Towards Secure Mobile Computing: Employing Power-Consumption Information to Detect Malware on Mobile Devices

Thomas Zefferer, Peter Teufl,
David Derler, Klaus Potzmader, Alexander Oprisnik, Hubert Gasparitz, and Andrea Höller

Institute for Applied Information Processing and Communications
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
Email: {thomas.zefferer|peter.teufl}@iaik.tugraz.at,
{dderler|klaus.potzmader|oprisnik|hubert.gasparitz|ahoeller}@student.tugraz.at

***Abstract*—Smartphones and related mobile end-user devices represent key components of mobile computing based solutions and enable end users to conveniently access services and information virtually everywhere and any time. Due to their continuously growing importance and popularity, mobile devices have recently become a common target for malware. Unfortunately, capabilities of malware-detection applications on smartphones are limited, as integrated security features of smartphone platforms such as sandboxing or fine-grained permission models restrict capabilities of third-party applications. These restrictions prevent malware-detection applications from accessing required information for the identification of malware. This renders the implementation of reliable malware-detection solutions on smartphones difficult. To overcome this problem, we propose an alternative malware-detection method for smartphones that relies on the smartphone's measured power consumption. We show that information contained in the measured power consumption of smartphones can in principle be used to identify certain kinds of malware by means of simple threshold-based approaches. We also propose two different machine-learning techniques that allow for a classification of applications according to their power consumption in situations, where disturbing influences prevent an application of simple threshold-based approaches. The capabilities of all proposed techniques have been assessed by means of an evaluation with real-world applications running on physical smartphones. The results of this evaluation process demonstrate the applicability of power consumption based classification and malware-detection approaches in general and of the two proposed machine-learning techniques in particular.**

***Keywords*–Android; power consumption; application classification; malware detection; machine learning.**

I. Introduction

With the growing popularity of mobile end-user devices such as smartphones or tablet computers, also malware for these devices has become an issue. The special architecture and characteristics of modern mobile end-user devices raise the demand for appropriate methods to detect such malware. Innovative approaches to detect malware on mobile end-user devices based on power-consumption measurements have been proposed by the authors in [1]. In this article, we further elaborate on the proposed techniques and on its underlying concepts.

During the past years, powerful mobile end-user devices have become part of our daily life and have significantly changed the way we access information, communicate, and interact with each other. During the past few years, smartphones and tablet computers have gradually replaced traditional end-user devices such as desktop PCs and laptops as preferred consumer devices. Considering current sales and usage statistics [2], it can be expected that mobile computing in general and smartphone-based solutions in particular will continue to play a major role in future.

The recent success of popular smartphone platforms such as Apple iOS [3] or Google Android [4] has unfortunately turned these platforms into attractive targets for attackers. This is especially problematic, as mobile end-user device typically store and process an increasing amount of security and privacy-sensitive data. In this context, malware tailored to the special characteristics of smartphone platforms has turned out to be a potential threat during the past few years. Recent reports [5] show that smartphone malware must be expected to evolve to a major issue in mobile computing in the future. By exploiting specific functionality provided by the infected smartphone platform, smartphone malware can cause financial losses, e.g., by calling premium-rate numbers or by compromising smartphone-based authentication schemes of e-banking solutions. During the past years, especially the Android platform has been frequently targeted by smartphone malware. A recent example is a malware called Eurograbber. In 2012, Eurograbber has been used to steal 47 million USD from European bank accounts by intercepting SMS-based authentication processes of e-banking portals [6]. Android seems to be especially prone to malware due to the platform's support of alternative application sources that usually lack extensive malware checks, and due to the broad functionality offered by Android's public APIs. This is advantageous for application developers and users, as it allows for mobile applications with increased functionality. At the same time, it also gives attackers the opportunity to implement more powerful malware. For instance, the Android APIs grant application developers as well as attackers full access to incoming and outgoing SMS messages, or facilitate the execution of arbitrary background tasks. While this enables the development of mobile apps that can make use of SMS functionality, it also facilitates the development of malware that intercepts or spies on incoming messages.

The factual vulnerability of the Android platform against malware raises the need for reliable methods to distinguish benign apps from malicious ones and to detect unwanted behavior on smartphones. On classical end-user devices such as desktop computers or laptops, this functionality is typically implemented by anti-virus software, which is able to detect malicious software at runtime. Unfortunately, the deployment of anti-virus software on smartphone platforms in general, and on Android in particular is difficult. This is mainly due to the fact that Android (as well as other smartphone platforms) implements several security features on operating-system level, which limit access rights and capabilities of third-party applications. For instance, all smartphone applications are executed in a so-called sandbox and are, thus, unable to access resources of other applications being installed and executed on the same device. This way, application-specific data remains protected from unauthorized access by other applications. While implemented security features definitely improve the system's basic security, they also render the implementation of supplementary security software difficult. For instance, the implemented sandbox feature prevents anti-virus software on Android smartphones from collecting information that is required to reliably detect smartphone malware at runtime.

Integrated security features that limit the capabilities of classical malware-detection methods can theoretically be bypassed by rooting the smartphone's operating system. Rooting has become common practice in the power-user community, as it gives users more control over the device and allows for additional functionality. However, rooting is not really an option to increase the capabilities of anti-virus software, as it significantly decreases the smartphone's overall security and enables additional attack vectors. Furthermore, non-rooted device still represent the majority of all mobile end-user devices. For these reasons, we focus on the class of non-rooted devices only.

The reliable detection of malware on non-rooted smartphones is still an unsolved problem that definitely needs to be addressed to assure the security of future mobile computing. To overcome this problem, we propose a new technique to detect malware on mobile end-user devices. The proposed technique compensates the lack of required information about running applications by making use of side-channel information being available on non-rooted Android smartphones. This way, this work answers two basic research questions. First, this work evaluates if and to what extend power-consumption information available on smartphones can be used to classify running mobile applications and to identify malware. Second, this work investigates capabilities of different machine-learning techniques to analyze available power-consumption information.

The results presented in this article extent first results that have been presented in 2013 [1]. The obtained results show that alternative approaches to identify malware on mobile end-user devices can be successful. Furthermore, obtained results indicate that the application of machine-learning techniques can improve the classification of applications according to their power consumption. Even though the described work is basically a first proof of concept, the obtained results are promising and open new possibilities for malware detection on mobile end-user devices. This way, this work contributes to the security of future mobile-computing applications.

The remainder of this paper is structured as follows. In Section II, existing malware-detection approaches for smartphones are briefly surveyed and limitations of these approaches are identified. Subsequently, Section III introduces the tool PowerTutor [7] and explains our approach to measure the power consumption of applications on smartphones. In Section IV, we show that measured power-consumption traces can indeed be used to enhance the security of critical applications on smartphones by means of a simple detection mechanism for SMS-based malware. In Section V, we focus on more sophisticated methods to analyze collected power-consumption information and propose two methods to analyze collected power-consumption measurements based on approved machine-learning techniques. We evaluate the capabilities of the proposed methods to classify applications and to distinguish benign applications from malicious ones in Section VI. Finally, conclusions are drawn in Section VII.

II. Related Work

During the past years, several approaches to detect and analyze malware on mobile platforms have been introduced. Basically, existing approaches can be classified into static and dynamic analysis methods. Static analysis refers to the inspection of an application's source code or binary package without running it, whereas dynamic analysis involves running the application to capture additional information.

Dynamic analysis includes techniques such as Information Flow Analysis, where private data is labeled and prevented from leaving the device. TaintDroid [8] is an Android kernel extension that follows this approach and allows for dynamic taint tracking. Dynamic approaches, which apply machine-learning techniques to distinguish benign applications from malicious ones, include [9] by Shabtai et al. and [10] by Burguera et al. Both references include extensive listings of related Android-based malware-detection systems. Most of these approaches run candidate applications in a sandbox to derive measurements, such as system-call intervals and networking usage. Our technique presented in this paper follows a similar approach. Similar to related work, we make use of side-channel information to classify smartphone applications. However, in contrast to other related work, we rely on power-consumption measurements for this purpose.

A comprehensive overview of dynamic malware-analysis techniques is also provided by Egele et al. in [11]. Many of these methods are highly advanced in detecting and analyzing malware. However, these methods usually require complex external analysis frameworks and can hardly be deployed on non-rooted end-user devices, due to their requirement to deeply integrate into the smartphone's operating system. Thus, these methods are usually less useful for detecting malware on typical end-user devices at runtime.

The technique presented in this paper addresses this problem and facilitates dynamic malware detection directly on non-rooted Android phones by analyzing the devices' power consumption. A related approach to analyze the power-consumption in order to detect malware has been followed by Jacoby and Davis [12], who have proposed an intrusion detection system that correlates various attack scenarios to typical power consumptions. Additional work has been published by Buennemeyer et al. [13] [14], who propose systems, which use power profiles of phones to detect malware targeting battery drainage. Our technique follows a similar approach but extracts more detailed information from the collected power-consumption measurements in order to classify applications and to detect malware.

Employing the power consumption to reveal additional information on an IT system is actually not a new idea. For instance, approaches to extract secret information stored on smart cards by means of measuring and analyzing their power consumption have already been proposed in 1999 by Kocher et al. [23]. Based on this work, various techniques to reveal secret data and information from IT systems have been proposed during the past few decades. Most of these techniques however require an elaborate measurement equipment and the application of complex analysis methods. This renders a real-time application of these approaches on current mobile end-user devices difficult. The techniques proposed in this paper follow a slightly different approach and rely on methods that basically allow for real-time application.

For the proposed techniques, the collection of accurate power-consumption measurements on smartphones is a key aspect and mandatory enabler of our technique. We discuss details of this aspect in the next section.

III. Measuring the Power Consumption of Smartphones

Measurements of a smartphone's power consumption build the basis of the proposed classification and malware detection techniques. To collect the required power-consumption measurements, we rely on the tool PowerTutor by Zhang et al. [7]. Another tool that would allow for the acquirement of this kind of information is Trepn [15]. In contrast to PowerTutor, Trepn uses hardware sensors and thus promises more exact measurements. However, Trepn is limited to the Snapdragon mobile development platform [16] and can hence not be applied on typical Android based end-user devices. Heading for a solution that is applicable on real end-user devices, PowerTutor has therefore been our tool of choice.

The tool PowerTutor is basically a smartphone application that measures the power consumption of all applications running on the same smartphone. For each application, the power consumption of six smartphone components is measured. In particular, the power consumption of the components *CPU*, *Audio*, *Display*, *Wi-Fi*, *3G* and *GPS* is measured separately. Figure 1 shows the mean per-component power consumption of six different applications. All information shown in Figure 1 has been directly derived from measurements created by the tool PowerTutor. The graphically prepared measurements shown in Figure 1 clearly indicate that there are significant differences between power-consumption measurements of different applications. Furthermore, Figure 1 shows that there are also differences in the power consumption of different components. Unsurprising, the smartphone display and the mobile network (3G) are responsible for most of the measured power consumption - at least for the six measured applications[1].

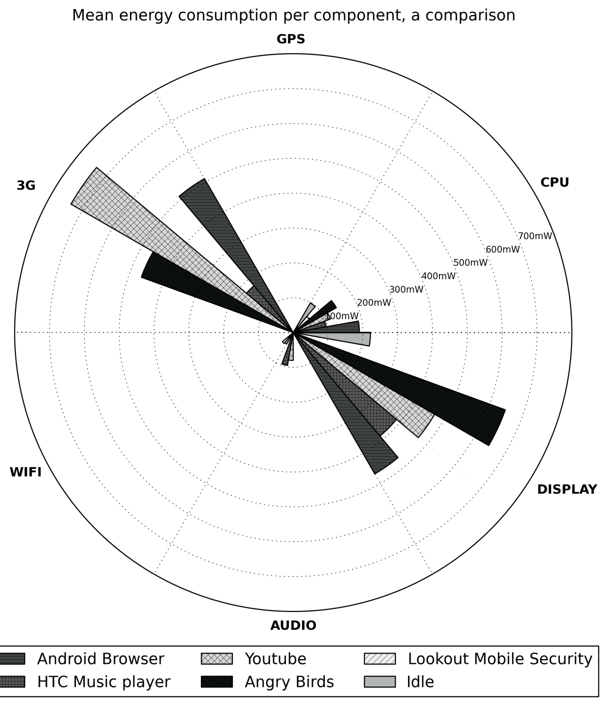


Figure 1: Comparison of mean power usage per component using six different applications

Figure 1 shows that power measurements provided by the tool PowerTutor indeed contain valuable information that might allow for a classification of running applications and for an on-the-fly identification of malicious apps at runtime. To improve efficiency and to appropriately cope with limited processing power on mobile end-user devices, we have decided to focus on measurements of one component only in a first step. Analysis of power-consumption measurements of several applications have shown that the smartphone CPU is actually the best suited component for profiling running applications by means of its power consumption. This is also illustrated by Figure 2. This figure shows the stacked power consumptions of the six measured components while running the app *HTC Music Player*. This figure shows that the CPU is basically the only component that shows a significant change in power consumption over time. Similar results have also been obtained for other applications, which justifies our decision to focus on the power consumption of the CPU only. At this point, it has to be noted that ignoring all other components of course reduces the amount of available information. However, considering all available power-consumption information from all components significantly increases complexity and is hence considered as future work.

Figure 3 shows the measured power consumption of the CPU component caused by the smartphone game Angry Birds, the Android Browser, the Idle process, and the security application Lookout Mobile Security. These measurements show

[1]GPS is also know to consume significant amounts of power. However, none of the six measured applications used GPS functionality during the measurements.

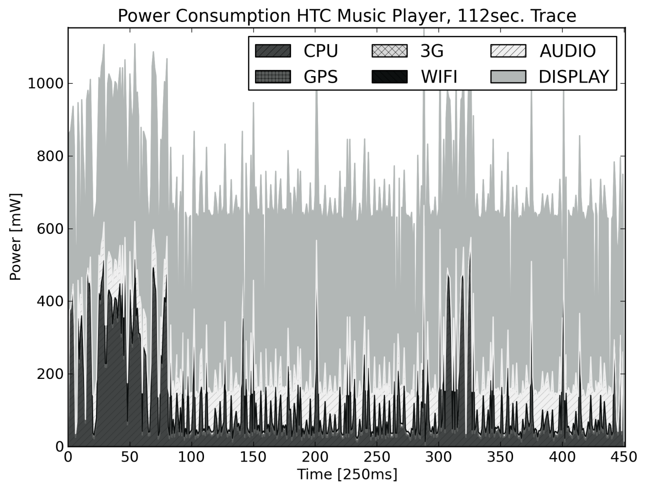


Figure 2: Power usage of HTC Music Player stacked per component.

that there are obvious differences in the measured power consumption of different applications running on the same device. Obviously, the application Angry Birds and the Web browser consume significantly more power than the Idle process or the application Lookout Mobile Security. The first question that arises from these observations is whether and to what extend these differences in the power consumption can be used to identify applications and suspicious behavior. We show in the next section that this question can be positively answered and that differences in measured power consumptions can in some cases be directly used to identify applications with suspicious behavior on smartphones at runtime.

IV. Detection of SMS-Controlled Spyware

SMS-controlled spyware has evolved to a serious threat on the Android smartphone platform. This kind of malware spies on private user data (e.g., position information, data received and sent via SMS, etc.) and is able to receive hidden control commands via SMS. These messages contain control commands and are sent to the victim's smartphone unnoticed by the legitimate user. Apart from spyware, capturing and processing SMS messages is also handled by malware (e.g., Eurograbber), which attacks two-factor authentication systems that are usually employed by online-banking systems. Spyware and malware related to SMS functionality are typically implemented for the Android platform, due to the availability of public APIs for accessing low-level SMS functionality. This is in contrast to iOS and Windows Phone, where this functionality is only handled by the operating system and cannot be used via public APIs.

In general, spyware and malware, which access SMS-related APIs can be assigned to two different categories according to the implemented functionality.

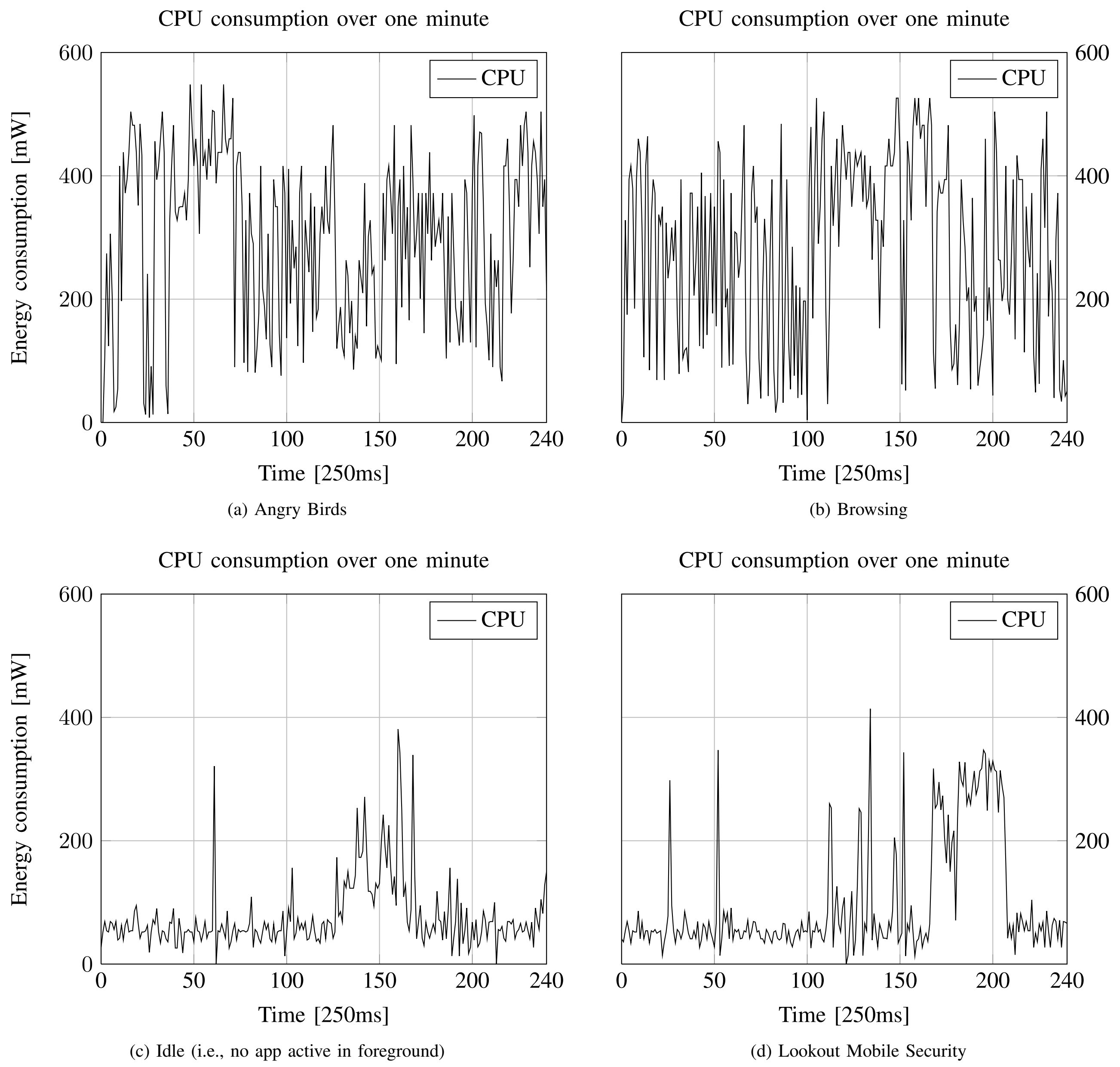
- **SMS Sniffers:** SMS sniffers, which are mostly relevant for spyware, only capture the information in the received SMS messages, and process and optionally forward this information.
- **SMS Catchers:** SMS catchers also suppress the SMS forwarding mechanism of the operating system, which would deliver the message to the default SMS client. Therefore, the user will not receive a notification on the newly arrived messages. This functionality is often required by malware that attacks online banking systems, because the received TAN must not be shown to the user, who would otherwise be alarmed by receiving a TAN without executing an actual transaction. The problem of SMS catchers has recently been addressed by Android 4.4., which requires definition of a default SMS application that always receives incoming messages without the possibility to suppress the notification of the user. However, the market share of Android 4.4. is still very limited and the principle problem of reading SMS messages remains.

To assess the general potential of power consumption based malware-detection approaches, we show how to identify suspicious SMS processing activities by analyzing the power consumption of an Android smartphone. For this purpose, we have developed the following two smartphone applications for the Android platform:

- **Malware Simulator:** This application simulates SMS-controlled spyware and is able to intercept incoming SMS messages as well as to silently forward copies of received SMS messages to arbitrary recipients. Furthermore, this application is able to determine the smartphone's current location either by reading out the last known position from the Android system, or by determining the current position using available position sensors. The determined position information can be forwarded to an attacker by SMS. This way, the developed malware simulator implements all typical features of SMS-controlled spyware, which receives commands via SMS, spies on the user's current location, forwards determined location information, and silently forwards copies of SMS messages to a remote attacker.
- **Malware Detector:** This application detects anomalies of local SMS processing activities by measuring and analyzing power-consumption information provided by PowerTutor. Thus, the basic goal of this application is to successfully identify the implemented **malware simulator** as spyware.

To evaluate the general capabilities of power consumption based malware detection approaches, we carried out the following steps: First, the malware detector has measured the power consumption for three seconds each time an SMS message has been received by and processed on the device. Second, from the obtained measurements, an appropriate model has been extracted. According to this model, all measured power traces have been classified into four categories depending on the SMS processing operation that has been taking place during the measurements. Finally, the extracted model has been integrated into the developed malware detector. With the help of this model, the malware detector has then been used to identify suspicious activities during SMS processing operations in real time.

Following this approach, both the malware simulator and



(a) Angry Birds

(b) Browsing

(c) Idle (i.e., no app active in foreground)

(d) Lookout Mobile Security

Figure 3: One minute plots of CPU energy consumption

the malware detector have been deployed on a Samsung Galaxy S2 smartphone running the Android 2.3 operating system. The device's power consumption has been measured during execution of the following SMS processing steps carried out by the malware simulator:

- **Step A – Normal SMS receive**: An incoming SMS message is received and forwarded to Android's default SMS application without any further action.
- **Step B – Pos. Command 1**: An incoming command SMS message is intercepted and an SMS message is returned that includes the last known location.
- **Step C – Pos. Command 2**: An incoming command SMS message is intercepted and an SMS message is returned that includes the currently determined location.
- **Step D – Forward SMS**: An incoming SMS is forwarded to Android's default SMS application and additionally forwarded to a given number.

The power consumption of each processing step has been measured for ten times. Figure 4 shows the average power consumption of the four different SMS processing steps. Obviously, there are significant differences in the power consumption depending on the executed processing step. If the SMS is forwarded to Android's default SMS application for further processing (**Step A** and **Step D**), more energy is consumed compared to processing steps, in which an incoming SMS message is intercepted and another SMS message is sent unnoticed by the user (**Step B** and **Step C**). Due to the significant differences in the measured power consumptions, an appropriate model for the detection of suspicious activities can be extracted by simply considering the average value of the power measurements. In the present case, an average power consumption of 150 has been chosen to define the lower bound of an SMS that is processed by Android's SMS application. If the average value is beneath this bound, it is likely that an incoming SMS message has been intercepted and hidden from the user.

The reliability of this rather simple model has been evaluated by sending 10 normal SMS messages that have simply been forwarded to Android's SMS application. Additionally, we have sent ten SMS messages, that have been intercepted by our malware simulator and discarded afterwards. By measuring the power consumption during execution of the SMS processing steps and by applying our simple model to the obtained measurements, the implemented malware detector was able to successfully distinguish between discarded messages and messages forwarded to Android's SMS application. This way, we have shown that it is basically possible to detect in real time whether an SMS message is received normally or whether it is intercepted by a third party application. This validates the postulated assumption that information contained in power-consumption measurements can indeed be used to identify malicious applications on smartphones and hence positively answers the first research question of this work.

V. Enhanced Classification Techniques

The successful realization of a malware detector that is able to identify suspicious SMS processing activities at run-time confirms the capabilities of application-classification and malware-detection techniques based on power-consumption information. However, even though there are obvious differences in the power consumption of these two applications (Figure 3), an immediate identification and classification of applications based on such measurements is often not possible. This is due to the fact that the measured power consumption is not only influenced by the application itself, but also by other effects, such as varying user inputs, the processed data, different screen orientations, the deployment of hardware acceleration techniques, or 3G or WiFi signal reception. This is illustrated in Figure 5, which shows two different measurements of one and the same application. Although stemming from the same application, the two measured power-consumption traces are quite different.

Disturbing influences render the determination of a simple and unique power-consumption signature for a given application or smartphone state impossible. Unambiguous results, such as the ones obtained for the implemented malware detector for identification of SMS-controlled spyware, are usually hard to achieve in practice. To overcome this problem, we propose two analysis techniques that rely on approved machine-learning approaches. The proposed techniques can be used to classify smartphone applications according to their power consumption, even if there are only minor differences in collected power measurements due to disturbing influences.

During the past years, different machine-learning techniques for the classification of data have been proposed. For the given scenario, i.e., the classification of smartphone applications based on their power consumption, two techniques have been chosen and adapted to the given requirements. Both techniques consist of a *learning phase* and a *classification phase*. During the learning phase, well-known input data is used to train a model. In the subsequent classification phase, the trained model is used to classify unknown input data. The two techniques are discussed in more detail in the following subsections.

A. Power-Consumption Histograms

This technique is rather simple and counts how often a specific application is on a certain power-consumption level. In order to model this, we have computed power histograms by dividing the interval between 0% power consumption and 100% power consumption into 15 disjoint and equal-sized intervals. A histogram is then created by simply assigning each data point to exactly one interval and counting the data points in each interval. In order to cope with differences in the absolute power consumption, the values have been normalized appropriately. During the learning phase, the average histograms have been created by measuring the power consumption of well-know applications. Figure 6 shows some examples of average histograms for different applications that have been obtained during the training phase.

In the classification phase, the histograms of applications to be classified are compared with the trained average histograms by applying distance-measures such as cosine similarity. To assess the capabilities of this approach, this technique has been evaluated in a real-world scenario. Results of this evaluation process are presented and discussed in Section VI.

B. MFC Coefficients and Gaussian Mixture Models

This technique makes use of *Mel Frequency Cepstral Coefficients (MFCC)* to classify smartphone applications based on their power consumption. This technique has originally been introduced for speaker-recognition systems [17][18] and is also frequently used for music similarity finders [19][20]. In such systems, MFC coefficients and their distribution are extracted from recorded voice or music using complex transformations as implemented by the *melcepst* function [21]. The distributions of the extracted MFCC are then used to create a *Gaussian Mixture Model (GMM)* for each MFCC. The resulting GMM define a unique representation of the recorded voice or music. Later recordings of voice or music can be compared to existing representations in order to implement voice-recognition and music-similarity finders.

Our intention behind using a speaker recognition approach was to map the problem of matching voice recordings to a person to the problem of matching power measurements to an application. Spoken voice recordings vary in pitch and frequency and are very unlikely to be equal between two recordings. This, naively speaking, resembles the problem we face with power-consumption measurements.

Our implementations bases on an existing speaker-recognition implementation by Anil Alexander [22]. This implementation relies on GMM and MFCC and can be customized with a number of parameters including the number of Gaussians and the number of MFCC to use. Experiments have shown that for our purposes best results can be achieved

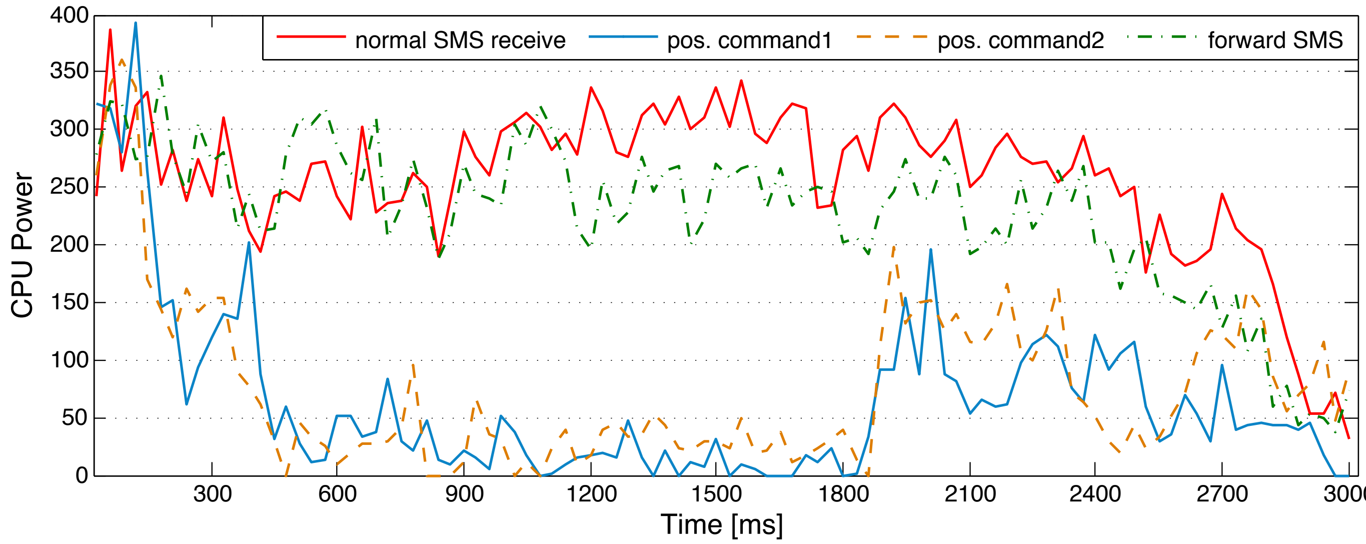
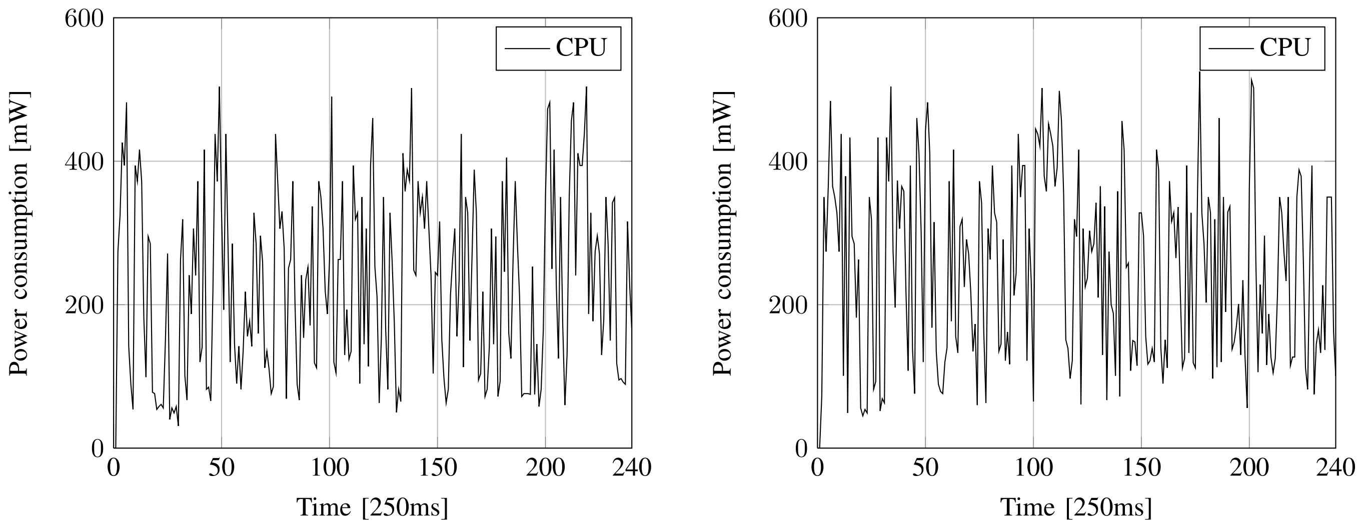


Figure 4: Average CPU power profile after receiving an SMS



(a) Angry Birds - Run 1

(b) Angry Birds - Run 2

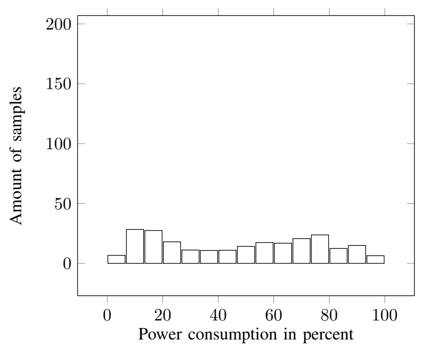
Figure 5: One minute plots of CPU energy consumption

with three Gaussians and twelve MFCC. Hence, during the learning phase the distributions of twelve MFCC are computed from power-consumption measurements for each class of application. The computed distributions of the twelve MFCC are then approximated using a GMM with three Gaussians. The resulting GMM finally represents the result of the learning phase. Figure 7 illustrates the distribution of twelve different MFCC and the resulting GMM.

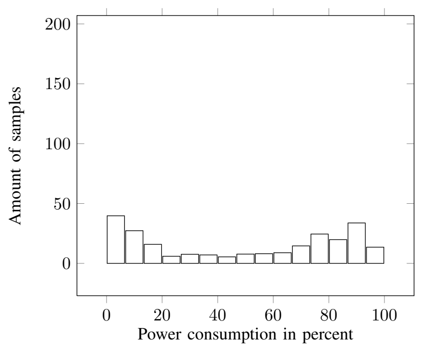
During the classification phase, MFCC are derived from power-consumption measurements of the application to be classified. For each derived MFCC, the best matching GMM is selected out of all GMM that have been obtained during the learning phase. By combining the classification results of all twelve MFCC, the best matching application class is finally determined.

VI. Evaluation

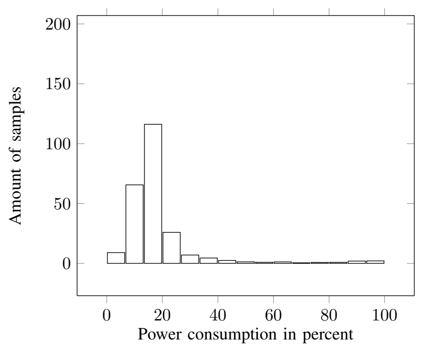
We have evaluated the reliability and efficiency of the proposed feature extraction techniques by testing prototype implementations of the two techniques in a real-world scenario. Required power-consumption measurements have been acquired using the tool PowerTutor. For convenience reasons, the classification itself has been performed off the mobile device, as the learning phase (especially for the speaker recognition based approach) is rather slow. This subsection describes the model that has been used to classify applications, discusses details of the dataset creation, and presents results that have been obtained by applying the two classification techniques introduced in Section V.



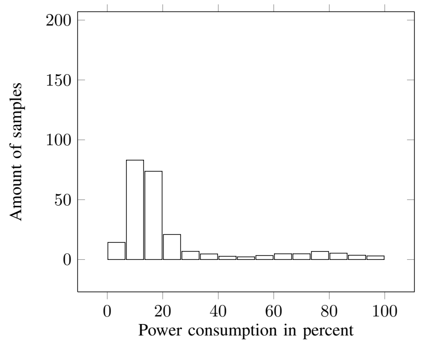
(a) Angry Birds



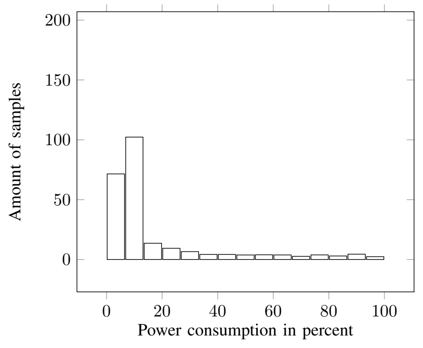
(b) Browsing



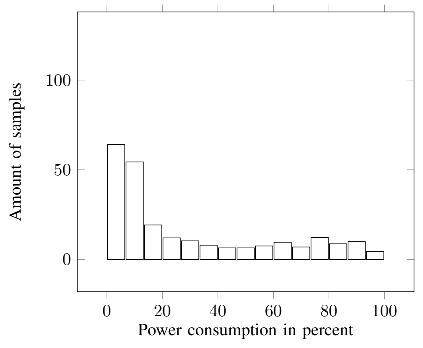
(c) Idle



(d) Lookout Mobile Security



(e) Music



(f) YouTube

Figure 6: Average histograms of different applications

A. *Classification Model*

Applications with the same or almost the same purpose are expected to cause similar power consumptions. Therefore, we have roughly grouped applications into distinct sets according to their purpose. The resulting list of groups is no comprehensive classification scheme of all available applications. It is merely a logical grouping of the power-consumption measurements we gathered in this experiment and does raise no claim to completeness. Based on the gathered measurements, the following six groups of applications have been defined: *Games*, *Internet*, *Idle*, *Malware*, *Music*, and *Multimedia*. Note that malware and security software have been assigned to the same group. Both malware and security software usually remain idle in the background until being activated by a certain event (e.g., reception of a command message via SMS). This comparable behavior leads to a comparable power consumption too and justifies a common classification of these both types of application. Of course, also other malware with different behavior and hence a different power-consumption profile exists. However, for a first proof of concept only malware with the above-described behavior has been considered. Consideration of other types of malware is regarded as future work.

B. *Dataset Creation*

PowerTutor provides specific measurements for each running application. However, in practice these application specific measurements have turned out to be not as reliable and accurate as desired. Therefore, we have refrained from using application specific measurements and have relied on system-wide power-consumption measurements provided by PowerTutor instead.

We have further limited subsequent analysis steps to the measured power consumption of the smartphone's CPU. Although PowerTutor also provides measurements for other smartphone components such as the display or the GPS receiver, measurements of these components have been omitted in order to reduce computation costs when learning and due to the fact that these components often lack activity.

To evaluate the proposed classification techniques, we finally created 96 system-wide power-consumption measurements (CPU) using a customized instance of PowerTutor. To facilitate a subsequent analysis, we have adapted PowerTutor such that beside the measurement values themselves also the device model, the capture date, and the sample rate have been stored. The 96 captured measurements (sixteen measurements per application group) have been limited to the length of about one minute, with a total of 247 data points per measurement. We have cut off the trace length after about one minute, as this is a realistic time-frame for real-world scenarios. Analysis of longer measurements is considered as future work. Similarly, increasing the number of analyzed applications is also considered as future work. For a first proof of concept, the used number of applications and the chosen time interval is however sufficient. In total, six devices have been used to collect the measurements (three Samsung Galaxy S2 smartphones and three HTC Desire devices). To reduce noise, only the application to be measured and PowerTutor have been active during the measurements.

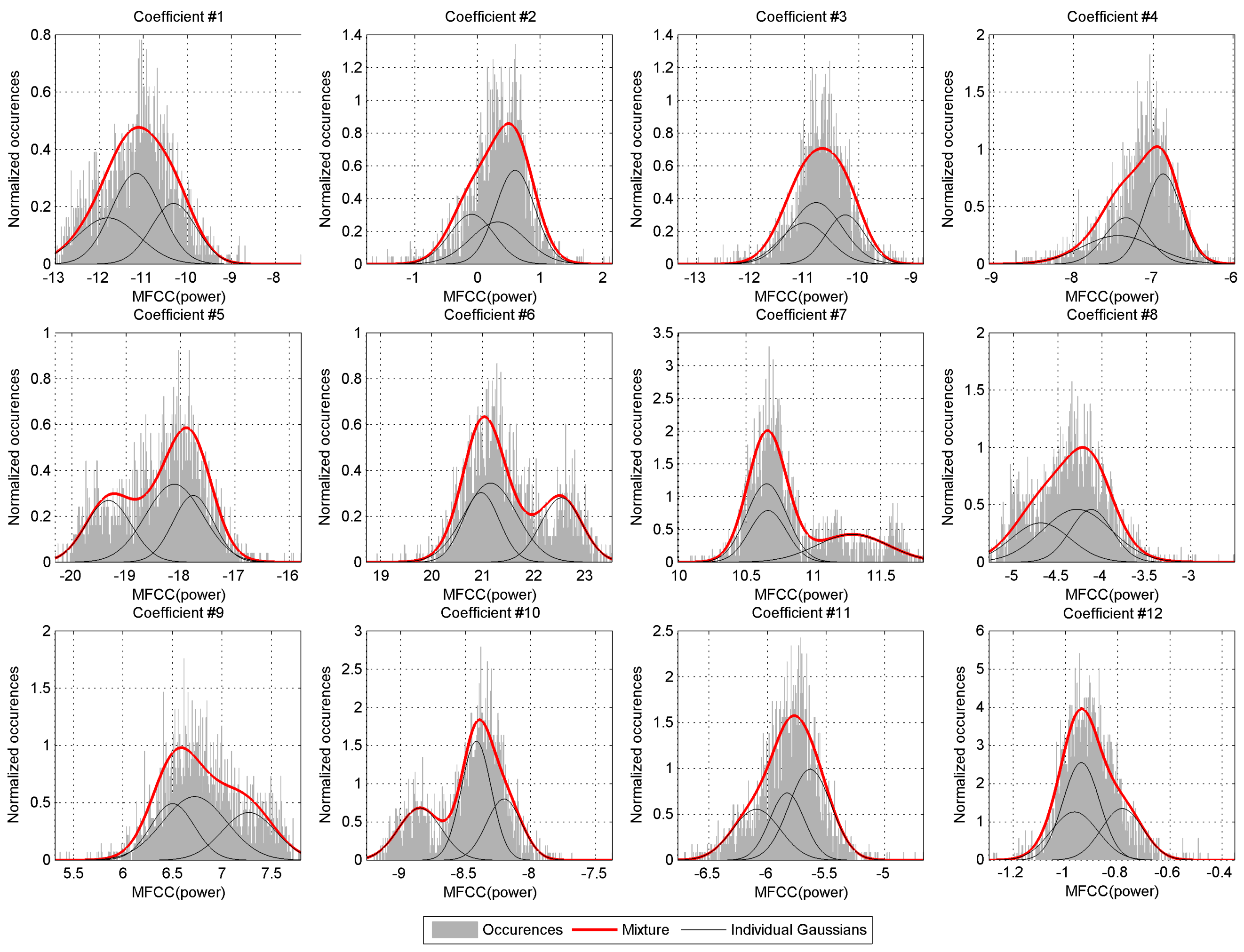


Figure 7: Gaussian Mixture Models of twelve MFCC derived from power-consumption measurements

C. Results

The 96 captured measurements have been used to evaluate the efficiency and reliability of the proposed classification techniques. As quality indicators, the positive predictive value (PPV, also referred to as precision), the true positive rate (TPR, also referred to as recall or sensitivity), the true negative rate (TNR, also referred to as specifity), the accuracy, and the area under the receiver operating characteristic (AUC) have been used. According to its definition, PPV refers to the correct positive classification in relation to all positive classifications. Accordingly, TPR refers to true positives given all real positives. TNR denotes true negatives (TN) given all negatives. Accuracy is the relation between correctly classified samples given all samples. The receiver operating characteristic is a graphical representation of the trade-off between TPR and FPR (1-TNR). AUC (also sometimes denoted as AUROC) refers to the area below this resulting curve.

In order to appropriately divide the available measurements in training and test data, we have folded the available dataset using 10-fold cross validation. To enhance the robustness of the obtained results, average values over 100 runs are presented.

TABLE I: Histogram based approach: confusion matrix for categories Games (G), Internet (IN), Idle (ID), Malware (MW), Music (MU), and Multimedia (MM)

	G	IN	ID	MW	MU	MM
G	13.98	1	0	1.02	0	0
IN	1	13.14	0	0	0	1.86
ID	0	0	12	4	0	0
MW	0	0	3.97	10.07	1.96	0
MU	0	0	0	2	9.24	4.76
MM	0.27	0.75	0	0	0	14.98

For our performance evaluation, a confusion matrix for the six predefined application categories has been created, which can be interpreted in the following way: Values in the diagonal of the matrix have been classified correctly (true positives), values within a row not in the diagonal represent false negatives and values within a column not in the diagonal represent false positives. Other values are considered true negatives.

Obtained results of the histogram based approach are

TABLE II: Classification results (histogram based approach)

Category	PPV	TPR	TNR	Accuracy	AUC
Games	0.87	0.92	0.98	0.97	0.95
Internet	0.82	0.88	0.98	0.95	0.93
Idle	0.75	0.75	0.95	0.92	0.85
Malware	0.63	0.59	0.91	0.87	0.75
Music	0.58	0.83	0.98	0.91	0.90
Multimedia	0.94	0.69	0.92	0.92	0.80

TABLE III: MFCC and GMM based approach: confusion matrix for categories Games (G), Internet (IN), Idle (ID), Malware (MW), Music (MU), and Multimedia (MM)

	G	IN	ID	MW	MU	MM
G	10.40	3.46	0.01	0	0.55	1.58
IN	2.07	12.67	0	0	0.26	1
ID	0.39	0	11.65	3.6	0.18	0.18
MW	0.07	0.01	2.56	13.15	0	0.21
MU	0.22	0.81	0	0.97	9.39	4.61
MM	0.96	2.97	0.01	0.03	1.20	10.83

shown in Table I and Table II. In case of the MFCC based approach, best results have been achieved with 3 Gaussians and twelve MFCC. The performance evaluation results of the MFCC based approach are outlined in Table III and Table IV.

D. Discussion

From these results, various findings can be derived. Mobile security applications and malware running in the background can generally be distinguished from application being active at the moment (with the exception of system services). Games, Internet, music and multimedia applications are distinguishable as well. Music and multimedia applications are more difficult to distinguish correctly, due to their similar power-consumption profile. However, given their related purposes this is plausible. Streaming a YouTube video with sound is not too different from listening to music while reading related information displayed by the music player.

The obtained results have also revealed that the MFCC based approach works better for the distinction between the categories Idle and Malware. Therefore, this approach seems to be more suitable for malware-detection purposes. On the other hand, the histogram approach constitutes a fast classification method, suitable for mobile devices with limited computational power.

It has to be noted that the obtained results basically represent a first proof of concept only. The number of measured applications and also the time period, in which the power consumption of applications has been measured, has been intentionally kept rather low for the sake of simplicity and both, an analysis regarding a larger dataset and an analysis regarding a larger number of applications is left open for future work.

Although the number of applications and also the length of measurements has been fixed to a relatively small value, obtained results are still useful due to using 10-fold cross validation, and, thus, answer the second research question of this work. Concretely, obtained results show that machine-learning techniques are suitable to analyze power-consumption measurements of smartphone applications for classification and malware-detection purposes.

TABLE IV: Classification results (GMM based approach)

Category	PPV	TPR	TNR	Accuracy	AUC
Games	0.65	0.74	0.95	0.90	0.85
Internet	0.79	0.64	0.91	0.89	0.78
Idle	0.73	0.82	0.97	0.93	0.90
Malware	0.82	0.75	0.94	0.92	0.85
Music	0.59	0.82	0.97	0.91	0.90
Multimedia	0.68	0.59	0.91	0.87	0.75

VII. Conclusion

Smartphones and related mobile end-user devices are frequently used to store and process security and privacy-critical data. Malware on smartphones is a growing threat for these data and hence a major challenge for future mobile computing solutions. To overcome this challenge, new and innovative methods to detect malware on smartphones and related mobile end-user devices are needed. In this paper, we have tested the hypothesis that the power consumption of smartphones correlates with the kind of applications being executed on the smartphone and that this correlation allows for a classification of applications and a detection of malicious software. To test this hypothesis, we have proposed a simple threshold-based method and two machine-learning techniques that can be used to classify unknown applications according to their power consumption. We have further assessed the validity of the general hypothesis and the capabilities of the proposed machine-learning techniques by means of a concrete prototype implementation and a succeeding evaluation in a real-world scenario. The conducted assessment has corroborated the constructed hypothesis and has shown the capabilities of the proposed techniques to correctly classify smartphone applications according to their power consumptions.

Although first results are promising, this work mainly represents a proof of concept and a solid basis for future work. In a next step, we plan to port the entire classification onto a smartphone in order to render external classification frameworks unnecessary. Power measurements can already be collected directly on the smartphone using tools such as PowerTutor. The development of a purely smartphone based application classification and malware detection solution that relies on the techniques presented in this paper is hence mainly a matter of computing resources available on smartphones. Since information on the smartphone's power consumption is publicly available on Android smartphones, our solution does not require root access to the operating system and is hence applicable on virtually all end-user devices. We are also planning to refine the proposed techniques and to enhance the current prototype in order to achieve even more accurate results and to be able to classify multiple applications running simultaneously on a smartphone.

References

[1] T. Zefferer, P. Teufl, D. Derler, K. Potzmader, A. Oprisnik, H. Gasparitz and A. Hoeller, "Power consumption-based application classification

and malware detection on Android using machine-learning techniques," *Future Computing, 2013. Proceedings from the fifth international conference on future computational technologies and applications*, pp. 26–31, May-June 2013.

[2] Go-Gulf, "Smartphone Users Around the World - Statistics and Facts," http://www.go-gulf.com/blog/smartphone/, 2013.

[3] Apple, "Apple iOS 6," http://www.apple.com/ios/, 2013.

[4] Google, "Android," http://www.android.com/, 2013.

[5] Lookout Mobile Security, "2011 Mobile Threat Report," https://www.mylookout.com/mobile-threat-report, 2011.

[6] InformationWeekSecurity, "Zeus Botnet Eurograbber Steals $47 Million," http://www.informationweek.com/security/attacks/zeus-botnet-eurograbber-steals-47-millio/240143837, 2012.

[7] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114.

[8] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[9] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, Feb. 2012.

[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 15–26.

[11] 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. 2008.

[12] G. A. Jacoby, R. Marchany, and N. J. Davis, "Battery-based Intrusion Detection: A First Line of Defense," *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pp. 272–279, Jun. 2005.

[13] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 2008, p. 296.

[14] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, pp. 239–252.

[15] Ben-Zur, Liat, "Developer Tool Spotlight - Using Trepn Profiler for Power-Efficient Apps," https://developer.qualcomm.com/blog/developer-tool-spotlight-using-trepn-profiler-power-efficient-apps, October 2011.

[16] Bsquare, "Snapdragon Based Products and Services," http://www.bsquare.com/products/snapdragon-based-products-and-services, 2013.

[17] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker Verification Using Adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[18] Kumar, G Suvarna and Raju, K.A. Prasad and Rao, Mohan and Satheesh, P, "Speaker Recognition using GMM," *International Journal of Engineering Science*, vol. 2, no. 6, pp. 2428–2436, 2010.

[19] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *ICME*, 2001.

[20] K. C. West and P. Lamere, "A model-based approach to constructing music similarity functions," *EURASIP J. Adv. Sig. Proc.*, vol. 2007, 2007.

[21] M. Brookes, "VOICEBOX: Speech Processing Toolbox for MATLAB," Web page, 2005.

[22] A. Alexander, "Automatic Speaker Recognition: A Simple Demonstration using Matlab," http://www.anilalexander.org/publications/, 2004.

[23] P. Kocher and J. Jaffe and B. Jun, "Differential power analysis," *Advances in Cryptology - CRYPTO*, pp. 388–397, 1999.