Powerslave: Analyzing the Energy Consumption of Mobile Antivirus Software

Iasonas Polakis[1], Michalis Diamantaris[2], Thanasis Petsas[2],
Federico Maggi[3(✉)], and Sotiris Ioannidis[2]

[1] Columbia University, New York, USA
polakis@cs.columbia.edu
[2] FORTH, Heraklion, Greece
{diamant,petsas,sotiris}@ics.forth.gr
[3] Politecnico di Milano, Milan, Italy
federico.maggi@polimi.it

Abstract. Battery technology seems unable to keep up with the rapid evolution of smartphones and their applications, which continuously demand more and more energy. Modern smartphones, with their plethora of application scenarios and usage habits, are setting new challenges and constraints for malware detection software. Among these challenges, preserving the battery life as much as possible is one of the most pressing. From the end users' perspective, a security solution, such as an antivirus (AV), that significantly impacts the battery's life is unacceptable. Thus, the quality and degree of adoption of malware-detection products is also influenced by their energy demands.

Motivated by the above rationale, we perform the first fine-grained measurement that analyzes, at a low level, the energy efficiency of modern, commercial, popular AVs. We explore the relations between various aspects of popular AVs, when handling malicious and benign applications, and the resulting energy consumption. Even though we focus on energy consumption, we also explore other dimensions such as the discrepancies between scanning modes, the impact of file size and scan duration. We then translate our findings into a set of design guidelines for reducing the energy footprint of modern AVs for mobile devices.

1 Introduction

The popularity of mobile devices has also resulted in them being heavily targeted by malware authors. Their built-in billing system, along with the plethora of personal information and account credentials that can be found on such devices, render them a highly profitable resource. According to practically every threat report (e.g., [24]), the amount of malware against Android, the most popular smartphone platform, is a real problem. The ample literature on the subject also confirms the importance of defending smartphones from these threats.

The vast amount of malicious software has resulted in most antivirus (AV) vendors releasing mobile versions of their software. Consequently, a wide range of free and paid solutions is readily available. Interestingly, several of the most

popular AV apps available on the Google Play Store are from new vendors [15]. Apart from the detection techniques and capabilities, the mobile nature of these devices introduces an important performance requirement for security solutions; energy consumption [12]. Understanding the energy consumption of a mobile security app is a key area of interest for end users, as well as developers [19]. An extra half hour of battery life really makes a big difference for the end user, and sets a strong decision boundary on whether or not to install a security app. Previous work investigated the energy efficiency of mobile browsers [25], among the most extensively used mobile apps, focusing on how the various components of a site (e.g., HTML, JavaScript) affect battery usage. Other work [26] explored whether battery usage can be leveraged as an early indicator of potential infection, despite previous work [13] demonstrating that a smart malicious app can circumvent such indicators by carefully diluting energy-intensive tasks over long time spans.

In this paper, we evaluate the energy efficiency of current AV tools on the Android platform, and explore whether the energy consumption is correlated to their detection quality. As the overall energy consumption of an AV is dominated by scanning benign apps during everyday use, a complete evaluation of the energy efficiency of AV engines mandates including benign apps as well. To this end, we conduct accurate and manually validated experiments on a real device, with a set of 250 malware samples and 250 goodware apps, on 6 of the most popular AVs. We measure the energy consumption during various scanning operations: (i) scanning the app upon installation, (ii) scanning the entire device, and (iii) scanning the SD card. We go beyond quantifying energy efficiency; we design and calculate metrics that quantify and "visualize" the relation between energy consumption and detection outcome. We break the energy consumption down to the energy consumed by each of the device's components. Finally, we provide a series of recommendations for designing the functionality and graphical interface of AV apps so as to minimize unnecessary energy consumption.

Our findings show that when handling benign apps, all but one of the AVs consume more energy than with malicious apps. One of the AVs consumes 8 times more energy, rendering usage under normal conditions considerably expensive in terms of energy. Our results also show that the most accurate AV is also efficient, consuming 46–65 % less energy than half of the AVs we tested. In most cases, size does matter, as larger apps will result in higher amounts of consumed energy. The AVs that rely heavily on hash-based signature matching techniques are an exception, and consume less energy for large apps compared to the others. The drawback of such AVs, however, is that their signatures can be easily bypassed. Furthermore, we find that design characteristics of the AV apps' graphical interfaces can significantly impact the overall energy consumption. Overall, the main contributions of this work are:

– We perform the first fine-grained, low-level study that quantifies the energy consumption of AV software in Android, which reveals the inefficiency of existing solutions and the impact on battery life.

– Our metrics and their visualization provide a detailed analysis of the energy consumption behavior of various aspects of AV software.
– We derive insights from our measurements and translate them into guidelines for the design of more energy-efficient security solutions.

2 Energy Measurements

The generation of energy-consumption traces that are accurate enough for our study require a framework that is sufficiently precise and fine grained, which will allow us to separate the consumption per application and per device component.

Software-Based Measurements. We reviewed, and tested when publicly available, several options that have been used in previous research. PowerTutor [29] works in user space and is designed to measure the cumulative energy consumed by a device within a time window. Unfortunately, this is not suitable for fine-grained measurements, i.e.,for measuring the energy consumption of each process or application individually. Any irregular, non-scheduled activity (that cannot be prevented or predicted due to the asynchronous nature of the Android runtime) may result in severely skewed readings. Moreover, the energy consumption readings returned by PowerTutor also contain the energy it consumes, which further skews results. Eprof [21] is an accurate fine-grained energy profiler for Android apps with routine level granularity. It is able to analyze the asynchronous energy state of an app, and model the tail-state energy characteristics of the hardware components appropriately. Unfortunately, this software is not publicly available. Yoon et al. proposed AppScope [28], which is designed to accurately estimate the energy consumed by each device component (CPU, display, wireless, GPS) per application and process, through monitoring of the Android kernel at a microscopic level. By following an event-driven approach that analyzes the traces of all system calls, along with any messages that pass through the Binder (used for inter-process communication in Android), AppScope is able to provide fine-grained readings at the process level. Additionally, AppScope is applicable to any Android device, without modification of system software, since it is implemented using a dynamic module in the Linux kernel. It has been calibrated for the power model of a specific device (HTC Google Nexus One), resulting in very accurate readings for experiments conducted on that device.

Hardware-Based Measurements. Many approaches have tried to perform accurate energy measurements on mobile devices based on readings from external hardware [8,9,11,23], e.g.,by using oscilloscopes and low current sensors. Even though such approaches can provide accurate energy measurements for the total power dissipation, none of them can provide fine-grained energy consumption information at an application or process level granularity.

For the aforementioned reasons, we selected AppScope for our experiments.

3 Experimental Setup and Datasets

Measurement Environment. AppScope best supports HTC Google Nexus One, as it is calibrated based on the power model of this specific device. We prepared our device by first unlocking the boot-loader through a standard rooting procedure, and installing the AppScope kernel module. Moreover, we implemented a custom component that parses the AppScope logs and extracts the energy samples for a specific process or app. During our experiments, the device had no SIM plugged in and WiFi was disabled (unless stated otherwise).

Measurement Unit. The power readings produced by AppScope every second are in units of Watts. Since a Watt expresses energy per time unit (Joules per second), we transform these samples to energy samples with $E(t) = P \cdot t$, as the goal of our experiments is to compute the total energy consumption of specific apps. In the rest of the paper, we use the energy unit in milli-Joules (mJ) to express the energy consumption of an app.

Measurements Automation. We automate our measurements through user interface (UI) actions, using the MonkeyRunner [1] API to produce specific actions (e.g., tapping on UI elements) on the device. We can perform actions such as installing-uninstalling an APK, rebooting the device, starting-stopping a scan. For this, we identify the main activity name of each AV and extract the package name of each malware, from their bytecode representation, obtained through reverse engineering using the `apktool` [3].

Antivirus Apps. We evaluate the 6 AVs listed in Table 1, which we obtained from the Google Play Store, on August 1st 2013. Most of the AVs are in the top ten list, with downloads ranging from 100K to 500M, and we denote their relative popularity within our selected set; as an exact number is not provided, but rather a range of downloads, two apps are tied for 1st and two for 4th place. The "Signature-based" column indicates whether the AV relies mainly on easily evaded hash- or package-name-based signatures for detecting malware. We provide more details on this aspect of the AVs in Sect. 4.2.

Even though some AVs offer in-the-cloud scanning (NQ and Sophos state it explicitly), our goal is to explore the energy consumption of the actual

Table 1. List of AV products tested in our study. Rank refers to the respective rank among the tested AVs based on their downloads in Google Play.

Product	Rank	Downloads	Updated	Signature-based
AVG Antivirus Free (2.12.3)	1	100–500M	05/03/13	✓
Symantec Norton Mobile (3.5)	4	10–50M	25/06/13	✗
Dr.Web (8.0)	3	50–100M	27/02/13	✗
Avast Mobile (2.0.4)	1	100–500M	09/05/13	✗
Sophos Mobile (2.5.1009)	6	100–500K	09/04/13	✓
NQ Mobile (6.8.6)	4	10–50M	21/05/13	✗

offline detection process. In addition, uploading times can significantly fluctuate depending on network conditions, causing non-negligible deviations on the usage and, thus, energy consumption of the wireless module. Last, we wanted to maintain the malware definitions stable for the duration of the experiments. Thus, we disabled Internet access to avoid updates, and selected AVs that also work in an offline mode. In Sect. 4.6 we provide preliminary insights on the influence of WiFi connectivity as a concluding experiment. Given the complexity introduced by this variable, our results pave the way for further research.

Malware Dataset. We select a set of 250 malware samples, collected from the Contagio Mini Dump [4] and the Android Malware Genome Project [2]. Note that, as the size of a malware sample may influence the overall energy consumption during the scanning process, we opt for a collection with a variety of sizes, ranging from 12.9 KB to 24.2 MB. The overall detection capability of each AV is reported in Table 2.

Goodware Dataset. We collected a benign set consisting of 250 apps from the Google Play Store. Specifically, we selected apps from the list of the most popular in the Google Play Store. The size of the apps ranges from 92.1 KB to 40.2 MB. Again, Table 2 summarizes the overall detection outcome.

Device Setup. Based on our requirements and experimental parameters, we use the following device setup for our experiments:

- *Snapshots.* We removed unnecessary, pre-installed apps (e.g., Youtube) and created a backup of a clean state of the device (one with each AV installed), containing AppScope components for determining the energy consumption.
- *Revert.* After each malware sample is installed and scanned, we revert to the clean state and erase the SD card.
- *Run.* We repeat each test five times in a row, and calculate the average energy consumed per device component (e.g., CPU, display) and time taken.
- *Luminosity.* We set the display luminosity to a fixed amount.

Measurement Methodology. Malware can infect the device internal storage (e.g., via simple copy operation during installation) or the external SD card (e.g., via simple copy operation). The per-app isolation security model of Android is such that an app—including AVs—can only read certain portions of the filesystem, unless granted root privileges. Regardless of the permission, no AV can possibly perform dynamic analysis of (malicious) apps at runtime, as Android exposes no system-level auditing API at the app level. The SD card filesystem is universally readable. As a result, AVs are designed to look for malware using a combination of the following access modes. As explained in [15], the *On Demand* mode scans the readable portions of the filesystem (internal or external), whereas in *Upon Installation* mode, the AV scans any APK being installed. Based on the these observations, we conduct two different experimental procedures, which we automate via MonkeyRunner:

1. *Device (Internal):* We install each APK individually and scan the app installed on the device's internal storage through two methods:

    (a) On Demand. After installing an APK, we perform a full scan of the device.

    (b) Upon Installation. After installing an APK, the AV automatically scans the installed app (e.g., with a `PACKAGE_ADDED` broadcast receiver).

2. *SD Card (External):* Copying (via `adb push`) all the APKs to the SD card, without installing them, and then scanning the entire SD card partition.

We extract the following data: (1) *time* taken to perform a scan, (2) *energy consumed* by the specific process for each device's component, and (3) whether or not the AV identified the malware or goodware correctly.

4 Experimental Results

We explored, with specific experiments, the relation between energy consumption and scan duration (Sect. 4.1), detection outcome (Sect. 4.2), on-demand or upon-installation scan (Sect. 4.3), app size (Sect. 4.4), use of display (Sect. 4.5), and, preliminary, the usage of Internet connectivity (Sect. 4.6).

4.1 Energy Consumption vs. Scan Duration

Once the scanning has completed, the AVs include a message in Android's logging system (which we fetch through `logcat`), enabling us to calculate the exact duration of the scan. Table 2 shows the aggregated results for each AV, and reports the minimum and maximum scan duration for both goodware and malware. The scan duration coincides fairly accurately with the aggregate energy consumption, with a few exceptions; Norton has a comparatively small duration for the energy it consumes, while Dr. Web exhibits the opposite behavior with a relatively small consumption and a long duration.

    We present the duration statistics for the "On Demand" scan and not the "Upon Installation". Whenever an app—during installation—is flagged as a threat, the AV produces feedback (i.e., a pop-up). This enables us to check at which point the AV has finished scanning the app, and calculate the duration. However, we can not extract the time for false negatives, because there is no feedback. One could count the number of seconds for which AppScope returns an energy consumption reading since the installation. This, however, only allows readings at a granularity of seconds, which can introduce significant error in these experiments. For example, if a scan lasts 3.1 s, we would retrieve readings for 4 s from AppScope, and calculate a significantly skewed energy consumption rate. As such, despite the "On Demand" scan lasting significantly longer than the "Upon Installation", it is useful for comparing the AVs in terms of time performance.

4.2 Energy Consumption vs. Detection Outcome

Malware. We explored the relation between energy consumption and the detection outcome. Our experiments show that there is not a restricting correlation

Table 2. Detection outcome for all scanning modes, and duration of the "On Demand" scan (Sect. 4.1). The "Correct" and "Error" columns refer to correct and erroneous labelling by the AV for the apps being scanned. The "Energy Penalty" expresses the percentage of energy wasted due to the AV misclassifying a malicious app as benign.

AV	Detection rate [%]				Energy Penalty [%]	Scan duration [s]			
	Malware		Goodware		Malware	Malware		Goodware	
	Device	SD	Device	SD		Correct	(Error)	Correct	(Error)
Norton	98.8	98.8	4.8	0.0	+20.00	22–28	(22–28)	23–31	(23–28)
Avast	98.0	98.0	0.0	0.0	+6.500	30–34	(33–36)	23–38	-
Sophos	97.2	97.2	0.0	0.0	+1,023	18–38	(18–24)	11-27	-
Dr. Web	96.4	96.0	1.2	1.2	+31.40	30–107	(30–43)	30–169	(33–80)
NQ	95.2	95.6	0.0	0.0	+49.00	26–50	(23–53)	25–54	-
AVG	94.8	72.0	0.0	0.0	−73.90	9–13	(10–13)	9–13	-

between the two, as can be seen in Fig. 1(a). For example, while AVG is the most energy-efficient and also achieves the lowest detection rate, we find that the most effective AV engine in terms of detecting malicious apps (Norton) is also highly efficient in terms of energy consumption. Specifically, while achieving a 98.8 % detection rate, it consumes 46.2 %–65 % less energy than three of the other AVs. Thus, our findings show that security vendors have the potential to create AVs that *achieve high detection rates, while maintaining reasonable energy consumption profiles.* Sophos exhibits the highest consumption of all, consuming 22.2 % more energy than Avast that has the second highest consumption. Compared to the remaining apps, Sophos consumes 53.8 % more than NQ, 185 % more than Norton, 215 % more than Dr. Web, and over 326 % more than AVG that is the most efficient.

Signature-Based Heuristics. As the type of heuristics employed by an AV can impact certain aspects of its energy consumption, we run the following experiment for identifying the AVs that rely heavily on signatures that detect specific package names or hash values. We altered the package name (and thus the file hash) of the malware samples using an automated process built on `apktool`, and repeat the scans for all the AVs. Sophos failed to detect a single sample, indicating that the effectiveness of its detection engine relies significantly on signatures that can be trivially evaded, all the while exhibiting the highest energy consumption out of all the AVs. AVG is also severely impacted, detecting only 30.2 % of the malware samples. As can be seen in Table 1 the remaining AVs were not impacted, and achieved the same detection rate as before.

Note that there are several ways to evade signatures and, in general, to stress test the obfuscation resilience of AVs, as demonstrated in previous work [22,30]. This specific point falls out of the scope of our paper, as our goal was to explore the correlation, if any, between signatures sophistication and energy consumption.
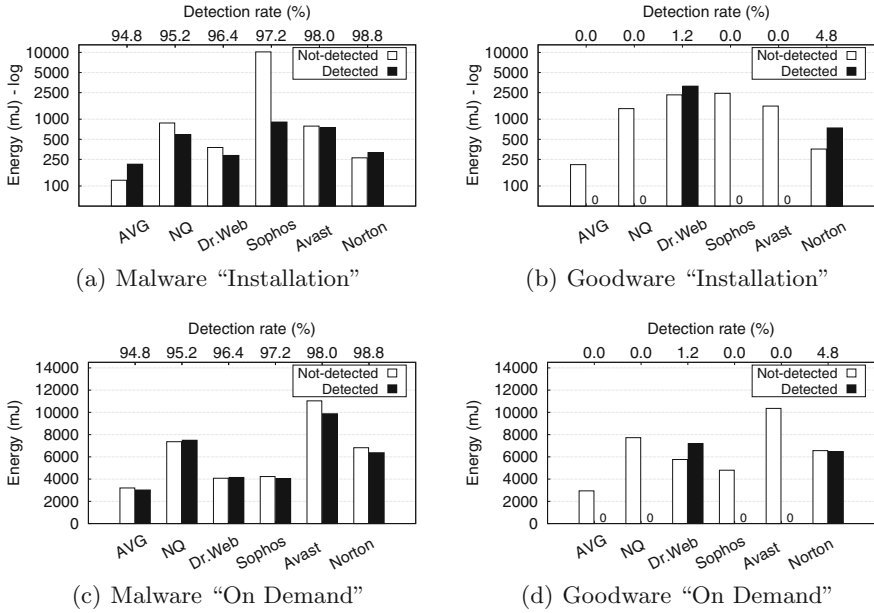
(a) Malware "Installation"

(b) Goodware "Installation"

(c) Malware "On Demand"

(d) Goodware "On Demand"

Fig. 1. Aggregate energy consumption by CPU during the "Upon Installation" and "On Demand" scan (Sect. 4.2).

Goodware. The majority of apps installed by users are benign. As such, a significant aspect of the energy consumption of AVs is how they handle such apps. When scanning an app, if one of the heuristics is triggered, the AV can flag the app as malicious and save energy by not executing the remaining heuristics. On the other hand, when scanning a benign app, no heuristics will be triggered (unless there is a false positive), resulting in all of them being executed, which will also result in a higher energy consumption.

Figure 1(a) and (b) shows the results for the CPU energy consumption for both app collections during the "Upon Installation" scanning. Our results demonstrate that most AV engines exhibit this behavior, apart from AVG where the energy consumption is approximately the same for malware and benign apps. Dr. Web presents the largest increase, with 8 times more energy consumed when scanning goodware. The higher average consumption for the 3 falsely detected apps, compared to the other benign apps, is in part due to their large size which affects the scanning process of Dr. Web as we discuss later on (we found that when scanning apps of comparable size, benign ones tend to have a higher consumption).

NQ, Sophos and Avast consume 64 %–169 % more energy for goodware. Norton is the most efficient with a 12.9 % increase, which may be correlated to the high false positive rate: in an effort to maintain a low energy consumption, they may employ a more "superficial" heuristic that is prone to false positives.

Table 3. Detection outcome and energy consumption of the seven malicious apps not detected by Sophos (Sect. 4.2).

Sample	Size	Energy Consumption (mJ)					
		Sophos	AVG	NQ	Avast	Dr. Web	Norton
thespyja	15.5 KB	9,748.3	102.4	612.9	202.1	314.3	111.9
vending.sectool.v1	96.37 KB	9,676.1	196.4	543.2	580.8	87.9	254.8
androiddefender	1.5 MB	9,834.5	184.4	895.9	568.4	384.7	211.3
carrierig.trial	1.69 MB	9,825.4	96.6	717.9	894.1	585.6	253.2
dropbox.android	3.03 MB	12,410.2	232.3	787.5	738.1	580.7	468.7
apps.DWBeta	4.28 MB	9,838.2	298.1	580.5	664.2	363.3	573.9
appgame7.candystar	5.76 MB	9,942.3	99.2	2,182.5	1,540.4	779.4	520.7

not detected — detected

Application Whitelisting. We explore whether any of the AVs employ whitelisting for popular apps (i.e., if really popular apps are matched against a simple signature and are not scanned in depth). We select the 10 most popular apps (e.g., Instagram, WhatsApp) and repeat the previous experiments. Results are in line with our previous measurements for each AV, with the app sizes once again affecting the energy consumption for certain AVs. There were no instances of an app being processed in a significantly faster manner or consuming less energy, indicating the lack of some form of whitelisting.

False Classification. We calculate the penalty, in terms of "wasted" energy consumption, that occurs when the AV misclassifies a malicious app as benign. As can be seen in Table 2, AVG is the only AV engine to actually benefit in terms of energy consumption when failing to detect a malicious app, with a 73.9 % decrease in consumption. While consuming more energy is, obviously, a negligible side-effect of a malicious app going undetected, in our case it serves as a potential indicator of instances where an AV exhibits strange behavior.

Indeed, one can see that Sophos exhibits a strange behavior and has the largest increase with 1023.8 % compared to the malware samples that are detected, and has a consumption 4–6 times larger than when scanning benign apps. As shown in Table 3, all of the malware samples have a significant impact on Sophos in terms of the amount of energy consumed when scanning them. We also explore the effect of those seven malicious apps on the other AV engines. Although 3 samples are not detected by any of the AVs, 2 of them were detected by all but Sophos. None, however, present the same effect on all of the remaining AVs. While some result in higher consumption than the average for the specific AV, they do not present a consistent effect on all AVs. This indicates that this is most likely caused by the internal implementation of Sophos, and not the malware employing detection or analysis evasions technique.

Regarding the undetected samples, we found that `dropbox.android` and `thespyja` remain undetected by several AV engines, as shown by popular online

scanning services. Indeed, `dropbox.android` was detected by only one AV engine (out of 52 in VirusTotal, and 7 in AndroTotal [15]), while `thespyja` was detected by 11 (∼21 %) in VirusTotal and 2 (∼28 %) in AndroTotal, which are still quite low. This could be attributed to AVs intentionally not labelling a sample as a threat after some time. However, in the case of the third sample (`androiddefender`), VirusTotal and AndroTotal report detection by 61.5 and 85.7 % of the AVs respectively.

To gain a better understanding of the behavior of `thespyja`, which has a size of merely 15.5 KB and is characterized as Spyware, we reverse engineer the app. We first obtain the Java bytecode using `apktool` and then get the actual source code (Java files) using `dex2jar` [5] and `JD-GUI` [6]. The source code is constituted by only two Java files, one Android Activity file and a graphical layout file. By inspecting the code of the activity file we found that the app's logic is simplistic. It opens a web view and transfers the user to a web page. Evidently any malicious action occurs when the user visits the page. The app does not contain any other functionality and only requires the INTERNET permission.

4.3 Upon Installation vs. on Demand Detection

Figure 1(c) and (d) shows the average energy consumed by the CPU for each AV when scanning the whole device "On Demand". As expected, the "Upon Installation" method consumes less energy than the "On Demand" method as it only scans the app being installed and not the entire device. Sophos exhibits the smallest increase for the "On Demand" consuming 4.5 times more energy for malware and 2 times more for benign apps. Norton has the largest increase, with a 20 times larger consumption for malware and 18 for goodware. The remaining AVs have an increased energy consumption by 12–14 times for malware and 3–6 for benign apps. AVG and Sophos remain fairly consistent, with the energy consumption being the same whether an app is malicious or benign, for both the "On Demand" and "Upon Installation" scanning modes. This can be attributed to our finding that both AVs rely heavily on hash-based signatures for detection. Overall, the "On Demand" scan proves to be a very expensive activity in terms of energy consumption, even when not taking into account the energy consumption from the device's display.

4.4 Size Does Matter

The app's size may affect the energy consumption of the AV, depending on the type of heuristics employed. One would expect larger apps to result in longer scans and increased consumption. However, as our experiments show, that is not always the case. Figure 2 plots the aggregate energy consumed by each AV, and the size of the scanned app. Dr. Web clearly exhibits a strong correlation between the two, and Avast mostly for benign apps. For the remaining AVs size does not seem to be a dominating factor. The lack of correlation may be attributed to fingerprinting techniques that create a compact signature without processing the entire file, or may leverage some indexing technique to speed up
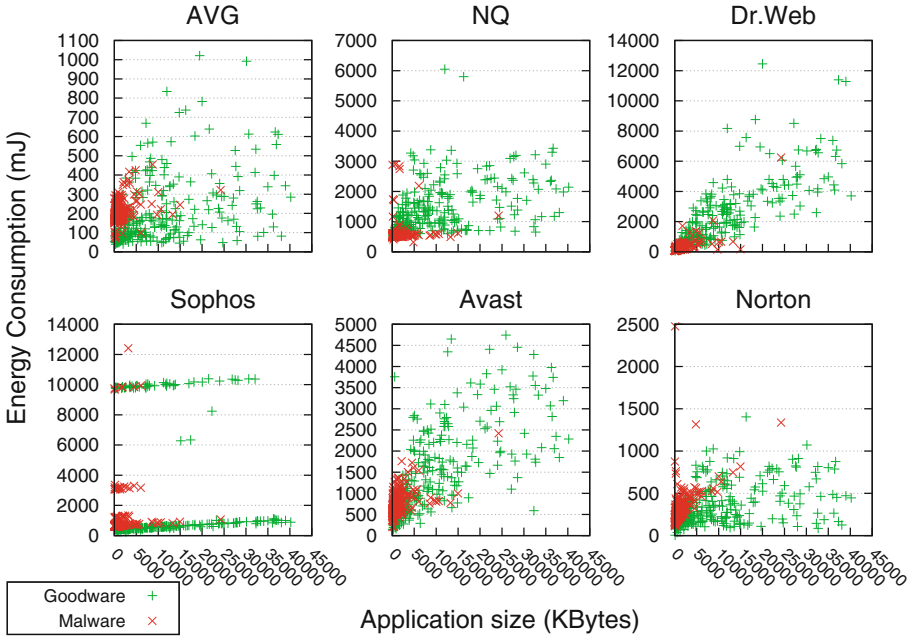
Fig. 2. Size and aggregate energy consumption for "Upon Installation" scan (Sect. 4.4).

matching. Sophos presents no correlation at all, as there are three clusters of consumption, which remain consistent regardless of app size. The three clusters of consumption suggest three sets of heuristics and termination of the scan upon detection by a set of heuristics.

To verify our analysis, we also calculate the Pearson's correlation coefficient, a value between $[-1, 1]$, between app size and energy consumption; 1 denotes perfect linear correlation between two variables, 0 denotes no correlation and -1 shows total negative correlation. Table 4 shows that for Dr. Web we have a correlation value of over 0.7 for both benign and malicious apps verifying our previous observations. Avast also demonstrates a strong correlation for benign apps but not for malicious ones. Furthermore, Sophos presents almost no correlation with 0.012 for benign apps and 0.098 for malware. Norton is the only AV to exhibit a higher correlation for the malicious apps compared to the benign, suggesting extensive heuristics that analyze the app, justifying its high detection rate. Interestingly, the two AVs with the lowest correlation between size and consumption, namely AVG and Sophos, are also the two apps that rely heavily on package name or hash-based signatures for detection.

To further explore the correlation, we plot the duration of the "On Demand" scan for Dr. Web, correlated to the app's size in Fig. 3. While it is evident that the size has an impact on the scan's duration, it is not the only factor. Whereas in the case of malware the duration is fairly comparable for samples of a similar size (apart from a few exceptions), things are more complicated for benign apps,

Table 4. Correlation between energy consumption and size (Sect. 4.4).

Antivirus	Pearson's correlation coefficient	
	Goodware	Malware
AVG	0.402	0.349
NQ	0.420	0.085
Dr.Web	0.793	0.721
Sophos	0.012	0.098
Avast	0.673	0.493
Norton	0.402	0.510

(a) Malware   (b) Goodware
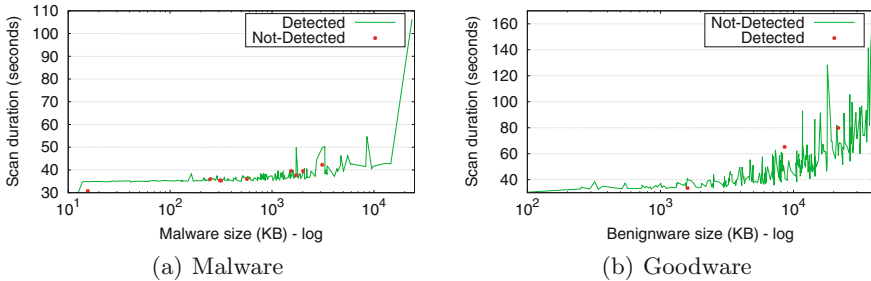
Fig. 3. Duration of the scan vs. size of the app for Dr. Web (Sect. 4.4).

with large fluctuations for almost identical sizes. In one case, an app with a 0.3 %
increase in size from the previous app, presents a 150 % increase in duration.
Thus, the scanning process can be greatly influenced by the internals of each
app and the type of code and system calls included.

4.5 Display vs. CPU Energy Consumption

The energy consumption of a display is not determined only by the choice of col-
ors. It is also heavily influenced by the amount of text, animations, and the gen-
eral design of the layout. We found that while some of the AVs we tested adopt a
darker design which consumes less energy [10], they consume more energy than
others that use white but follow a more minimalistic design. We were not able
to collect information about the energy consumption of the display for NQ, as
AppScope is not able to collect that information for the specific app, due to a
bug of the app.

Figure 4(a) shows the average energy consumption per second for the display.
Results exhibit a significant disparity between certain AVs. Specifically, Dr. Web
is the least efficient, consuming 25.4 % more energy than AVG that is the most
efficient. The app's energy consumption behavior is completely different for the
CPU and display, as Dr. Web was the second most efficient for the CPU experi-
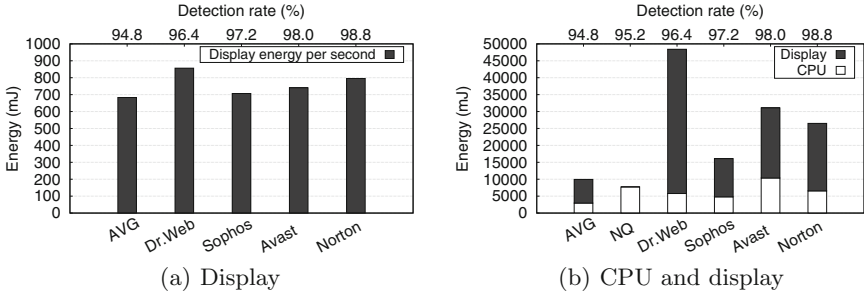ments, while AVG was the second least effective. Figure 4(b) shows the aggregate

(a) Display   (b) CPU and display

Fig. 4. Average energy consumed per second during "On Demand" scan (Sect. 4.5).

consumption of the CPU and display. The apps' consumption is heavily dominated by the display, with the CPU accounting for 11.9 %–33.3 % of the overall consumption, leaving room for significant improvement. The aggregate energy consumption during the "On Demand" scan by Dr. Web is 5 times larger than AVG. This is due to both the scan duration being comparatively much longer, as well as specific design aspects of the Dr. Web app that we describe in Sect. 5.

Note that during "Upon Installation" scans, the AV runs in the background, and does not use the display. Consequently, the contribution of the AV to the display energy consumption is dominated by the foreground app, not by the on-screen notifications—visualized only in cases of detected malware. Thus, we measured the energy consumption for the "On Demand" scan only, which keeps the AV's view in the foreground, and provides a better baseline for comparison.

Impact on Battery Life. We calculate the effect of an "On Demand" scan (CPU and display) on the battery life for each AV, i.e., what percentage of the battery is consumed by a single scan. We calculate a lower bound based on the value for the specific battery (the value will be higher for older batteries with decreased duration). Dr Web presented the largest impact by consuming 0.19 % of the total capacity of the battery, whereas AVG has the smallest impact with 0.05 %. The remaining AVs consume 0.10–0.18 % of the battery's energy.

4.6   Internet Connectivity (WiFi)

We conducted a preliminary experiment to investigate the impact of Internet connectivity during the "Upon Installation" scans of the 250 malicious samples. Even though such a study needs a large number of runs to eliminate any discrepancies in the energy consumption readings due to unpredictable external factors (e.g., network load, uploading times), our goal was to test whether the detection rates were affected, and to provide an estimation of the consumption. Our experiments reveal some interesting results.

Detection Rate. Given the availability of cloud-based scanning, we expected much higher detection rates. However, that was not the case. Specifically, we turned on the WiFi antenna, which triggered a signature update for Dr. Web,
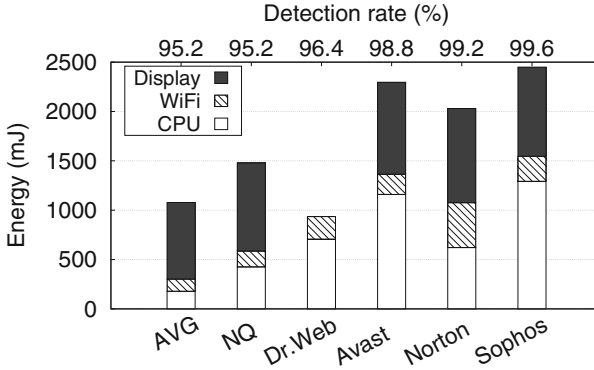
Fig. 5. Aggregate energy consumed by CPU, WiFi and display for malware detected "Upon Installation" (Sect. 4.6).

Sophos, Norton and AVG (experiment conducted late May 2014). Even though we recorded slight improvements in Sophos (+2.4 %) and Norton (+0.4 %), still none of the AVs was able to reach 100 % detection rate, verifying previous results [18]. After this upgrade, we recorded no further increase, neither due to subsequent upgrades, nor in-the-cloud scanners.

Energy Consumption. Figure 5 shows the average consumption per component for "Upon Installation" scans, calculated only for the malware samples for which each AV used the network. The display's energy consumption is from the pop-up informing the users of the detected threat. Dr.Web uses the Android notification system and, thus, presents no consumption for the display. Our results show that the energy consumption for the WiFi component is always smaller than the other components. There is also significant deviation between the WiFi energy consumption of each AV. Norton has the highest consumption, being 79 % higher than that of Sophos. As further motivated in Sect. 6, these findings support our initial decision of conducting our measurements offline, and to postpone an in-depth exploration of WiFi connectivity for future research.

5 Efficiency Guidelines

Here we present a series of guidelines for designing more energy-efficient AVs, based on the insights from our results. Most AVs adopt some of the techniques we propose, indicating that efficient power consumption is an important and relevant aspect of their design. All of them, however, could significantly improve their power consumption behavior by following the guidelines we propose.

5.1 Detection Heuristics and Behavior

Our experimental findings negate concerns of a strong correlation between security and efficiency. Norton had the highest detection rate, yet was quite efficient

Table 5. Visual characteristics of each AV (sorted in descending order of consumption).

Antivirus	Colors	Design	Text	Animation
AVG	White	Minimal	Scarce	✕
Sophos	White	Minimal	Scarce	✕
Avast	Grey/Orange	Normal	Substantial	✓
Norton	Dark/Blue	Normal	Substantial	✓
Dr. Web	Green/Orange	Heavy	Excessive	✓

in terms of energy consumption. We suggest the adoption of the following optimizations for further reducing the power consumption of the scanning process:

– *Early termination:* Norton and AVG consume essentially the same energy for detected malware and undetected benign apps, suggesting that all the heuristics are executed in both cases instead of stopping after a heuristic is triggered. The remaining AVs present lower consumption for detected malware, indicating that this is a feasible technique. While running all the heuristics may in principle help minimize false positives, our experiments demonstrated that 3 of the 4 AV apps that adopt an "early stop" strategy do not present any false positives, while Norton has a 4.8 % false positive rate.
– *Whitelisting*[1]: whitelists can be used to avoid scanning known benign apps. The AV will have a list of signatures for popular benign apps. If the user installs an app contained in the list, the AV simply has to match it to the signature and can avoid all the intensive, energy consuming heuristics that analyze the app in depth. The list should be updated periodically to include the latest versions of apps, and add new and popular apps that are released. Such a mechanism does not affect the detection accuracy of the AV. Even if a malware author releases a sample that exhibits a dual behavior, as demonstrated in [27], if the sample has been previously detected by the vendor it will not be whitelisted but detected as a threat. If the sample has not been previously detected, the whitelisting does not change the outcome, as it would have not been labelled as a threat regardless of the whitelisting. Furthermore, the whitelisting should only be employed for very popular apps, rendering the scenario of a whitelisted malware sample unlikely.
– *Background tasks:* Run "On Demand" scans in the background to minimize the rendering of visual components, as is done for the "Upon Installation" scans, which leverage the notification system when necessary.

5.2   Visual Design

While not pertinent to the security aspect of our study, we also provide a series of guidelines regarding the design of the apps, for reasons of completeness.

---

[1] Not to be confused with application whitelisting (e.g., [7]), where only known applications are allowed to be executed on the system.

Our results revealed that the energy consumption of an AV is largely affected by the display. Minimizing the energy consumption of the display will significantly reduce the app's consumption and minimize the impact on the battery's life.

Table 5 provides details of the visual design of each AV, which directly affect the energy consumption. Most of the design elements of the apps we tested could easily be changed towards minimizing the consumption. Even though appearance is an important aspect of apps, which can affect their popularity, AVs present a "special" case, because their success is based on their functionality (i.e., good detection accuracy, low CPU load and energy consumption), rather than their look and feel. This, of course, is more constraining for other categories, such as websites [25], as they have to be visually appealing and entice users. On the other hand, the popularity of AVs is primarily affected by their ability to detect malware, regardless of their visual design. This could be further supported by the fact that AVG, which is one of the two most popular AV apps, is also the most energy-efficient one. Nonetheless, even though AVG follows the most "clean" visual design, it also relies heavily on bright colors, significantly draining more energy than dark ones. Overall, taking into account the current hype about "green" or "sustainable" designs, following our guidelines and advertising its power efficiency, may further boost the popularity of an AV app.
We propose the following guidelines in terms of visual design:

– *Colors:* Maximize the dark areas (e.g., backgrounds). In OLED displays black pixels are produced by turning them off, which greatly reduces consumption.
– *Text:* Reduce text to bare essential (e.g., result of analysis).
– *Design:* Follow minimalistic and clear designs. Animations and "heavy" designs unnecessarily increase energy consumption.

Note that the default theme for system apps (e.g., system settings) in vanilla Android is dark and minimalistic. Indeed, development guidelines encourage the adoption of built-in widgets, which ensure a memory-efficient, fast and sleek UI.

6   Limitations and Future Work

Multiple Devices. All experiments have been conducted on a specific smartphone model, as AppScope's power consumption model has been calibrated for this specific device. Building models for AppScope and calibrating them for other devices is out of the scope of this work. While the power consumption of these AVs might present differences on other devices (e.g., devices with larger displays), their inherent behavior will remain the same across devices. Furthermore, our study is comparative, and all measurements are completed on the same model. Thus, the insights we derive are not bound to the specific device, and our design guidelines will improve energy consumption regardless of the device.

WiFi. Apart form the preliminary WiFi experiment, our study has been conducted with the WiFi connectivity disabled. Several reasons guided that choice, with the inability to maintain all aspects of the experiments constant for the

entirety of their duration being the most significant. Fluctuations of uploading times and differences in network loads can significantly alter the duration and, thus, energy consumption of the experiments. Furthermore, the apps may "silently" update certain aspects of their functionality or signature database. As the experiments are conducted on a real device and not within a simulator, reverting the device to a clean state after every sample installation significantly increases the duration of each experiment. Taking into account the number of apps explored, and the multiple iterations per sample, it is possible that before a set of experiments has completed, the AV engines might have updated, thus creating different "testing environments" within a single experiment. As such, we plan to explore methods to enable us to include WiFi connectivity, while maintaining a constant testing environment across all experiments.

7 Related Work

Battery-Based Malware Detection. Hoffmann et al. [13], proposed the idea that every action performed drains a specific amount of energy from the battery. As such, they performed tests to check whether they could analyze the behavior of benign and malware applications in terms of energy consumption. They used PowerTutor to conduct their experiments. First they conducted tests on different components (CPU, Display, 3G, WiFi) on a clean device and compared the results with experiments on 2 malware samples. According to their results, they could not identify the malicious activity from the energy traces due to the high "noise" ratio. All the experiments, were conducted on two different Android devices.

Merlo et al. [17] contradicted the conclusions drawn in [13], arguing that it is possible to identify malicious activity by monitoring the energy consumption of a device. To show the correctness of their proposal, they analyzed two different approaches to the problem. First, with a high level approach similar to [13], they attempted to measure the energy consumption based on hardware resources usage counters. They found that this approach provides inaccurate results, not capable of identifying a crafted network attack. Then, with a low level approach based on energy measurements at the battery level, they found the produced results reliable for identifying the aforementioned attack. However, this approach induces a high level OS intrusiveness that considerably limits its portability.

Additionally, Truong et al. [26] recently found that the in-the-wild malware is not sophisticated enough, or simply does not bother, to hide the extra energy consumption. To this end, they used MIT's Carat infrastructure to conduct a large-scale infection-rate measurement on 55,000 real devices.

In an older study, Hahnsang et al. [14], developed a framework for detecting energy-greedy anomalies based on power measurements, performed on a device running Windows Mobile OS. Their framework consists of two basic components: a power monitor that collects power samples of a candidate application and a data analyzer that generates power signatures from those samples. The detection is performed through comparing the generated signatures with a predefined database. The framework was evaluated on custom worms and proven

to successfully detect samples sharing common behavior with known malware variants, as well as applications exhibiting unknown energy-greedy anomalies.

Previous studies [16, 20] have focused on the detection of battery exhaustion attacks on mobile devices. In [16], Marting et al., presented a list of methods that an attacker can use to drain the battery of a device, along with a power-secure architecture to thwart those attacks. In [20], Nash et al., proposed an IDS for battery exhaustion attacks based on power consumption estimations as derived from several parameters of the system, such as CPU load and disk accesses.

Security vs. Energy Efficiency. Bickford et al. [8] studied the potential trade-offs between security monitoring and energy consumption for a specific class of malware detectors: rootkit detectors. They propose a framework to asses security versus energy tradeoffs along two axes, attack surface and malware scanning frequency. Their results demonstrate that protection against code-driven attacks is relatively cheap, while protection against all data-driven attacks is prohibitively expensive. Their study was performed on a mobile device running a version of Linux, and was limited for a specific class of malware (rootkits) which currently does not constitute a threat for mobile devices. In contrast, our study was performed on a mobile device running the Android OS and we use popular commercial AVs and real malware samples which pose a significant threat.

Android AV Evaluation. Rastogi et al. in [22], performed an evaluation, in the most popular AVs, regarding malware evasion techniques. They developed DroidChameleon a framework which transforms malware from different malware families by changing the package name, renaming the files and the identifiers, reordering code etc. They tested the new transformed malware samples in ten AV products. Results indicate that all the tested AV products are susceptible to common evasion techniques. In a similar work [30], Zhen et al.present an automated and extensible platform, called ADAM, which evaluates the detection of Android anti-malware tools through different static transformation techniques. applied on a number of malicious apps. They tested their repackaged malware samples on a popular analysis service and found that the detection rates dropped compared to the original samples.

8   Conclusion

Battery technology seems unable to keep up with the rapid evolution of smartphones and their applications, which demand more and more energy. The popularity of external power banks and slim batteries camouflaged as covers are a sign that smartphone users do need extra power for their devices. In a similar vein, the rise of applications that promise to make the battery draining slower (e.g., by killing processes) show that users are concerned about this aspect. Security is generally a cost, and in the case of mobile AVs, the risk is that users may decide to uninstall an AV because it drains too much battery. In other words, the less energy an AV consumes, the more usable it is.

Research efforts such as the one presented in this paper are preparatory for building design methodologies and tools that enable application vendors and

developers to balance the trade off existing between security effectiveness, user experience and speed. We derived our design guidelines from thorough measurements obtained on a real-world device running the most popular mobile operating system and, thus, we believe they offer an accurate reference for practitioners and future researchers.

In particular, our measurements show that there are significant deviations of CPU energy consumption among different AV products. Interestingly, the energy consumed by the CPU due to the AV is 3–4 times lower when compared to other hardware components, which are energy greedy. This means that there is room for increasing the detection accuracy, for example by running more energy greedy algorithms. Indeed, we found that no correlation exists between the detection accuracy and energy consumption of AV applications on mobile devices. Moreover, in most AV products a pattern can be seen: the bigger the size of an application, the higher the energy consumption required for scanning it, but that depends on the AV and the specific scanning technique. Clearly, energy efficiency and complex detection algorithms can coexist only if the UI designers are cautious and create UI components that minimize the illumination of the pixels. Indeed, the display is the most energy greedy component.

Based on our findings, we argue that security vendors are not bound by an "Efficiency vs Accuracy" trade off, and can produce antivirus apps that are effective while maintaining modest energy consumption profiles.

References

1. Android developers - monkeyrunner. http://developer.android.com/tools/help/monkeyrunner_concepts.html
2. Android malware genome project. http://www.malgenomeproject.org/
3. Apktool. https://code.google.com/p/android-apktool/
4. Contagio - mobile malware. http://contagiominidump.blogspot.com
5. Dex2jar. https://code.google.com/p/dex2jar/
6. Java decompiler. http://jd.benow.ca/
7. Beechey, J.: Application whitelisting: Panacea or propaganda (2010). http://www.sans.org/reading-room/whitepapers/application/application-whitelisting-panacea-propaganda-33599
8. Bickford, J., Lagar-Cavilla, H.A., Varshavsky, A., Ganapathy, V., Iftode, L.: Security versus energy tradeoffs in host-based mobile malware detection. In: MobiSys (2011)
9. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: USENIX ATC (2010)
10. Chen, X., Chen, Y., Ma, Z., Fernandes, F.C.A.: How is energy consumed in smartphone display applications? In: HotMobile (2013)

11. Friedman, R., Kogan, A., Krivolapov, Y.: On power and throughput tradeoffs of wifi and bluetooth in smartphones. In: INFOCOM (2011)
12. Harlalka, R.: How to stop your mobile app from being a serious battery drain (2013)
13. Hoffmann, J., Neumann, S., Holz, T.: Mobile malware detection based on energy fingerprints — a dead end? In: Stolfo, S.J., Stavrou, A., Wright, C.V. (eds.) RAID 2013. LNCS, vol. 8145, pp. 348–368. Springer, Heidelberg (2013)
14. Kim, H., Smith, J., Shin, K.G.: Detecting energy-greedy anomalies and mobile malware variants. In: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, MobiSys 2008 (2008)
15. Maggi, F., Valdi, A., Zanero, S.: Andrototal: a flexible, scalable toolbox and service for testing mobile malware detectors. In: CCS SPSM (2013)
16. Martin, T., Hsiao, M., Ha, D., Krishnaswami, J.: Denial-of-service attacks on battery-powered mobile computers. In: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications(PerCom 2004) (2004)
17. Merlo, A., Migliardi, M., Fontanelli, P.: On energy-based profiling of malware in android. In: HPCS (2014)
18. Mohaisen, A., Alrawi, O.: AV-meter: an evaluation of antivirus scans and labels. In: Dietrich, S. (ed.) DIMVA 2014. LNCS, vol. 8550, pp. 112–131. Springer, Heidelberg (2014)
19. Nacci, A.A., Trovò, F., Maggi, F., Ferroni, M., Cazzola, A., Sciuto, D., Santambrogio, M.D.: Adaptive and flexible smartphone power modeling. Mob. Netw. Appl. **18**(5), 600–609 (2013)
20. Nash, D.C., Martin, T.L., Ha, D.S., Hsiao, M.S.: Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In: PerCom Workshops. IEEE Computer Society (2005)
21. Pathak, A., Hu, Y.C., Zhang, M.: Where is the energy spent inside my app? Fine grained energy accounting on smartphones with eprof. In: EuroSys (2012)
22. Rastogi, V., Chen, Y., Jiang, X.: Droidchameleon: evaluating android anti-malware against transformation attacks. In: ASIA CCS (2013)
23. Rice, A.C., Hay, S.: Decomposing power measurements for mobile devices. In: PerCom (2010)
24. Symantec: Android Madware and Malware Trends (2013)
25. Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., Singh, J.P.: Who killed my battery? Analyzing mobile browser energy consumption. In: WWW (2012)
26. Truong, H.T.T., Lagerspetz, E., Nurmi, P., Oliner, A.J., Tarkoma, S., Asokan, N., Bhattacharya, S.: The company you keep: mobile malware infection rates and inexpensive risk indicators. In: WWW (2014)
27. Wang, T., Lu, K., Lu, L., Chung, S., Lee, W.: Jekyll on iOS: when benign apps become evil. In: Proceedings of the 22Nd USENIX Conference on Security, SEC 2013 (2013)
28. Yoon, C., Kim, D., Jung, W., Kang, C., Cha, H.: Appscope: application energy metering framework for android smartphones using kernel activity monitoring. In: USENIX ATC (2012)
29. Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: CODES/ISSS (2010)
30. Zheng, M., Lee, P.P.C., Lui, J.C.S.: ADAM: an automatic and extensible platform to stress test android anti-virus systems. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 82–101. Springer, Heidelberg (2013)