Thomas Zefferer, Peter Teufl, David Derler, Klaus Potzmader, Alexander Oprisnik, Hubert Gasparitz, Andrea Höller: "Power Consumption-based Application Classification and Malware Detection on Android Using Machine-Learning Techniques". In IARIA FUTURE COMPUTING 2013: The Fifth International Conference on Future Computational Technologies and Applications, 2013, IARIA.

Power Consumption-based Application Classification and Malware Detection on Android Using Machine-Learning Techniques

Thomas Zefferer*, Peter Teufl*, David Derler*, Klaus Potzmader*
Alexander Oprisnik*, Hubert Gasparitz* and Andrea Hoeller*
*Insitute for Applied Information Processing and Communications, Graz University of Technology
thomas.zefferer@iaik.tugraz.at, peter.teufl@iaik.tugraz.at

***Abstract*—Mobile computing has significantly gained importance during the past years and is expected to remain one of the most relevant future computing trends. Smartphones represent a key component of mobile computing based solutions and allow end users to conveniently access services and information. Due to their continuously growing importance and popularity, smartphones have recently become a common target for malware. Unfortunately, capabilities of malware-detection applications on smartphones are limited, as implemented security features such as sandboxing or fine-grained permission models restrict capabilities of third-party applications. These restrictions prevent malware-detection applications from accessing information, which is required to identify malware, and hence render the implementation of reliable malware-detection solutions on smartphones difficult.**

To overcome this issue, we propose an alternative malware-detection method for smartphones that relies on the smartphone's measured power consumption. We propose two different machine-learning techniques that allow for a classification of applications according to their power consumption and hence facilitate the identification of suspicious and potentially malicious software components. The capabilities of the 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

Powered by the emergence of smartphones, mobile computing has significantly gained popularity and relevance during the past years. Smartphones have become part of our daily life and have significantly changed the way we access information, communicate, and interact with each other. Considering current sales and usage statistics [1], 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 [2] or Google Android [3] has unfortunately turned these platforms into attractive targets for malware. Recent reports [4] show that smartphone malware must be expected to evolve to a major issue in mobile computing in future. By exploiting specific functionality provided by the infected smartphone platform, smartphone malware can cause financial losses 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 Eurograbber. In 2012, this Android malware has been used to steal 47 million USD from European bank accounts by intercepting SMS based authentication processes of e-banking portals [5]. 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. These APIs grant application developers as well as attackers for instance full access to incoming and outgoing SMS messages, or facilitate the execution of arbitrary background tasks.

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. In the desktop-computer domain, 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 that limit access rights and capabilities of third-party applications. For instance, all smartphone applications are executed in a sandbox and unable to access resources of other applications being installed and executed on the same device. While implemented security features definitely improve the system's basic security, they 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.

The integrated security features that limit the capabilities of classical malware-detection methods can theoretically be

bypassed by rooting the smartphone's operating system. However, this is not really an option in practice, as it significantly decreases the smartphone's overall security and enables additional attack vectors.

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 that compensates the lack of required information about running applications by making use of side-channel information being available on non-rooted Android smartphones. Concretely, our technique measures and analyses the power consumption of applications running on Android smartphones. We show that an application's power consumption correlates with its implemented functionality and that applications can hence be classified according to their power consumption. We further show that this classification can be used to identify malicious 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 PowerTutor tool [6] and explains our approach to measure the power consumption of applications on smartphones. In Section IV, we 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 V. Finally, conclusions are drawn in Section VI.

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 [7] 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 [8] by Shabtai et al. and [**?**] 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.

A comprehensive overview of dynamic malware-analysis techniques is provided by Egele et al. in [9]. 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.

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 [10], 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. [11] [12], 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.

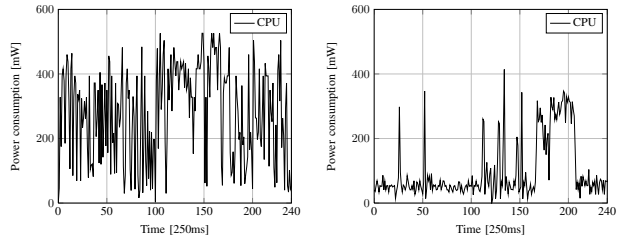
Obviously, the collection of accurate power-consumption measurements on smartphones is a key aspect of our technique. We discuss details of this aspect in the next section.

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 PowerTutor tool by Zhang et al. [6]. Another tool that would allow for the acquisition of this kind of information is Trepn [13]. In contrast to PowerTutor, Trepn uses hardware sensors and thus promises more exact measurements. However, Trepn is limited to the Snapdragon mobile development platform [14] and can hence not be applied on typical Android based end-user devices.

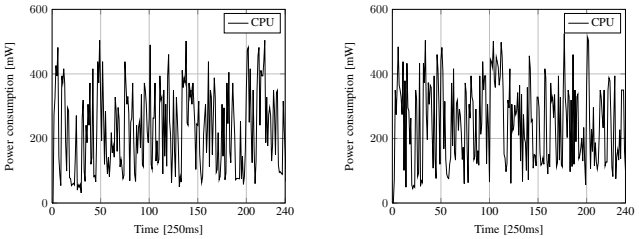
The PowerTutor tool is basically a smartphone application that measures the power-consumption of all applications running on the same smartphone. For each application, the power consumption of the six smartphone components *CPU*, *Audio*, *Display*, *Wi-Fi*, *3G* and *GPS* is measured separately. Figure 1 shows the measured power consumption of the CPU component caused by the two applications Android Browser and Lookout Mobile Security.

Although there are obvious differences in the power consumption of these two applications, an immediate identification and classification of applications based on such measurements is usually 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. Figure 2, which shows two different measurements of the same application, illustrates this fact. Although stemming from the same



(a) Android Browser (b) Lookout Mobile Security

Figure 1: One minute plots of CPU power consumption



(a) Angry Birds - Run 1 (b) Angry Birds - Run 2

Figure 2: One minute plots of CPU energy consumption

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. 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. Details of the proposed techniques are discussed in the next section.

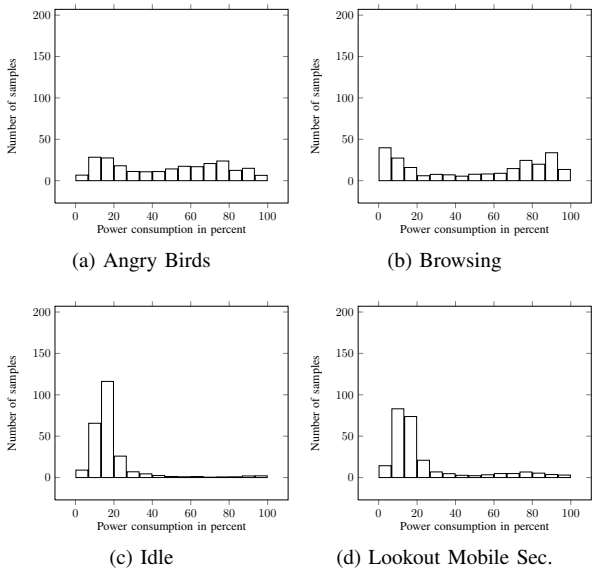
IV. Classification Techniques

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 consumptions, 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 3 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 V.



(a) Angry Birds (b) Browsing

(c) Idle (d) Lookout Mobile Sec.

Figure 3: Average histograms of different applications

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 [15] [16] and is also frequently used for music similarity finders [17][18]. In such systems, MFC coefficients and their distribution are extracted from recorded voice or music using complex transformations as implemented by the *melcepst* function [19]. 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 are based on an existing speaker-recognition implementation by Anil Alexander [20]. 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 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 4 illustrates the distribution of two 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.

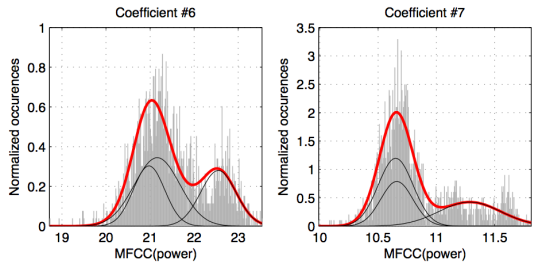


Figure 4: Gaussian Mixture Models of two MFCC derived from power-consumption measurements

V. 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 PowerTutor tool. 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 IV.

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.

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. 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.

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.

	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

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

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 II: Classification results (histogram based approach)

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. 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 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.

	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

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

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

Table IV: Classification results (GMM based approach)

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. However, given their relating 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.

VI. Conclusion and Future Work

Malware on smartphones is a growing issue and a major challenge for future mobile computing solutions. To overcome this challenge, new and innovative methods to detect malware on smartphones 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 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. 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] GO-Gulf.com, "Smartphone Users Around the World - Statistics and Facts," 2013. [Online]. Available: http://www.go-gulf.com/blog/smartphone/

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

[3] Google, "Android," 2013. [Online]. Available: http://www.android.com/

[4] Lookout Mobile Security, "2011 Mobile Threat Report," Tech. Rep. [Online]. Available: https://www.mylookout.com/mobile-threat-report

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

[6] L. Z. L. Zhang, B. Tiwana, R. P. Dick, Z. Q. Z. Qian, Z. M. Mao, Z. W. Z. Wang, and L. Y. L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," p. 105, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1878961.1878982

[7] W. Enck, L. P. Cox, P. Gilbert, and P. Mcdaniel, "TaintDroid : An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Design*, pp. 1–6, 2010. [Online]. Available: http://www.usenix.org/event/osdi10/tech/full_papers/Enck.pdf

[8] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromalyâ: a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2011. [Online]. Available: http://www.springerlink.com/index/10.1007/s10844-010-0148-x

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2089125.2089126

[10] G. A. Jacoby, R. Marchany, and N. J. I. Davis, "Battery-based intrusion detection a first line of defense," 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1437827

[11] 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," pp. 296–296, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4439001

[12] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," *Power*, pp. 239–252, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1378600.1378627

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

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

[15] 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, Jan. 2000. [Online]. Available: http://dx.doi.org/10.1006/dspr.1999.0361

[16] G. Kumar, K.A.Prasad Raju, Dr.Mohan Rao CPVNJ, and P.Satheesh, "Speaker Recognition Using GMM." [Online]. Available: http://core.kmi.open.ac.uk/display/910430

[17] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001.* IEEE, 2001, pp. 745–748. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1237829

[18] K. West and P. Lamere, "A Model-Based Approach to Constructing Music Similarity Functions," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, p. 024602, Jan. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1288980.1289110

[19] Mike Brooks, "VOICEBOX: Speech Processing Toolbox for MATLAB." [Online]. Available: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html

[20] Alexander Anil, "Automatic Speaker Recognition: A Simple Demonstration using Matlab," 2004. [Online]. Available: http://www.anilalexander.org/publications/Speaker_Recognition_Intro_A-A.pdf