The Best of Both Worlds. A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones

Dimitrios Damopoulos
Stevens Institute of Technology
Hoboken, NJ, USA
ddamopou@stevens.edu

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

Georgios Portokalidis
Stevens Institute of Technology
Hoboken, NJ, USA
gportoka@stevens.edu

ABSTRACT

Smartphone ownership and usage has seen massive growth in the past years. As a result, their users have attracted unwanted attention from malicious entities and face many security challenges, including malware and privacy issues. This paper concentrates on IDS carefully designed to cater to the security needs of modern mobile platforms. Two main research issues are tackled: (a) the definition of an architecture which can be used towards implementing and deploying such a system in a dual-mode (host/cloud) manner and irrespectively of the underlying platform, and (b) the evaluation of a proof-of-concept anomaly-based IDS implementation that incorporates dissimilar detection features, with the aim to assess its performance qualities when running on state-of-the-art mobile hardware on the host device and on the cloud. This approach allows us to argue in favor of a hybrid host/cloud IDS arrangement (as it assembles the best characteristics of both worlds) and to provide quantitative evaluation facts on if and in which cases machine learning-driven detection is affordable when executed on-device.

1. INTRODUCTION

Smartphones have become an integral part of the way we communicate and work. Sixty one percent of mobile subscribers in the US owned a smartphone in 2013 [15], while their European counterparts are closely following [5]. Unfortunately, their use also introduces a variety of security threats. Android-based devices have drawn a lot of unwanted attention from malware authors [12], while users of different devices are all facing privacy issues due to leaks of personal information [10], either performed accidentally by buggy applications, or intentionally by malicious ones.

Over the last few years, many different anomaly-based (AD) mechanisms have been proposed [2–4, 9, 18], specifically targeting smartphones, because of the reasons above. AD techniques profile normal behavior and attempt to identify anomalous patterns of activities that deviate from a predefined profile. They begin by extracting features based on various characteristics, like user actions, API calls, utilization of system resources, and network traffic, and use them to construct normal behavior profiles in a training phase. During regular operation they search for significant deviations from these profiles to detect security related problems.

There are two major directions taken by these systems. *Host-based systems* [2] are designed to operated on the device, and their proponents argue that it is necessary because it is the only way to ensure timely reaction to identified issues. On the other hand, *cloud-based systems* [4] are designed to offload a significant part of their operation to the cloud, where their computationally intensive algorithms and analyses are running. Their designers argue that it is not possible to deploy such mechanisms on-device because of the prohibitive slowdown they would incur on normal phone operations. This is because smartphones usually have limited processing and memory resources compared to those of PCs.

This paper argues that *both* host- and cloud-based protection systems need to operate concurrently to complement each other. The reasoning is straightforward; although cloud-hosted mechanisms do save system resources, on their own they cannot offer real-time protection, and they can leave devices vulnerable when connectivity with the server is poor, something which malware already present on the device could pursue intentionally to masquerade an infection. Nevertheless, we cannot ignore the potential benefits of offloading certain mechanisms to the cloud, such as low overhead and increased battery life.

We propose a new framework that supports both host- and cloud-based protection mechanisms, and facilitates synergy. We focus on anomaly-based intrusion detection systems (IDS) used primarily for malware detection and privacy invasive software [2, 6, 14]. The framework aims to decouple the IDS from where it is hosted, enabling the "movement" of mechanisms from host to cloud and vice-versa transparently, and without any modifications to the IDS. This means that an IDS component will be able to operate autonomously on the device (e.g., for run-time protection or when connectivity issues occur), as well as with the assistance of the cloud for relieving the device from sophisticated but computationally expensive operations.

Ideally, our solution would automatically decide where to deploy a particular IDS, by balancing resource consumption and security requirements such as the time required to detect a threat. Policies regarding resource allocation could also be applied, for instance, if the phone is fully charged we may decide to run more IDS on the device, and as the battery depletes offload them to the cloud.

The focus of this paper is, however, the design, implementation, and evaluation of the framework enabling such hybrid IDS. We demonstrate its capabilities by incorporating four different IDSs introduced in previous work [6–8, 14]. First, we show that we can successfully combine multiple mechanisms to provide broader de-

EuroSec'14, April 13 - 16, 2014, Amsterdam, The Netherlands

fensive capabilities. To the best of our knowledge, we are the first to propose such a system. Second, we evaluate the performance of all four IDS running under our framework, when running on the device and on the cloud, both for training and detection (in earlier work, these mechanisms were only evaluated in terms of effectiveness). This is important because it allows us to make several observations about the various benefits and downsides of each deployment option. While the results we obtained are closely related to the implemented detection mechanisms, they are indicative and can be used as a reference for future work in the subject. Our results can help put similar works [3], which focus mostly on accuracy and do not provide experimental results regarding performance and scalability, in perspective.

Summarizing the above, the contributions of this work consist of the following:

- We design a new framework that facilitates the development of hybrid IDS for mobile devices. Our architecture can support diverse anomaly-based mechanisms to be concurrently applied, either directly on the device, or by offloading computation on the cloud, maximizing synergy between device and cloud-hosted defenses. To our knowledge, literature lacks such a comprehensive proposal and most work on this issue is fragmentary.
- We implement the framework for the iOS architecture. Note, however, that an implementation for the Android platform is straightforward (discussed in 3.1).
- We utilize the framework to develop an IDS that combines four diverse detection mechanisms, introduced in previous work [6–8, 14]. The combination of diverse techniques provides protection against a broader set of attacks.
- Our prototype is evaluated along three main axes: overhead 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. The results provide insights on the actual advantages and disadvantages of hosting a defense on the device or the cloud and can be used as a reference for future work. Our observations show that an optimal deployment strategy varies based on the technique.

The rest of the paper is organized as follows. The next section reviews recent literature for IDS on mobile platforms. Section 2 introduces the proposed framework, and its use to implement a hybrid, four-way IDS as discussed in Sec. 3. In Sec. 4 we present the evaluation of our prototype. Section 6 concludes the paper and gives pointers to future work.

2. THE FRAMEWORK

In this section we describe our framework for developing hybrid IDS for mobile devices. The term "hybrid" refers to the ability to deploy the mechanisms on the host and on the cloud and have them work in synergy. We first present the components of the architecture, and then briefly describe how they interact.

2.1 On-device Components

2.1.1 Event Sensors

Event Sensors monitor and collect information from different layers of the OS. For instance, system calls, inter-process communications (IPC), hardware sensors, API calls, system services (e.g., user SMS), and generally any library call can be monitored and intercepted. An IDS can deploy one or more Event Sensors based on the data it requires to create a valid usage profile and monitor the device for intrusion, as shown in Fig. 1 (1). The preferred way to transparently intercept information exchanges in modern mobile devices is by hooking and overwriting [7] the appropriate functions and system calls. Hooking is lightweight and enables multiple Event Sensors to intercept the same data.

The information collected by the sensors may include sensitive user data, such as SMS, e-mails, etc. Because an IDS could be deployed in the cloud, it may be undesirable to submit raw sensor data to the cloud due to privacy concerns. Sensors can anonymize sensitive data using cryptographic primitives available on most mobile devices, such as fast (sometimes hardware supported) implementations of one-way hash functions like SHA-1 and SHA-2. By obfuscating parts of the the user data early on, we reduce the possibility of inadvertently exposing sensitive information later.

2.1.2 System Manager

The System Manager, shown in Fig. 1 (2), is the brain of the framework as it coordinates all other components. It is the component responsible for deciding which detection mechanisms are going to be applied and their "sensitivity" in terms of detecting intrusions. Different users may have different requirements, so flexibility is an important property.

When the System Manager receives an event from a sensor, it forwards it to the Detection Manager. Along with new events, it also retrieves older events and their characterization from the detection engines from a local database, shown as "Behavior Profiles" in Fig. 1, and also sends them forward.

2.1.3 Detection Manager

The Detection Manager is in charge of running the defensive analysis mechanisms hosted by our framework. It supports two types of engines, a *host* and a *cloud* engine. A host engine runs an IDS on the device, utilizing the information provided by the System Manager (i.e., events and previously generated behavior profiles) to identify malicious behavior. The outcomes of the analysis, which could indicate an intrusion, are sent to the Response Manager. Respectively, a cloud engine executes an IDS on the cloud and is appropriate for computationally demanding analyses. It receives the sensor data forwarded by the System Manager, but it can use its own copy of behavior profile data, instead of the device-local data, to minimize communications with the cloud. The Detection Manager received decisions from the running IDS and forwards them to the Response Manager.

2.1.4 Response Manager

This component is in charge of deciding, and if needed, responding to suspicious activity reported by the Detection Manager. If the analyses signal that an event is suspicious, it decides, based on the "sensitivity" configured by the System Manager, to either ignore it, or perform an action. Response modules can be used to support different actions. For instance, display a notification to the user requesting his input, block the suspicious event, terminate an application, require re-authentication, etc. In all cases, the decision of the IDS, the action taken, and any user feedback are stored back into the behavior profiles database to be used in future decision making. Finally, it informs the Cloud Manager about the incident to ensure that it can also update its database, if it maintains its own.

2.2 The Cloud Manager

Detection engines running on the cloud are managed by the Cloud Manager, which communicates with the Detection Manager on the device. The Cloud Manager can keep its own store of Behavior

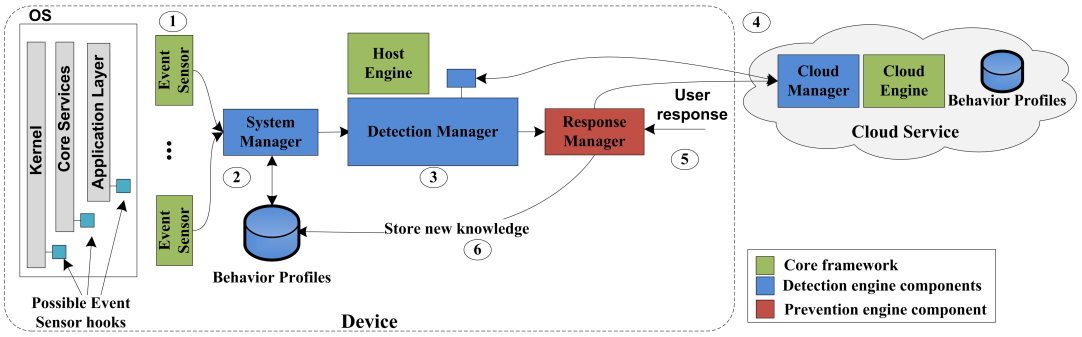

Figure 1: Proposed cross-layer IDS framework for mobile platforms.

Profiles into the cloud, synchronizing with the device when necessary. However, when continuous connectivity with the cloud can be maintained, newly generated knowledge is pushed back to the cloud by the Response Manager.

2.3 Overall Operation

As depicted in Fig. 1 the operation of an IDS based on our framework includes seven steps. As soon as the Event Sensor (1) intercepts a new event, it forwards it to the System Manager (2), which is responsible for: i) loading the behavior profile that corresponds to a detection mechanism, ii) forwarding the necessary data (i.e., the new event and the existing behavior profile) to the Detection Manager (3). The Detection Manager passes the data for classification to the detection engines, which can be running either on the host, or on the cloud (4). The results from the engines are sent to the Response Manager which determine which actions, if any, need to be taken for the particular event (5). Finally, the event itself and the corresponding decision is stored into profiles database, enriching the knowledge of the system.

3. IMPLEMENTATION

To evaluate the architecture described in the previous section, we developed a proof-of-concept implementation of the framework and a hybrid IDS using it.

3.1 System description

We implemented our prototype on Apple's iOS 7 platform. Specifically, it consists of a main daemon, written in Objective-C, containing the functionality for the System, Detection, Response, and Security managers. To implement the event sensors for the prototype we created dynamic libraries (dylibs), utilizing the MobileSubstrate framework. This framework provides a lightweight way to interact with the OS at a low level, and thus enables us to hook, overwrite, and monitor the kernel, application APIs, etc. As for the detection engine, we rely on the OpenCV library which runs natively on the iOS platform and supports a variety of machine learning classifiers. Note that although our prototype targets the iOS platform, it can be easily ported to Android. This stands true because MobileSubstrate is supported by Android as well, while the WEKA framework can be imported as a decision engine to any Android application. Bear in mind, that both these popular OSs need to be jailbroken/rooted before we can install and run our system on the device.

The cloud service part of our framework is implemented as a simple client-server, where the server is considered trusted. The communication link between the smartphone and the cloud entity is also considered to be secure, say, by means of an SSL tunnel. The Cloud Manager and cloud-based detection engines are all contained within a single multi-threaded server written in Java, which uses the popular WEKA framework. Note that WEKA provides the same machine learning classifiers as the OpenCV, so we have capabilities equivalent with the ones on the device.

In a nutshell, the system works both on the host and the cloud. Host-side detection is autonomous, while cloud-based detection relies on the host for sending the behavioral data (in the form of vectors of behavioral attributes) to the cloud, waits for the cloud to process the received data, and makes a decision, which is sent back to the host.

3.2 Detection Mechanisms

We implemented four smartphone detection mechanisms based on previous work [6–8, 14]. The first two detection mechanisms, namely SMS Profiler and iDMA, aim to detect the illegitimate use of system services and identify unknown malware. The other two, coined iTL and Touchstroke, can provide (post) authentication to ensure the legitimacy of the current user.

An overall description of the four detection mechanisms can be found in Tab. 1. The features column corresponds to the sensor events that are required to build the corresponding behavior profile for each detection mechanism. Data Training and Data Testing represent the minimum amount of data required during the training and testing process by the classifiers. Accuracy corresponds to the effectiveness of the detection mechanism, while the behavior profiles are built using the aforementioned specifications. The popular Random Forest algorithm is used as a classification engine, because it has yielded the best results in previous works.

To summarize, these four mechanisms present the following capacities: (a) the SMS Profiler detects misuses pertaining to SMS by observing user behavior, (b) iDMA is able to monitor the behavior of running applications as means to detect malware, (c) iTL is a touch logger that profiles the user of the device based on touching behavior, and (d) Touchstroke provides a keystroke-based authentication system for mobile devices equipped with a touchscreen. A detailed description of these four mechanisms can be obtained from the corresponding works [6–8, 14].

4. EVALUATION

This section provides results derived from the evaluation of the proposed IDS prototype. We utilized the latest Apple smartphone, iPhone 5s, as the host of our IDS. The cloud service was located within the AWS Amazon's EC2 infrastructure, and the client was connected to the cloud service through a local 802.11bg hotspot. The results correspond to the performance of the system when all four detection mechanisms were running either directly on the host

Table 1: Overview of the four detection mechanisms used.

Detection Mechanism	Features	Training	Testing Frequency	Accuracy	Reference
SMS Profiler	Number, Timestamp, Flags, Country, Intruder/Legit	1 week	1 SMS	99.6 %	[8]
iDMA	Method, Malicious/Legit	Legit and Malicious App methods	10 sequences methods	99.1 %	[7]
iTL	Touch Type, X, Y, Timestamp, Intruder/Legit	1 day touch patterns	1 day touch patterns	99.2%	[6]
Touchstroke	Hold-time, Inter-time, Distance, Speed, Intruder/Legit	12 sentence samples	1 sentence	80.6 %	[14]

or on the cloud. Our experiments essentially evaluate two extreme scenarios, where we could either deploy entirely on the host or in the cloud.

Although the effectiveness in identifying intrusions is one of the most critical metrics when designing and evaluating an IDS, two other attributes that indicate how the system behaves during the detection phase are of equal importance. Specifically, the Performance and Timeliness metrics are directly associated with the effectiveness of any IDS with regard to real-time detection (real time means that an intrusion is detected before any, or significant, damage occurs). In particular, these metrics are significant because of the following:

Timeliness An IDS has to react as quickly as possible in order to prevent the attacker from subverting the system. In the literature, two metrics are widely used to decide if the timeliness of a given system; Train Time, which is concerned with time required to build the behavior model and Test Time, which is the time required to classify an anomaly pattern.

Performance The performance of a system can be measured using two metrics; the CPU and memory consumption during the detection process (for both training and testing the mechanism of interest). Bear in mind, that although modern smartphones are powerful, they are significantly more constrained than PCs.

These characteristics are well-known in the area of desktop IDS systems, but they have been neglected in the smartphone domain because the IDS needs to run directly on the device.

Table 2 presents the performance results in terms of Classification Time, CPU, Memory, Battery metrics, for all four detection mechanisms for both host- and cloud- based scenarios. We used the Random Forest algorithm in the detection engine for all experiments. Additionally, the same behavioral set of data were provided to both scenarios during the training and test phases according to Tab. 1. The user-behavior profiles used in every scenario were created based on real data collected from a critical mass of participants. More details on these datasets and the way the profiles were constructed can be obtained from the references provided in Tab. 1. For the cloud scenario, all data produced by the IDS Event Sensors were sent to the cloud service for further analysis via the use of *Cloud Engine*. On the other hand, for the Host scenario, the *Host Engine* analyzed the data outright on the device. Both Train and Test Times have been calculated from the time either *Host Engine* or *Cloud Engine* get the data, until their processing is finished. Bear in mind, that the amount of data the *Host* and *Cloud* Engines have to process is different for each mechanism and it depends on the behavior profile structure, as well as on the number of features used for their creation. For all the metrics included in Tab. 2, we consider the statistical average of the values resulting from the total number of cases. Also, during the measurements, all non- essential services running on the device were disabled.

What we observe from the results is that host-based detection is affordable for the high-end iPhone device used throughout testing. In all cases, the time required to detect and intrusion (i.e., decision making) remains less than 0.4 secs. As expected, the training phase was the one with the longest time penalty requiring almost 6 secs in the worst case (iTL). Generally, both cloud -based training and testing times are lower than host-based times, ranging between 0.35 and 0.45 secs, however they are burdened by communication delays between the smartphone and the cloud. As a result, host-based detection requires significantly less time to classify a new event. On the other hand, CPU performance almost reached 100% when detection occurs on the host itself, while during the training phase it varies between 90% and 97%, depending on the mechanism used.

Table 2 also shows that the training phase stresses the battery to a greater degree than detection. Specifically, between 0.04 and 0.27 mAh\sec, making decisions for new events can be considered a rather lightweight operation. Classifying new events can cause a decrease rate in battery life between 0.05 to 0.22 mAh\sec per event. Bear in mind that detecting intrusions can become a energy-demanding process when the frequency of events rises. This means that the power consumption of the iTL mechanism when performing detection is 0.21 mAh per second (or 2.80 Watt (W)) and that it can be performed for 24.116 events based on a 1.570 mAh battery and 3.8 V. It is important to underline that the average number of touch events collected during our experiments were 1200 per day. We should mention that applications like Safari, iBooks, Maps, and Infinity Blade have a power consumption of 1.1 W, 1.4 W, 2.6 W respectively [1]. Due to the fact that battery lifetime is affected by on-device detection, cloud-based approaches seem to be the best alternative when battery life matters. Of course these results are indicative as they depend on several factors including the smartphone model, the OS, and the battery itself.

Looking at the results derived from the cloud-based scenario, one can observe that CPU consumption remains mostly around 20% and 25% both for the training and testing phases, and only in the iTL case it increases up to 41% and 45% respectively. Lastly, memory consumption fluctuates between 78% and 88% for the host-based scenario depending on the detection mechanism used, and between 62% and 63% for the cloud-based scenario. This is rather straightforward as the first scenario requires more memory to execute and build the necessary classification model, while the second one needs to only construct the proper sockets to initiate communication with the server. From the results it is also obvious that the iDMA and iTL mechanisms are the most demanding in terms of resources because inherently they need to produce a considerably greater mass of data in comparison with the other two.

Summarizing all the above one can arrive to the following conclusions regarding the basic question posed by this paper. It seems that host-based detection and classification consumes every available resource on the device and therefore is far more demanding than that executed in the cloud. This excessive overhead may induce battery drain and cause user discontent. Cloud-based IDS op-

Table 2: Classification, CPU, Memory and Battery performance results.

Detection Mechanism	Scenario	Random Forest							
		Classification (sec)		CPU (%)		Memory (%)		Battery (mAh/sec)	
		Train	Test	Train	Test	Train	Test	Train	Test
SMS Profiler	Device	2.51	0.04	90	97	78	79	0.16	0.10
	Cloud	0.25	0.23	22	21	63	62	0.07	0.07
iDMA	Device	4.83	0.29	96.2	100	82	83	0.27	0.20
	Cloud	0.45	0.33	25	25	63	62	0.06	0.07
iTL	Device	5.42	0.31	97	100	87	88	0.26	0.21
	Cloud	0.35	0.39	41	45	63	62	0.05	0.05
Touchstroke	Device	2.43	0.07	96.5	98.7	79.4	79.7	0.19	0.14
	Cloud	0.26	0.24	25	26	63	62.0	0.04	0.05

eration on the other hand is far more relieving for the mobile device but it cannot be considered real-time. This is obvious from Tab. 2 where for all the detection mechanisms, but one, the test phase lasts far more time than that required when executed on the host. For example, for the first mechanism (SMS profiler) the time needed when detection is conducted on the host and cloud is 0.04 and 0.23 secs correspondingly (i.e., we perceive an additional time penalty of 0.19 secs). Of course, this time gap is due to network communications required for the host to upload the detection data to the cloud and receive the result back. Naturally, as already pointed out, this is not the only problem with an IDS consigned to cloud care. Temporary or even permanent network outages due to the lack of network coverage or malware activity may provide the necessary time window for an attack to successfully complete and compromise the IDS. Considering all the above, we argue that a hybrid IDS deployment may be the King's Solomon solution the problem. That is, having the most heavyweight detection mechanisms running on the cloud and keeping others, which are considered more sensitive mostly in terms of Timeliness, on the device. Of course, this decision needs to be taken on a case by case basis as it is absolutely associated with the detection mechanisms at hand. For instance, in case the IDS affords a user (post)authentication sensor (as that of Touchstroke) it is better to assign it on the cloud. However, when it comes to malware detection (as that of iDMA) it is better to keep it on the host. Last but not least, although battery consumption during the training and testing phases can be considered quite low (having in mind modern smartphones are equipped with a larger 2.000 mAh battery) the detection procedure can still deplete the battery and greatly reduce the autonomy of the device.

5. RELATED WORK

This section reviews host and cloud anomaly-based IDS proposals for mobile devices presented in the recent literature. Note that, theoretical approaches or individual detection mechanisms, that do not provide a comprehensive IDS solution, have been deliberately left out.

One of the first remote monitoring frameworks for the Android platform was proposed in [17]. Specifically, this mechanism was able to analyze the Android kernel to detect anomalies in the system. The authors in [3] have proposed a behavioral detection framework able to detect mobile worms, viruses and Trojans infecting devices under the Symbian OS. However, the detection mechanism was able to run only within the emulator.

TaintDroid a real-time monitoring system for Android OS was presented in [11]. Although, it doesn't use anomaly detection, but "taint tracking" analysis, it was the first security mechanism able to run directly on the device. Paranoid Android prototype was introduced by the work in [16]. This system was the first malware detection system with the ability to analyze on the cloud Android mobile phone replicas.

The authors in [13] presented a cloud-based intrusion detection and response engine for smartphones. According to the authors, the system was able to perform forensics analysis to detect any misbehaviors. Crowdroid, a cloud-based anomaly detection system for Android platforms, was presented in [4]. Crowdroid was able to obtain the traces of applications' behavior, utilizing a crowdsourcing system, to detect malicious applications.

The first host-based detection framework, named Andromaly, was proposed for Android OS [18]. Andromaly was able to detect new types of malware based on samples of known malware families. Also, the authors in [9] presented a host-based anomaly detector for Android malware, coined as MADAM. This system was able to monitor the Android OS to detect malware infections with the help of machine learning techniques. The authors indicate that their system produces an average 7% of CPU overhead and of 3% MB of RAM space, without providing any detailed performance analysis on their system.

Very recently, DREBIN, a host-based detection mechanism for Android OS was introduced [2]. This system was able to detect malware using machine learning techniques and more specifically the Support Vector Machine (SVM) classifier. The authors shortly discuss about the performance of their system, which was found to be between 10 and 20 secs for high and low-end smartphones respectively.

From the above it seems that several research efforts have been done so far for deploying IDS systems in the smartphone ecosystem. However, it is obvious that none of them introduces a dual host/cloud IDS architecture and (in addition) no work addresses performance issues in detail.

On the top of that we found that nearly all the solutions proposed so far in the literature do not occupy themselves with performance issues, but only care about increasing the accuracy of the system and reducing critical factors including false positives and negatives. However, while the aforementioned issues are indeed the most important for every IDS, one should also consider performance at least when it comes to the smartphone realm. This is because although modern mobile devices are quite capable of coping with high computation and memory demands that are required by, say, machine learning algorithms for the classification procedure, this leads to a high cost regarding battery resources and user discontent.

6. CONCLUSIONS

Given the increased popularity of smartphones and the services they can support, more and more people will be threatened by upcoming instances of mobile malware and illegitimate usage of services by unauthorized users. In this context, few will argue that the

Smartphone ecosystem needs specially designed, custom-tailored IDS that are lightweight and effective, as well as easy to use.

In this paper, we address a number of still unresolved issues around mobile IDS. First off, we present our design for a mobile IDS architecture and a framework supporting it for deploying defensive mechanisms both on the host and the cloud, and having them work in synergy. Second, we implement a proof-of-concept IDS, implemented for state-of-the-art mobile hardware and including four different detection mechanisms. Third, we conduct a thorough performance evaluation of the detection mechanisms in a effort to approximate the cost of a real IDS. In this respect, the results obtained by this study can be used as a reference to any future work in the mobile IDS domain. Still, research on mobile IDS has a long road ahead as several important issues remain semi-explored, including the design of novel lightweight decision engines, the lowering of energy consumption, and the exploration of new advance methods to detect intrusions.

7. REFERENCES

[1] F.-A. L. S. an Brian Klug. Apple iphone 4s: Thoroughly reviewed. Anandtech, 2011. http://www.anandtech.com/show/4971/apple-iphone-4s-review-att-verizon/15.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. Drebin: Efficient and explainable detection of android malware in your pocket. In *Proc. of 17th Network and Distributed System Security Symposium*, NDSS '14. IEEE, 2014.

[3] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 225–238. ACM, 2008.

[4] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 15–26, New York, NY, USA, 2011. ACM.

[5] comScore. Smartphones reach majority in all EU5 countries. comScore Data Mine, 2013. http://www.comscoredatamine.com/2013/03/smartphones-reach-majority-in-all-eu5-countries/.

[6] D. Damopoulos, G. Kambourakis, and S. Gritzalis. From keyloggers to touchloggers: Take the rough with the smooth. *Computers & Security*, 32(0):102 – 114, 2013.

[7] D. Damopoulos, G. Kambourakis, S. Gritzalis, and S. Park. Exposing mobile malware from the inside (or what is your mobile app really doing?). *Peer-to-Peer Networking and Applications*, pages 1–11, 2012.

[8] D. Damopoulos, S. A. Menesidou, G. Kambourakis, M. Papadaki, N. Clarke, and S. Gritzalis. Evaluation of anomaly-based ids for mobile devices using machine learning classifiers. *Security and Communication Networks*, 5(1):3–14, 2012.

[9] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra. Madam: A multi-level anomaly detector for android malware. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, MMM-ACNS'12, pages 240–253. Springer-Verlag, 2012.

[10] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2011.

[11] 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 *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6. USENIX Association, 2010.

[12] F-Secure. Mobile threat report july-september 2013. F-Secure Research, 2013. http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q3_2013.pdf.

[13] A. Houmansadr, S. Zonouz, and R. Berthier. A cloud-based intrusion detection and response system for mobile phones. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 31–32, 2011.

[14] G. Kambourakis, D. Damopoulos, D. Papamartzivanos, and E. Pavlidakis. Introducing touchstroke: Keystroke-based authentication system for smartphones. *Security and Communication Networks, Special Issue on: Challenges and Opportunities in Next Generation Cyberspace Security*, To appear.

[15] nielsen. Mobile majority: U.s. smartphone ownership tops 60%. nielsen Mobile Newswire, June 2013. http://www.nielsen.com/us/en/newswire/2013/mobile-majority--u-s--smartphone-ownership-tops-60-.html.

[16] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: Versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM.

[17] A.-D. Schmidt, F. Peters, F. Lamour, and S. Albayrak. Monitoring smartphones for anomaly detection. In *Proceedings of the 1st International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, MOBILWARE '08, pages 40:1–40:6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[18] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.