DOI:
10.1109/SURV.2013.101613.00077

Document Version
Peer reviewed version

Evolution, Detection and Analysis of Malware for Smart Devices

Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda

***Abstract*—Smart devices equipped with powerful sensing, computing and networking capabilities have proliferated lately, ranging from popular smartphones and tablets to Internet appliances, smart TVs, and others that will soon appear (e.g., watches, glasses, and clothes). One key feature of such devices is their ability to incorporate third-party apps from a variety of markets. This poses strong security and privacy issues to users and infrastructure operators, particularly through software of malicious (or dubious) nature that can easily get access to the services provided by the device and collect sensory data and personal information. Malware in current smart devices –mostly smartphones and tablets– have rocketed in the last few years, in some cases supported by sophisticated techniques purposely designed to overcome security architectures currently in use by such devices. Even though important advances have been made on malware detection in traditional personal computers during the last decades, adopting and adapting those techniques to smart devices is a challenging problem. For example, power consumption is one major constraint that makes unaffordable to run traditional detection engines on the device, while externalized (i.e., cloud-based) techniques rise many privacy concerns.**

This article examines the problem of malware in smart devices and recent progress made in detection techniques. We first present a detailed analysis on how malware has evolved over the last years for the most popular platforms. We identify exhibited behaviors, pursued goals, infection and distribution strategies, etc. and provide numerous examples through case studies of the most relevant specimens. We next survey, classify and discuss efforts made on detecting both malware and other suspicious software (grayware), concentrating on the 20 most relevant techniques proposed between 2010 and 2013. Based on the conclusions extracted from this study, we finally provide constructive discussion on open research problems and areas where we believe that more work is needed.

***Index Terms*—smart devices, malware, grayware, smartphones, security, privacy.**

I. Introduction

SMART devices are rapidly emerging as popular appliances with increasingly powerful computing, networking and sensing capabilities. Perhaps the most successful examples of such devices so far are smartphones and tablets, which in their current generation are far more powerful than early personal computers (PCs). The key difference between such "smart" devices and traditional "non-smart" appliances is that they offer the possibility to easily incorporate third-party applications through online markets. The popularity of smart devices –intimately related to the rise of cloud-computing paradigms giving complementary storage and computing services – is backed by recent commercial surveys, showing that they will very soon outsell the number of PCs worldwide [1]. For example, the number of smartphone users has rapidly increased over the past few years. In 2011, global mobile handset shipments reached 1.6 billion units [2] and the total smartphone sales reached 472 million units (58% percent of all mobile devices sales in 2010) [3]. In fact, the number of ANDROID OS and IOS users alone increased from 38 to 84 million between 2011 and 2012 according to a report by Nielsen [4]. The same report also indicates that the average number of applications per device increased from 32 to 41 and the proportion of time spent by users on smartphone applications almost equals the time spent on the Web (73% vs. 81%). Furthermore, the number of worldwide smartphone sales saw a record of 207.7 million units during 2012, rising up 38.3% with respect to the same period in the previous year [5]. Specifically, the global mobile Operating System (OS) market share shows that ANDROID OS reached 69.7% at the beginning of 2013, racing past SYMBIAN OS, BLACKBERRY OS and IOS as depicted in Figure 1.

New smart devices are appearing at a steady pace, including TVs [6], watches [7], glasses [8], clothes [9] and cars [10]. This is not only playing a key role in bringing to reality much-discussed paradigms such as wearable computing or the Internet of Things, but also finding innovative and very attractive applications in critical domains such as, for example, healthcare. Both medical staff and patients are increasingly taking advantage of such devices, from regular tablets and smartphones [11] to smart pillboxes [12], and the new generation of smart wearable systems (SWS) for health monitoring (HM) or implantable medical devices (IMDs) [13], among others.

A. Ubiquitous Networking and Smart Devices

One key element behind the popularity of smart devices is their mobile nature along with their capabilities to provide pervasive user connectivity. Wireless communication technologies offer smart devices the ability to ubiquitously communicate with an ample variety of Internet services, remotely located personal appliances, and wearable or implantable objects. The most common wireless technologies used by current smart devices are *infrared* (IR) and *radio frequency* (RF) communication. While the use of IR has gone unnoticed during the proliferation of smartphones, it has recently become popular again [15]. A wide variety of RF technologies are present

Manuscript received March 22, 2013; revised May 28, 2013 and August 27, 2013.

G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda are with the Computer Security Lab (COSEC), Department of Computer Science, Universidad Carlos III de Madrid, Av. de la Universidad 30, 28911 Leganés, Madrid, Spain (e-mail: guillermo.suarez.tangil@uc3m.es, {jestevez, pperis, arturo}@inf.uc3m.es).

Digital Object Identifier 10.1109/SURV.2013.101613.00077

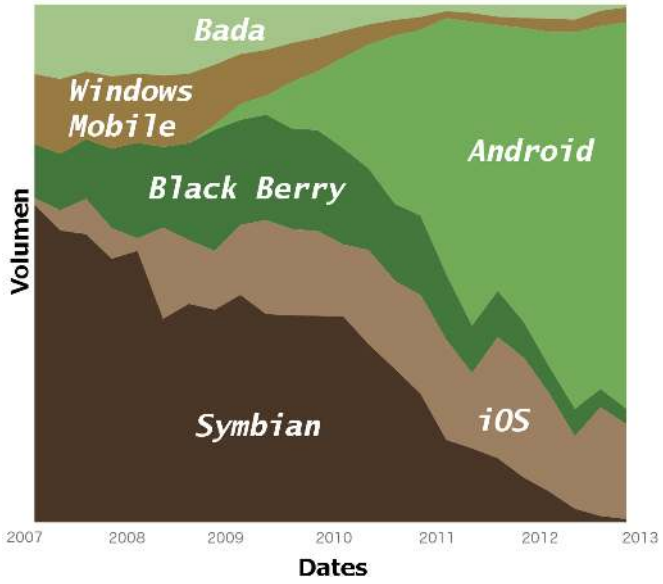

Fig. 1. Main smartphone platforms by market share from 2007 to 2012 [14].

in wireless communication capabilities for smart devices. Perhaps the most notorious ones are Near field Communication (NFC), IEEE 802.15.1 (Bluetooth), IEEE 802.11 (WiFi), Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS), Radio Data System (RDS), Global Positi oning System (GPS), Software Defined Radio (SDR) and Cognitive Radio (CR).

Integrated wireless communications also provide smart devices with new sensor capabilities. Current sensors have evolved from mechanical transducers featured with network connectivity (e.g., Wireless Sensor Networks [16] or Smart Grids [17]) to communication-centric systems where many information is acquired via communications interfaces. For instance, some communication techniques allow devices to sense their position based on radio signals transmitted either by a local positioning system (e.g., cellular base stations, WiFi access points, etc.) [18] or by a global positioning system such as GPS. Additionally, communication standards such as Bluetooth Low Energy (LE), namely *Bluetooth SMART*, allow smart devices to sense information by simply communicating with them. Similarly, the use of RFID and NFC can be used to sense near field information by encoding it in programmable tokens or tags (e.g., *SmartTags* [19]). Both *Bluetooth SMART* and *SmartTags* technologies transform everyday objects into powerful data sensors.

All these heterogeneous communication and sensing capabilities pull together several opportunistic networking paradigms [20], such as: (i) Device-to-Cloud, (ii) Device-to-Device, and (iii) Device-to-Environment, which have played an important role in the proliferation of communication-based services. For instance, paradigm (i) offer users the possibility to remotely manage their devices, back-up data, or access online software markets. In addition to this, other paradigms such as (ii) and (iii) allow users to interact with their environment for a better social experience, such as for example multi-player games. Furthermore, the combination of different communication and networking paradigms has made possible the rise of very promising services, such as NFC-based e-payment schemes, Location-Based Services (LBS), or even novel forms of authentication in anonymous networks [21]. Most researchers agree that this trend towards a rich ecosystem of wireless technologies will continue in the near future, quite possibly in a more versatile way as (smart) devices are increasingly capable of adaptively incorporating new software-based communication capabilities via *RadioApps* [22].

While this fruitful environment of cheap, fast and heterogeneous communications capabilities has been key to the success of smart devices, it has also brought about a number of security and privacy concerns. Attack vectors have multiplied ([23], [24]), and the availability of a myriad of networking paradigms has given rise to new epidemic behaviors (see, e.g., [25]). Even services that historically have been exceptionally harmless have suddenly turned into a potential menace: one of the most recent examples is the advent of AM/FM radio-based attacks [26], which have proved to be particularly viral due to the broadcast nature of RDS and the increasing popularity of SDR and CR systems [27] based on RadioApps.

Recent communication-centric sensors rise new privacy problems. For instance, sensors such as GPS can potentially leak users' location, and NFC-equipped devices can pose traceability issues. Other sensors, such as for example the accelerometer or the gyroscope, can be used to infer the location of screen taps and, therefore, be used to guess user passwords. These Device-to-Environment communication paradigms can be especially harmful when correlated with others such as Device-to-Cloud or Device-to-Device. All these features pose a security threat to communications and fundamental research in this regard is therefore required. In fact, several approaches have tackled privacy leakage from the sensor's perspective [28], [29]. We next provide a closer look at some of these issues.

B. Malware and Smart Devices

In many respects, smart devices present greater security and privacy issues to users than traditional PCs [30]. For instance, many of such devices incorporate numerous sensors that could leak highly sensitive information about users location, gestures, moves and other physical activities, as well as recording audio, pictures and video from their surroundings. Furthermore, users are increasingly embedding authentication credentials into their devices, as well as making use of on-platform micropayment technologies such as NFC [31].

One major source of security and privacy problems is precisely the ability to incorporate third-party applications, primarily from available online markets but also by other means. There are currently two established models of smart devices according to how users can access such markets [32]. In the open-market model, users are free to install applications from any online market, whereas the so-called walled-garden market model restricts the market from which users can install applications. (In spite of this, users have found ways of circumventing such restrictions by modifying the device so that other markets will be accessible too.) Many market operators carry out a revision process over submitted apps, which presumably also involves some form of security testing

to detect if the app includes malicious code. So far such revisions have proven clearly insufficient for several reasons. First, market operators do no give details about how (secu rity) revisions are done. However, the ceaseless presence of malware in official markets reveals that operators cannot afford to perform an exhaustive analysis over each submitted app. Second, determining which applications are malicious and which are not is still a formidable challenge. This is further complicated by a recent rise in the so-called grayware [33], namely apps that are not fully malicious but that entail security and/or privacy risks of which the user is not aware. And finally, a significant fraction of users rely on alternative markets to get access for free to apps that cost money in official markets. Such unofficial and/or illegal markets have repeatedly proven to be fertile ground for malware, particularly in the form of popular apps modified (*repackaged*) to include malicious code.

The reality is that the rapid growth of smartphone technologies and its widespread user-acceptance have come hand in hand with a similar increase in the number and sophistication of malicious software targeting popular platforms. Malware developed for early mobile devices such as *Palm* platforms and *featured mobile phones* was identified prior to 2004. The proliferation of mobile devices in the subsequent years translated into an exponential growth in the presence of malware specifically developed for them (mostly SYMBIAN OS), with more than 400 cases between 2004 and 2007 [34], [35]. Later on that year, IPHONE and ANDROID OS were released and shortly became the predominant platforms. This gave rise to an alarming escalation in the number and sophistication of malicious software targetting these platforms, particularly ANDROID OS. For example, according to the mobile threat report published by Juniper Networks in 2012, the number of un ique malware variants for ANDROID OS has increased by 3325.5% during 2011 [2] and by 614% between 2012 and 2013 [36]. A similar report by F-Secure reveals that the number of malicious ANDROID OS applications received during the first quarter of 2012 increased from 139 to 3063 when compared to the first quarter of 2011 [37], and by the end of 2012 it already represents 97% of the total mobile malware according to McAfee [38].

The main factors driving the development of malware have swiftly changed from research, amusement and the search for notoriety to purely economical –and political, to a lesser extent. Current malware industry already generates substantial revenues [39], and emergent paradigms such as Malware-as-a-Service (MAAS) paint a gloomy forecast for the years to come. This admits a simple explanation from an economic point of view: all in all, attackers seek to minimize the cost required to achieve their goals and, therefore, aim at obtaining the maximum revenues with minimal efforts. For example, the inequality

$$Cost(Attack) < Potential\ Revenue \tag{1}$$

is used in [40] to give a cost-benefit analysis of mobile attacks. This fits perfectly the case of smart devices such as smartphones, where malware is rather profitable due to (i) the existence of a high number of potential targets and/or high value targets; and (ii) the availability of reuse-oriented development methodologies for malware that make exceedingly easy to produce new specimens. Both points are true for the case of ANDROID OS and explain, together with the open nature of this platform and some technical particularities, why it has become such an attractive target to attackers (see for example Figure 2, where the correlation between the market share and the number of unique malware cases reported is straightforward).

Correlations –if not causations– such as those discussed above are paramount to understand future tendencies and threats, not only in the case of smartphones or tablets but also in other devices that soon will likely proliferate. For instance, it has been recently reported that medical devices are plagued with malware [42]. In the near future, it is quite plausible that similar risks will affect vulnerable IMDs [43], leaving users and patients exposed to exfiltration of highly-sensitive medical information or even malicious manipulation [44].

C. The Malware Challenge for Smart Devices

Thwarting malware attacks in smart devices is a thriving research area with a substantial amount of still unsolved problems. In the case of smartphones, one primary line of defense is given by the security architecture of the device, one of whose foremost features is a permission system that restricts apps privileges. This has proven patently insufficient so far. For example, in the case of ANDROID OS apps request permissions in a non-negotiable fashion, in such a way that users are left with the choice of either granting the app everything it asks for at installation time or it will not be possible to use it. Most users simply do not pay attention to such requests; or do not fully understand what each permission means; or, even if they do, it is hard to figure out all possible consequences of granting a given set of privileges. For example, applications requesting permission to access the accelerometer of a smartphone or a tablet are rather common. However, it has been demonstrated that it is possible to infer the keys pressed by the user on a touchscreen from just vibrations and motion data [45]. Thus, using such a permission in conjunction with Internet access –another rather common privilege– could pose a serious risk of data exfiltration. On top of that, the problem aggravates in platforms where apps can interact with each other and share information, as one needs to consider the privileges acquired by potential collusions.

Many of these problems cannot be solved by market operators alone or by enhanced security models, as they really depend on each user's privacy preferences. For example, a leakage of data such as one's location or the list of contacts might well constitute a serious privacy issue for many users, but others will simply not care about it.

Even if a piece of malware gets it way into a device, it remains unclear how it is possible to detect its presence. Traditional signature-based antimalware techniques suffer from inherent limitations: they can only detect malware for which a signature is available, and are useless against polymorphic and metamorphic code. For example, a recent report by Zhou et al. [46] shows that common smartphone antivirus software detects only between 20.2% and 79.6% of analyzed malware. More optimistic studies such as AV-Test [47], performed with a much more restricted dataset, shows that 31 out of 41 solutions

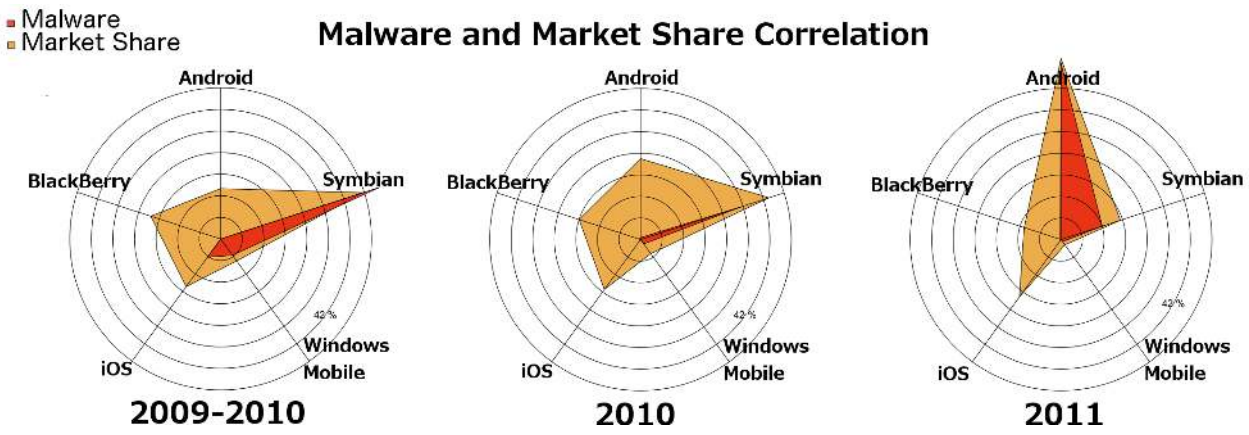

Fig. 2. Correlation between the number of malware cases and platform market share during a) 2009-2010 [41], b) 2010 [2], and c) 2011 [2].

tested presented a detection rate lower than 90%. Approaches based on dynamic code analysis [48] are promising, but adopting and adapting them to smart devices is not straightforward. For instance, many devices suffer from strong limitations in terms of power consumption, so a constant monitoring executed on the platform may be simply unaffordable. External analysis performed on the cloud in near real time constitute an alternative, although it is not exempted from privacy-related risks.

D. Scope and Organization

In this article, we present a comprehensive survey of the evolution and current state of malware for smart devices and techniques proposed to thwart malware attacks. Our analysis is strongly biased towards smartphones, since they currently are the most extended class of smart devices and the platform of choice for malware developers and security researchers. However, our discussion and conclusions apply to other devices as well, and can help to better understand the problem and to improve upon current defense techniques. In this regard, our survey complements and extends other works such as [24].

The rest of this paper is organized as follows. In Section II we describe current smartphone security architectures and discuss a number of research works that have recently proposed enhanced models to provide protection against malicious applications. In Section III we provide a characterization of the various categories of malware developed for smart devices by identifying possible attack goals, distribution and infection strategies, and exhibited behavior. Other authors (e.g., [33], [46]) have previously discussed similar issues for smartphone malware, but not to the extent covered by this work. Furthermore, our taxonomy is used to analyze the evolution of malware using a representative sample of specimens that have gained notoriety over the last few years.

Section IV analyzes and discusses malware detection approaches specifically developed for smart devices. Again, we first identify a number of features according to which each technique can be classified and use them to provide a systematic review of the most relevant works proposed so far. Among our contributions, we identify an extensive number of indicators that can be monitored to detect the presence of malware and that apply to any kind of smart device – not only smartphones or tablets. Additionally, we correlate these features with our malware characterization, pointing out how each class of malicious behavior manifests in terms of observable indicators.

Finally, in Section V we discuss open research topics and in Section VI describe our main conclusions.

II. Security Models in Current Smart Devices

In this section we provide an overview of the security models and protection measures incorporated in current smart devices, with particular emphasis on smartphones. The two major mobile platforms –iOS and Android OS– are built upon traditional desktop Operating Systems (OS) and inherit some security features from them. However, they also employ more elaborated security models designed to better fit the architecture and usage of these devices.

A. Security Features

A number of recent works (e.g., [49], [50]) have provided detailed account of the major security features incorporated in smartphones. In what follows we restrict ourselves to highlight the fundamentals about:

1) security measures implemented at the market level;
2) security features incorporated in the platform; and
3) an overview of recently proposed security mechanisms

with particular emphasis on the protection against malware that they provide.

1) Market Protection: A primary line of defense against malicious software consists of preventing it from entering available distribution markets. To this end, two basic security measures are applied at the market level:

- **Application review.** Some official markets analyze submitted apps before making them available for download and install. Operators do not give details about the particularities of such reviews, but it is generally understood that some form of security testing is carried out. Furthermore, in walled-garden models devices can only access some markets, which presumably only distribute reviewed apps.

- **Application signing.** Most markets force authors to sign their apps. This allows authors to claim authorship and also has some technical consequences in certain platforms (e.g., apps signed with the same certificate can share resources). Thus, a device can be sure about the integrity of an app by verifying the associated signature against the corresponding certificate authority.

Both measures have proven so far insufficient to combat malware. Manually reviewing applications is a a difficult and time-consuming task, impossible to perform in full extent due to the massive number of applications being submitted every day. Automated approaches have been recently explored as an affordable alternative [51], [52], [53], [54]. For instance, in 2012 Google announced an application approval tool named Google Bouncer [53] for ANDROID OS. Also in this line, Zhou et al. proposes *DroidRanger* for detecting smartphone malware in Android markets [54], [55]. Their analysis shows that the infection rate in alternative marketplaces is one order of magnitude higher than the official marketplace. Additionally, they found that about 0.1% of the 204.040 analyzed applications are malicious. We however believe that such a fraction is much higher for two reasons. On the one hand, samples were taken during a two-month period in the first and third quarter of 2011. However, according to McAfee Threat Report [56], the number of ANDROID OS malicious samples experimented an exponential growth of 400% during the fourth quarter of that year. On the other hand, the detection heuristics used by authors present a high false negative rate, ranging from 5.04% to 23.52%.

Even if application review processes were perfect, many devices install applications through unofficial markets in which there are no guarantees whatsoever about the trustworthiness of such apps. Application signing can give users some assurance about the integrity of software downloaded from a questionable source, particularly when such software claims to be an unmodified copy of the same available in official markets. But most of the time users do not perform such verifications, nor it is possible to do so in many cases as signatures are stripped off.

2) Platform Protection: Current platforms incorporate a number of mechanisms to confine and limit the actuation of malicious apps once installed in the device:

- **Permissions**. Most platforms provide a permission-based system aimed at restricting the actions that an app can execute on the device, including access to stored data and available services (e.g., networking, sensors, etc.). Au et al. [57] examine the permission system of several smartphone OS, focusing on:
  1) The amount of control users have over app permissions. Depending on the granularity offered by the OS, users can grant privileges using precise or coarse permissions. Additionally, such permissions cannot always be individually enabled or disabled.
  2) The information they convey to the user. Several platforms offer the users specific information about how applications are using resources. While some OS only inform of what resources the application may use, others track the actual use of permissions throughout execution.
  3) The interactivity of the system. Some permission systems require a heavy intervention of the user. Typically, fine-grained permissions require more interaction than coarse-grained. Furthermore, permissions can either be requested only once (assuming they will remain the same) or they can be requested periodically.

  A summary of their analysis is shown in Table I. These results will be further discussed later on Section II-B when discussing the security features of the most important platforms. A recent study by Felt et al. [58], [59] on the effectiveness of app permission systems concludes that they are rather effective at protecting users. However, in the case of ANDROID OS it points out that many apps request a significant amount of permissions identified as potentially dangerous and that frequent exposure to warnings drastically reduces effectiveness. Furthermore, authors also conclude in [59] that apps are often overprivileged due to a lack of documentation and development bad practices. In this regard, Barrera et al. [60] propose a methodology for analyzing permission-based security models and suggest to increase the expressiveness without maintaining the total number of permissions.
- **Sandboxing**. Trusted execution environments are a security mechanism used by some platform architectures to isolate running applications based on mandatory access control policies. Sandboxing can provide protection against malicious applications to a certain extent, but are ineffective if users overlook the permissions entitled to installed apps. Furthermore, sandboxing do not prevent apps from exploiting system or kernel vulnerabilities and, besides, can also be bypassed in some cases [61]. In this regard, several works [62], [32], [63], [64] propose the use of hypervisors that run directly on the hardware. Other authors (e.g., [65]) have focused on optimizing the virtual machine manager, as virtualization introduces a trade-off between security and performance [66].
- **Interactions between apps**. Some platforms provide the developer with a rich inter-application communication system to facilitate component reuse. Such Inter Component Communication (ICC) systems introduce several security issues. For example, in a compromised device messages exchanged between two components could be intercepted, stopped, and/or replaced by others, as they generally are not encrypted or authenticated. Additionally, two or more malicious applications can collude to violate app security policies, such as for example in the so-called re-delegation attacks [67]. Chin et al. [68] have recently identified a number of security risks derived from the app interaction system in ANDROID OS. Their reported results show that 97% of the analyzed applications are exposed to activity hijacking; 57% to activity launch; 56% to broadcast injection; 44% to broadcast theft; 19% to service hijacking; 14% to service launch; and 13% to system broadcast without action check.
- **Remote management**. Some market and network operators, as well as platform manufacturers, are empowered with the ability to remotely remove apps from the device

TABLE I
PERMISSION MODELS IN THE MAIN SMARTPHONE PLATFORMS [57].

Platform	#Perm.	Control	Information	Interactivity
ANDROID OS	75	Medium	High	Low
WINDOWS MOBILE	15	Medium	Medium	Low
IOS	1	Low	Low	Low
BLACKBERRY OS	24	High	High	High

and even repair damages caused by malware. This can be seen as an extension of other functionalities already present, such as for example updating the OS or applying patches. However convenient, this feature can be seen by many users as too intrusive and is not exempt from risks, both privacy-wise but also in case of compromise of the remote management function.

3) Other Proposals: Over the last few years there has been an explosion of proposals suggesting enhanced security models and alternative policy languages to improve upon the limitations discussed above. The interested reader can find a summary in recent surveys, such as for example [50]. The majority of them fall in one or more of the next categories:

1) **Rule driven policy** approaches [69], [70], [71], [72] propose richer languages based on rules, aiming at palliating insufficient policy expressibility on current protection systems.
2) **High-level policy** protection techniques focus on enforcing information flow throughout the system. Several approaches focus on applying different labeling systems [73], while others enforce full isolation based on distinct security profiles within a single device [65].
3) **Platform hardening** aims at simplifying underlying platform layers, i.e., bootloader and kernel, to mitigate the risk of unpatched vulnerabilities [32]. SELinux-based systems [74] and remote attestation [75] approaches can be applied to improve trusted computing base protection.
4) **Multiple-users** protection assumes scenarios where different users share the same device. Several approaches focus on applying different access control mechanisms such as DifUser [76] or RBACA [77] (a Role Based Access Control for Android).

Most of these proposals would certainly provide enhanced protection against malicious apps. However, in many cases they ultimately rely on richer –and more complex– policies that users must specify. But users generally lack security expertise [78], and developing complete and consistent security policies is far from being an easy task even for experts with the appropriate background. It can be argued that devices could use policies created by others, but it is unclear to what extent "one size fits all." Furthermore, there is an incipient interest on intentionally bypassing the platform protection mechanisms to gain full control of the device and, for example, install apps otherwise forbidden.

B. Security Features in Dominant Platforms

When compared with traditional PCs, smartphone platforms have taken an innovative approach to securing the device and the distribution of software. We next provide an overview of some of the security features present in the five platforms that currently dominate the market.

1) Symbian: SYMBIAN OS security model is based on a basic permission system. Phone resources are controlled by the OS using a set of permissions called "capabilites". Furthermore, applications run in user space, while the OS run in kernel space. Those applications requiring access to protected libraries must be signed using a certificate issued by Symbian, while all others can be self-signed [49]. Protection at the market level is inexistent or very low.

2) BlackBerry: BLACKBERRY security model is based on a coarse-grained permission protection model. Applications have very limited access to the device resources and, as in the case of BLACKBERRY OS, they must be signed by the manufacturer (RIM) to be able to access resources such as, for example, the user's personal information. Additionally, applications must get user authorization to access resources such as the network. However, once the user grants access to an application to use the network, the application can both send SMSs and connect to Internet [79]. Although applications are not executed in a sandbox, some basic process and memory protection is offered. For instance, a process cannot kill other processes nor access memory outside the app bounds.

3) Android: Google's ANDROID OS security model relies on platform protection mechanism rather than on market protection, as users are free to download applications from any market. Applications declare the permissions they request at installation time through the so-called manifest. If the user accepts them, the operating system will be in charge of enforcing them at running time.

Many researchers have pointed out that ANDROID OS's permissions are overly broad and have proposed alternatives and extensions. For example, Ongtang et al. propose a fine-grained permission model called *Saint* to limit the granularity at which resources are accessed [72]. Similarly, Jeon et al. [80] propose a framework that enhances ANDROID OS's security policies and extends permission enforcement both an installation time and during runtime. Schreckling et al. introduced in [81] *Constroid*, a framework to define data-centric security policies for access management. Security policies are here defined for each individual resource, instead of specifying permissions for each app. Furthermore, such definition can be done at a fine-grained level, allowing users to, for example, grant an app access to a part of the address book only. A major consequence is that security policies are therefore defined by the user, not by the developer. However, this approach can easily overwhelm users as they are held responsible of specifying security and privacy policies.

Additionally, ANDROID OS uses sandboxing technique and Address Space Layout Randomization (ASLR) to protect applications from malicious interference of others apps. Although ANDROID OS isolates each running process, apps can still communicate with each other using ICC, a rich functionality that, however, introduces risks such as those discussed before. Bugiel et al. introduce a security framework called TrustDroid [82] to separate trusted an untrusted applications into domains, firewalling ICCs among these domains. Similarly, Dietz et al. propose *Quire* [83], a signature scheme that allow developers to specify local (ICC) and remote (RPC) communication restrictions. Other proposals such as TaintDroid [84], AppFence [85] or XManDroid [70] closely monitors apps to enforce given security policies. The first two uses dynamic taint analysis to prevent data leakage and protect user's privacy, while the last one extends ANDROID OS's security architecture to prevent privilege escalation attacks at runtime. The main difference between TaintDroid and AppFence is that the latter tries to covertly anonymize private information prior to blocking leakages.

Furthermore, all ANDROID OS applications must be signed with a certificate to identify the developer. However, the certificate can be self-signed, in which case no certificate authority verifies the identity of the developer.

Several articles discuss ANDROID OS security model [86], [87], providing a deep understanding of android architecture. Enck et al. [88] also present a study of Android security by analyzing 1100 free applications. We refer the reader to these works for further details.

4) iOS: Apples IOS security model [89] relies on market protection mechanisms rather than enforcing complex permission polices on the device at installation time. Apple's App Store is a walled-garden market with a rigorous review process. Those processes are essential for preventing malware from entering the devie, as runtime security mechanisms are limited to sandboxing and user supervision. IOS isolates each third-party application in a sandbox. However, most of the device's resources are accesible[1] and misuse of a few of them –such as GPS, SMS, and phone calls– can only be detected by the user after installation. Furthermore, IOS sandboxing model is weaker than ANDROID OS's, as Apple only uses one sandbox to run all applications, whereas Google separates each application in a sandbox [91].

Specific details on Apples App Store application review are unknown. In July 2009 Apple revealed that at least two different reviewers study each application [92]. However, it is probable that Apple uses also static and dynamic analyses.

Applications distributed on Apple's App Store must be signed by a valid certificate issued by Apple. Developer certificates are issued to individuals and/or companies after obtaining a verified Apple credential. IOS dynamically verifies that the application is signed, and therefore it is trusted, before executing it. Nevertheless, IOS can be tampered with (jailbroken) to install applications from alternative markets. This practice violates Apple policies, causes the device to lose its warranty, and avoids prevention of shellcode injection.

[1]In IOS version 5, although Apple is likely to introduce some modifications in IOS version 6. Specifically, the new version will restrict access to most of the device's resources [90].

Latest versions of IOS provide a number of features to protect user data based on master encryption keys and protected by a passcode. The entire file system is encrypted using block-based encryption and can only be decrypted when the phone is unlocked. Additionally, IOS supports ASLR and Data Execution Prevention (DEP) to prevent the execution of arbitrary code at runtime.

5) Windows Mobile: Microsoft's market protection model for WINDOWS MOBILE systems is based on application review. Developers are also validated prior to application's approval. Platform protection in WINDOWS MOBILE is similar to ANDROID OS. It uses a trusted boot component and code signing to protect the integrity of the operating system. It also provides signed drivers and applications through the *Windows Phone Store* online market.

Latest versions of WINDOWS MOBILE (Windows Phone 7 and 8) incorporate isolation among different sandboxes [93], and each app is executed in its own sandbox, named "chamber." While chambers are defined and implemented using a number of system policies, each security policy defines what permissions are given to an app, known as capabilities. In this regard, users are informed of the capabilities of an application prior to install. However, the only control users have over these capabilities at runtime is quite limited, as only GPS needs user authorization the first time an application request access to it [57].

III. MALWARE IN SMART DEVICES: EVOLUTION, CHARACTERIZATION AND EXAMPLES

Malicious applications for smart devices –notably smartphones– have rocketed over the last few years, evolving from relatively simple apps causing annoyance to complex and sophisticated pieces of code designed for profit, sabotage or espionage. In this Section we first provide a brief overview of such evolution from early mobile platforms to current devices. We subsequently propose a number of features that can be used to classify, characterize and better understand malware for smart devices.

A. Evolution

As in the case of traditional PCs, where malware evolution was intimately connected to the increase in computing resources and the advent of the Internet, the complexity and hostility of malicious software has intensified from early mobile handsets to the current generation of smart devices. In the early 2000s, *Palm* platforms were affected by malicious software that mimicked strategies well-known in PC malware. For example, *Symb/Liberty*, *Symb/Vapor* and *Symb/Skuller* were popular trojans at the time, i.e., applications that perform some useful function while simultaneously conducting malicious activities. Others such as *Symb/Phage* employed classical virus propagation strategies to infect additional programs present in the handset. Their malicious payload varied, but in all cases it was sought to inflict damage over user information or corrupt system files in order to cause a device failure.

The rise of *featured mobile phones* brought about a variety of distinctive infection vectors when compared to traditional

PCs, primarily through the communication and networking functions offered by 3G, Wi-Fi, EDGE, Bluetooth, the SMS/MMS messaging system, and NFC [94], [95]. For instance, *Symb/Cabir* was one of the first SYMBIAN OS worms using Bluetooth to infect other devices. Additionally, when handsets were given Internet connectivity and the possibility to easily install third-party applications, more sophisticated infection strategies appeared. One early example was *Symb/Yxes*, which used the SMS channel and support from remote servers to propagate and configure itself.

The availability of mobile networking and pay-per-use services contributed to a rapid escalation of the malware phenomenon, both in featured phones and smartphones. Examples such as *Android/YZHCSMS.A* and *WinCE/Fakemini* send premium-rate SMSs without the user's knowledge, which results in very significant revenues for the owner of the registered number. Others such as *Android/Smspacem* have been also driven by economic incentives: sending spam through SMSs.

In recent years, the proliferation of smartphones with improved sensing and networking capabilities has translated into more sophisticated threats. For example, *Android/DroidKungFu* and *iPhone/FindAndCall* steal a variety of personal information stored in the device and exfiltrate it through the network to a remote server. Other pieces of malware such as *Android/Spybubble*, *Android/Nickispy* and *FinSpy Mobile*[2] have evolved into fully fledged spy instruments with the ability to monitor, record and exfiltrate the device's current location, ongoing and past phone calls and SMS logs to name a few. Although more illustrative examples are provided later on this section, readers interested in a more in-depth study are referred to the recent work of Zhou and Jiang [46], where a study of more than 1200 malware samples is presented.

It is plausible to believe that similar threats will soon affect other smart devices such as smart TVs or IMDs. For example, Auriemma [96] has recently shown that several versions of Samsung's Smart TV [6] are vulnerable to buffer-overflow attacks that could allow an attacker to remotely control the device. Many security vendors are already releasing security frameworks for smart TVs, including antimalware products [97]. The situation may become similar for medical devices too, particularly for those designed to remotely monitor a patient's condition and/or control body functions. We are not aware of any malware reported so far that affects existing IMDs or other medical smart devices, although researchers believe that malicious programs will certainly target them sooner or later [98], [99].

B. Malware Characterization

Current malware for PCs have evolved into complex and reuse-oriented pieces of software. Traditional classifications have focused on factors such as the propagation strategy (e.g., viruses vs. worms) or the malicious activity carried out (trojan horses, spyware, adware, rootkits, etc.), among others [100], [101], [33], [46]. However, these categories are rather imprecise and do not contribute to a better understanding in terms of detecting the presence of malware, particularly in current times where most malware present multiple and constantly changing features.

We next identify several criteria according to which malware in smart devices can be described and classified. Each provided criterion will be subsequently associated with some observable behavior in one or more features of the device. Thus, our classification will serve both to better understand the functionality of malware, but also to point out where to look for detecting malicious activities. We believe this can be of help to improve upon current detection strategies.

We classify malware for smart devices in terms of the following three features (a graphical summary is provided in Figure 3):

- **Attack goals and behavior**: Identifying malware's motivation on smart devices is paramount to have a better understanding of its behavior and can be used to develop targetted detection strategies. Such goals range from fraud and service misuse driven by economic incentives, to spamming, espionage, data theft and sabotage.
- **Distribution and Infection**: Malware creators can use a variety of techniques to distribute malicious applications and infect devices, from self-propagation mechanisms based on vulnerabilities and misconfigurations, to simply tricking the user into installing it by means of social-engineering techniques.
- **Privilege acquisition**: Once the malicious code is installed on the device, it often needs to acquire enough privileges to carry out its goals. This is automatic in many cases, as the user might already have granted them to the app, whereas in other cases technical vulnerabilities and/or misconfigurations are exploited.

In the remaining of this section we describe each criterion in detail and discuss some illustrative examples.

C. Attack Goals and Behavior

Felt et al. [33] analyze the main incentives behind IOS, ANDROID OS, and SYMBIAN OS malware using a dataset containing 46 specimens found between 2009 and 2011. According to their analysis, the most common malicious activities are related to the exfiltration of personal information and user credentials (44%), followed by premium-rate SMSs (33%) and, to a lesser extent, research, novelty, or amusement purposes. It is also pointed out that the majority of the analyzed pieces exhibited behaviors related to more than one incentive, and that they often incorporate secondary goals such as SMS advertisement, spamming, search engine optimization and, in a few cases, ransom. About the 33% of the studied malware changed their behavior based on commands received from a Command and Control (C&C) server.

More recently, new pieces of malware such as *Android/NotCompatible* [102] are demonstrating that attackers' interests are not only limited to the scope of a smartphone and its user, but to large private networks. By turning an infected device into a TCP relay/proxy –capable of forwarding network traffic–, smartphones can be used to support many

[2]FinSpy is a surveillance component part of a commercial surveillance toolkit called *FinFisher*, designed to spy over a wide range of mobile platforms. The mobile version is capable to monitor apps, emails, text messages, etc. on Android, iOS, BackBerry, Symbian, etc.

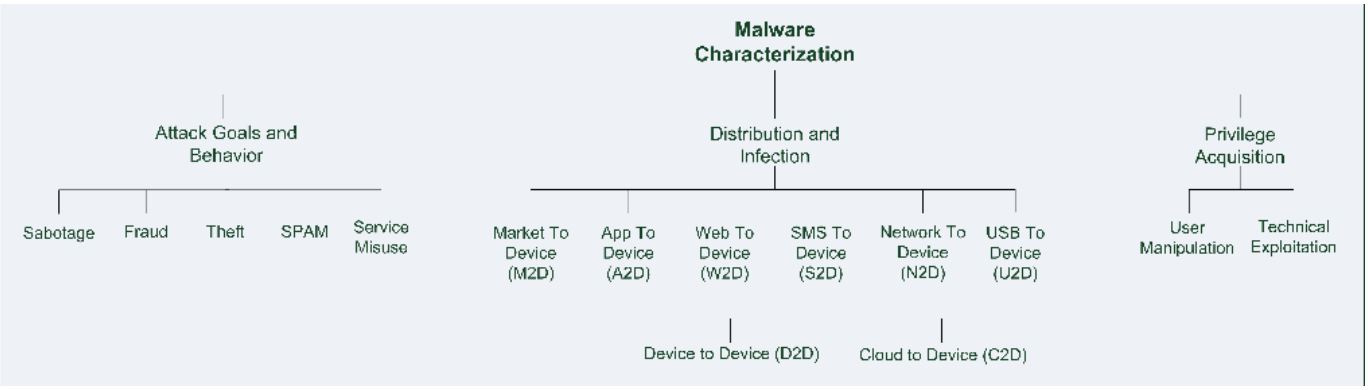

Fig. 3. Malware characterization for smart devices.

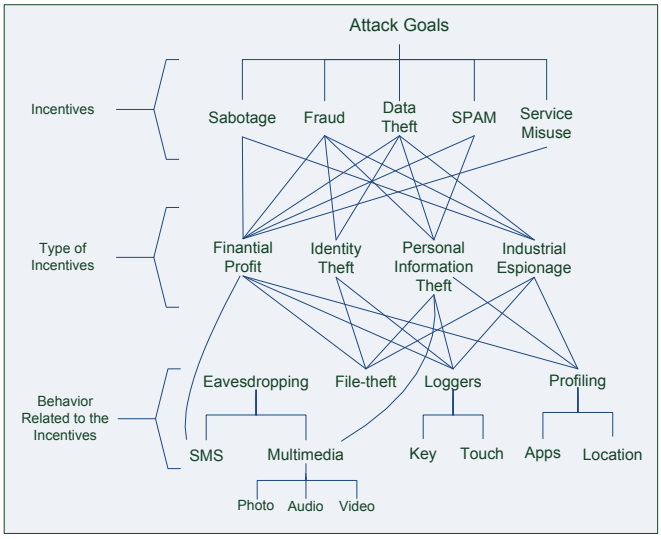

Fig. 4. Main attack goals, associated incentives, and exhibited behavior for malware in smart devices.

infection vectors. For instance, an attacker could establish an encrypted point-to-point session via HTTP with a device located behind the firewall. Using such tunnel, the attacker might be able to probe the private network and run exploits against assets within the corporation. Thus, malware such as *Android/NotCompatible* opens new opportunities for penetrating corporate networks.

Understanding the motivations behind malware can lead to a better identification of its behavior. Figure 4 presents the relation between most common incentives and the behavior associated with them. Common behaviors can be classified in *monitoring* (eavesdropping, profiling, etc.), *service misuse* (SMS, call, email, other services used for spamming, etc.), *sabotage* (draining the battery, deleting critical files, etc.), *data exfiltration*, and *fraud*. Note that some behaviors could affect two or more categories. For example, the unauthorized use of SMSs for spamming might well be both a service misuse and a fraud.

1) Example: Smartphone-based Botnets: A botnet is a collection of compromised devices that can be remotely controlled by an attacker (i.e., the bot master). As the number of smartphones is rapidly approaching the number of PCs, botnets for such platforms have gained momentum using a variety of distribution strategies to harvest as many devices as possible.

Traynor et al. [103] were among the first to study the potential theoretical impact of mobile-phone botnets in cellular networks. As far as we are aware, the first mobile botnet –named *SymbOS/Yxes*– appeared in 2009 and targetted SYMBIAN OS platforms, using a rudimentary HTTP-based command and control (C&C) channel. *iPhone/Ikee* appeared later on that same year, infecting around 21000 IPHONEs within two weeks. One remarkable feature of *Ikee* was that it showed how easy it can be to hijack a smartphone platform when root exploits are available. Specifically, it exploited IPHONEs that were left with the SSH port open and a default password after having been jailbroken. Such simple but very effective attack vectors can enable an attacker to control thousands of devices through an easy-to-implement C&C mechanism, as *Ikee.B* did [104].

C&C resilience is essential for a botnet to survive. In this regard, smartphones are very attractive devices, as they offer multiple communication alternatives that can be leveraged to implement a C&C channel, including rather non-standard means such as SMSs [105]. Nulliner et al. implemented and evaluated an IPHONE-based mobile botnet named *iBot* and demonstrated that thwarting them is more challenging than in computer networks, in particular because of employing multiple C&C channels (HTTP, SMS, etc.) in a peer-to-peer (P2P) fashion.

Android/Andbot [106] introduced a new energy-aware C&C strategy named URL Flux for ANDROID OS botnets. *Android/Andbot* uses URL Flux to eliminate the single point of failure problem present in *Ikee.B* and also reduces the SMS fees incurred by *iBot*. URL Flux is a domain name conversion used by *Confiker* –a Windows worm that infected millions of computers between 2009 and 2011– based on a domain generation algorithm seeded with a public key. Recently, more advanced IOS rootkit-like malware such as iSAM [107] integrates multi-functional tools also capable of self-propagating to other IPHONE devices in ways similar to *Ikee*'s.

Obfuscation is becoming popular in botnets, both by encrypting communications exchanged over the C&C channel and also local resources that might facilitate detection through

static analysis, such as server names and URLs, keywords, file names, etc. *AnserverBot* makes extensive use of some of these techniques, and also relies on posts made on public blogs to retrieve code updates and communicate with other members of the botnet.

2) Example: Grayware: The so-called grayware apps gather potentially sensitive user and/or device information, sometimes without user knowledge, and use it for dubious purposes or in contexts that the user might well not approve. For example, *Aurora Feint* is an app that sends the whole address book to an unknown destination and was quickly delisted from Apple's market in July 2008. Similarly, the author of *Storm8* –a popular game– was sued for collecting users' phone numbers, and *Twitter* has been widely criticized for sending the phone's contact list without informing the user.

Most grayware apps claim to retrieve such information for legitimate purposes and that it is crucial to improve the quality of the service offered to users. This, however, has recently become a major privacy threat for users' privacy, as apps collect excessive amounts of personal information and it remains unclear whether the service provider will use that data for legitimate purposes or not. Some platform manufacturers are increasingly deploying measures to prevent this. For example, in IPHONE a strict control is carried out to guarantee that personal information is not sent to the cloud unless really needed.

D. Distribution and Infection Strategies

Malicious programs employ a number of distinctive techniques to distribute themselves. We next discuss the most relevant and propose a taxonomy to classify them according to the channel used to enter the device. Distribution techniques are primarily influenced by malware in desktop computers, although the emergence of app markets have opened new possibilities. Two main approches exist: (i) self-propagation and (ii) social engineering. A self-propagating piece of malware can use different strategies to automatically install the payload into a device, whereas social engineering-based distribution strategies exploit the securiy unawareness of users to trick them into manually installing the application (e.g., *Andr/Opfake-C* by Sophos [108], which spreads via Facebook and, once installed, allows the attacker to perform premium-rate calls).

We have identified six different distribution vectors that can be used to infect devices:

- **Market to Device (M2D)**: This propagation strategy is based on market-borne attacks. An attacker uploads a malicious application to a market, sometimes using a stolen identity. Users can only get infected if markets accept such malicious apps and users install them. Open markets, in particular those performing little or no security revisions, are particularly vulnerable to this distribution method. For instance, malware using devious exploits (e.g.: *Android/DroidKungFu*[3]), might compromise the device by these means.
- **Application to Device (A2D)**: This propagation strategy is based on application-borne attacks. An attacker might rely on a specific, vulnerable application to spread itself. For instance, instances such as *Andr/Opfake-C* can use Facebook to post links with a copy of the malicious code. The main difference with M2D is that attackers assume the presence of other installed applications (presumably "goodware") to achieve infection. In this regard, even walled-garden models can be vulnerable to this type of infection vector.
- **Web-browser to Device (W2D)**: W2D uses web-borne attacks to propagate the malware in way similar to A2D. In this regard, we can consider W2D an specific type of A2D. The difference is that A2D strategies are limited by the possibilities offered by the application, whereas in W2D malware can exploit general drive-by-download strategies. This attack vector has recently gained popularity due the widespread use of vulnerable multi-platform components such as WebView [110].
- **SMS to Device (S2D)**: This strategy is used by malware that propagates via SMS or MMS or attacks that distribute a malicious payload by these means.
- **Network to Device (N2D)**: This propagation strategy is based on exploiting vulnerabilities or misconfigurations in the device. We distinguish between:
  - **Device to Device (D2D)**: When distribution is driven by another device in a P2P-fashion, and
  - **Cloud to Device (C2D)**: When distribution is done by a powerful computer such as a workstation or a server.
- **USB to Device (U2D)**: This strategy is used by malware that enters the device through a port (typically a cable) when connected to an infected PC.

1) Example: Repackaging: One of the most common distribution strategy for smartphone malware consists of repackaging popular applications and distributing them through alternative markets (M2D) with additional malicious code attached. Repackaging is not a phenomenon exclusive of the current generation of smartphones, although the proliferation of these platforms and the impressive growth in available apps have certainly contributed to make it a popular infection strategy. As far as we know, M2D repackaging started with SYMBIAN OS trojans such as *SymbOS/Skuller* and *SymbOS/Dampig*, which replaced system applications and antivirus files with modified ones. The focus has recently shifted towards ANDROID OS apps, particularly by repackaging popular games and tools [111], including banking apps. For example, *Android/FakeToken* trojan implements a man-in-the middle attack to forward SMS messages with mTANs (Mobile Transaction Numbers).

Zhou et al. present in [55] a systematic study of six popular third-party marketplaces for ANDROID OS. Their report concludes that between 5% and 13% of all available apps online are malware using repackaging, and the most common incentive is fraud in the form of replaced in-application advertisements to re-route revenues. The study also identifies a few cases with planted backdoors and other malicious payloads.

[3] *Android/DroidKungFu* uses an exploit called 'Rage Against The Cage" [109] for privilege escalation

2) Example: Malicious Code Transference via Network: In some cases, malware creators do not repackage an app with the full malicious code. Instead, the modified app only encloses a short piece of code that downloads and install the malicious payload once the app is installed on the device. One example of this variant –sometimes known as update attacks [46]– is *Android/DroidKungFuUpdate*. Remarkably enough, repackaged apps can enter the device without the user being aware of it. By exploiting some technical vulnerabilities and misconfigurations, some malware samples have even been able to replace another installed app by a repackaged version of the same one.

Repackaged apps often rely on obfuscation techniques to avoid detection and to make static analysis harder [112]. For example, in the case of update attacks the transferred payload is often encrypted. In other cases, encryption is applied to malicious components that are distributed together with the repackaged app, usually as if they were class files, images or other raw resources. For instance, *Android/RootSmart* and *Android/Fjcon* use AES to hide domain names and URLs; *Android/Geinimi* conceals URLs by encrypting them with DES; and *Android/OpFake* simply makes an XOR with a predefined key.

E. Privilege Acquisition

Exploitation strategies comprise a variety of techniques used by malware to gain the privileges required to achieve its goals. We distinguish two broad classes:

- **User Manipulation**: In many cases, privileges are directly granted by users who are not aware of the potential repercussions of doing so. These strategies, which rarely involve any technical sophistication, can be surprisingly effective and very damaging. Common forms of user manipulation include:
  - Social engineering.
  - Malware and/or grayware installed by novice users who do not understand –or do not pay atention to– the permission model.
  - Repackaged applications found in alternative markets.

  As in other similar security problems in computing, these methods can be prevented by raising awareness about the dangers of malicious apps.
- **Technical Exploitation**: In other cases the malicious app can escalate by exploiting technical vulnerabilities or misconfigurations of the platform. Even though the particular technical means greatly depend on each platform, the most common current attacks include [68], [61]:
  - API vulnerabilities.
  - Buffer overflows.
  - Code injection attacks.
  - ICC vulnerabilities.
  - Return-oriented Programming (ROP) and ROP without return flaws.
  - System vulnerabilities.
  - Netwoking protocol flaws.
  - Bootloader vulnerabilities.
  - Rooted device-based vulnerabilities.

1) Example: Rootkits: Current smartphone platforms are becoming increasingly complex, including not only the operating system itself but also dozens of libraries that give support to the services offered by the device. Kernel-level rootkits similar to those known for traditional PCs have recently appeared with identical purposes, namely to hide the existence of malicious software from the operating system. Most rootkits infect devices via N2D vectors, but app markets –official or not– are increasingly playing a key role. For example, it is pointed out in [46] that repackaged apps that implement technical exploits to gain root access once installed in the device do exist. Such exploits are often distributed with the repackaged app or acquired from a remote server as they become available. Contrarily, other exploits involve user manipulation to acquire privilege escalation. For example, *iPhone/Mobileconfigs* [113] allows an attacker to remotely hijack the device by installing malicious system-level settings into the device through social engineering.

Root exploits in iPHONE are often quickly patched by Apple and it is difficult to find malware samples exploiting these vulnerabilities [114]. The first exploit known for iOS was identified as early as 2007 and exploited a buffer overflow in the *libtiff* library. Other known exploits affected the SMS service –*SMS fuzzing*, presented at Black Hat USA 2009 by Miller and Mulliner– and PDF-related functionalitites –as the one used by *iPhone/JailbreakMe* to root iOS 4.3.3 and earlier versions via a web browser. Later in 2011, Miller submitted *iPhone/InstaStock* [115], which, after being approved, disclosed a hidden payload endowing *InstaStock* with remotely controlled root capabilities.

Hypervisors are a common strategy to counteract rootkits. Although there are some approaches to incorporate them on smartphones, such architectures are heavyweight and not widely available yet. Bickford et al. [116] implemented three proof-of-concept rootkits for Android. Firstly, they rootkit the GSM Linux Kernel Module (LKM) in a way that a remote attacker can listen to the victim's conversations. Secondly, they rootkit the GPS LKM so that the attacker compromises the victim's location privacy. And thirdly, they exploit a number of power-intense services so that the battery is drained in two hours. They conclude that there is currently no effective nor efficient technique to detect infection by rootkits.

F. Discussion

Table II shows a representative set of smarphone malware and provides, for each one of them, sought attack goals and the distribution and privilege acquisition strategies implemented. Various conclusions can be drawn:

- M2D strategies clearly dominate other distribution and infection strategies. This conforms the study conducted in [46] over 1200 samples of ANDROID OS malware, which points out that 86% of them use repackaging techniques.
- Privileges are mostly acquired by simple user manipulation, i.e., by simply asking the user to grant them to the app. This is certainly worrysome and motivates many recent works dealing with enhanced permission models and novel ways of communicating requested privileges to users. Even though repackaging is nowadays the primary

TABLE II
SAMPLES OF SMARTPHONE MALWARE FOR THE MAIN OS AND THEIR MOST RELEVANT CHARACTERISTICS.

	Attack Goals					Distribution / Infection						P.A.	
Charact. / App	Theft	Misuse	Sabotage	SPAM	Fraud	M2D	A2D	W2D	N2D	U2D	S2D	User	Exploit
FinSpy Mobile	●	□	□	–	–	–	●	●	●	●	●	●	●
Symb/Cabir	◊	◊	◊	◊	◊	–	–	–	●	–	–	●	–
Symb/Skuller	□	□	●	□	□	●	–	–	–	–	–	●	–
Symb/Yxes	●	–	●	–	–	●	–	–	–		●	●	–
Sym/ZeusMitmo	●	□	□	□	□	●	–	–	–	–	–	●	●
BB/FlexiSpy	●	–	–	–	–	●	–	–	–	–	–	●	–
BB/BBproxy	–	●	–	–	–	●	–	–	–	–	–	●	–
BB/ZeusMitmo	●	□	□	□	□	●	–	–	–	–	–	●	●
And/YZHCSMS	●	–	–	–	●	●	–	–	–	–	–	●	–
And/SpyBubble	●	–	–	–	–	●	–	–	–	–	–	●	–
And/SimChecker	●	–	–	–	–	●	–	–	–	–	–	●	–
And/BaseBridge	●	–	–	–	–	●	–	–	–	–	–	●	–
And/GinMaster	●	–	–	–	–	●	–	–	–	–	–	●	–
And/DroidKungFu	●	–	–	–	–	●	–	–	–	–	–	●	–
And/AutoSPSubs	–	–	–	–	●	●	–	–	–	–	–	●	–
And/Nickispy	●	–	–	–	–	●	–	–	–	–	–	●	–
And/Smspacem	–	●	–	●	–	●	–	–	–	–	–	●	–
And/Crusewind	●	–	–	–	–	●	–	–	–	–	–	●	–
And/Zsone	–	●	–	–	–	●	–	–	–	–	–	●	–
And/GGTracker	●	●	–	●	–	●	–	–	–	–	–	●	–
And/AdSMS	●	●	–	–	–	–	–	●	–	–	–	–	●
And/Fakeplayer	–	●	–	–	–	●	–	–	–	–	–	●	–
And/Bgserv	●	–	–	–	–	●	–	–	–	–	–	●	–
And/Lightdd	●	–	–	–	–	●	–	–	–	–	–	●	–
And/Rootcager	●	–	–	–	–	●	–	–	–	–	–	●	●
And/Opfake	–	●	–	–	–	●	●	–	–	–	–	●	–
And/OneClickFraud	–	–	–	–	●	●	–	–	–	–	–	●	–
And/FakeToken	–	–	–	–	●	●	–	–	–	–	–	●	–
iP/MogoRoad	–	–	–	–	●	–	–	●	–	–	–	–	●
iP/JailbreakMe	–	◊	–	–	–	–	–	●	–	–	–	–	●
iP/InstaStock	◊	◊	◊	◊	◊	●	–	–	–	–	–	–	●
iP/FindAndCall	●	–	–	●	–	●	–	–	–	–	–	●	–
iP/Mobileconfigs	□	□	□	□	□	–	–	●	–	●	–	●	–
iPJ/iKee.A	◊	◊	◊	◊	◊	–	–	–	●	–	–	–	●
iPJ/iKee.B	□	□	□	□	□	–	–	–	●	–	–	–	●
iPJ/Dutch 5	–	–	–	–	●	–	–	–	●	–	–	–	●
iPJ/Privacy.A	●	–	–	–	–	–	–	–	●	–	–	–	●
WinCE/Duts.A	◊	◊	◊	◊	◊	●	–	–	–	–	–	●	–
WinCE/Fakemini	–	●	–	–	–	●	–	–	–	–	–	●	–
WinCE/Pmcryptic	–	●	–	–	–	–	–	–	–	●	–	●	–
WinCE/Terred	–	●	–	–	–	●	–	–	–	–	–	●	–
WinCE/ZeusMit.	●	□	□	□	□	●	–	–	–	–	–	●	●

Legend:

Symb: Symbian *iPJ*: Jailbroken iPhone *iP*: iPhone
And: Android *WinCE*: Windows Mobile *BB*: BlackBerry

●: The referred characteristics are applied to the application.
◊: Proof-of-concept for demonstration, novelty or amusement purposes.
□: Multi-purpose malware having multiple goals.

entry point for malware, it is pointed out in [46] that 36.7% of studied specimes attempt to leverage technical exploits to obtain root privileges.

- In terms of behavior, malware with just one goal is rare. Most samples spy on users and steal personal data, but also attempt to commit fraud or misuse services. A possible explanation for this is the reconfigurable nature of most malware specimens through updates, as in the case of botnets. Thus, attackers basically seek to plant a basic bot engine in the device, and then to provide it with instructions and further code to perform specific tasks. Again, this conforms similar studies carried out recently. For example, in [46] it is pointed out that 90% of the samples turn the compromised device into a bot; almost half of them (45.3%) try to misuse SMS or call services to obtain financial profit; and 51.1% harvest user information. Finally, sabotage is quite unusual, with only a few examples that drain the device's battery or remove selected files.
- There are remarkable differences between ANDROID OS and IPHONE malware in the three criteria of our taxonomy
  - First, most ANDROID OS malware is distributed by markets, notably in the form of repackaged applications. IPHONE barely suffers from such infection vectors, and the majority of malware enters via web and network exploits. In part, this is a consequence of the walled-garden model of Apple's market.
  - The differences in their respective permission models and the way of granting privileges also show up: while a significant fraction of ANDROID OS malware is entitled with sufficient privileges by the user –even if it later escalates by other means–, in IPHONE most specimens depend on technical exploits.
  - Finally, in contrast with ANDROID OS malware, most IPHONE specimens discovered so far have been created for demonstration or amusement purposes.

  A word of caution is appropriate, though: because of its openness, ANDROID OS is the *de facto* platform-of-choice for security research in smartphones, which may have also negatively contributed to the malware phenomenon; and, furthermore, Apple follows a less communicative strategy about IPHONE malware.

IV. MALWARE DETECTION AND ANALYSIS

As detailed in the previous section, current malware pose severe threats to security models in smart devices. In this section we classify and describe the most significant advances in malware detection systems for such devices [117]. More precisely, we show how such systems build their foundations based on a variety of detection techniques. These techniques aim at identifying where and how malware manifests by constantly monitoring various device-based features. We also show how detection systems are driven by these features, as they represent the key elements for malware identification. We believe that this comprehensive study is paramount for researchers and practitioners in order to facilitate the construction of new detection systems.

A. A Taxonomy of Detection Techniques

Malware detection is a complex process pulling together monitoring, analysis and identification tasks. In order to organize and better understand current detection systems, we next propose a taxonomy based on the following seven characteristics (see Figure 5 for a graphical summary):

- **Type of Detection (ToD)** There are two common types of malware detection techniques according to how code is analyzed:
  - *Static analysis*: this type of technique attempts to identify malicious code by unpacking and disassembling (or decompiling) the application. This technique is a relatively fast approach and it has been widely used in preliminary analysis to search for suspicious strings or blocks of code.
  - *Dynamic analysis* techniques seek to identify malicious behaviors after deploying and executing the application on an emulator or a controlled device. These techniques require some human or automated interaction with the app, as malicious behavior is sometimes triggered only after certain events occur.

  Static analysis techniques are well known in traditional malware detection and have recently gained popularity as efficient mechanisms for market protection [118]. As a major drawback, these techniques fail to identify malicious behavior when it is obfuscated or distributed separately from the app. Contrarily, dynamic analysis are arguably more powerful in these cases. In fact, the only way of learning what the app is really doing necessarily requires to run the code and observe its actions. However, the inputs generated by most dynamic analysis tools are generally produced by using random streams of user events, which might not trigger the execution of the malicious payload, resulting in malicious apps that avoid being detected. This particular shortcoming can be tackled by modelling users' behavior and providing human-like inputs. Dynamic analysis can be used both in the cloud for market protection or directly in the device, although resource consumption is certainly a issue (see later discussion on this).
- **Type of Monitoring (ToM)** Malware can be detected by analyzing various features that serve to tell apart benign from malicious activities. A monitoring system can collect *user-level*, *kernel-level*, or *hypervisor-level* activity, depending on the type of features that will be extracted. Monitoring approaches include the collection of: (i) system calls (SYS); (ii) network activity (NET); (iii) event logs (EL); (iv) user activity; (v) instructions (I); (vi) permissions (P); or (vii) program traces (PT); to name a few. Each type of monitoring activity requires the deployment of different instruments to intercept and format the corresponding events. For instance, SYS requires the use of a system trap technique with root privileges, while NET requires capturing all packets from the network interface. Additionally, monitoring any of these features when the app is run in an hypervisor requires the introspection of a virtual environment.
  Monitoring can be potentially expensive in terms of

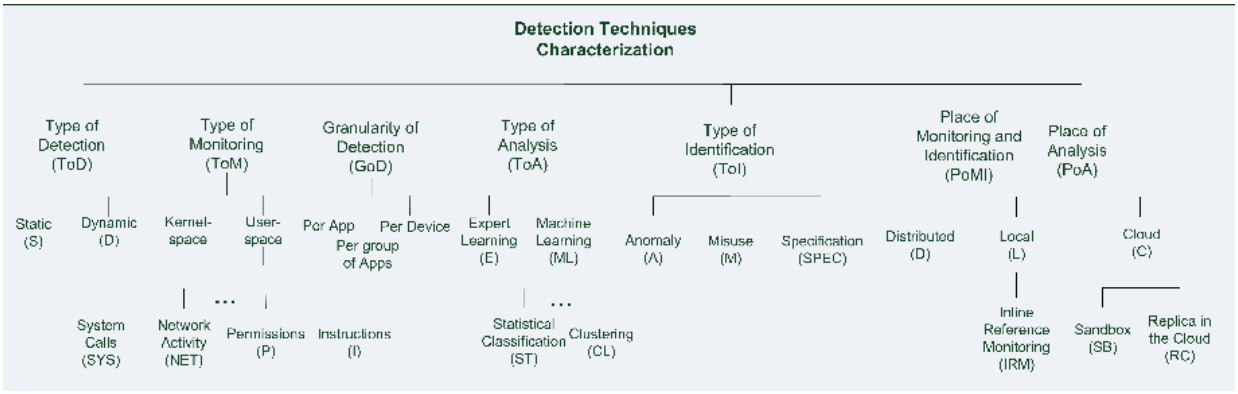

Fig. 5. Taxonomy of malware detection techniques for smart devices.

resource consumption, particularly if a large number of events is collected directly over the platform being monitored. As far as we are aware, no power consumption analysis has been carried out yet, but practical experience suggests that intensive monitoring is prohibitive for current smart devices.

- **Granularity of Detection (GoD)** A point related to the ToM discussed above is how collected data is filtered in order to select the detection scope. Monitoring can be carried out at different levels:
  - *Per App*: features related to a specific application are monitored and analyzed independently from other apps in the system. This type of feature classification presents good performance when malware is a standalone application.
  - *Per group of apps*: in this case, data from a collection of applications is gathered and analyzed. This is potentially useful when malware's goals are achieved in a distributed way by several collaborating apps.
  - *Per device*: detecting certain types of malware, such as for example rootkits, requires a more general detection approach focused on monitoring the device itself rather than particular apps executed on it.
- **Type of Analysis (ToA)** The monitored information is subsequently analyzed to extract evidence on the presence of malware. Such analysis can be carried out by a human expert (E), although this possibility is becoming increasingly unaffordable, at least without the support of automated analysis tools. There are several types of techniques for analyzing data obtained after monitoring, including: Clustering (CL), Support Vector Machines (SVM), Self-Organizing Maps (SOM), other general Machine Learning (ML) algorithms, Control Flow Graphs (CFG), Data Flow Graphs (DFG), Program Dependency Graphs (PDG), etc.
- **Type of Identification (ToI)** Depending on the type of identification carried out, detection systems can be classified as either *anomaly*-based (A), *misuse*-based (M), or *specification*-based (SPEC) system. This feature refers to the principle guiding the identification of malicious activities and follows the same ideas explored in Intrusion Detection Systems [119], [120].
  - *Anomaly-based* identification attempt to model the "normal" behavior of the monitored system, classifying as anomalous any other behavior reported. Anomaly detection techniques have the potential to detect previously unseen malware. However, they generally present a high rate of false positives, i.e., they are prone to detect rare legitimate behaviors as malicious.
  - *Misuse-based* identification –also known as signature-based– aims at identifying known malicious activity by means of predefined patterns of signatures. Thus, only "malicious" behaviors are modeled here. The main benefit of misuse detection lies in its accuracy detecting well-known attacks. Generally, for each know malicious behavior, misuse systems are equipped with one or more signatures. In this regard, maintaining an up-to-date database with a massive amount of signatures poses a major challenge. Furthermore, resource-constrained devices are not capable of processing big amount of signatures.
  - *Specification-based* identification works on the basis of predefined authorized behaviors (specifications) and assumes that any activity deviating from them violates the system policy and, therefore, is malicious.
- **Place of Monitoring and Identification (PoMI)** Monitoring, analysis, and identification techniques are generally resource-intensive tasks that cannot be afforded in battery-constrained devices. As a consequentce, in recent years it has been proposed to extenalize many of such tasks to more powerful platforms, even though some processing still needs to be taking place in the device. We distinguish three main classes of detection schemes according to where monitoring and identification takes place:
  - *In the device*: both monitoring and identification are placed locally in the device. This requires very lightweight approaches and their scope may be quite limited. There are two types of local monitoring or identification techniques according to where the monitoring is taking place:
    - Local out-line (L): this type of technique aims at monitoring the device by installing itself in one of

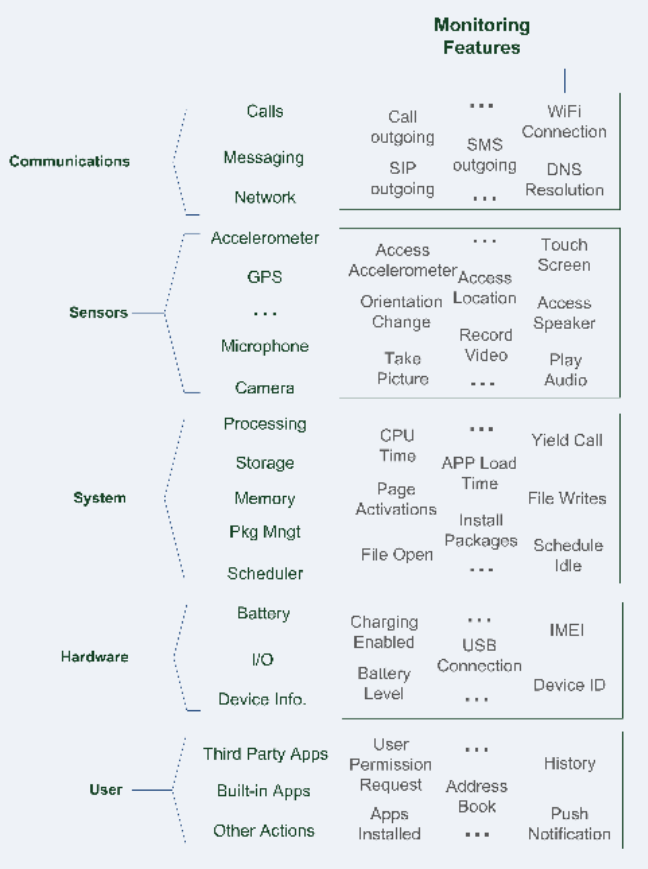

Fig. 6. Taxonomy of monitorable features for smart devices.

the lower layers of the device's architecture, and generally require root privileges.

- ∗ Local in-line, also known as *Inline Reference Monitor (IRM)*: this type of technique rewrites untrusted applications so that the monitoring code is embedded into the app, and does not require root privileges.

– *Distributed (D)* among other devices. Performs any monitoring, analysis or identification task in a cooperative way among different trusted devices.

– *In the cloud (C)*. Uses virtual environments for running several devices on a single server machine without reducing the battery life.

- ∗ Sandbox (SB): uses a tightly controlled set of resources for running dynamic analysis over target apps.
- ∗ Replica in the cloud (RC): uses remote security servers for hosting exact replicas of the device. Monitoring and identification techniques that are placed on the replicas require complex synchronization systems to ensure that the replica is at all times identical to the actual device, as well as collaboration with the service provider (e.g., the internet provider for general purpose devices or phone provider for smartphones).

– **Place of Analysis (PoA)** Finally, depending on where the analysis component is placed –i.e., locally or in the cloud– the approach used poses different challenges. On the one hand, *cloud-based* approaches require local preprocessing of the monitored traces, transmitting them to the cloud, and waiting for the results. Finally, results may be included for further identification of malware. On the other hand, *local* approaches might accelerate the delay in obtaining the response, especially when traces are too big and/or the connection is very slow.

TABLE III
MONITORABLE HARDWARE FEATURES AND EXAMPLES OF ATTACKS THAT COULD AFFECT THEM.

Features	Attacks	Botnet	DoS & DDoS	SMS–of–death	Phising	Pharming	Monitoring	Misuse SMS service	Malicious QRCode
Battery	Charging_Enabled	•	–	–	–	–	–	–	–
	Battery_Voltage	•	•	–	–	–	–	–	–
	Battery_Current	•	•	–	–	–	–	–	–
	Battery_Temp	•	•	–	–	–	–	–	–
	Battery_Level_Change	•	•	–	–	–	–	–	–
I/O	LED	–	–	–	–	–	–	–	–
	USB_Connection	–	–	–	–	–	–	–	–
	Coverage_Range	–	–	–	–	–	–	–	–
	Press_Key	–	–	–	–	–	•	•	–
Device Info.	IMEI	–	–	–	–	–	•	–	–
	Device_Id	–	–	–	–	–	•	–	–
	SIM_Card	–	–	–	–	–	•	–	–
	Phone_State	–	–	–	–	–	•	–	–
	UID_Access	–	–	–	–	–	•	–	–
	UID_Removal	–	–	–	–	–	•	–	–

B. Monitorable Features in Smart Devices

According to the monitoring approaches discussed above, we next identify and classify a number of device-based features that can provide evidence of malware activities. We subsequently explore how the behavior of some representative classes of malicious activities manifest in subsets of these features. A summary of this taxonomy –excluding the full list of features for each class– is given in Figure 6.

– **Hardware**: this kind of features identify the state of the hardware (HW) components of the device. We group HW features in three subclasses: (i) *battery*, (ii) *input/output HW*, and (iii) *device info*. Table III provides a detailed list of features for each subclass. The state of the battery or the access to the unique device identifier can be used to detect a specific type of malware. For instance, some botnets check first that the battery is charging before performing heavy operations. Another example of the use of HW-based features for malicious purposes is access to the IMEI of a smartphone with the goal of exfiltrating it.

– **Communications**: communications represent an essential infection vector in smartphones. They include the following features: (i) phone and internet *calls*, (ii) phone and internet *messaging*, and (iii) *network* usage (data other than calls and messaging), as identified in Table IV.

- **Sensors**: on-platform sensors allow the device to interpret the physical context of a user [121]. Currently the most common sensors are: (i) *accelerometer*, (ii) *GPS*, (iii) *compass*, (vi) *gyroscope*, (v) *microphone*, (vi) *touch*

TABLE IV
MONITORABLE COMMUNICATIONS FEATURES AND EXAMPLES OF ATTACKS THAT COULD AFFECT THEM.

Features		Attacks	Botnet	DoS & DDoS	SMS–of–death	Phising	Pharming	Monitoring	Misuse SMS service	Malicious QRCode
Calls	Phone	Phone_Outgoing	–	●	–	–	–	●	–	–
		Phone_Incoming	–	●	–	–	–	●	–	–
		Phone_Missed	–	●	–	–	–	●	–	–
		Phone_Privileged	–	●	–	–	–	●	–	–
	Internet	SIP_Incoming	–	●	–	–	–	–	–	–
		SIP_Outgoing	–	●	–	–	–	●	–	–
Msg.	Phone	SMS_Incoming	●	●	●	–	–	●	–	–
		SMS_Outgoing	●	●	–	–	–	●	–	–
		SMS_Read	●	●	–	●	–	●	–	–
		SMS_Privileged	–	●	–	–	–	●	●	–
		MMS_Incoming	●	●	●	–	–	●	–	–
		MMS_Outgoing	●	●	–	–	–	●	–	–
		MMS_Read	●	●	–	–	–	●	–	–
		MMS_Privilege	–	●	–	–	–	●	●	–
	Internet	XMPP_Incoming	●	–	–	–	–	●	–	–
		XMPP_Outgoing	●	●	–	–	–	●	–	–
Net.	Byte	WiFI_TX_Bytes	●	–	–	–	●	●	–	–
		Phone_TX_Bytes	●	●	–	–	–	●	–	–
		Bluetooth_TX_Bytes	●	●	–	–	–	●	–	–
		WiFI_RX_Bytes	●	●	–	–	–	●	–	–
		Phone_RX_Bytes	●	●	–	–	–	●	–	–
		Bluetooth_RX_Bytes	●	●	–	–	–	●	–	–
	Packets	WiFI_TX_Pckts	●	●	–	–	●	●	–	–
		Phone_TX_Pckts	●	●	–	–	–	●	–	–
		Bluetooth_TX_Pckts	●	●	–	–	–	●	–	–
		WiFI_RX_Pckts	●	●	–	–	–	●	–	–
		Phone_RX_Pckts	●	●	–	–	–	●	–	–
		Bluetooth_RX_Pckts	●	●	–	–	–	●	–	–
	Connections	WiFI_CX	●	●	–	●	●	●	–	–
		Phone_CX	●	●	–	●	●	●	–	–
		Bluetooth_CX	●	●	–	●	●	●	–	–
		DNS_Resoluc.	●	●	–	●	●	●	–	–

TABLE V
MONITORABLE SENSORS FEATURES AND EXAMPLES OF ATTACKS THAT COULD AFFECT THEM.

Features	Attacks	Botnet	DoS & DDoS	SMS–of–death	Phising	Pharming	Monitoring	Misuse SMS service	Malicious QRCode
Accelerometer	Access_Accelerometer	–	–	–	–	–	●	–	–
	Current_Roll_Pitch_Yaw	–	–	–	–	–	●	–	–
	Orientation_Changing	–	–	–	–	–	●	–	–
GPS	Access_Location	–	–	–	–	–	●	–	–
	Current_Location	–	–	–	–	–	●	–	–
	Location_Changing	–	–	–	–	–	●	–	–
Compass	Access_Compass	–	–	–	–	–	●	–	–
	Current_Cardinal_Orientation	–	–	–	–	–	●	–	–
	Cardinal_Orientation_Changing	–	–	–	–	–	●	–	–
Gyroscope	Access_Gyroscope	–	–	–	–	–	●	–	–
	Current_Angular_Moment	–	–	–	–	–	●	–	–
	Angular_Moment_Changing	–	–	–	–	–	●	–	–
Microphone	Record_Audio	–	–	–	–	–	●	–	–
	Access_Audio	–	–	–	–	–	●	–	–
Touch	Touch_Screen_Preasure	–	–	–	–	–	●	–	–
	Touch_Screen_Area	–	–	–	–	–	●	–	–
Speaker	Access_Speakers	–	–	–	–	–	●	–	–
	Play_Audio	–	–	–	–	–	●	–	–
Camera	Take_Picture	–	–	–	–	–	●	–	●
	Access_Picture	–	–	–	–	–	●	–	●
	Record_Video	–	–	–	–	–	●	–	–
	Access_Video	–	–	–	–	–	●	–	–
	Calculate_Depth (RGDB)	–	–	–	–	–	●	–	–

TABLE VI
MONITORABLE SYSTEM FEATURES AND EXAMPLES OF ATTACKS THAT COULD AFFECT THEM.

Features	Attacks	Botnet	DoS & DDoS	SMS–of–death	Phising	Pharming	Monitoring	Misuse SMS service	Malicious QRCode
Processing	CPU_Time	–	●	●	–	–	–	–	–
	Runnable_Entities	–	●	–	–	–	–	–	–
	Context_Switching	–	–	–	–	–	–	–	–
	Wakelocks	–	–	–	–	–	–	–	–
	Processes_Changing	–	●	–	–	–	–	–	–
Storage	File_Open	●	–	–	–	–	–	–	–
	File_Reads	●	–	–	–	–	–	–	–
	File_Writes	●	–	–	–	–	–	–	–
	File_Read_Bytes	●	–	–	–	–	–	–	–
	File_Write_Bytes	●	–	–	–	–	–	–	–
Memory	Dirty_Pages	–	–	–	–	–	–	–	–
	Active_Pages	–	–	–	–	–	–	–	–
	Anonymous_Pages	–	–	–	–	–	–	–	–
	Page_Activations	–	–	–	–	–	–	–	–
	Page_Desactivations	–	–	–	–	–	–	–	–
	Page_Faults	–	–	–	–	–	–	–	–
	DMA_Allocations	–	–	–	–	–	–	–	–
	Garbage_Collections	–	–	–	–	–	–	–	–
	Page_Frees	–	–	–	–	–	–	–	–
	Inactive_Pages	–	–	–	–	–	–	–	–
	File_Pages	–	–	–	–	–	–	–	–
	Mapped_Pages	–	–	–	–	–	–	–	–
	Writeback_Pages	–	–	–	–	–	–	–	–
Pkg Mgmt	App_Load_Time	●	–	–	–	–	–	–	–
	Install_Packages	●	–	–	–	–	–	–	–
	Delete_Packages	●	–	–	–	–	–	–	–
	Change_Package	●	–	–	–	–	–	–	–
	Restart_Package	●	–	–	–	–	–	–	–
	Master_Clear	●	–	–	–	–	–	–	–
Scheduler	Yield_Calls	–	–	–	–	–	–	–	–
	Schedule_Idle	–	–	–	–	–	–	–	–
	Running_Jiffies	–	–	–	–	–	–	–	–
	Waiting_Jiffies	–	–	–	–	–	–	–	–

sensors, (vii) *speakers*, and (viii) *camera*, as illustrated in Table V. Access to sensors can be monitored to identify malicious use. For instance, profiling malware will typically access the user's current location. Thus, if an application is constantly accessing the GPS and sending this information through the network, it could be an indication of malicious –or, at least, potentially dangerous– usage.

- **System**: access to system resources can be used to identify malicious behaviors by monitoring: (i) *processes*, (ii) *storage*, (iii) *memory*, (iv) *package management*, and (v) *scheduler*, as identified in Table VI.
- **User**: there are a number of features that generally involve user interaction and that could also provide evidence of malicious behavior. We identify (i) *user-permissions* frequency requests (applications can be classified into categories by monitoring the frequency at which they request permissions [122]), (ii) *third-party apps*, (iii) *built-in apps*, and (iv) *other actions*, as detailed in Table VII.

1) Discussion: Malicious apps –as any other app– rely on the device's system and sensors to achieve their goals. Different components of the device are therefore interrogated by the malware to operate. For instance, the behavior of botnets is deeply related to almost any kind of communication feature as all bots rely on a C&C back-end. Additionally, they could also require some system interactions in order to store and update themselves. However, they are not likely to access any sensor –unless the master commands it through a remotely transmitted payload. Another interesting example is given by fraud attacks such as *Phising* or *Pharming*. In these cases, the malware is likely to use network connections in order to get to the victim, access to SMS messages to steal, for example, One Time Passwords (OTPs), or change the DNS resolution of the device, but it will definitely not access sensors.

Accessing those components in a stealthy manner is still, to the best of our knowledge, a limitation for attackers.

TABLE VII
MONITORABLE USER FEATURES AND EXAMPLES OF ATTACKS THAT COULD AFFECT THEM.

Features	Attacks	Botnet	DoS & DDoS	SMS–of–death	Phising	Pharming	Monitoring	Misuse SMS service	Malicious QRCode
User–permissions	#_requests	•	•	•	•	•	•	•	•
Third Party Apps	Apps_Installed	•	–	–	–	–	–	–	–
	Apps_Usage	•	–	–	–	–	–	–	–
	Apps_Delete	•	–	–	–	–	–	–	–
Built–in Apps	Address_Book	–	–	–	–	–	•	–	–
	History	–	–	–	–	–	•	–	–
	Bookmarks	–	–	–	–	–	•	–	–
	Calendar	–	–	–	–	–	•	–	–
	Feeds	–	–	–	–	–	•	–	–
	Email	–	–	–	–	–	•	–	–
Other Actions	Push_Notifications	–	–	–	–	–	•	–	–
	Unlock	•	•	•	•	•	•	•	–

Nevertheless, there are some technical exploitation vectors that allow a malware to root the device, which could thwart detection at some levels. In those cases, access to hypervisor-level monitoring is paramount to identifying such cases.

Tables III through VII present various examples of malicious activities and the features that would likely allow a detection system to identify them. Several conclusions can be drawn:

- Monitoring can be a very heavy consuming task. Thus, identifying a monitoring strategy as well as an appropriate *type* of features is crucial to reduce workload and improve detection efficacy. For instance, if a user is interested in using his device in a Bring-Your-Own-Device (BYOD) context, avoiding exfiltration of sensitive information may be critical, and therefore monitoring only some specific features would be a good strategy.
- From all eight cases studied, the most relevant group of features affects communications (Table IV). In this regard, it is also interesting to identify adaptive monitoring strategies based on the appropriate *amount* of features. Thus, if a detection system can likely identify the most popular malware by only monitoring, say, 40% of the features, then monitoring the remaining ones can be eventually switched off, e.g., when the battery is lower than a given threshold.

Finally, we emphasize that the list of detection features presented in Tables III through VII are only an excerpt of all those that can be used by a detection system. In general, each type of device will offer a more or less exhaustive list of available features for each category given above.

C. Overview of Detection Systems

In the last few years several works have been proposed to detect malware on smart devices –mostly smartphones and, more specifically, for ANDROID OS platforms. We have classifed the 20 most representative detection systems according to the taxonomy provided above. The result, shown in Table VIII, summarizes current research directions.

Even though all detection systems are strongly interrelated, some general characteristics are evident. For example, while some techniques are more versatile and, therefore, are used more often, others are used mainly for certain detection systems. Thus, both static and dynamic analysis are used for both device and market protection. However, it is more frequent to use dynamic analysis for device-oriented detection and static analysis for market protection. Despite this, dynamic analysis is becoming an important technique for market detection as well, as new paradigms based on Security-as-a-Service, such as *Replicas in the Cloud*, are gaining popularity.

For the sake of organization, in the remaining of this section we describe current research proposals grouped into three main categories:

i) Device monitoring systems.
ii) Automatic app-review systems for market protection.
iii) Attack-specific malware identification systems (both for user and market protection).

D. Device-based Monitoring Systems

Device-based malware detection systems have received much attention lately. They mostly use dynamic analysis techniques, although some combine them with static analysis to improve the detection strategy. In this regard, both anomaly and misuse detectors are proposed.

1) Anomaly Detectors: Schmidt et al. [137] leverage both static and dynamic analysis for detecting malware in SYMBIAN OS and ANDROID OS devices. On the one hand, function calls are first extracted, and monitored data is then analyzed using decision trees. Classifiers are trained to recognize normal and malicious apps. On the other hand, an anomaly-based malware detection is used for dynamic analysis. Features such as free RAM memory, CPU usage, SMS count, etc. are monitored for further analyzing behavior. Analysis is done in the cloud using machine learning algorithms such as Artificial Immune Systems (AIS), Self-Organizing Maps (SOM), Support Vector Machines (SVM), and Tree Kernels.

A somewhat similar approach is Andromaly [127], which uses dynamic analysis for periodically monitoring a number of features and machine learning anomaly-based detectors for classifying apps as goodware or malware. In Andromaly, however, classification is done locally in the device. The scheme monitors various system features such as CPU consumption, number of network packages, number of running processes and battery level. Redundant features are first eliminated using three feature selection algorithms: Chi-Square, Fisher Score, and Information Gain. Furthermore, collected observations are classified using K-Means, Logistic Regression, Histograms, Decision Trees, Bayesian Networks and Naive Bayes. Evaluation was performed testing a small number of self-implemented malware samples, and results show a detection rate accuracy ranging from 44% to 100%. More precisely, they show that Fisher Score with 10 top features selected, and using Naive Bayes and Logistic Regression, perform better than the other classifiers. Although no real malware is studied, their experiments help to understand which machine learning algorithms are superior as well as their degradation. In fact, their experiments show a 10% of performance degradation in the worst scenario, i.e., 8 different classifiers with 30 features. However, it is not clear how this performance has been measured and whether the consumption

TABLE VIII
MALWARE DETECTION SYSTEMS.

LEGEND

Platform
And: Android
Win: Windows
Sym: Symbian

Type of Detection (ToD)
S: Static
D: Dynamic

Other
∅: Unavailable

Type of Monitoring (ToM)
SYS: System calls
NET: Network
EL: Event Log
I: Instructions
P: Permissions
PT: Program Traces
PCB: Process Control Block
API: API Calls
K: Kernel-level
U: User-level

Type of Analysis (ToA)
E: Expert
ML: Machine Learning
CL: Clustering
DG: Dependency Graphs
ST: Statistical
PRO: Probabilistic Models

Type of Identification (ToI)
A: Anomaly
M: Misuse
SPEC: Specification

Place of Monitoring and Identification (PoMI) and Place of Analysis (PoA)
L: Local Outline
IRM: Local Inline (IRM)
C: Cloud
DB: Distributed
HP: Honeypot
RC: Replica in the Cloud
SB: Sandbox
H: Hybrid

		Detection Approach									
	Plat.	ToD	ToM	ToA	ToI	PoMI	PoA	Consumption	Features	Attack	Observations
Dendroid (2013) [118]	And	S	I	ST, DG, CL	∅	C	C	Discussed: deals efficiently with large databases of malware instances	Code Chunks – a high-level representation of the CFG	Automatic classification of unknown malware samples	The classification is done using text mining and information retrieval techniques. Hierarchical Clustering is also used to extract evolutionary analysis
AppProfiler (2013) [123]	And	S, D	API, PT	E	M	L	L, C	Not available	Permissions, and API Calls	Privacy leakage	API calls are analyzed statically using signatures and apps are traced dynamically through tainting analysis
Apps Playground (2013) [124]	And	D	SYS, PT	∅	∅	C	C	Not applicable	Taint tracing, SYS call, etc.	Any kind	Heuristic-based UI interaction based on contextual exploration
Secloud (2013) [125]	And	*	*	*	*	RC	C	Device consumption not available	Any kind	Any kind	Detection techniques: AV scanning, file integrity checking, SYS call monitoring, or network intrusion detection and response
TStructDroid (2013) [126]	And	D	PCB	STAT, ML	A	L	L	Performance degradation of 3.73% on average	Frequencies of 99 preliminary parameters: page frames, context switches, page faults, virtual memory, etc.	Any kind	Type of analysis: theoretic analysis, time–series feature logging, segmentation and frequency component analysis of data, and machine learning classifier
Andromaly (2012) [127]	And	D	*	ML	A	L	L	$16,78Kb$ ±32 RAM (≈ 8.8%), 5.52% ±2.11 CPU, and 10% Battery (unclear)	Detection Method: monitorization of features. Feature selection: Subset of selected features from 88 initial categories	Any kind of anomaly	Training Method: Classification with labelled data. Experimental evaluation
AppGuard (2012) [128]	And	D	PT	∅	M	IRM	C	Not available	Program traces and generated events	Privacy leakage and user–level misuse —kernel-level is not monitored	Analysis is done off-line, prior to repackaging the app, i.e., in the cloud
Crowdroid (2011) [129]	And	D	SYS	CL	A	L	C	Not available	System calls per application	Any kind of anomaly	Training Method: Clustering with k–means: i) malware, and ii) goodware. Evaluation: Experimental and wild malware
DroidScope (2012) [130]	And	D	*	∅	∅	SB	C	Not applicable	Any kind	Any kind	ToM: Syscalls, etc. Ad–hoc plugins for monitoring features and analyzing data (authors provide several proof of concepts, e.g.: tainting
MADAM (2012) [131]	And	D	K, U	ML	A	L	L	Overhead of 3% memory utilization, 7% CPU and 5 % battery	K: SYS, proc., memory, CPU usage. U: user–state, key strokes, called numbers, SMS, NET	Any kind of anomaly	K-NN (with K=1) for classification. 10 malicious apps and 50 benign. 93% detection rate and 5% FP
Peng et al. (2012) [132]	And	S	P	PRO	N/A	C	C	Not applicable	Permissions	Effectiveness of apps permissions	
RiskRanker (2012) [133]	And	S	I/P/API	DG	M	C	C	Not applicable	Vulnerability signatures, permissions, API calls: crypto, dynamic code, IPC, and JNI, etc.	Any kind	Checks a pre-defined set of malicious operations (e.g.: known exploits) to rate the severity of stealthy applications
SmartDroid (2012) [134]	And	H	*	*	*	SB	SB	Unavailable	Any	UI–based obfuscation	Improved detection by generating UI-based trigger conditions. Any kind of detection system might be plunged, but no further details are given

		Detection Approach									
	Plat.	ToD	ToM	ToA	ToI	PoMI	PoA	Consumption	Features	Attack	Observations
Woodpecker (2012) [135]	And	S	I/P	DG	$\emptyset$	C	C	Time consuming analysis: 1 hour per phone image	Executing paths and 13 representative privileged permissions	Capability leaks. Confused deputy attacks	Uses CFG for detecting explicit capability leakages and permission analysis for implicit capability leakage
PiOS (2011) [136]	iOS	S	N/A	DG	N/A	C	C	Not applicable	Instructions	Obfuscation	Uses CFG for detecting capability leakages
Schmidt et al. (2011) [137]	Sym/And	S/D	SYS	CL	A	L	C/DB	Not available	Free RAM, User Inactivity, Process count, CPU usage, SMS sent, and others not specified	Any kind of anomaly	Training method: SVM–light and user's statistical data
Elish et al. (2013) [138]	And	S	I	DG	$\emptyset$	C	C	Not applicable	Data event–specific control	Component hijacking for information leakage and unauthorized access	Uses DDG to track the user's private information
CHEX (2012) [139]	And	S	I	DG	$\emptyset$	C	C	Not applicable	User's data	Component hijacking for information leakage and unauthorized access	Uses system dependence graphs to track the user's private information
AASandbox (2010) [140]	And	D	*	CL	M	SB	C	Not applicable	Not available	Any kind	Training method: Unspecified type of clustering. Evaluation: Self–written malware
Paranoid Android (2010) [141]	And	D	*	*	*	RC	C	Discussed. Apparently larger than expected	Not available	Any kind	Training method: Dynamic analysis and AV Analysis. Evaluation: Not performed
TaintDroid (2010) [84]	And	D	PT	E	M	L	L	Uses 14% CPU and 4.4% memory overhead. Power consumption not available	Variables, methods, file, and message	Explicit information flow leakage	Type of monitoring: label-based tracking of variables, methods, files and IPC via dynamic tainting, and enforced by the user. Tainted variables are propagated according to data flow rules
Kim et al. (2008) [142]	Win	D	HW	ST	M	L	L/C	Not available	Energy consumption	Energy–depletion attacks	The consumption is monitored using physical hardware (HW) and the analysis is done either at the phone or at the server (no performance comparison is provided). The signatures are generated sampling the power consumption history and matching is computed using χ^2–distance

exhibited is in the same conditions with the malware detector or without it.

Similarly to Andromaly [127], MADAM [131] uses dynamic analysis for periodically monitoring a number of features, and machine learning anomaly detectors for classifying goodware and malware, locally in the device. However, MADAM is evaluated using real malware samples, and consequently needs a higher number of features to model user behavior. Furthermore, collected observations are classified using K-Nearest Neighbor (K-NN) with $K = 1$ (1-NN). The evaluation was carried out with more than 50 goodware applications and 10 malware samples along with several user behaviors, improving the detection accuracy (93%) with respect to the same classifier used in Andromaly [127]. The results show an average number of number of 5 false positives per day. The reported performance overhead is 3% of memory consumption, 7% of CPU overhead and 5% of battery.

More recently, TStructDroid [126] presents a real-time malware detection system for ANDROID OS devices. The proposed system monitors Process Control Blocks (PCB) and uses theoretical analysis, time-series feature logging, segmentation and frequency component analysis of data, and a learned classifier to analyze monitored data. Evaluation shows a 98% accuracy and less than 1% false alarm rate, togetther with a 3.73% of performance degradation.

Finally, Crowdroid [129] is another anomaly-based malware detection system for ANDROID OS devices. The main difference with Andromaly [127] and MADAM [131] is that authors analyze the monitored featured in the cloud, whereas the other two approaches train their classifiers locally in the device. Collected observations are classified using K-Means. Evaluation was also carried out using a self-implemented set of malware samples, showing a detection rate of 100%. Additionally, they also test their system with two malware instances observed in the wild, showing a detection rate of 85% and 100% respectively. A key limitation in their study is that they assume that outsourcing the analysis should present a lower battery degradation than approaches that classify locally. However, we consider that this assumption has to be formally proven as some detection approaches are quite lightweight and might consume less than continuously transmitting all traces through the network.

2) Misuse Detection: AppGuard [128] is a malware prevention system for ANDROID OS in which the monitoring system is placed inline (IRM) with the application. Applications are manipulated using the repackaging technique, and the monitoring system is, therefore, inserted inside the applications. Applications can thus trace themselves and a number of security policies can be defined to enforce system permissions at runtime. Evaluation was performed using 13 apps, each of which

was inlined with 9 policies. One noteworthy characteristic is that inlined apps incur a negligible increment in their size.

Reported experiments in [128] also compare the execution of three function calls in both the original and the inlined app (the latter with no policies set), showing a degradation of 5.0%, 6.2%, and 1.0% of overhead respectively. In this regard, we consider that the three micro-benchmarks used are not conclusive due to their simplicity. Additionally, we consider that these results cannot be compared with Andromaly as they were not tested under the same conditions.

3) Replicas in the Cloud: Approaches such as Paranoid Android [141] or Secloud [125] have focused on performing malware detection tasks over synchronized replicas of the device maintained in the cloud. Thus, all security monitoring, analysis and identification tasks can be done in an environment not subject to battery constraints. Additionally, multiple detection techniques can be applied simultaneously, as several replicas can be run at the same time.

The proposed systems introduce several attack detection mechanisms for dynamic analysis in the replicas such as AV scanners and tainting analysis. However, Secloud [125] extends those mechanisms and deploys a number of response and prevention techniques, including file removal, process termination, periodic backups, network filtering, and device quarantining.

Experiments on Paranoid Android [141] show that synchronizing the device with the replicas does not introduce more than 2KB/s and 64B/s of trace data for high-load and idle operation environments, respectively. This performance, however, cannot be compared with Secloud [125], as for the latter no information about the consumption of the device being replicated is provided.

E. Market Protection

Most of the aforementioned techniques are typically designed to monitor physical devices, although they can also be used in virtual environments for market protection. Using specific monitoring techniques for virtual environments can bring about a number of benefits, such as (i) performing a resource-intensive security analysis, (ii) enabling virtual machine introspection [143] to intercept OS-level semantics, or (iii) enabling the possibility of hosting exact replicas of the device in the cloud (e.g.: CloneCloud [144], and ThinkAir [145]) as mentioned before.

1) Sandboxing: Several approaches have been proposed for malware detection in the form of sandboxes. For example, AASandbox [140] is an ANDROID OS analysis sandbox for both static and dynamic analysis. AASandbox uses an android emulator, pre-loaded with a SYS call monitoring service.

DroidScope [130] is another sandbox for ANDROID OS based on virtualization. It allows to monitor app features at the three layers of ANDROID OS's architecture, i.e., hardware, OS, and Dalvik Virtual Machine. Different types of monitoring can be enabled by developing custom plugins over DroidScope. In this regard, the authors include (i) a collector for native and Dalvik instructions traces, (ii) a profiler for API-level activity, and (iii) a tracking system for information leakage using taint analysis.

2) Smart Interaction: Sandbox analysis poses a limitation when interacting with samples in an automated way, due to the fact that some malicious apps hide their malicious activity through the User Interface (UI). In this regard, SmartDroid [134] presents an hybrid static and dynamic detection method to reveal UI-based trigger conditions in ANDROID OS. While static analysis is used to generate Activity and Function Call Graphs (ACG and FCG, respectively), dynamic analysis is used to explore such paths.

AppsPlayground [124] presents a similar approach combining detection techniques (ranging from taint tracing to SYS call monitoring) along with automatic exploration strategies. The proposed framework uses heuristics to guide the UI inputs, avoiding redundant explorations and using contextual information to fill editable text boxes.

3) Risk Analysis: Risk analysis techniques are emerging as a mechanism to palliate the ineffective way in which permissions are used to communicate potential threats to the user [33]. Here, Grace et al. propose the use of static assessment metrics to measure dangerous behaviors in ANDROID OS called RiskRanker [133]. Their proposal focuses on conducting a scalable, efficient and accurate proof-of-concept rather than leveraging on sophistication. Contrary, Peng et al. [132] propose the use use probabilistic generative models for risk ranking and scoring schemes. More precisely, they evaluate a range of models starting from simple Basic Naive Bayes (BNB) to advanced hierarchical mixture models, showing that these models offer a promising mechanism for risk scoring.

4) Similarity detection: Researchers have explored different ways to detect repackaging in markets by detecting similarity dependencies among population of applications. While early approaches use syntactic analysis such as string-based matching [146], recently approaches elaborate on semantic analysis [147], [118], e.g., PDG, as it is resilient to code obfuscation. However, semantic analysis is generally more expensive than syntactic analysis.

A different approach is presented in [146], where several compression algorithms are used to compute normalized information distances between two applications based on Kolmogorov complexity measurement. Their algorithm first identifies which methods are identical and calculates the similarity of the reminder methods using Normalized Compression Distances (NCD). In order to reduce complexity, the authors use a representation of each method based on structured control flow signatures [148]. Finally, authors apply Longest Common Subsequence (LCS) algorithm to identify differences between similar elements.

Zhou et al. [55] propose a system called DroidMOSS for detecting repackaged applications based on a fuzzy hashing technique. Distinguishing features are first extracted in the form of fingerprints, and then compared with those from other applications in order to identify similarities. These features are computed by applying traditional hash functions to pieces of code of variable size. The size of the pieces is bounded by smaller chunks of fixed size called reset points. A chunk is considered a reset point when the resulting hash is a prime number. Then, the edit distance is calculated between two applications by comparing their fingerprints on identical

matching-basis. More recently, authors have extended their work in [149]. While their former work is designed to detect repackaging in unofficial markets, the latter is capable of detecting repackaging among apps in the same market.

Authors in [150] present Juxtapp, a system for detecting app similarity. They propose an optimization over the representation of the applications as an alternative to k-grams based on feature hashing and then use hierarchical clustering to classify similar applications.

Authors in [147] present DNADroid, a system for detecting cloned applications based on dependency graphs between methods. PDG is used to detect semantic similarities by comparing graph isomorphism. Prior to similarity detection, authors group applications based on meta-information retrieved from each application, and they use several filters to enhance efficiency. Although their experiments show better results than similar approaches such as [146], the scheme is less efficient in terms of performance. In fact, their experimental testbed is deployed in a small cluster composed of one server and three desktop computers over *Hadoop*. Even there, the analysis rate is 0.7 applications per minute.

More recently, Suarez-Tangil et al present Dendroid [118], a text mining approach to analyzing and classifying code structures in Android malware families. By adapting the standard Vector Space Model and reformulating the modelling process followed in text mining applications, authors present a novel way to measure similarity between malware samples. This similarity is used to automatically classify samples into families. Authors also investigate the application of hierarchical clustering over the feature vectors obtained for each malware family. The resulting dendograms resemble the so-called phylogenetic trees for biological species, allowing researchers to conjecture about evolutionary relationships among families. Experimental results suggest that the approach is remarkably accurate and deals efficiently with large databases of malware instances.

F. Attack-specific Malware Identification Systems

The majority of the approaches described above focus on general detectors using either anomaly or misuse detection for both static and dynamic analysis. However, due to the diversity of malware goals and incentives, other schemes are narrowing the complexity towards detecting specific classes of malware, such as privileged escalation, battery-depletion attacks, or money stealing.

1) Privilege Escalation: There are two common types of privilege escalation attacks according to whether the exploitation strategy focuses on inter-process capability leakage or system vulnerabilities. Approaches such as XManDroid [70], Woodpecker [135], Elish et al. [138] or CHEX [139] focus on the first class, while others such as [151] concentrate on the latter.

XManDroid [70] is a privilege escalation detection tool for ANDROID OS devices. Dynamic analysis is used to identify covert channels using DFG. Woodpecker [135] is capable of identifying both explicit and implicit leakage by combining static with dynamic analysis. Static analysis is used to identify possible execution paths by means of CFG, and inter-procedural data flow analysis is used to filter out non-dangerous paths. Additionally, app permissions are examined to broaden leakage search. Similarly, Elish et al. [138] use DDG providing user-interaction dependencies of more than 1000 benign and malign apps, while CHEX [139] employs system dependence graphs over more than 5000 applications from *Google Play*.

ROPdefender [152] is a generic ROP detection tool for Windows and Linux–based OS capable of enforcing a return address check. Although ROPdefender is not built for smart devices, the proposed framework can be applied in this context.

2) Grayware: As discussed early in this paper, grayware poses a serious challenge to privacy leakage detection system. Several approaches have focused on detecting such privacy leakages, such as TaintDroid [84] for ANDROID OS devices and PiOS [136] for IOS.

TaintDroid [84] uses dynamic taint analysis to track sensitive information. It monitors variables, methods, files, and messages throughout the program execution according to data flow rules, and label the variables as they use the sensitive data. When a piece of sensitive information attempts to leave a taint sink, e.g., through the network interface, TaintDroid requests user consent to do so. The authors studied 30 popular applications, showing that at least 20 of them misused users' private information. Experiments also show that TaintDroid incurs 14% CPU and 4.4% memory overhead. A major limitation of TaintDroid is its inability to distinguish between legitimate and non-legitimate exfiltrations, especially when facing grayware. In fact, their experiments show that 37 out of 105 instances (35%) were incorrectly classified as false positives. Additionally, techniques such as tainting can be circumvented through leaks via implicit flows, i.e., using program control flow to disclose information.

AppProfiler [123] uses dynamic tainting analysis along with static analysis to extract privacy-related behaviors. The scheme builds a *knowledge base* that maps application behaviors with API calls observed during static analysis, providing the user with valuable information about their apps.

Finally, PiOS [136] is an information leakage detection system for IOS devices that uses static analysis on apps. PiOS constructs CFG paths from the sources of sensitive information to data sinks by means of data-flow analysis. So far, static analysis of IOS apps does not have to face the obfuscation challenge, as obviously obfuscated apps would not pass the revision process. However, this might change in the coming years if non-walled-garden models such as *Cydia* gain popularity.

3) Battery-depletion: Traditional anomaly and misuse detection techniques have not paid much attention to unknown energy-depletion attacks. In this regard, Kim et al. [142] proposes a power-aware malware detection system for smart devices. It uses dynamic analysis to monitor power samples and build a consumption model. Power signatures are generated from monitoring malicious samples in the device, and results are analyzed in the device or in the cloud using noise filtering and data compression algorithms. After building the model, malware is identified by using χ^2-distance and comparing the results with a set of signatures.

V. Open Research Topics

Malware in smart devices still pose many challenges and a number of important issues need to be further studied and addressed with novel solutions. This section identifies some open issues where research is needed. Some of these problems are not specific to smart devices, such as for example the case of botnets. Others, such as Online Social Networks (OSNs) –which have attracted millions of active users in the last years– are increasingly related to smart devices, as users mostly access them through their smartphones and smart TVs. Thus, security problems in these domains (e.g. *socialbots* [153]) will likely target these platforms soon.

A. Automatic Malware Analysis and Classification

The impressive growth both in malware and benign apps is making increasingly unaffordable any human-driven analysis of potentially dangerous apps. Dynamic analysis techniques such as those surveyed in [48] are progressively playing a key role in detecting malware for smart devices. Current trends in malware engineering suggest that malicious software will continue to evolve its sophistication, in part due to the availability of reuse-oriented development methodologies. From the defender's point of view, this should be exploited to facilitate analysis and detection. For example, some works conducted over the last years have explored the possibility of *clustering* malware instances [154], [147] into classes according to some similarity metric. Such classes can be later used to automatically classify newly discovered specimens, thus facilitating their analysis. We believe that further efforts along this line are required, in particular by developing more fine-grained techniques. For example, instead of just pointing out what malware samples are similar to a given one, it would be more useful to decompose the sample in components and perform the similarity search at that level (we refer the reader to Dendroid [118] for more details in this regard).

B. Trusted Software

In the case of current smartphones and tablets, trust on the non-malicious nature of an app is based on two factors: (i) the implicit assumption that the market operator has conducted some security review before making the app available for download; and (ii) the identity of the developer, given by the signature attached to the app, which also provides some evidence of the app's integrity. The first point is not fully reliable, as operators cannot afford to carry out an exhaustive analysis over every submitted app; and, even if they could, there is still some non-negligible probability of sophisticated malware evading detection. As for the identify of the developer and the app's integrity, evidence suggests that most users do not pay much attention to them, or positively ignore them when downloading apps from alternative markets.

We believe that further efforts to improve trust in software are required. This will be increasingly necessary in the near future, as the number of developers –and, hence, apps– will likely grow very significantly. Reputation systems [155], [156] adapted to this context might offer some added value, in particular by exploiting interactions in large user communities such as, for example, those provided by online social networks or mobile adhoc networks [157]. But other mechanisms for building trust could also apply, such as for example remote attestation protocols [158], [156], [75] or any other schemes to ensure the authenticity and integrity of software.

C. Malware in Other Smart Devices

The experience gained from current smartphones suggests that malware will also hit other smartdevices as soon as they appear. Evidence in other pervasive technologies already exists. For example, nowadays Radio Frequency Identification (RFID) systems are used in a wide range of applications, such as transport tickets, access control systems, e-passports, e-health applications, etc. The benefits of adopting RFID technology for identification purposes are clear, but its associated security risks need to be addressed. One of them –often underestimated– is malware. The use of Internet-enabled mobile devices as RFID readers makes this sort of attacks potentially more harmful. Most previous works have focused on the securing the communication link between the tag and the (mobile) reader. There are, however, some preliminary works [159], [42] on RFID malware, but further studies and solutions are required. Similarly, IMDs and other medical devices will likely be an attractive target for attackers due to the economic value of the information they can provide [43], [160]. These devices are not prune to the software problems like malfunctions and corrupted updated versions [161], [162].

D. Grayware and Other Privacy Issues

Applications are increasingly requiring the user to authorize the transference of personal information to the cloud as part of the normal use of the application. For instance, *WhatsApp* sends the user's address book to establish friendship connections [163]. However, even if the user authorizes such transference, it does not mean that it will be used for purposes other than those conveyed to the user, such as for example market research. In other cases users are only informed that some personal information will be sent, but the particulars about what specific items or how it will be used are not given. Identifying misuse of personal information, both on-platform and in the cloud, is a challenging process that is typically tackled by legal enforcement mechanisms, but technical approaches should be explored. For instance, in the same way that Google App Engine [164] is used to deploy in-the-cloud applications –monitored by Google–, back-end services for smartphones and other smart devices could be moved to a cloud controlled and monitored by a trusted third party. This could make feasible to monitor behavior and enforce security policies in the cloud-end of the service, thus complementing other security mechanisms applied in the device.

Similar privacy-related problems arise in cloud-based monitoring schemes, primarily in those that maintain a virtualized replica of the device to carry out monitoring tasks that are unaffordable to perform directly on the device. Privacy-preserving monitoring systems for this scenario are required, but also more lightweight monitoring and detection mechanisms that can run on the device with an appropriate balance between efficacy and power consumption.

E. Cooperative security

In the near future it is very likely that many users will own a *network* of smart devices, including smartphones, smart TVs and other home appliance, and wearable computing platforms. Such networks could be leveraged to implement cooperative security functions, as a complement to cloud-based and on-platform monitoring and analysis mechanisms. Ideally, several connected devices could cooperate to improve security in a number of ways. For example, resource-intensive tasks can be delegated to devices with a permanent power source to preserve the battery of mobile platforms. Similarly, mutually monitoring schemes could be interesting, where each device monitors the behavior of others to detect compromise.

F. Forensics-based analysis for smart device protection

Sometimes malicious programs uninstall themselves after achieving their goals. However, analyzing evidences that they leave behind could be used as an input for detecting future propagations using the same infection vector. Identifying such traces is a great challenge, particularly due to the availability of anti-forensic tools for devices such as smartphones [165]. In this regard, two different approaches might be worth exploring. On the one hand, deleting evidences or attempting to neutralize any source of evidence usually produces fresh new evidences. On the other hand, new paradigms such as the aforementioned replicas in the cloud, allow the creation of novel forensic approaches on the cloud based on virtual introspection.

VI. Conclusions

In this paper, we have presented a comprehensive survey on the evolution of malware for smart devices and recent results on detection and analysis techniques. We have first provided an overview of the security models and protection mechanisms present in current platforms for smart devices, mostly smartphones. Next we have proposed a characterization of malware in terms of three key factors: pursued goals and associated behaviors; distribution and infection channels; and privilege acquisition strategies. Our analysis of some representative samples shows that malware is becoming increasingly complex and adaptive, with constantly changing goals and using multiple distribution and infection strategies.

We have also provided an analysis of the 20 most significant proposals for detecting and analyzing malware for smart devices proposed between 2010 and 2013. Instead of merely enumerating and describing each one of them, we have first identified and classifed all device features where malware behavior could manifest. This taxonomy is complemented with additional elements, such as where the monitoring and analysis tasks takes place, or the specific detection technique used.

Finally, we have discussed a number of open research problems in the hope of stimulating further research in this thriving area.

Acknowledgements

We thank the anonymous reviewers for valuable suggestions that helped to improve the quality and organization of this paper.

References

[1] H. Dediu, "When will tablets outsell traditional pcs?" March 2012, http://www.asymco.com/2012/03/02/when-will-the-tablet-market-be-larger-than-the-pc-market/.

[2] Juniper, "2011 mobile threats report," Juniper Networks, Tech. Rep., 2012.

[3] L. Goasduff and C. Pettey, "Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth," Visited April 2012, http://www.gartner.com/it/page.jsp?id=1924314.

[4] Nielsen, "State of the appnation —a year of change and growth in u.s. smartphones," Nielsen, Tech. Rep., 2012.

[5] R. van der Meulen and J. Rivera, "Gartner says worldwide mobile phone sales declined 1.7 percent in 2012," Visited March 2013, http://www.gartner.com/newsroom/id/2335616.

[6] Samsung. (Visited March 2013) Samsung smart tv. [Online]. Available: http://www.samsung.com/us/2012-smart-tv/

[7] Sony. (Visited March 2013) Smartwatch. [Online]. Available: http://www.sonymobile.com/us/products/accessories/smartwatch/

[8] Google. (Visited March 2013) Google glass. [Online]. Available: http://http://www.google.com/glass/

[9] CuteCircuit. (Visited March 2013) T–shirtos: The future is getting closer. [Online]. Available: http://www.cutecircuit.com/tshirtos-the-future-is-getting-closer/

[10] D. Newcomb. (Visited March 2013) Weblink aims to bridge the nagging smartphone-car disconnect. [Online]. Available: http://www.wired.com/autopia/2013/03/weblink-abalta-auto-apps/

[11] S. Larner, "Smartphones and tablets in the hospital environment," *British J. of Healthcare Management*, vol. 18, no. 8, pp. 404–405, 2012.

[12] IIH. (Visited March 2013) Smart pillbox. [Online]. Available: http://www.innovatorsinhealth.org/solutions/

[13] M. Chan, D. Estve, J.-Y. Fourniols, C. Escriba, and E. Campo, "Smart wearable systems: Current status and future challenges," *Artificial Intelligence in Medicine*, vol. 56, no. 3, pp. 137 – 156, 2012.

[14] H. Dediu, D. Schmidt, and R. Salle. (Visited March 2013) Asymco. [Online]. Available: http://www.asymco.com/

[15] D. Seifert. (Visited May 2013) Back from the dead: why do 2013's best smartphones have ir blasters? [Online]. Available: http://www.theverge.com/2013/4/24/4262074/is-this-the-year-of-the-ir-blaster

[16] F. Wang and J. Liu, "Networked wireless sensor data collection: Issues, challenges, and approaches," *IEEE Commun. Surveys & Tutorials*, vol. 13, no. 4, pp. 673–687, 2011.

[17] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A survey on smart grid communication infrastructures: Motivations, requirements and challenges," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 1, pp. 5–20, 2013.

[18] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surveys & Tutorials*, vol. 11, no. 1, pp. 13–32, 2009.

[19] C. MacManus. (Visited Agoust 2013) Sony's smarttags could change phone habits. [Online]. Available: http://news.cnet.com/8301-17938_105-57359901-1/sonys-smarttags-could-change-phone-habits/

[20] Y. Lee, Y. Ju, C. Min, J. Yu, and J. Song, "Mobicon: Mobile context monitoring platform: Incorporating context-awareness to smartphone-centric personal sensor networks," in *9th Annu. IEEE Commun. Society Conf. on Sensor, Mesh and Ad Hoc Commun. and Netw. (SECON 2012)*, 2012, pp. 109–111.

[21] D. Kelly, R. Raines, R. Baldwin, M. Grimaila, and B. Mullins, "Exploring extant and emerging issues in anonymous networks: A taxonomy and survey of protocols and metrics," *IEEE Commun. Surveys & Tutorials*, vol. 14, no. 2, pp. 579–606, 2012.

[22] M. Mueck, V. Ivanov, S. Choi, J. Kim, C. Ahn, H. Yang, G. Baldini, and A. Piipponen, "Future of wireless communication: Radioapps and related security and radio computer framework," *IEEE Wireless Commun.*, vol. 19, no. 4, pp. 9–16, 2012.

[23] C. Szongott, B. Henne, and M. Smith, "Evaluating the threat of epidemic mobile malware," in *Proc. IEEE 8th Int. Conf. on Wireless and Mobile Computing, Netw. and Commun. (WiMob 2012)*, 2012, pp. 443–450.

[24] La Polla, M. and Martinelli, F. and Sgandurra, D., "A Survey on Security for Mobile Devices," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 1, pp. 446–471, 2013.

[25] X. Wei, N. C. Valler, B. Prakash, I. Neamtiu, M. Faloutsos, and C. Faloutsos, "Competing memes propagation on networks: A network science perspective," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 6, pp. 1049–1060, 2013.

[26] Fernandes, Earlence and Crispo, Bruno and Conti, Mauro, "FM 99.9, Radio Virus: Exploiting FM Radio Broadcasts for Malware Deployment," *IEEE Trans. Inf. Forens. Security*, 2013.

[27] G. Baldini, T. Sturman, A. Biswas, R. Leschhorn, G. Godor, and M. Street, "Security aspects in software defined radio and cognitive radio networks: A survey and a way ahead," *IEEE Commun. Surveys & Tutorials*, vol. 14, no. 2, pp. 355–379, 2012.

[28] S. Amini, J. Lindqvist, J. Hong, J. Lin, E. Toch, and N. Sadeh, "Caché: caching location-enhanced content to improve user privacy," in *Proc. 9th Int. Conf. on Mobile systems, applications, and services*. ACM, 2011, pp. 197–210.

[29] A. Parate, M.-C. Chiu, D. Ganesan, and B. M. Marlin, "Leveraging graphical models to improve accuracy and reduce privacy risks of mobile sensing," in *Proc. 11th Int. Conf. on Mobile Systems, Applications and Services*. ACM, 2013, pp. 83–96.

[30] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Symp. on Usable Privacy and Security*. Washington: Advancing Science, Serving Society, March 2012.

[31] J. Fenske, "Biometrics in new era of mobile access control," *Biometric Technology Today*, vol. 2012, no. 9, pp. 9–11, 2012.

[32] N. Husted, H. Saïdi, and A. Gehani, "Smartphone security limitations: conflicting traditions," in *Proc. 2011 Workshop on Governance of Technology, Information, and Policies*, ser. GTIP '11. New York, NY, USA: ACM, 2011, pp. 5–12.

[33] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 3–14.

[34] K. Dunham, *Mobile malware attacks and defense*. Syngress, 2008.

[35] D. Shih, B. Lin, H. Chiang, and M. Shih, "Security aspects of mobile phone virus: a critical survey," *Industrial Management & Data Systems*, vol. 108, no. 4, pp. 478–494, 2008.

[36] Juniper, "2013 mobile threats report," Juniper Networks, Tech. Rep., 2013.

[37] F-Secure, "Mobile threat report q1 2012," F–Secure, Tech. Rep., April 2012, "http://www.f-secure.com/weblog/archives/MobileThreatReport_Q1_2012.pdf".

[38] McAfee, "Threats report:fourth quarter 2012," McAfee, Tech. Rep., January 2013, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2012.pdf.

[39] M. Schipka, "Dollars for downloading," *Network Security*, vol. 2009, no. 1, pp. 7–11, 2009.

[40] D. Guido and M. Arpaia, "Mobile exploit intelligence project," 2012, http://www.trailofbits.com/resources/mobile_eip-04-19-2012.pdf.

[41] McAfee, "Threats report:fourth quarter 2010," McAfee, Tech. Rep., January 2011, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2010.pdf.

[42] M. R. Rieback, P. N. Simpson, B. Crispo, and A. S. Tanenbaum, "Rfid malware: Design principles and examples," *Pervasive and mobile computing*, vol. 2, no. 4, pp. 405–426, 2006.

[43] W. P. Burleson, S. S. Clark, B. Ransford, and K. Fu, "Design challenges for secure implantable medical devices," in *Proc. 49th Design Automation Conf.*, ser. DAC'12. New York, NY, USA: ACM, 2012, pp. 12–17.

[44] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *Proc. 29th Annual IEEE Symp. Security and Privacy*. USENIX Association, May 2008, pp. 129–142.

[45] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proc. 6th USENIX conf. on Hot topics in security*, ser. HotSec'11, Berkeley, CA, USA, 2011, pp. 9–9.

[46] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proc. 33rd IEEE Symp. Security and Privacy (Oakland 2012)*, May 2012.

[47] AV-Test, "Anti–malware solutions for android," AV Test, Tech. Rep., 2012, "http://www.av-test.org/fileadmin/pdf/avtest_2012-02_android_anti-malware_report_english.pdf".

[48] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2012.

[49] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan, "Old, new, borrowed, blue –: a perspective on the evolution of mobile platform security architectures," in *Proc. 1st ACM conf. on Data and application security and privacy*, ser. CODASPY '11. ACM, 2011, pp. 13–24.

[50] W. Enck, "Defending users against smartphone apps: techniques and future directions," in *Proc. 7th int. conf. on Information Systems Security*, ser. ICISS'11. Springer-Verlag, 2011, pp. 49–70.

[51] L. Batyuk, M. Herpich, S. Camtepe, K. Raddatz, A. Schmidt, and S. Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications," in *6th Int. Conf. on Malicious and Unwanted Software (MALWARE 2011)*, October 2011, pp. 66–72.

[52] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: automated security validation of mobile apps at app markets," in *Proc. int. workshop on Mobile cloud computing and services*, ser. MCS '11. New York, NY, USA: ACM, 2011, pp. 21–26.

[53] H. Lockheimer. (Visited January 2013) Android and security. [Online]. Available: http://googlemobile.blogspot.com.es/2012/02/android-and-security.html

[54] 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. 19th Annu. Network and Distributed System Security Symp. (NDSS)*, 2012.

[55] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proc. 2nd ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 317–326.

[56] McAfee, "Threats report:fourth quarter 2011," McAfee, Tech. Rep., January 2012, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2011.pdf.

[57] K. Au, Y. Zhou, Z. Huang, P. Gill, and D. Lie, "Short paper: a look at smartphone permission models," in *Proc. 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 63–68.

[58] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proc. 2nd USENIX conf. on Web application development*, ser. WebApps'11. USENIX Association, 2011, pp. 7–7.

[59] Felt, Adrienne Porter and Chin, Erika and Hanna, Steve and Song, Dawn and Wagner, David, "Android permissions demystified," in *Proc. 18th ACM conf. on Computer and commun. security*. ACM, 2011, pp. 627–638.

[60] Barrera, David and Kayacik, H Güneş and van Oorschot, Paul C and Somayaji, Anil, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proc. 17th ACM conf. on Computer and communications security*. ACM, 2010, pp. 73–84.

[61] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Information Security*, ser. Lecture Notes in Computer Science, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds. Springer Berlin / Heidelberg, 2011, vol. 6531, pp. 346–360.

[62] K. Gudeth, M. Pirretti, K. Hoeper, and R. Buskey, "Delivering secure applications on commercial mobile devices: the case for bare metal hypervisors," in *Proc. 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. ACM, 2011, pp. 33–38.

[63] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4android: a generic operating system framework for secure smartphones," in *Proc. 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 39–50.

[64] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in *Proc. 23rd ACM Symposium on Operating Systems Principles*, ser. SOSP '11. ACM, 2011, pp. 173–187.

[65] G. Russello, M. Conti, B. Crispo, and E. Fernandes, "Moses: supporting operation modes on smartphones," in *Proc. 17th ACM symp. on Access Control Models and Technologies*, ser. SACMAT '12. New York, NY, USA: ACM, 2012, pp. 3–12.

[66] Y. Xu, F. Bruns, E. Gonzalez, S. Traboulsi, A. Mott, and A. Bilgic, "Performance evaluation of para-virtualization on modern mobile phone platform," in *Proc. Int. Conf. on Computer, Electrical, and Systems Science and Engineering*, ser. ICCESSE '10. Waset, 2010, pp. 272–280.

[67] A. Felt, H. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: Attacks and defenses," in *Proc. 20th USENIX Security Symp.*, 2011.

[68] E. Chin, A. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proc. 9th int. conf. on Mobile systems, applications, and services*. ACM, 2011, pp. 239–252.

[69] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," *Information Security*, pp. 331–345, 2011.

[70] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," Technische Universitat Darmstadt, Tech. Rep., 2011.

[71] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM conf. on Computer and communications security*. ACM, 2009, pp. 235–245.

[72] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," in *Computer Security Applications Conf., 2009. ACSAC '09. Annu.*, December 2009, pp. 340–349.

[73] C. Mulliner, G. Vigna, D. Dagon, and W. Lee, "Using labeling to prevent cross-service attacks against smart phones," in *Detection of Intrusions and Malware and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, R. Bschkes and P. Laskov, Eds. Springer Berlin Heidelberg, 2006, vol. 4064, pp. 91–108.

[74] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing android-powered mobile devices using selinux," *IEEE Security & Privacy*, vol. 8, no. 3, pp. 36–44, 2010.

[75] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert, "Beyond kernel-level integrity measurement: enabling remote attestation for the android platform," in *Trust and Trustworthy Computing*. Springer, 2010, pp. 1–15.

[76] X. Ni, Z. Yang, X. Bai, A. C. Champion, and D. Xuan, "Diffuser: Differentiated user access control on smartphones," in *IEEE 6th Int. Conf. Mobile Adhoc and Sensor Systems, 2009. MASS'09.*. IEEE, 2009, pp. 1012–1017.

[77] F. Rohrer, Y. Zhang, L. Chitkushev, and T. Zlateva, "Poster: Role based access control for android (rbaca)," Boston University, MA USA, Tech. Rep., 2012.

[78] S. Kraemer and P. Carayon, "Human errors and violations in computer and information security: The viewpoint of network administrators and security specialists," *Applied Ergonomics*, vol. 38, no. 2, pp. 143 – 154, 2007.

[79] J. O'Connor, "Blackberry security: Ripe for the picking?" Symantec, Tech. Rep., 2006.

[80] J. Jeon, K. Micinski, J. Vaughan, N. Reddy, Y. Zhu, J. Foster, and T. Millstein, "Dr. android and mr. hide: Fine-grained security policies on unmodified android," University of Maryland, Tech. Rep., 2011.

[81] D. Schreckling, J. Posegga, and D. Hausknecht, "Constroid: Data-Centric Access Control for Android," in *Proc. 27th Symp. on Applied Computing (SAC): Computer Security Track*, 2012.

[82] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in *Proc. 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 51–62.

[83] N. Husted, H. Saïdi, and A. Gehani, "Smartphone security limitations: conflicting traditions," in *Proc. 2011 Workshop on Governance of Technology, Information, and Policies*, ser. GTIP '11. New York, NY, USA: ACM, 2011, pp. 5–12.

[84] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX conf. on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–6.

[85] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proc. 18th ACM conf. on Computer and communications security*. ACM, 2011, pp. 639–652.

[86] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *IEEE Security & Privacy*, vol. 8, no. 2, pp. 35–44, 2010.

[87] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 50–57, 2009.

[88] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proc. 20th USENIX conf. on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 21–21.

[89] Apple. (May 2012) ios security. [Online]. Available: http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf

[90] D. Chubb. (Visited May 2013) Data privacy of ios 6 release notes. [Online]. Available: http://www.product-reviews.net/2012/06/15/ios-6-release-notes-show-heightened-security/

[91] C. Miller. (Visited May 2013) Comparision of ios and android security. [Online]. Available: http://www.accuvant.com/blog/2011/10/20/dr-charlie-miller-compares-security-ios-and-android

[92] Apple. (Visited May 2013) Apple answers fcc questions. [Online]. Available: http://www.apple.com/hotnews/apple-answers-fcc-questions/

[93] Microsoft, "Windows Phone 8 Security Overview," Microsoft Coorporation, Tech. Rep., December 2012, http://go.microsoft.com/fwlink/?LinkId=266838.

[94] C. Fleizach, M. Liljenstam, P. Johansson, G. Voelker, and A. Mehes, "Can you infect me now?: malware propagation in mobile phone networks," in *Proc. 2007 ACM workshop on Recurring malcode*. ACM, 2007, pp. 61–68.

[95] R. Verdult and F. Kooman, "Practical attacks on nfc enabled cell phones," in *3rd Int. Workshop on Near Field Commun. (NFC)*, February 2011, pp. 77–82.

[96] L. Auriemma. (Visited May 2013) Samsung devices with support for remote controllers. [Online]. Available: http://aluigi.org/adv/samsux_1-adv.txt

[97] Sophos. (Visited May 2013) First anti–virus software for connected tv. [Online]. Available: http://goo.gl/Ww67D

[98] D. Halperin, T. Kohno, T. Heydt-Benjamin, K. Fu, and W. Maisel, "Security and privacy for implantable medical devices," *IEEE Pervasive Comput.*, vol. 7, no. 1, pp. 30–39, January 2008.

[99] M. Vockley, "Safe and secure? healthcare in the cyberworld." *Biomedical instrumentation & technology/Association for the Advancement of Medical Instrumentation*, vol. 46, no. 3, p. 164, 2012.

[100] S. Corporation, "Symantec security threats," Visited May 2013, http://www.symantec.com/security_response/landing/threats.jsp.

[101] F-Secure, "F–secure mobile threats," Visited May 2013, http://www.f-secure.com/en/web/labs_global/mobile-security.

[102] Lookout. (Visited May 2013) Notcompatible. [Online]. Available: http://goo.gl/yJEgn

[103] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta, "On cellular botnets: measuring the impact of malicious devices on a cellular network core," in *Proc. 16th ACM conf. on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 223–234.

[104] P. A. Porras, H. Saidi, and V. Yegneswaran, "An analysis of the ikee.b iphone botnet," in *Security and Privacy in Mobile Information and Communication Systems (MobiSec), 2nd Int. ICST Conf.*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, A. U. Schmidt, G. Russello, A. Lioy, N. R. Prasad, and S. Lian, Eds., vol. 47. Springer, May 2010, pp. 141–152.

[105] C. Mulliner and J.-P. Seifert, "Rise of the iBots: 0wning a telco network," in *Proc. 5th IEEE Int. Conf. Malicious and Unwanted Software (Malware)*, Nancy, France, October 2010, pp. 71–80.

[106] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning, "Andbot: towards advanced mobile botnets," in *Proc. 4th USENIX conf. on Large-scale exploits and emergent threats*, ser. LEET'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 11–11.

[107] D. Damopoulos, G. Kambourakis, and S. Gritzalis, "isam: An iphone stealth airborne malware," in *Future Challenges in Security and Privacy for Academia and Industry*, ser. IFIP Advances in Information and Communication Technology, J. Camenisch, S. Fischer-Hbner, Y. Murayama, A. Portmann, and C. Rieder, Eds. Springer Berlin Heidelberg, 2011, vol. 354, pp. 17–28.

[108] S. Ldt., "Sophos endpoint protection," Visited May 2013, http://www.sophos.com.

[109] Kramer, S, "Rage against the cage," 2010.

[110] Luo, Tongbo and Hao, Hao and Du, Wenliang and Wang, Yifei and Yin, Heng, "Attacks on WebView in the Android system," in *Proc. 27th Annu. Computer Security Applications Conf.*. ACM, 2011, pp. 343–352.

[111] NakedSecurity, "Malicious cloned games attack google android market," Visited May 2013, http://nakedsecurity.sophos.com/2011/12/12/malicious-cloned-games-attack-google-android-market/.

[112] A. Apvrille, "Cryptography for mobile malware obfuscation," in *RSA Conf.*, RSA, Ed. Fortinet, October 2011.

[113] Skycure. (Visited May 2013) Malicious Profiles - The Sleeping Giant of iOS Security. [Online]. Available: \textcolor{black}{http://blog.skycure.com/2013/03/malicious-profiles-sleeping-giant-of.html}

[114] N. Seriot, "iphone privacy," *Black Hat DC*, p. 30, 2010.

[115] D. Goodin. (Visited June 2012) Apple expels serial hacker for publishing iphone exploit. [Online]. Available: http://www.theregister.co.uk/2011/11/08/apple_excommunicates_charlie_miller/

[116] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones: attacks, implications and opportunities," in *Proc. 11th Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 49–54.

[117] F. Shahzad, M. A. Akbar, and M. Farooq, "A survey on recent advances in malicious applications analysis and detection techniques for smartphones," National University of Computer & Emerging Sciences, Islamabad, Pakistan, Tech. Rep., 2012.

[118] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. B. Alis, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," *Expert Systems with Applications*, 2013, in Press.

[119] J. M. Estévez-Tapiador, P. Garcia-Teodoro, and J. E. Díaz-Verdejo, "Anomaly detection methods in wired networks: a survey and taxonomy," *Computer Communications*, vol. 27, no. 16, pp. 1569–1584, 2004.

[120] P. Garcia-Teodoro, J. E. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[121] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, "Survey of context provisioning middleware," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 3, pp. 1492–1519, 2013.

[122] I. Rassameeroj and Y. Tanahashi, "Various approaches in analyzing android applications with its permission-based security models," in *2011 IEEE Int. Conf. Electro/Information Technology (EIT)*, May 2011, pp. 1–6.

[123] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," in *Proc. 3rd ACM conference on Data and application security and privacy*. ACM, 2013, pp. 221–232.

[124] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: automatic security analysis of smartphone applications," in *Proc. 3rd ACM conference on Data and application security and privacy*. ACM, 2013, pp. 209–220.

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

[126] F. Shahzad, M. Akbar, S. Khan, and M. Farooq, "Tstructdroid: Real-time malware detection using in-execution dynamic analysis of kernel process control blocks on android," National University of Computer & Emerging Sciences, Islamabad, Pakistan, Tech. Rep., 2013.

[127] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""andromaly": a behavioral malware detection framework for android devices," *J. of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.

[128] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. Styp-Rekowsky, "Appguard —-real–time policy enforcement for third-party applications," Universitats- und Landesbibliothek, Postfach 151141, 66041 Saarbracken, Tech. Rep., 2012. [Online]. Available: http://scidok.sulb.uni-saarland.de/volltexte/2012/4902

[129] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 15–26.

[130] L. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proc. 21st USENIX conf. on Security symp.*. USENIX Association, 2012, pp. 29–29.

[131] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "Madam: a multi-level anomaly detector for android malware," in *Proc. 6th int. conf. on Mathematical Methods, Models and Architectures for Computer Network Security: computer network security*, ser. MMM-ACNS'12. Springer-Verlag, 2012, pp. 240–253.

[132] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *Proc. 2012 ACM conf. on Computer and communications security*. ACM, 2012, pp. 241–252.

[133] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proc. 10th int. conf. on Mobile systems, applications, and services*. ACM, 2012, pp. 281–294.

[134] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications," in *Proc. 2nd ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2012, pp. 93–104.

[135] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock android smartphones," in *Proc. 19th Annu. Symp. on Network and Distributed System Security*, 2012.

[136] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in *Proc. Network and Distributed System Security Symp.*, 2011.

[137] A.-D. Schmidt, "Detection of smartphone malware," Ph.D. dissertation, Universitätsbibliothek, 2011.

[138] K. O. Elish, D. D. Yao, B. G. Ryder, and X. Jiang, "A static assurance analysis of android applications," Virginia Polytechnic Institute and State University, Tech. Rep., 2013.

[139] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proc. 2012 ACM conf. on Computer and communications security*. ACM, 2012, pp. 229–240.

[140] T. Blasing, L. Batyuk, A. Schmidt, S. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *5th Int. Conf. on Malicious and Unwanted Software (MALWARE 2010)*. IEEE, 2010, pp. 55–62.

[141] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: versatile protection for smartphones," in *Proc. 26th Annu. Computer Security Applications Conf.*, 2010, pp. 347–356.

[142] H. Kim, J. Smith, and K. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proc. 6th int. conf. on Mobile systems, applications, and services*. ACM, 2008, pp. 239–252.

[143] T. Garfinkel, M. Rosenblum *et al.*, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Network and Distributed Systems Security Symp.*, 2003.

[144] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. 6th conf. on Computer systems*, 2011, pp. 301–314.

[145] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*. IEEE, 2012, pp. 945–953.

[146] A. Desnos, "Android: Static analysis using similarity distance," in *System Science (HICSS), 2012 45th Hawaii Int. Conf. on*. IEEE, 2012, pp. 5394–5403.

[147] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," *Computer Security–ESORICS 2012*, pp. 37–54, 2012.

[148] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Proc. 8th Australasian Symp. on Parallel and Distributed Computing-Volume 107*. Australian Computer Society, Inc., 2010, pp. 61–70.

[149] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of piggybacked mobile applications," in *Proc. 3rd ACM conf. on Data and application security and privacy*. ACM, 2013, pp. 185–196.

[150] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among android applications," in *Proc. 9th Conf. on Detection of Intrusions and Malware & Vulnerability Assessment*, 2012.

[151] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy, "Return-oriented programming without returns," in *Proc. of CCS 2010*, A. Keromytis and V. Shmatikov, Eds. ACM Press, Oct. 2010, pp. 559–72.

[152] L. Davi, A.-R. Sadeghi, and M. Winandy, "Ropdefender: A detection tool to defend against return-oriented programming attacks," in *Proc. 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 40–51.

[153] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "Design and analysis of a social botnet," *Computer Networks*, vol. 57, no. 2, pp. 556–578, 2013.

[154] F. Yamaguchi, F. Lindner, and K. Rieck, "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning," in *Proc. 5th USENIX conference on Offensive technologies*. USENIX Association, 2011, pp. 13–13.

[155] G. Zacharia, A. Moukas, and P. Maes, "Collaborative reputation mechanisms for electronic marketplaces," *Decision Support Systems*, vol. 29, no. 4, pp. 371–388, 2000.

[156] W. Viriyasitavat and A. Martin, "A survey of trust in workflows and relevant contexts," *IEEE Commun. Surveys & Tutorials*, vol. 14, no. 3, pp. 911–940, 2012.

[157] K. Govindan and P. Mohapatra, "Trust computations and trust dynamics in mobile adhoc networks: A survey," *IEEE Commun. Surveys & Tutorials*, vol. 14, no. 2, pp. 279–298, 2012.

[158] S. Saroiu and A. Wolman, "I am a sensor, and i approve this message," in *Proc. 11th Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 37–42.

[159] Q. Yan, Y. Li, T. Li, and R. Deng, "A comprehensive study for rfid malwares on mobile devices," in *5th Workshop on RFID Security (RFIDsec 2009 Asia)*, January 2009.

[160] S. S. Clark and K. Fu, "Recent results in computer security for medical devices," in *Wireless Mobile Communication and Healthcare*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 83. Springer Berlin Heidelberg, 2012, pp. 111–118.
[161] K. Fu, "Trustworthy medical device software," in *In Public Health Effectiveness of the FDA 510(k) Clearance Process: Measuring Post-market Performance and Other Select Topics: Workshop Report*. Washington, DC: National Academies Press, 2011.
[162] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, "Take two software updates and see me in the morning: The case for software security evaluations of medical devices," in *Proc. 2nd USENIX Workshop on Health Security and Privacy (HealthSec)*. USENIX Association, Aug. 2011, pp. 6–6.
[163] WhatsApp. (Visited March 2013) Legal info. [Online]. Available: http://www.whatsapp.com/legal/?l=en_en
[164] Google. (Visited May 2013) Google app engine. [Online]. Available: www.google.com/enterprise/cloud/appengine
[165] A. Distefano, G. Me, and F. Pace, "Android anti-forensics through a local paradigm," *Digital Investigation*, vol. 7, Supplement, pp. S83 – S94, 2010.

Guillermo Suarez-Tangil is a PhD student in the Computer Security (COSEC) Lab at Universidad Carlos III de Madrid, Spain. His research focuses on security in smart devices, intrusion detection, event correlation, and cyber security. He has participated in various research projects related to network security and trusted computing. He holds a B.Sc. and a M.Sc. in Computer Science from Universidad Carlos III de Madrid.

Juan E. Tapiador is Associate Professor of Computer Science in the Computer Security (COSEC) Lab at Universidad Carlos III de Madrid, Spain. Prior to joining UC3M, he was Research Associate at the University of York, UK. His work back there was funded by the ITA project (www.usukita.org), a joint effort between the UK Ministry of Defence and the US Army Research Lab led by IBM. His main research interests are in computer/network security and applied cryptography. He holds a M.Sc. in Computer Science from the University of Granada (2000), where he obtained the Best Student Academic Award, and a Ph.D. in Computer Science (2004) from the same university.

Pedro Peris-Lopez is Visiting Lecturer in the the Computer Security (COSEC) Lab at Universidad Carlos III de Madrid, Spain. He holds a M.Sc. in Telecommunications Engineering and a Ph.D. in Computer Science. His research interests are in the design and analysis of cryptographic protocols and primitives and in lightweight cryptography. His current research is focused on Radio Frequency Identification Systems (RFID) and Implantable Medical Devices (IMD). In these fields he has published many papers over the last years in specialized journals and conference proceedings.

Arturo Ribagorda is Professor of Computer Science at Universidad Carlos III de Madrid, where he also serves as Head of the Computer Security (COSEC) Lab in the Computer Science Department. He holds a M.Sc. in Telecommunications Engineering and a Ph.D. in Computer Science. He is one of the pioneers of computer security in Spain, having more than 30 years of R&D experience in this field. He has authored 4 books and more than 100 articles in several areas of computer, network and information security. He also serves as program committee member for several conferences related to cryptography and information security.