloud-based system that operates under a crowdsourcing logic, with the aim to provide i) a real-time privacy-flow tracking service, ii) a collaborative infrastructure for exchanging information related to apps' privacy exposure level, and iii) potentially a behavior-driven detection mechanism in an effort to take advantage of the crowdsourcing data to its maximum efficasy.

Categories and Subject Descriptors

[**Security and privacy**]: Malware and its mitigation, Mobile platform security, Mobile and wireless security; [**Computer systems organization**]: Cloud computing

Keywords

Mobile Security and Privacy; Crowdsourcing; IDS; Behavior-driven Detection; Cloud Computing.

1. INTRODUCTION

Today, more and more users rely on smartphones and other ultra-mobile devices to meet their everyday needs. As a result, the use of such mobile devices to store sensitive personal data, carry out financial transactions, and even socialize with other people becomes significantly more frequent for virtually everyone of us. In fact, mobile devices has become an integral part of people's life, resulting in attracting the attention of ill-motivated entities who intend to exploit the devices and the data stored in it for their own profit. A characteristic example of this situation is that of the flooding of mobile app markets with malicious apps, aiming either to expose user privacy or manipulate services and data stored on the device. It is therefore more than obvious that there is an urgent need for the design of robust detection mechanisms capable of deterring, if not eliminating, malicious actions and thus offer end-users the opportunity to protect their data and preserve their privacy.

According to recent reports [9], Android seems to be the most widely used mobile platform since it currently holds 75% of the total mobile market share. This popularity, along with the certain augment in malicious apps destined to it, has rendered Android quite vulnerable when it comes to user privacy. This is actually fairly prominent as the vast majority of Android apps require from the user an - often extravagant - large number of permissions for its installation. Unfortunately, this allegation is confirmed by recent researches [10] reporting that the 98% of malicious applications in the wild target to Android. The reasons why this happens are quite a few, but the cardinal one is that the market is deluged with 3rd party app repositories which unfortunately do not fall in step with the appropriate security policies. Overall, the decision on whether or not to install a given app is left entirely up to the user.

On the other hand, there are more than a few cases where even approved apps from Google Play act maliciously and do not respect users' privacy. As already pointed out, every app, malicious or not, asks for many privacy-sensitive permissions and in most of the cases the Android market does not perform any privacy-exposure related control. Nevertheless, this negligent attitude motivates the research community to take external measures for safeguarding the privacy of the end-user. This usually pertains to Intrusion Detection Systems (IDS) that is lately on the rise in the mobile ecosystem. However, implementing a robust detection mechanism for such devices is not straightforward. This is mainly to the inherent limitations of these devices, including memory and storage constraints, limited battery capacity, etc. For this reason, early approaches on this topic invest on cloud's "unlimited" processing capabilities in order to deal with the aforementioned impediments. Yet, as discussed further down in section 3, most of them aim to provide detection mechanisms under sophisticated architectures, not concentrating on protecting from privacy leaks.

In this context, this paper details on a host and cloud-based synergistic mechanism for preserving mobile users' privacy. Specifically, our approach comprises an easily implementable architecture for sharing apps' behavioral profile among the participating users and thus offer them the ability to control privacy exposures. Also, gives them the ability to immediately get informed about any sus-

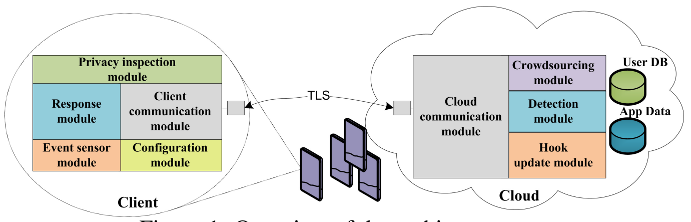


Figure 1: Overview of the architecture

picious app activity and block it if needed. The highlights of our solution are:

- The formation and diffusion of a common knowledge about virtually any mobile app. This knowledge is dynamically and constantly updated to not only reflect the attitude of the app during its interaction with privacy-sensitive assets, but also the user's preferences regarding its operation.
- In continuation to the previous point, the decisions on the user's side on how to handle an app during its installation or operation (per event of interest), is the continually revised product of a collaborative effort fusing the user's preferences, the common knowledge about that app, and the feedback received during its operation. Note that the latter also affects the former.
- The solution at the client side is not self-sufficient, but it relies on the cloud. Nonetheless, it is practical to implement and discharges the smartphone from resource intensive tasks.

The rest of this paper is organized as follows. The architecture of the proposed system is given in the next section. The same section also offers some preliminary evaluations of the proposed solution in terms of CPU and memory utilization. The related work is addressed in section 3. The last section concludes and provides pointers to future work.

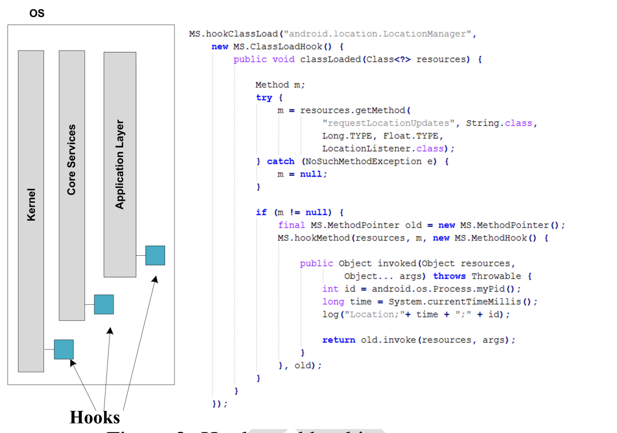
2. ARCHITECTURE

Figure 1 illustrates a high-level overview of the proposed architecture composed of two main components; the client and cloud. Overall, three main services are provided to the participating users for enabling them to be in charge and keep control of their privacy; privacy-flow tracking, crowdsourcing, and detection and reaction against privacy violations. These services are conceived to get materialized by the synergistic operation of both the components. The client app has been implemented in Android ver. 4.1.2, while the cloud infrastructure has been deployed on the Greek Research and Academic Community cloud service Okeanos.

2.1 The Client

This section details on the client app running on the mobile device. This app is responsible for tracking and recording privacy-sensitive events, inform users about privacy leaks, and exchange information with the cloud infrastructure. To do so, the client comprises of five collaborating modules discussed in the following.

Event sensor module: The starting point of every monitoring service is the event sensing mechanism. In our case, to record a privacy-sensitive event, the monitoring service has to be notified whenever an app attempts to access any resource associated with users' privacy. Naturally, these privacy-sensitive assets need to be decided beforehand. Here, we follow an approach based on method invocation and object creation hooking. It is stressed that



```
MS.hookClassLoad("android.location.LocationManager",
    new MS.ClassLoadHook() {
        public void classLoaded(Class<?> resources) {

            Method m;
            try {
                m = resources.getMethod(
                        "requestLocationUpdates", String.class,
                        Long.TYPE, Float.TYPE,
                        LocationListener.class);
            } catch (NoSuchMethodException e) {
                m = null;
            }

            if (m != null) {
                final MS.MethodPointer old = new MS.MethodPointer();
                MS.hookMethod(resources, m, new MS.MethodHook() {

                    public Object invoked(Object resources,
                            Object... args) throws Throwable {
                        int id = android.os.Process.myPid();
                        long time = System.currentTimeMillis();
                        log("Location;"+ time + ";" + id);

                        return old.invoke(resources, args);
                    }
                }, old);
            }
        }
    });
```

Figure 2: Hooks and hooking process

the hooking of OS operation aims solely to have one get notified about privacy intrusive events and not to modify the app behavior per se. In an effort to collect more accurate information regarding the exact way each app behaves, our approach takes into account app layer calls along with kernel and core service ones. This situation is characteristically shown in figure 2 providing also an example of a typical hook. For more information about the hooking process the interested reader can refer to [5].

But how is this event sensing mechanism configured? Naturally, this has to do with one's perspective on what she considers to lead in privacy leaks. In this direction, we have to track down certain API calls and parameterize the event sensor to listen to their invocation event. Therefore, the event sensor wakes up only when a sensitive invocation happens, making it more effective for devices with limited processing power and battery capacity. To implement the aforementioned event sensing mechanism and get notified when an app attempts to access critical resources, we capitalize on the Cydia substrate framework [5]. Table 1 contains a list of all the methods currently hooked by our scheme. As observed, all these hooks represent important operations invoked by virtually every app, and specifically those that matter most in terms of privacy. Putting it another way, these methods seem to be a point of reference not only for spying and malicious apps, but also for a plethora of benign ones.

All the sensors store an Event Vector (EV) in the local database. This vector has the following structure:

EV={appID, hookID, hookedMethodID, hookMode, rFlag, timestamp}

where appID is the unique identifier of the app, hookID is the ID of the accessed resource, hookedMethodID is the ID of the method used to access the resource, hookMode is the configured access mode as activated by the user (described in the next module), rFlag designates the user's response to this event, and timestamp is the exact time of the generation of the event.

Configuration module: This module enables the users to parameterize the execution environment of any app, thus providing them a straightforward way of blocking privacy intrusive actions. In this direction, the current module offers a GUI where users can personalize their resource accessing preferences per application. Also, the module provides 3 possible modes of operation for every available hook.

- Mode 1: Always Allowed (default permit)
- Mode 2: Ask me first
- Mode 3: Always Blocked (default deny)

Table 1: Hooks list related to user's privacy

Hooks	Class	Method	Tag
Contacts	ContentResolver	Query(...)	contacts
SMS/MMS/Photo Album	ContentResolver	Query(...)	sms/mms
Photo Album	ContentResolver	Query(...)	images
Sending SMS	SMSManager	sendTextMessages(...)	sendSMS
	SMSManager	sendDataMessages(...)	sendSMS
	SMSManager	sendMultipartTextMessages(...)	sendSMS
Location	LocationManager	requestSingleUpdate(...)	location
	LocationManager	requestLocationUpdates(...)	location
	Location	getLatitude()	location
	Location	getLongitude()	location
Subscriber ID	TelephonyManager	getSubscriberID()	subscriberID
Location	TelephonyManager	getCellLocation()	cellLocation
SIM Serial Number	TelephonyManager	getSimSerialNumber()	simSerialNum
Device ID	TelephonyManager	getDeviceID()	deviceID
Camera	Camera	Open(...)	Camera
	Camera	takePicture(...)	Camera
Command Execution	Runtime	Exec()	commandExec
Audio Recording	AudioRecord	startRecording()	audioRecording
Accounts	AccountManager	getAuthToken()	accounts
	AccountManager	getAccounts()	accounts

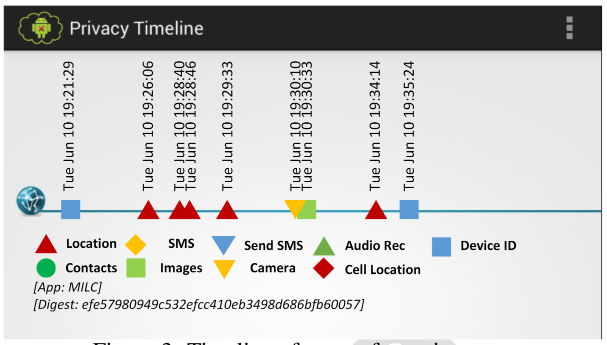
When the first mode is selected, the module is not involved in the app's execution process (rFlag=0). Opposite to that, when the third mode is activated, the hooked process is always blocked (rFlag=1). The second mode displays a prompt to the users allowing them to decide about the execution of the process at hand (rFlag=0/1). It has to be mentioned that no matter what is the hook mode, the corresponding method invocation is logged in order to store the actual behavior of the app for later reference and transmission to the cloud.

The users' preferences about the above mentioned modes are based on decisions taken by either inspecting the locally-generated knowledge pertaining to the behavior of a given app and the crowdsourced one as stored in the cloud. So, every time an app is installed on the device, this module prompts the user to accept the recommended hook parameterization as it is generated by the cloud. If agreeing to this recommendation the hook modes are configured automatically. Naturally, users can revise their preferences for a given app at any time by taking under consideration the behavior profile provided by the privacy inspection module as described in the following.

Privacy inspection module: This module undertakes the semantic interpretation of system calls triggered by apps to human understandable form. In this respect, the goal of this module is to provide a privacy flow tracking service able to inform users about their privacy exposure level. Putting it another way, this service is responsible for quantifying and visualizing events happening on privacy-sensitive assets. To do so, it provides a GUI where the behavior profile of an app can be displayed in timeline format aiming to inform users about its attitude. An example of such a GUI screen is given in figure 3. The timeline is generated by reading and processing the events list logged for that app by the event sensor module. The module enables the user to select among three presentation modes; daily, weekly, and monthly.

Client communication module: This module always connects to the corresponding one residing on the cloud side. So, in the uplink this module transmits Enhanced EVs (EEVs), i.e., the original EVs logged by the client plus some additional data used to uniquely identify an event per client. That is, the original EV is populated with the client's device identifier (deviceID), the name of the app that triggered the event (appName), and an SHA-1 hash of the the app's executable (apkDigest). Note that these pieces of data are not logged for every event by the sensor module for reasons of optimization.

EEV= {deviceID, appID, appName, apkDigest, hookID, hookedMethodID, hookMode, rFlag, timestamp}



[App: MILC]
[Digest: efe57980949c532efcc410eb3498d686bfb60057]

Figure 3: Timeline of events for a given app

On the downlink, this module receives the crowdsourced knowledge as generated by the cloud. This common knowledge is used to inform the receiver about the actual behavior of the app as perceived by the others and for providing recommendation about configuring preferences per app. Ideally, the client should be always connected to the cloud for sharing information corresponding to apps, but also for retrieving behavior statistics contributed by other participants in realtime. However, in most of the cases, this is not a realistic scenario. Therefore, this module only wakes up when the device gets online.

Response module: This module is fed from the privacy settings configured per app per hook, and provides counteraction to each occurring event depending on the mode of operation. Specifically, based on the privacy preferences provided by the user with the help of the configuration module, this one acts upon any privacy sensitive method invocation. Also, in case of a "Mode 2" setting, this module constantly tracks the user response for that app, hook. If the user mostly selects to block the corresponding event, this module intervenes and suggests the user to automatically change that mode to the strictest one. The opposite situation happens when the majority of user's responses in a "Mode 2" setting is "allow". Simply put, the current module monitors the user's responses and through them tries to calibrate the privacy settings per app, per hook.

2.2 The Cloud

The cloud offers a central point where aggregated knowledge (statistics) for every smartphone app can be generated taking as input the behavioral data submitted by all the participating clients. Specifically, it is comprised of the following modules used to correspondingly display, analyze and disseminate the cumulative knowledge to the community.

Cloud communication module: It is responsible for enabling communication with the participating clients. On the one hand, it receives data related to the behavior of the various apps running on the clients, and on the other, disseminates collective data about the apps. A downlink communication may happen due to the request of a client or independently. For example, in the first of the aforementioned cases, the client needs to install a new app and requests collective behavior data about it, while in the other the cloud multicasts an update for one or several apps to a group of clients.

Crowdsourcing module: It receives information pertaining to the behavior of the apps running in every client. Bear in mind that this information reflects the privacy exposure level of any given app. More specifically, the information submitted to this module include app EVs collected by the Event sensor module as detailed in section 2.1, and user's privacy preferences related to the level of access she allows (tolerate) each app to have. The latter is collected and contributed by the Configuration module on the client side. Based on the collected data, this module makes an overall assessment of the level of privacy for every app running on the clients. From that, it is also possible to provide privacy configuration suggestions upon request from a client (e.g., when installing a new app), thus helping the users to take optimal decisions regarding their privacy.

However, for the module to be able to estimate the level of trust that can be assigned to an app, it needs to perform some basic calculations taking as input the EEVs sent per app by the population of users. Recall that the structure of such an EEV is discussed in detailed in section 2.1. Also, as already pointed out, every hook can have one of the values 1, 2, or 3 corresponding to the three modes of operation imposed by the Configuration module. Hence, for N hooks pertaining to an app the maximum level of trust could be succeeded if all of them are set at the first operation mode, thus allowing always the event to occur. However, each user can set a different value for every hook leading to different levels of trust per app among the users. In this respect, the current module is responsible for calculating the mean value of the level of trust for every app as this has been measured from the users' preferences. This metric is provided in percentage form.

Nevertheless, this metric alone is not enough to offer an objective estimation of an app's level of trust. Different users may have varying preferences and opinions about the nature and the behavior of an app or even a different perspective of what action is considered to be privacy-offensive. In this regard, the computation of the standard deviation metric over the mean trust values contributed by all users is needed. The calculated standard deviation value is displayed in one of three colors: green, orange, and red. This way it is easier for the end-user to perceive the degree of concordance between the contributors about that app. The aforementioned knowledge is delivered to every client (e.g.,when installing an app) in an effort to assist the receiver in making the safest decisions.

To exemplify the above process, let us assume that the framework has currently 10 hooks in use. So, the maximum level of negative trust for an app is 10*3. Also, suppose that one of the users has configured the hooks in the following modes: 3 to Mode 3, 5 to Mode 2, and the rest to Mode1. As a result, the local level of negative trust for the same app is 3*3+5*2+2=21, which corresponds to a percentage of 70% (also meaning that the user's confidence in this app is 30%). Recall that the module computes the mean value of trust submitted by all the participants. Hence, if this app has been rated by 5 users with percentages 30, 80, 90, 95, and 85% correspondingly, then the average level of trust for it is 76%. On the other hand, the st. deviation is approximately 26.3, thus classifying the app as "orange". The exact values for transitioning from the one category (color) to the other can be a matter of debate, but naturally depend on the system and situation at hand.

Detection module: The crowdsourcing of app execution traces described previously contributes toward the design of a behavior-based detection mechanism for exposing suspicious and/or sudden changes in the behavior of an app. Under this prism, the current module is destined to deliver such a service by detecting new undocumented privacy threats and zero-day attacks when an app gets updated or a new feature is added. We envisage this module to use machine learning to make decisions about an app, however its implementation is left for future work. In fact, recent research [3] has shown that the use of popular classifiers in behavior-based detection can deliver quite accurate results when it comes to the identification of the real nature of an app.

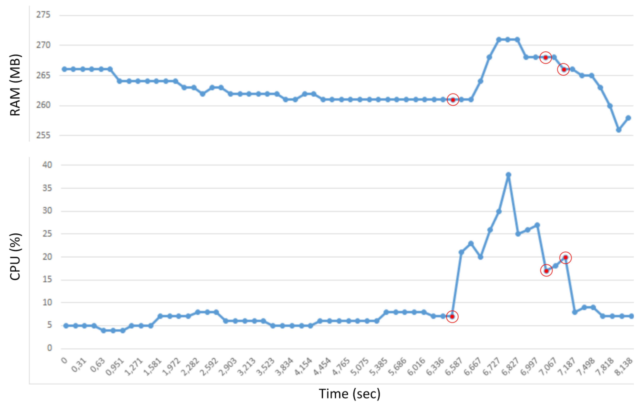


Figure 4: RAM and CPU usage

Hook update module: As discussed in section 2.1, the efficiency of the proposed architecture is directly associated with the selected app hooks on the client side. Therefore, a module that allows the update of the hooks list in the client is deemed necessary. This functionality is also directly associated with the open nature of the Google's Android platform which allows smartphone manufactures to design their own custom APIs to meet their specific needs. This calls for a hook update module enabling custom hooks depending on the situation and the device at hand. Overall, the current module is responsible for adapting the hooks to device's custom OS API after, say, an OS update.

2.3 Hooking performance evaluation

In this section we present some preliminary performance results for our mechanism. As already pointed out, our goal is to design a lightweight client app capable of performing efficiently on even low-end mobile devices. So, its operation solely relies on the detection and logging of privacy sensitive events. By doing so, its performance is directly associated with the Cydia Substrate tool which is used to detect any method call pertaining to a privacy sensitive resource. Our tests have been conducted using a Sony Xperia L device which incorporates a dual core 1 GHz CPU and 1 GB of RAM.

Some pilot results on CPU and memory utilization for our app is presented in figure 4. As observed from the figure, we monitored the application for about 8 secs. From time moment 0 to 6.587 the app was idle, consuming about 7% of the CPU for maintaining its services running in the background. During the same period of time, the total free memory on the device was 264 MB. From time moment 6.587 to 7.067 the app detects a privacy-sensitive event and takes action to handle it. The maximum CPU consumption during this time interval is 38% while the memory increased by 10 MB. From time moment 7.067 to 7.187 the application logs the detected event in the database. In this interval the CPU consumption was about 17%-20% while the memory usage was being gradually decreased. Considering the above observations one can argue that the application performs efficiently. Overall, we can say that although there exists a noticeable augment in CPU and memory usage during the detection and logging of events, this happens only for a tiny period of time.

3. RELATED WORK

So far, a considerable mass of works in the literature have been devoted to the preservation of user's privacy and/or the detection of malware in the smartphone ecosystem. Therefore, in a high-level, the contributed solutions can be classified into those aiming at detecting malware, and ones focusing on preserving user's privacy. Other classification is also possible, for example based on the behavior of the app when interacting with either low-level or high-level system assets, but in the following we opt to select the former.

3.1 Malware detection

The work in [12] presents one of the first network-based intrusion detection services. According to the authors, their scheme is able to provide enhanced protection capabilities to mobile devices, while achieving reduced complexity and resource consumption at the same time. Precisely, the authors propose an off-device network service employing multiple virtualized antivirus engines to protect the devices from new undocumented malware. In the same context, the authors in [13] propose an alternative solution for malware detection, where security checks are applied on remote security servers. The latter entities are used to host an exact replica of each smartphone participating in the infrastructure. The servers are able to apply multiple detection techniques simultaneously, supporting more than a hundred replicas running on a single server. So, rather than running the security measures locally, the smartphone records a minimal execution trace, and transmits it to the corresponding security server, which is in charge of replaying the original execution.

The collection of Linux kernel system calls is the cornerstone of the proposal given in [12]. Actually, the system described uses crowdsourcing to obtain traces of app's behavior and distinguish between benign ones and those containing malware. This is done by utilizing anomaly detection techniques. More specifically, the authors choose the k-means algorithm with a known number of k=2 clusters to distinguish between legitimate and privacy-invasive apps. A cloud-based intrusion detection and recovery system for smartphones devices is given in [8]. According to the authors, their system provides continuous in-depth forensics analysis to detect any misbehavior in the network. The proposed scheme performs investigations on emulated smartphone devices residing in the cloud with the aim to take the optimal counteractions. Another interesting work is presented in [2]. The authors proposed a crowdsourcing architecture towards a behavior-based malware detection system for Android based on system call traces. Contrary to our approach which offers a privacy preservation scheme, this one aims at providing a detection mechanism.

The work in [1] advocates the idea of C2C, a peer-to-peer network of smartphone clones running in the cloud. According to the authors, this architecture can be used to prevent worm attacks spreading from one device to another. The authors try to apply a healing strategy by providing a software patch to a relatively low number of smartphones in order to reduce the probability of worm spreading. The authors in [15] proposed a permission-based scheme to detect new samples of already known Android malware and applied a heuristics-based technique to reveal new ones. Their framework, called *DroidRanger*, was able to unveil 211 malicious apps among 204,040 ones collected from different Android markets. The authors argue that generally the marketplaces are functional and relatively healthy, but there is a need for stricter security policies. Another proposal focusing on malware detection for mobile devices is given in [14]. Specifically, the authors describe and evaluate a detection framework destined to mobile malware in the Android platform. This solution is capable of analysing immutable information ranging from app permissions to API calls in an effort to portray apps' behavior. To evaluate their framework, the authors employed several machine learning classifiers aiming to identify the detection mechanism that profiles malicious apps more efficiently.

The authors in [17] also detail on a cloud IDS solution for smartphones, namely *Secloud*. The latter is able to emulate any registered smartphone with the help of a virtual machine running in the cloud. That is, *Secloud* mirrors all of the events happening on the device (e.g., user input, communications, etc) to the emulated environment for keeping the emulated version synchronized with the actual device. The process of the events occurs mainly in the cloud in an effort to alleviate the smartphone from computationally intensive tasks. A hybrid host/cloud IDS able to detect malware and privacy invasive software is contributed in [3]. The proposed architecture can support diverse anomaly-based mechanisms to be concurrently applied, either directly on the device, or in the cloud, thus maximizing the synergy between the two endpoints. The prototype presented by the authors was evaluated in terms of CPU load, memory and battery consumption, and timeliness (i.e., the time it takes for the IDS to respond to an attack).

3.2 Detection of privacy leaks

The work in [4] presents *TaintDroid*. This solution aims to provide a way of tracking the personal information accessed by the running apps. To meet its purpose, *TainDroid* merges analysis done in four axes (levels), namely variable, message, method, and file. The authors used *TaintDroid* to study the behavior of 30 randomly selected popular apps and arrive to the conclusion that the two-thirds of them indicated suspicious activity. Also, after experimentation, *TaintDroid* was found to achieve 14% performance overhead. A couple of other interesting works were introduced in [16], [7], aiming to protect the user from privacy violations. The first one contributes a privacy solution called *TISSA* for smartphones in the Android platform. According to the authors, their framework empowers the end-user to control the exact kind of personal information is made accessible to the different apps. They evaluate their scheme by testing apps which are known to leak private information. The second work proposes an architecture called *AppInspector* aiming to analyze apps submitted to a marketplace. The authors correctly point out that this centralized way of app distribution gives the chance for large-scale validation. In this respect, *AppInspector* analyzes apps and reports back potential security and privacy shortcomings.

Another proposal focusing on privacy leaks on Android platform is given in [6]. The authors present a static analysis tool called *AndroidLeaks* for examining potential leaks of personal information in massive scale. They created a mapping between the API calls and Android permissions. During their experiments they examined 24,350 different apps and were able to found 57,299 leaks in 7,414 of them. Moreover, the authors in [11] introduced a privacy model called *Privacy As Expectations*. The cardinal idea behind this model is that users are usually able to tell whether or not a privacy-susceptible resource should be allowed to be accessed by an app. The authors try to crowdsource such users' expectations and provide through it a service that is able to distinguish those that seem to violate the conventional expectation. Lastly, the work in [2] attempts to quantify privacy for mobile users through the definition of appropriate metrics. To do so, the authors try to estimate the sensitivity of each asset in terms of privacy and produce a user understandable rating. Their approach takes into account three categories of personal data, namely Location, Contacts, and Content. Nevertheless, the authors conclude that the proposed privacy metrics are by no means foolproof.

3.3 Discussion

As discussed above, several IDS-s have been proposed in the literature of smartphone security to cope with the detection of suspicious, aggressive or privacy-invasive behavior caused by different kind of apps. Generally, as analysed in this section, this timely problem remains in large part open to exploration. One can argue that this is mainly due to impediments imposed by (a) the great diversity of mobile apps, (b) the plethora of third-party app markets, especially for the Android platform, (c) the open nature of some mobile OS, (d) the various restrictions which originate from the idiosyncrasies of mobile hardware, and (e) the privacy awareness level of mobile users and their perception of security. That is, in relation to the latter parameter, the design of a truly effective mobile IDS needs to take into account the human factor along with the technological aspect. Under these circumstances, in this paper, we offer a solution which tries to optimize the decisions taken by a user on how to manage its privacy settings related to any given app.

That is, our proposal relies on the cloud to crowdsource privacy related information about virtually any app. This has the triple benefit of (a) sharing a common knowledge (the more participants the better the quality of this knowledge), (b) the decisions about an app stem from both the user's personal preferences and the cloud (this relationship is bilateral; the former affects the latter and vice versa), and (c) the device is relieved from resource demanding tasks. This is actually the main difference of the work at hand in relation to all the others in this area. Precisely, it tries to equally distribute the task and responsibility of controlling app privacy to all the community, rather than dealing with specific issues in an isolated manner. On top of that, our solution is flexible enough, practical to implement and shows promising results regarding the utilization of resources on the client device.

4. CONCLUSIONS

Today, with the proliferation of smartphones and related services, the need for end-user privacy is fast becoming a sine qua non. Unfortunately, the mushrooming of legacy and third party app markets has put the user in a constant dilemma on whether or not to install a downloaded app. Also, it creates excess worries about the behavior of the app after installation. This is especially prominent for Android app markets, but it seems to propagate to iOS and others as well. So, few would argue that the need for advanced mechanisms that put the user in control of their privacy is apparent. Motivated by this fact, in this paper, we propose a cloud-based, crowdsourcing-driven privacy monitoring mechanism that relies on the synergistic action of both a local knowledge base kept in the device and the disseminated data stemming from the population of participants. In this way, the proposed scheme is able to detect privacy violations caused by the running apps and alert the user of the device and the community about any misbehaviors. This can contribute towards stopping, say, the propagation of malware, and narrow down privacy violations in a promptly manner.

5. REFERENCES

[1] M. Valerio Barbera, S. Kosta, J. Stefa, P. Hui, and A. Mei. Cloudshield: Efficient anti-malware smartphone patching with a p2p network on the cloud. In *P2P*, 2012.

[2] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2011.

[3] D. Damopoulos, G. Kambourakis, and G. Portokalidis. The best of both worlds: A framework for the synergistic operation of host and cloud anomaly-based ids for smartphones. In *Proc. of EuroSec '14*. ACM, 2014.

[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *proc. of the 9th USENIX conf. on Operating Systems Design and Implementation*, 2010.

[5] J. Freeman. Cydia substrate - hacking and tweaking tool., 2011.

[6] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *proc. of the 5th Int'l conf. on Trust and Trustworthy Computing*. Springer-Verlag, 2012.

[7] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung. Vision: Automated security validation of mobile apps at app markets. In *proc. of the Second Int'l Workshop on Mobile Cloud Computing and Services*. ACM, 2011.

[8] A. Houmansadr, S. A. Zonouz, and R. Berthier. A cloud-based intrusion detection and response system for mobile phones. In *proc. of the 2011 IEEE/IFIP 41st Int'l conf. on Dependable Systems and Networks Workshops*. IEEE CS, 2011.

[9] IDC. Worldwide smartphone os market in 4q12, May 2013.

[10] Kaspersky. Kaspersky security bulletin. overall statistics for 2013.

[11] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *proc. of the 2012 ACM conf. on Ubiquitous Computing*. ACM, 2012.

[12] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. Virtualized in-cloud security services for mobile devices. In *proc. of the First Workshop on Virtualization in Mobile Computing*. ACM, 2008.

[13] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: Versatile protection for smartphones. In *proc. of the 26th Annual Computer Security Applications conf.* ACM, 2010.

[14] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *proc. of the 2012 Seventh Asia Joint conf. on Information Security*. IEEE CS, 2012.

[15] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In *proc. of the 19th Annual Network & Distributed System Security Symposium*, 2012.

[16] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. Taming information-stealing smartphone applications (on android). In *proc. of the 4th Int'l conf. on Trust and Trustworthy Computing*, 2011.

[17] S. A. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. H. Sanders. Secloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers & Security*, 37, 2013.