Seeing the Unseen: Revealing Malware Hidden Communications Using Mobile Device's Energy Consumption

Luca Caviglione, Mauro Gaggero, Jean-Francois Lalande, Marcin Urbański, and Wojciech Mazurczyk

***Abstract*—Modern malware uses advanced techniques to hide from static and dynamic analysis tools. To this aim, an effective approach to achieve stealthiness when attacking a mobile device is the use of a covert channel allowing two colluding applications to locally exchange data. Since this process is tightly coupled with the used hiding method, its detection is a challenging task, also worsened by the very low involved bitrates. As a consequence, researchers are investigating how to reveal the presence of malicious software by using general indicators such as the energy consumed by the device. In this perspective, this paper proposes a framework based on artificial intelligence tools such as neural networks and decision trees that exploit power measurements to detect when data is hiddenly exchanged for malicious purposes. Two detection methods are proposed that are able to learn from a set of collected battery drains whether hidden communication is present or not. To verify the effectiveness of the proposed methods, we implemented seven covert channels, five from the related literature and two new ones, and developed a measurement testbed using Android devices. Experimental results show the feasibility and effectiveness of the approach in detecting the hidden data exchange between the colluding applications.**

I. Introduction

Modern malware uses advanced techniques to defeat static analysis tools or live detection systems. Even if designing a malware is nowadays considered quite common [1], the most advanced programmers try to hide malicious behaviors by using different techniques, such as the repackaging of legitimate applications or the obfuscation/ciphering of malicious code. Besides, by automating such mechanisms, a single attacker can add malicious code to several applications that can be sent to alternative markets. As a consequence, classical detection approaches using signature-based methods have limited results [2].

Nowadays, one of the most advanced mechanisms used by malware to exfiltrate information or to prevent limits imposed by security frameworks of mobile devices, relies upon utilization of information hiding techniques to exchange data between different processes. Especially, as in the case of smartphones, a local covert channel can be used to setup a communication path between two colluding applications to extract personal information from a targeted device [3], [4]. As it was recently observed in [5] mobile devices are potentially more suitable for hidden communication purposes due to rich variety of resources which can be exploited as they natively incorporate cameras, GPS, WLAN, Bluetooth, cellular networks, and other various sensors. Morever, currently malware developers turned a significant portion of their attention to mobile devices, leading to a 1800 percent increase in mobile malware over the past two years [6]. Therefore there is an urge for research efforts to design original countermeasures and enable early prevention. Unfortunately, applying effective countermeasures is very difficult, as they strictly depend on the type of covert channel. For example, exploiting electromagnetic signals to covertly transmit data is very different from manipulating the statistics of the available RAM to embed secrets [5]. Additionally, covert channels typically achieve quite a modest bandwidths, thus increasing the complexity of finding out whether a hidden exchange is ongoing or not.

In this perspective, we believe that a promising approach aims at exploiting general information in order to detect covert channels. In more details, a recent debate has emerged about the possibility of using the power consumption as an indicator to identify malicious activities. Despite [7] shows that malware cannot be detected by high level applications measuring energy consumptions of processes, other works demonstrate that proper power measurements can reveal some threats [8]–[10].

In this paper, we prove the feasibility of malware detection exploiting covert channels using measurements of the consumed power. In order to support this claim, we have reimplemented five popular covert channels available in the literature targeting the Android platform [4], [11] and proposed and implemented two new ones. We developed an experimental setup to quantify the energy consumption of the software components running on a mobile device. In more details, we utilized measurements provided by the high level model of PowerTutor [8] with values available in the `/sys` portion of the filesystem [10].

To perform the detection using the collected energy measurements, we developed an approach based on two artificial intelligence tools: neural networks [12] and decision trees [13]. These tools have been proven to be effective in modeling different aspects of networking [14], and detecting malware in mobile devices [15]. The two detection methods are able to learn from a set of past collected data whether hidden

L. Caviglione and M. Gaggero are with the Institute of Intelligent Systems for Automation, National Research Council of Italy, Genoa, Italy (e-mail: luca.caviglione@ge.issia.cnr.it; mauro.gaggero@cnr.it).

J-F. Lalande is with INSA Centre Val de Loire, Bourges, France and with CIDRE team, CentraleSupélec / Inria, Rennes, France (e-mail: jean-francois.lalande@insa-cvl.fr).

W. Mazurczyk and M. Urbański are with the Warsaw University of Technology, Institute of Telecommunications, Warsaw, Poland (e-mail: wmazurczyk@tele.pw.edu.pl; m.urbanski@stud.elka.pw.edu.pl).

communication is present, and are able to detect threats on line based on energy measurements. The first approach requires the solution of a regression problem to predict the future behavior of energy consumption. A hidden communication is spotted if the difference between the actual and predicted consumption is large. The second method is based on a classification problem, and gives information on hidden communications using a specific set of features characterizing the energy behavior of the device. Even if such artificial intelligence tools have been already used to detect malicious code [16] or to prevent the execution of hazardous software on Android devices [17], no previous work focused on a general way of detecting information hiding-capable malware using energy consumption as the basis for the detection.

To summarize, the main contributions of the paper are: *i*) supporting the feasibility of using energy footprints for the detection of malware implementing information hiding techniques, *ii*) the creation of an ad-hoc testbed and a hybrid measurement method to characterize seven popular covert channels, including two new covert channels, and *iii*) the development of two "intelligent" frameworks to perform the detection having low computational requirements.

The rest of this paper is structured as follows. Section II reviews the literature dealing with the detection of malware by using energy measurements. Section III details the reference scenario and the considered covert channels, while Section IV describes the methodology used for the measurements of power. Section V introduces the detection methods based on neural networks and decision trees, and Section VI discusses experimental results. Finally, Section VII concludes our work.

II. Related Works

Anomaly detection using energy footprints has been already partially investigated in the literature, primarily for malware and network attacks. However, at best of the authors' knowledge, it has never been applied to covert channels.

In general, energy-based anomaly detection methods can be grouped according to how measurements are performed. In more details, we have the following approaches:

- *System-based* [7], [18]–[20]: the energy footprint is created by considering the whole consumption of the device or some specific sets of applications and/or hardware parts. The obtained data represents the "clean system", which serves as a baseline for potential malware discovery.
- *Application-based* [21]: similarly to the previous case, the energy footprint is created for a well-defined pool of applications (e.g., games), and each one is measured separately. The collected traces are then compared at runtime against the data obtained with a single-process granularity.
- *User-based* [22], [23]: the energy footprint is created by analyzing the typical behavior of users and the related power consumption. This also includes, for instance, what applications and device features are active as well as their timing statistics.
- *Attack-based* [9], [10], [24]–[26]: measurements are done while real attacks or malicious malware activities are performed in a controlled environment. Acquired traces form a database of energy signatures, which is used for detection.

It is worth nothing that methods belonging to the first three groups can potentially detect unknown threats, while the last only allows to recognize attacks for which signatures are available.

Concerning system-based methods, Jacoby et al. [19] introduced a battery-based Intrusion Detection System (IDS) relying upon the analysis of the power consumption of the device, as well as the use of levels of some critical hardware/software components like the CPU. The proposed solution was proven to be effective for the detection of network attacks. Later, Nash et al. [20] proposed to estimate the energy footprint of a desktop computer by using a multiple linear regression model considering the CPU load, read/write accesses to the storage unit, and network activity. The measured parameters are combined with performance data counters available in the operating system, and the results are compared with different thresholds to evaluate the presence of malicious activities. Such approach was proven to be effective, but its main issue is the complexity to compute proper numerical values for the needed thresholds. Liu et al. [18] proposed the VirusMeter tool for Symbian smartphones that relies on the calculation of the energy profile for the whole "clean" system. Then, a heuristic is used to infer the actual energy consumption, which is compared against a reference value. If an anomaly is detected, the user is alerted. Lastly, Hoffman et al. [7] performed comprehensive experiments with both artificial and real-world malware using observation windows of different lengths. They created energy footprints in a controlled setting for the WiFi and 3G hardware of a smartphone, and the resulting power consumption is treated as a baseline for detection. Even if such an approach is effective, the main contribution of the work concerns the noise level of tools used for measurements: the additional power consumed by a malware is often too small to be detectable with the resolution of many measurement software.

For the techniques using application-based methods, the most notable work has been proposed by Kim et al. [21]: they monitor the power consumption in order to detect malware in Windows Mobile smartphones. Their proof-of-concept detection solution is based on the energy footprints of the applications running on the smartphone, which are compared against "clean" consumption templates.

Concerning methods explicitly considering the behavior of the user, Dixon et al. [22], [23] showed that there is a strong correlation between the battery drain of a mobile device and the user's location. This can be exploited to determine the average power consumption for different locations and make the detection of abnormalities more efficient. In fact, instead of considering the required power only as a function of time, location-specific energy profiles are used as additional indicators.

Regarding approaches exploiting consumption of attacks, Buennemeyer et al. [25] considered the energy signatures of some network threats against mobile devices. In more details, the power consumption of a device is correlated with IEEE

802.11 communication activities. If an irregularity is discovered, it is compared with existing signatures to perform the detection of the attack. Then, each mobile device exchanges alerts with peers, thus implementing a distributed network IDS. This work has been further extended by modifying the rates at which the battery status is polled, and by considering the activity of the Bluetooth air interface as to increase the performance in terms of correct detections [26]. Caviglione and Merlo [9] researched how network attacks such as port scan or ping floods, as well as common security mechanisms like antivirus, impact over the battery depletion of different smartphones. They state the need for "green security" mechanisms to effectively develop consumption-based malware detection systems is emphasized. Curti et al. [24] studied the energy footprints for benign applications like Skype or Youtube and also for network attacks like Denial of Service (DoS). They also provided a power consumption model for the hardware involved in WiFi communications allowing to distinguish a normal traffic pattern from a network attack. This work has been further extended by Merlo et al. [10] by analyzing the feasibility of porting the two aforementioned approaches on Android devices with the aim of developing a malware detection framework. Unfortunately, the proposed solutions are not very suitable for the purpose, mainly due to implementation issues, which can be overcome by introducing the direct observation of power consumption from the battery hardware without the need to dwell deeply into the drivers.

Summarizing, a large number of works on the topic are available however the presented results are not always conclusive. For instance, Dixon et al. [23] and Curti et al. [24] report promising results, whereas Hoffmann et al. [7] are rather pessimistic about the usability of energy as a metric for malware detection. Literature also indicates that future research direction should consider hybrid approaches, i.e., the power consumption should be enriched with additional information such as the memory usage. Even if hybrid approaches are also possible, they are typically implemented in a standalone fashion [19] or as a part of a larger network-based IDS (see, e.g., [25]). Furthermore, mobile device evolve very dynamically, thus studies performed even few years ago could quickly become obsolete as functionality and capabilities of modern devices significantly outrun the old ones. In this paper we introduce a local covert channel detection approach for mobile devices that belongs to the system-based methods group presented above and that relies on neural networks and decision trees. This approach has been not pursued before especially in the information hiding-capable malware context.

III. Covert Channels

In this section, we describe how a malware prototype that exploits a local covert channel to secretly leak sensitive data has been developed, and how we studied its main characteristics.

A. Reference Scenario

We consider the typical scenario depicted in Figure 1, where a malware composed of two colluding applications exchanges

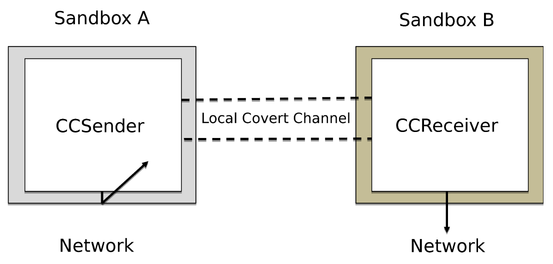


Fig. 1. Typical communication scenario of two colluding parts of a malware: `CCSender` and `CCReceiver` exchange data through a local covert channel.

data through a local covert channel in order to exfiltrate sensitive information from a mobile device [3], [4], [11]. In more details, the process `CCSender` has access to the data but has not the permission to use the network. Instead, the colluding application `CCReceiver` has access to the network and will be able to exfiltrate the received data to an external server or a Command & Control (C&C) facility. Obviously, the communications of `CCReceiver` towards the C&C could be detected, for instance by inspecting the traffic produced by the device. Thus, it is common to use another information hiding method to build a network covert channel within the produced network flow. In addition, some malware uses a TOR client to reach the server anonymously, thus making classic traffic analysis ineffective [27]. Consequently, analyzing an encrypted flow of information produced by the `CCReceiver` does not help to reveal the presence of a malware, and this is why this work focuses on the detection of the local covert channel itself.

Moreover, we also assume that the malware monitors user's activity in order to transmit data when he/she is not active [28]. Indeed, activating the covert channel during user's activity could degrade the performances of the device and reveal the presence of the malware. To this aim, many modern malware delay their activation in order to be invisible or not rise attention. Thus, even if the triggering of the malware may occur at any time, it is more likely to happen when the user is not active. For example, the families of malware DroidKungFu 1 and 2 include a time bomb mechanism that triggers the malicious behavior after a predefined period of time [29]. With such a protection mechanism, a malware would be running statistically when the smartphone is idle.

B. Implemented Covert Channels

For experimental purposes we chose and implemented the following seven local covert channels between the processes `CCSender` and `CCReceiver`. Five of them have been already proposed in the literature and they are:

- *Type of Intent* [11]: the secret receiver registers 256 types of Intent listeners and the secret sender encodes data by choosing and sending an intent of the right type.
- *File Lock* [30]: the secret sender communicates by locking a file. The secret receiver also tries to lock the same file: if it succeeds, a `0` is inferred. Otherwise, an exception is raised, meaning that the secret sender has locked the file before the secret receiver. In this case, a `1` is received.
- *System Load* [11]: the secret sender sends a `1` by burdening the CPU of the smartphone. The secret receiver

TABLE I
MEASURED BANDWIDTHS (IN BITS PER SECOND) FOR THE CONSIDERED COVERT CHANNELS COMPARED TO THE THEORETICAL VALUES.

Covert Channel	Th. Bandwidth	Meas. Bandwidth
Type of Intent	15K	74.62
File Size	2K	73.96
Memory Load	500	74.00
File Lock	250	9.31
System Load	5	6.43
Volume Settings	450	33.39
Unix Socket Discovery	100	74.57

checks how many clock ticks the sender has got since the previous iteration. If the value is greater than a certain value, it is assumed that a `1` should be inferred and a `0` on the other case.

- *Volume Settings* [30]: the secret data is encoded into the ringtone volume level of the device. If the sender can use eight levels of volume, it can send up to three bits per iteration.
- *Unix Socket Discovery* [11]: the secret data is sent by encoding the information within the state of a socket. Specifically, a closed socket is equal to `1`, while an opened one is equal to `0`.

Additionally, we are proposing two new covert channels that have never been tested in the literature, namely:

- *File Size*: the secret sender sets the size of a shared file and the secret receiver interprets it as a byte.
  Two new covert channels
- *Memory Load*: the secret receiver acquires the initial memory load of the secret sender. Then, the secret sender inflates the allocated data to send a `1`, or releases memory to send a `0`.

Before investigating the related energy footprints, we implemented the selected covert channels on the Android platform. Then, we conducted a performance evaluation to test their correctness, also in the perspective of removing possible power-hungry bugs, which can make our measurements incorrect. Besides, the literature related to such covert channels provided theoretical limits and/or performance evaluations in a very mixed set of testbed **Wojciech: we should either explain the huge difference in theoretical vs. measured bandwidth or we should remove th band row as it does not bring nothing new**. Thus, we also investigated the data throughputs achieved in our setup using our unique testbed. For this round of tests, we used a Samsung Galaxy SIII smartphone. Table I reports the mean bitrate achieved by each covert channel averaged over 100 repeated trials together with the estimated capacities as provided in the related references. In more details, the best obtained throughput is equal to 74.62 bit/s for the case of the Type of Intent covert channel, while the lowest is 6.43 bit/s for the System Load method. Moreover, at least four methods have similar maximum bandwidths, which suggests that we have reached some limits within the Android platform.

IV. MEASUREMENT METHODOLOGY

In this section we present the methodology used to measure the power consumption of the colluding applications using different covert channels. In order to have reliable data, we have preliminary investigated the literature to find the most suitable ways to collect information on the power used by applications running on Android devices. As pointed out in [10], the energy can be measured at high level using the Android APIs, or at low level by probing the battery driver. Performing measures at low level is a difficult task, as it requires to patch the battery driver to get access to such fine-grained data. Unfortunately, using high-level APIs may lead to values with a poor degree of reliability. For example, [10] and the references therein report that such values are not accurate enough to detect some form of network attacks, and also proposes an alternative mechanism defined as “middle level measure”, which exploits information stored in the `/sys` portion of the filesystem such as the current voltage of the battery.

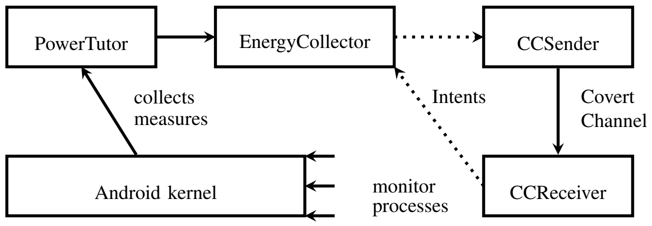


Fig. 2. Box diagram of the energy measurement architecture.

Based on these results, we decided not to collect low-level energy values for two main reasons: the technique requires deep changes in the operating system thus lacking of portability, and it has a non-negligible consumption that can mask the one of the covert channel used by the colluding applications. Additionally, since our objective is the detection of a threat without recurring to data related to the power consumed by the network subsystem, our effort of using a mixed high and middle level methodologies is novel [31].

In more details, to quantify the power depletion we relied upon: *i*) high level energy consumption of each process running on the system. To this aim, we used a modified version of PowerTutor [8], i.e., we developed a patch to enable the tool to send different process consumption to a proper data collector via an Android Intent; *ii*) middle level energy consumptions acquired by gathering current and voltage values stored in `/sys`.

Then, we developed a measurement framework both for collecting data and automatize the experiments. Figure 2 depicts the box diagram and the major interactions among the different subsystems. In particular, the EnergyCollector is the controller of our experiments and it is in charge of running repeated trials and collecting data measured by PowerTutor from APIs and `/sys`. The latter sends gathered data each second by using an intent. Obviously, the communication flow between the EnergyCollector and the `CCsender`/`CCreceiver` would not exist on a real system, but it is needed for automating the experiments. Nevertheless, after measuring such an overhead, we can assume that the impact of a single intent is negligible in terms of additional measured power.

For each covert channel, we conducted repeated trials according to the following flow:

1) the EnergyCollector waits a random time;
2) the EnergyCollector sends an intent to the `CCsender` to start a covert transmission;
3) `CCsender` and `CCreceiver` exchange a message;
4) `CCreceiver` sends an intent to notify the EnergyCollector that the transmission is finished;
5) the EnergyCollector waits a random time;
6) the experiment is repeated.

The random time inserted in 1) ensures that the beginning of the covert channel transmission is not known a priori and that two different exchanges are not "triggered" within a timeframe. This enables to have an unpredictable behavior, which is more coherent with real-world use cases. The duration of the transmission depends on the type of the covert channel, since each one has a different bandwidth.

V. Energy Consumption and Black-box Modeling

The basic idea of our work is to detect covert communications among colluding applications by using information about the energy requirements of the processes running on a smartphone. Each process correspond to a software entity (Activity with eventually additional services) using the resources of the device. The main benefit of using energy drains as a marker is that they allow to decouple the detection of the covert channel from the method used for its implementation: hidden communications are tightly coupled with the used carrier and are characterized by a very low bitrate, thus making their detection very difficult and poorly generalizable.

For detecting covert channels, two general problems are addressed: *i*) developing an approximate model for the power consumption of a process, and *ii*) using the obtained information to detect whether a covert channel is present. Achieving such goals is a very difficult task, especially due to the different sources of power drains, such as transmissions over air interfaces, memory operations, CPU- or I/O-bound behaviors, user to kernel space switches, just to mention a few. Thus, we propose two general detection methods that are based on the black-box modeling approach.

The first method is based on a regression that exploits a collection of past values of the consumption to predict its future behavior. Then, if the expected energy footprint deviates too much from the real one, a trigger is raised in order to spot hidden communications. In the following, we will refer to such technique as regression-based detection (RBD).

The second method exploits a reduced set of features describing the energetic behavior of the device. More specifically, a classification problem is solved to detect covert communications between colluding applications. For the sake of brevity, we will refer to such an approach as classification-based detection (CBD).

In both cases, well-known artificial intelligence tools such as one-hidden-layer feedforward neural networks and binary decision trees are used to model the power consumption. In more details, RBD and CBD are built in two steps, as shown in Figure 3. First, a model of power consumption is constructed based on a set of past collected measurements. This requires the solution of an optimization problem to find the best values of the parameters the model depends upon. Such a step is called "training" and is usually performed offline, i.e., it is not performed on the mobile device. The collected measurements used in this phase are called "training set". Second, the optimized model is used to detect covert communications based on the new, actual measurements of power consumption. Such a step can be performed online, i.e., when the mobile device is running.

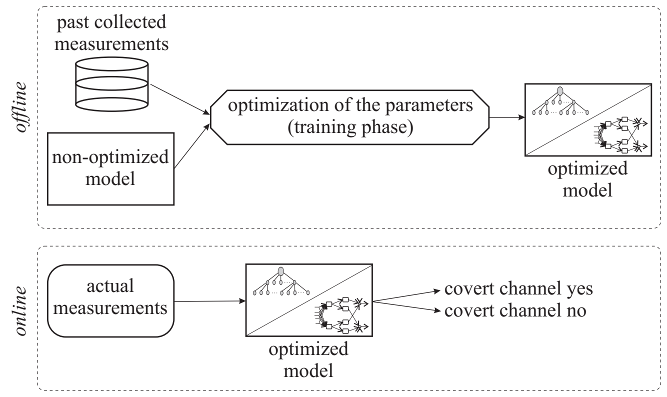


Fig. 3. Two-step procedure for the detection of covert channel communications.

A. Neural Networks and Decision Trees

In this section, we report some preliminary concepts about the tools used to construct the approximate model of power consumption, i.e., one-hidden-layer feedforward neural networks and binary decision trees.

Both tools are widely employed in the literature to approximate unknown relationships between input and output variables without any a-priori knowledge on the underlying dynamics. They can be used to solve either regression or classification problems. Regression models map the input space into a real-valued domain. Instead, classifiers map the input space into predefined classes. The relationships between input and output variables is learned from the collected data and then used to associate an output to new, unseen inputs. Neural networks belong to the family of "parametric" approximators, as their output significantly depends on the parameters that define the structure of the model, whereas decision trees are usually referred to as "non-parametric" approximators, i.e., they strongly depend from the available data and, less significantly, from a a reduced set of parameters or meta-parameters.

As said, the goal is to approximate an input/output mapping $\underline{x}_i \mapsto \underline{y}_i$, where $\underline{x}_i \in \mathbb{R}^n$ is the input and $\underline{y}_i \in \mathbb{R}^m$ is the corresponding output. The unknown functional relationship between inputs and outputs is then approximated by a function γ belonging to a suitable family Γ of parametrized functions, i.e.,

$$\underline{\tilde{y}_i} = \gamma(\underline{x}_i, \underline{\alpha})$$

where α is a vector of parameters and $\underline{\tilde{y}_i}$ is the estimated value of $\underline{y}_i$.

In recent years, the scientific literature of learning from data mainly concentrated on the use of nonlinear models for the approximation of complex systems. In fact, it has been proved that classical linear models may be computationally intractable for the approximation of complex functions, especially in the case of high dimensionality of the input variables (see, e.g., [32], [33] and the references therein). On the contrary, it has been shown that nonlinear structures have a greater flexibility, and allow to obtain good approximations with a smaller number of parameters, while guaranteeing the same accuracy.

One-hidden-layer feedforward neural networks are a very popular nonlinear architecture already used to model a variety of systems (see, e.g., [14], [34] and references therein). The function γ is given by a linear combination of parametrized basis functions, in which a certain number of free parameters to be optimized is inserted. For scalar outputs, we have

$$\gamma(\underline{x}, \underline{\alpha}) = \sum_{n=1}^{\nu} c_n \sigma \left(\sum_{j=1}^{q} a_{nj} x_j + b_n \right) + c_0$$

where $\sigma(\cdot)$ is the activation function, ν is the number of basis functions (i.e., the neurons) and $\underline{\alpha} \triangleq \operatorname{col}(a_n, b_n, c_n, c_0)$ is the vector of free parameters. In this work we focus on sigmoidal neural networks, i.e., the activation function is a sigmoid. It is well known that this family of nonlinear models is equipped with the so-called "universal approximation" property, i.e. the ability to approximate any well-behaved function with arbitrary precision. This makes one-hidden-layer feedforward neural networks good candidates to model the behavior of the power consumption considered in this paper.

Concerning binary decision trees, they consist of a piece-wise constant function on a partition $T = \cup_{m=1}^{M} B_m$ of the input domain, where B_m are the "leaves", i.e., proper subsets with sides parallel to the coordinate axes. The partition is called a "tree" because the leaves are added recursively with binary splits to form a tree structure. More specifically, a first split of the entire input domain is performed to obtain two regions divided by a hyperplane that is parallel to one of the coordinate axes. Then, each of such regions is further divided into two subregions, and so on until some stopping criterion is reached. In this paper we focus on binary trees, in which each step of a prediction involves checking the value of only one variable at a time. The functional relation (**??**) is made up by a sequence of "if-then" rules, representing the splits of the input domain into the above-describe subregions. **Mauro: expand if space allows checking reference [13] JFL: where is equation eq:gamma ? JFL: additionally, I found the end of this paragraph extremely difficult to understand.**

After choosing the family of functions Γ, we need to find the element γ^* that is capable of reproducing at best the system behavior. This corresponds to a training procedure that consists in the optimization of the parameters on the basis of the available data. To this purpose, a suitable cost index is used that measures the discrepancy between the estimated model output $\underline{\tilde{y}_i}$ for a certain value of the parameters and the real measured output values $\underline{y}_i$. The most used index in regression problems corresponds to a mean squared error (MSE) criterion: the optimal parameter vector $\underline{\alpha}^*$ is the one that minimizes the quadratic difference between the real output variables and the estimated ones, i.e.,

$$\underline{\alpha}^* \triangleq \min_{\underline{\alpha}} \left\{ \frac{1}{N} \sum_{i=1}^{N} (\underline{y}_i - \gamma(\underline{x}_i, \underline{\alpha}))^2 \right\}$$

where N is the number of samples used for the training.

In general, the training phase may be computationally demanding, especially for large values of N. However, many ad-hoc developed procedures are available in the literature **Wojciech: I think that few citations are needed here...**. Usually, they are implemented in efficient software packages, and allow to obtain a solution to the training problem in a reduced amount of time.

B. *Regression-Based Detection*

The RBD method is composed of two steps: the first one consists in modeling the power consumed in a "clean" system, i.e., when no colluding applications performing hidden data exchanges are present in the device. To this end, a collection of past values of the consumption are used to predict the future ones. The second step is based on a comparison between the forecasted consumption and the actual one. Hidden communications are spotted if the expected energy footprint deviates too much from the real one.

In more details, we define the following quantities at each time t (we assume measurements occur at discrete time instants $t = 0, 1, \ldots$). Let $p_t \in \mathbb{R}_+$ be the power consumed by a process at time t. Such a quantity is measured and can be considered as an input. Let $w_{t+1} \in \mathbb{R}_+$ be the prediction of the power needed by the process itself at the next time instant $t+1$ (i.e., it is an output).

At least in principle, the consumption of a process at time $t+1$ may depend on the consumption at the previous time instants, from time 0 to time t. However, to avoid dealing with vectors of increasing dimension as the time grows, we make a "fading memory" assumption, which consists in assuming that the output of the system (i.e., the power consumption of a process at time $t+1$) depends only on a finite number q of past inputs. This assumption is commonly made in the literature of regression-based approximations [35]. In this perspective, we define a regression vector (also called simply regressor) as the collection of the past input variables from time $t-q+1$ up to time t, i.e., $\underline{p}_t \triangleq \operatorname{col}(p_{t-q+1}, p_{t-q+2}, \ldots, p_t)$, where q is a positive constant value.

Thus, the training set for the creation of the model takes on the form of a set of input/output pairs

$$\Sigma_N \triangleq \left\{ \underline{p}_t, w_{t+1} \right\}_{t=1}^{N}$$

where N is the total number of available measures. The goal is to find a model that is able to capture, at each time t, the functional relationship between past and future consumption, i.e., the mapping $\underline{p}_t \mapsto w_{t+1}$. To this end, let $\tilde{w}_{t+1}$ be the estimated output at time $t+1$, i.e.,

$$\tilde{w}_{t+1} = \gamma(\underline{p}_t, \underline{\alpha})$$

where γ is a function belonging to a family Γ of functions with preassigned structure, and $\alpha \in \mathbb{R}^p$ is a vector of parameters. Clearly, the model has good predicting properties if $\tilde{w}_{t+1}$ is "sufficiently" close to w_{t+1}.

As said, we use as family Γ both one-hidden-layer feedforward neural networks and binary decision trees. Notice that, when the regressor is made up by a long series of past observations, the dimensionality of the problem may become very high, hence the need of efficient approximators arises.

Once the best model within the class Γ has been found by the training procedure, the second step of the RBD method requires the definition of a suitable detection rule. Specifically, the detection is performed by feeding at each time step t the trained model with the past q measurements of the consumption, collected into the regressor $\underline{p}_{t-1}$. Then, the estimated consumption $\tilde{w}_t$ is compared with the real measured value w_t in a time window moving over time. The same procedure is repeated at the next time steps, and a prediction error e_t is defined as follows:

$$e_t \triangleq \frac{1}{\tau} \sum_{k=t-\tau+1}^{t} |\tilde{w}_k - w_k|$$

where τ is a given time horizon. Then, we assume that there is a covert communication among two colluding applications if the prediction error is greater than a certain positive threshold ξ. Otherwise, we consider it absent. The rationale for this approach is that the approximate model, if properly trained, is assumed to predict with a good level of accuracy the future behavior of the energy consumption of the "clean" system. Hence, severe deviations reveals the presence of two processes covertly exchanging data.

Clearly, the performance of the RBD method depend on the quality of the approximating model and on the parameters q, τ, and ξ, which have to be properly tuned. We will observe their impact on the quality of detection in a real scenario in Section VI.

Since the approximate model γ is obtained in a "clean" system, it can be used to detect covert communications with all the seven covert channel types described in Section III.

C. Classification-Based Detection

The CBD method consists in the solution of a classification problem starting from a set of measurements both in the presence and in the absence of colluding applications covertly exchanging data. The method requires the definition of a set of "features" representing the power consumption of the device in a synthetic, but effective, manner.

In this paper we focus on three different features characterizing the energetic behavior of a process at each time t, collected into the vector $\underline{x}_t \in \mathbb{R}^3$: *i*) the average power consumption from time t to time $t-\lambda+1$, where λ is a positive constant defining a window of past measurements, *ii*) the total variation of the power consumption from time t to time $t-\lambda+1$, and *iii*) the instantaneous consumption at time t. Thus, we focus on the following vector of features at each time t:

$$\underline{x}_t \triangleq \text{col}\left(\sum_{l=t-\lambda+1}^{t} p_l, \ \sum_{l=t-\lambda+1}^{t} |p_l|, \quad p_t \right)$$

Each vector $\underline{x}_t$ refers to a single measurement and is associated to a certain class k among two possible ones, corresponding to the cases in which covert channels are used to exchanging data between colluding applications ($k = 1$) and no covert channels are established ($k = 0$).

The training set for the creation of the model takes on the form of a set of N input/output pairs

$$\Sigma_N \triangleq \{\underline{x}_t, y_t\}_{t=1}^{N} \tag{1}$$

where the scalar output y_t is equal to k if the input vector $\underline{x}_t$ belongs to the class k. Notice that in (1) we have assumed to have a total of N measurements.

The goal is to find a model that is able to recognize the class to which an input vector (that is not among the N used for the training) belongs. As in the case of the RBD method, we rely upon models belonging to a certain family Γ of functions with a preassigned structure, i.e.,

$$\tilde{y}_j = \gamma(\underline{x}_j, \underline{\alpha})$$

where γ is a function belonging to the family Γ and $\underline{\alpha} \in \mathbb{R}^p$ is a vector of parameters. As in the case of the RBD method, both one-hidden-layer feedforward neural networks and binary decision trees are used as models for the family Γ. The output of the model corresponding to a certain input must be one of the two possible classes, or directly as in the case of decision trees (in the sense that the output value is just one of the two classes) or through an assignment procedure based on the output value as in the case of neural networks (specifically, a rounding of the output to the nearest value between 0 and 1 is performed).

Thus, $\tilde{y}_j$ is the class assigned to the input vector $\underline{x}_j$ by the model. Let y_j be the actual class of $\underline{x}_j$. Clearly, the goal of the classification is to ensure that the assignment of the model is correct, i.e., the difference between $\tilde{y}_j$ and y_j is as small as possible. To this end, a suitable training of the model has to be performed to find the optimal values of the parameter vector $\underline{\alpha}$.

Once the best model within the family Γ has been found as the result of the training, at each time t the detection whether a covert channel between colluding applications is present or not is performed by feeding the trained model with the vector $\underline{x}_t$ of the current features and analyzing the value of the output $\tilde{y}_t$.

Clearly, the accuracy of detection of the CBD method depends on the quality of the approximating model and on the parameter λ used to construct the vector of features. We will see its impact on the quality of detection in a real scenario in Section VI. Notice that the CBD requires the tuning of only this parameter, whereas the RBD depends on three parameters. Another difference between the RBD and the CBD is that the former uses the same approximate model to detect covert communications with all the seven methods introduced in Section III, whereas the latter requires the training of a different model for each specific covert channel.

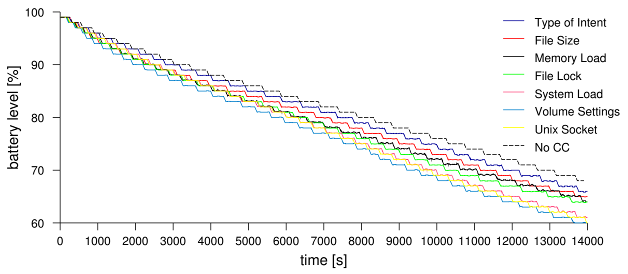


Fig. 4. Battery drain for each type of covert channel during 4 hours of repeated experiments.

VI. Numerical Results

To evaluate the effectiveness of the proposed approach, we conducted experimental trials with a testbed using two different smartphones, i.e., a Samsung Galaxy SIII and a LG Optimus 4X HD P880. A dataset containing the consumption of all the software components running on the smartphones was generated with a modified version of PowerTutor using the methodology presented in Section IV. In more details, the consumption was measured with a per-process granularity, both in a "clean" set-up and when a local covert channel among two processes is created. We also focused on a scenario in which information hiding-capable malware increases its stealthiness by acting when the device is idle i.e. there is no user activity involved. As regards the exchanged messages, we set a fixed size of $1,000$ bytes. Such a value is large enough to represent the exfiltration of sensitive information such as a bank account or a collection of contacts stored in the address book. Each covert channel was tested by performing 100 data transmissions starting at random time instants. To have a proper statistical relevance, each trial was repeated 10 times.

Figure 4 shows the trend of the battery drain for a period of 4 hours in the presence of colluding applications transmitting data with the seven implemented covert channels. By inspecting the trends, it can be noticed that the most consuming methods are the Volume Settings, the Memory Load, and the System Load, whereas the lowest one is the Type of Intent. We underline that, even if it is one of the most consuming, the System Load covert channel is also the one with the smallest bitrate.

Concerning the detection of covert communications with neural networks and decision trees, we tested the RBD and CBD methods using Matlab and the Neural Networks and Statistics Toolboxes. All the experiments were done on a computer with a 2.5 GHz Intel i7 CPU and 4 GB of RAM.

Preliminary investigations showcased that the overall power consumption of the smartphone is well represented by the energy required by processes with an OS-wide dynamics. Surprisingly, using only the process `System` gave us enough condensed information to capture the general trend. Indeed, the System server is the core of all mechanisms required for communicating with the low level drivers of Android. It is used for manipulating notifications, lights, audio, alarms, telephony just for naming a few. Therefore, for testing our methods, we relied upon the information provided solely by the process `System`. Then, using the measures of this process, we made several experiments: the first experiment measures a clean environment without any covert channel. Then, the next ones use one of the seven implemented covert channels.

The performances of the proposed detection methods were quantified by means of the percentage of correct detection of hidden communications, defined as follows:

$$d = \frac{1}{T}\sum_{t=0}^{T-1} 1 - |y_t - \tilde{y}_t| \tag{2}$$

where T is the length of each trial (dependent on the type of used covert channel) and y_t and $\tilde{y}_t$ denote the actual and spotted hidden communications at time t, respectively. In more details, as described in Section V-C for the CBD method, $y_t = 1$ if colluding applications are exchanging data and $y_t = 0$ otherwise.

A. RBD method

A training set composed of 5000 samples (i.e. energy consumption measurements) was used to optimize the parameters of neural networks and decision trees for the RBD method. Such samples were obtained in a clean system, i.e., when no hidden communication between colluding applications is present. The same approximator was used to detect all the seven implemented covert channels.

As said in Section V-B, the first part of the RBD approach requires the solution of a regression problem. The training was performed using the Levenberg-Marquardt algorithm [36] for neural networks and by minimizing the MSE of the predictions compared to the training data for decision trees.

The impact of the parameters q, ξ, τ, and ν on the detection performance has been investigated in a thorough simulation campaign in order to determine their best values. Concerning neural networks, Figure 5(a) reports the percentage of correct detections (d) of each covert channel defined as in Equation (2), averaged over 10 different trials, when ν is varied from 5 to 50 and the other parameters q, ξ, and τ are fixed to 20, 30, and 20, respectively. Figures 5(b) and 5(c) show the behavior of the average percentage of correct detection obtained with neural networks and decision trees, respectively, when varying the length of the regressor (q) from 5 to 30 and the other parameters are fixed. Figures 5(d) and 5(e) depict the average percentage of correct detections using neural networks and decision trees, respectively, as a function of the threshold ξ used for the detection rule, whereas the other parameters are fixed. Lastly, Figures 5(f) and 5(g) showcase the average percentage of correct detections obtained with neural networks and decision trees, respectively, as a function of the length of the time window (τ) used for the detection rule, with the other parameters fixed to specific values.

From the obtained results, it turns out that the best number of activation functions for neural networks is $\nu = 40$. Indeed, the percentage of correct detection increases with ν up to $\nu = 40$. For larger values, the phenomenon of overfitting is experienced, i.e., the number of basis functions is too large for the available data, and minor fluctuations in the available

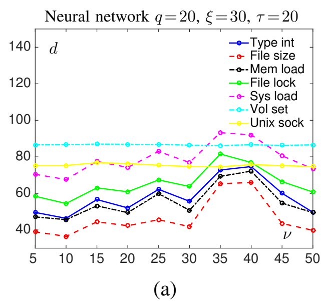
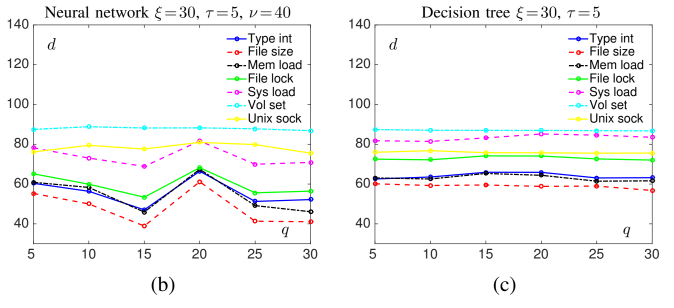
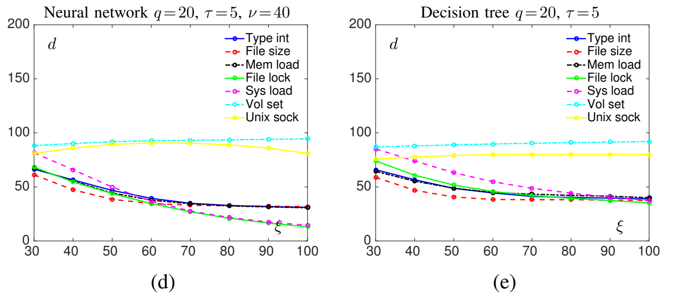
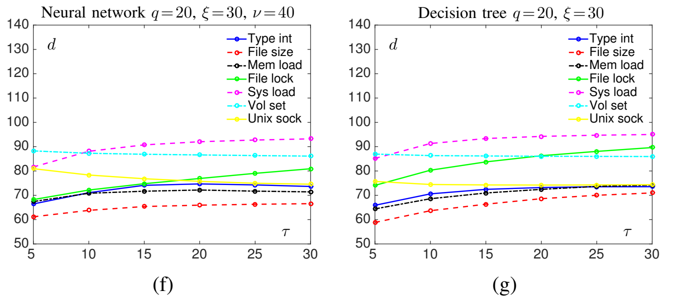





Fig. 5. Average percentage of correct detection for each covert channel using the RBD method when varying the parameters of neural networks (a, b, d, and f) and decision trees (c, e, and g).

data may be overemphasized, thus resulting into bad detection rates. Concerning the length of the regressor (q), the best value is $q = 20$. In this case, the behavior of neural networks is more affected by the chosen value if compared to decision trees, for which all the values of q guarantee the same results. Instead, neural networks and decision trees exhibit the same behavior when varying the thereshold ξ, for which the best value appears to be $\xi = 30$. Lastly, the percentage of correct detections increases if τ increases up to $\tau = 20$ and then remains almost constant. Thus, in the perspective of saving computational time, $\tau = 20$ appears to be the best value.

Figure 6 shows the boxplots of the percentages of correct detections (d) for each information hiding method computed over 10 different trials by using neural networks and decision trees with their best parameters' values, i.e., $\nu = 40$, $q = 20$, $\xi = 30$, and $\tau = 20$. We conclude that the performance of neural networks and decision trees are comparable, i.e., on the average the accuracy of the detection is the same in both cases. The most easily detectable method appears to be the System Load covert channel, whereas the method that is least detectable is the File Size.

Figure 7 depicts the measured trend of the consumption of the `System` process compared with its estimation provided by neural networks and decision trees when using the Volume Settings covert channel to exfiltrate data. The presence or absence of hidden communication is denoted by high or low values of the binary signal at the bottom of each picture. As it can be seen, the prediction of the energy consumption is more accurate when no covert communication is active, whereas the prediction is not accurate in the presence of colluding applications exchanging data by means of information hiding. More specifically, neural networks underestimate the power consumption, whereas decision trees saturate to a certain value. This is not surprising since, as said, the models have been built using “clean” system without colluding covert applications. The bad prediction when covert channels are present is fundamental to spot hidden communications.

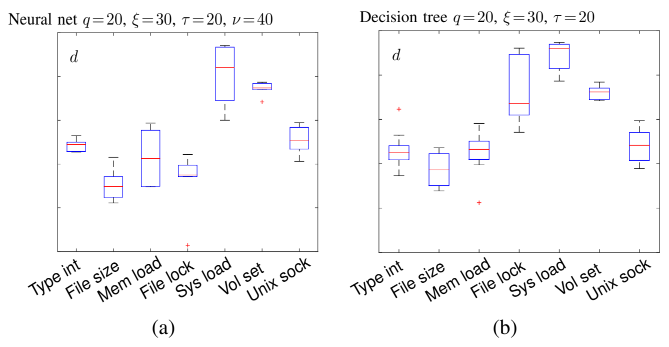


Fig. 6. Boxplots for the percentages of correct detection for each covert channel using the RBD method with neural networks (a) and decision trees (b).

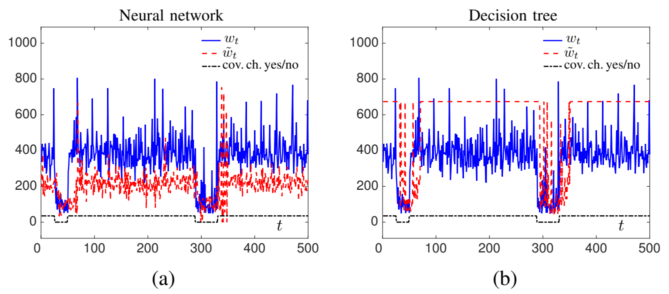
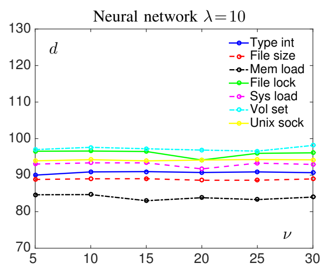


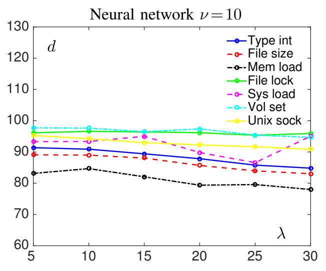
Fig. 7. Comparison of real and estimated power consumption of the `System` process for the Volume Settings method by using the RBD with neural networks (a) and decision trees (b).

B. CBD method

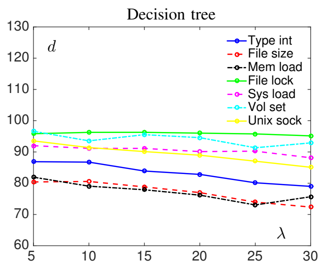
To test the effectiveness of the CBD method, we used again a training set made up of 5000 samples. Differently from the RBD method, the training was performed both when the colluding applications are active and inactive. Moreover,



(a)

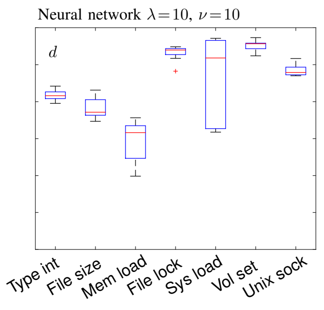


(b)

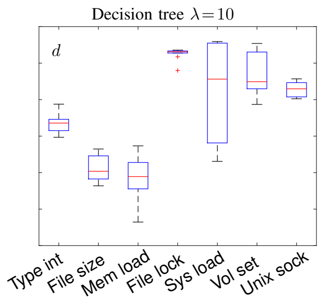


(c)

Fig. 8. Average percentage of correct detection for each covert channel using the CBD method when varying the parameters of neural networks (a and b) and decision trees (c).

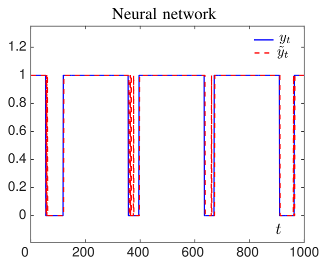


(a)

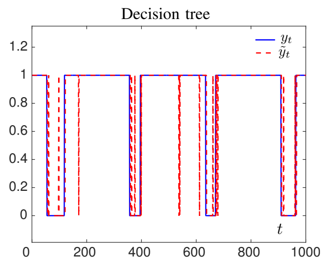


(b)

Fig. 9. Boxplots for the percentages of correct detection for each covert channel using the CBD method with neural networks (a) and decision trees (b).



(a)



(b)

Fig. 10. Comparison of true covert channel activity over time against the estimated one for the Volume Settings method by using the CBD with neural networks (a) and decision tree (b).

different approximators were trained for each of the seven implemented covert channels.

Since we have to solve a classification problem, the output of the neural networks (real numbers) were rounded either to 1 or 0 depending on whether hidden communication was spotted or not. Concerning decision trees, we adopted the so-called classification trees, whose output is directly one of the classes defined during the training. The training of neural networks was performed using the Levenberg-Marquardt algorithm, whereas classification trees were trained using the Gini's diversity index as split criterion [37].

For the case of neural networks, we varied both the number of neurons ν and the number of time instants λ for the computation of the features, in order to investigate their influence on the accuracy of the detection. Figure 8(a) depicts the percentage of correct detections (d) of each covert channel, averaged over 10 different trials, when ν is varied from 5 to 30 and λ is fixed to 10. Figure 8(b) presents the average percentage of correct detections for each information hiding method when λ ranges from 5 to 30 and ν equals to 10. It turns out that the number of neurons does not affect the accuracy of the detection too much. Therefore, to save memory and computational time, one might choose $\nu = 5$ or $\nu = 10$. Concerning the effect of λ, a small decay of performance is experienced for large values. Thus, $\lambda = 5$ or $\lambda = 10$ appears to be the best choice.

Concerning decision trees, we investigated the effect of the parameter λ on the accuracy of detection. The results are reported in Figure 8(c) for the average d. Also in this case, λ varies from 5 to 30. Similarly to the case when using neural approximators, the percentage of detection decreases if λ increases, hence, optimal values are again $\lambda = 5$ or $\lambda = 10$.

Figure 9 depicts the boxplots of the percentages of correct detections for each covert channel computed over 10 different trials using the approximate model with the best values of their parameters, i.e., $\nu = 10$ and $\lambda = 10$ for neural networks and $\lambda = 10$ for decision trees. In general, neural networks guarantee better performance compared with decision trees, i.e., on the average the accuracy of the detection is higher and with a lower variance. In both cases, the most easily detectable methods are the File Lock and the Volume Settings. Instead, the Memory Load method is the most difficult to detect despite its high consumption. In both cases, the System Load is characterized by the largest dispersion.

Figure 10 portraits the estimated covert channel activity compared to the real one for the Volume Settings method. As it can be seen, both neural networks and decision trees are able to correctly spot the channel activity most of the time, thus showcasing their effectiveness for run-time or static analysis purposes within the security framework of the device.

C. Comparison between RBD and CBD

To sum up, Table II reports the percentage of correct detections averaged over 10 trials for the RBD and CBD methods. In both cases, the System Load and the Volume Settings are the most easily detectable covert channels. This may be ascribed to the fact that such methods are also the most power-consuming, i.e., their energy footprint is more evident. In this case, the hidden communication is correctly spotted 9 times over 10. The most difficult methods to detect turns out to be the File Size and the Memory Load. However, even if lower than in the case of the best-performing methods, their average percentage of correct detection is about 65% when using the

TABLE II
AVERAGE PERCENTAGES OF CORRECT DETECTIONS FOR THE DIFFERENT DETECTION METHODS AND COVERT CHANNELS.

Covert channel	RBD		CBD	
	Neural net	Dec tree	Neural net	Dec tree
Type of Intent	74.3	73.1	90.8	86.7
File Size	65.3	68.6	88.9	80.5
Memory Load	71.5	72.4	84.6	79.0
File Lock	65.9	86.2	96.5	96.2
System Load	90.4	94.2	93.3	91.1
Volume Settings	87.1	86.0	97.6	93.5
Unix Socket	75.4	74.2	94.2	91.4

RBD and 85% for the CBD, which is a satisfactory result. The Memory Load covert channel seems the most difficult to detect. We think that this is due to the fact that there is no code in the System Server process that allocates memory: this task is done at high level by the Dalvik virtual machine and at low level by the Linux kernel.

According to the obtained results, in general the CBD method outperforms the RBD one in terms of percentage of correct detections. Moreover, it is worth noting that the RBD has three parameters to be tuned, i.e., q, τ, and ξ, instead of only one for the CBD, i.e., λ. As a consequence, the implementation of the CBD in a production quality detection tool should be preferred, both in terms of complexity and performance.

Concerning the computational complexity, the average time for the training was equal to 82.5 seconds for the RBD with $\nu = 40$ and $q = 20$ and to 15.6 seconds for the CBD with $\nu = 10$ and $\lambda = 10$ when using neural networks. The training times of the two detection methods when using decision trees with $q = 20$ and $\lambda = 10$ were equal to 1.92 and 0.27 seconds, respectively. The higher times of the RBD may be ascribed to the fact that the input vector has a greater dimension than the CBD. In fact, in the first case the dimension of the input is equal to the length $q = 20$ of the regressor, whereas in the second one it is equal to 3 (the number of features). This also requires a larger number of neurons to obtain satisfactory approximations. In general, neural networks appears to be more computationally demanding compared to decision trees. However, the RBD method requires the training of only one model, whereas an approximate model for each covert channel is required by the CBD approach, thus resorting to seven different training procedures.

As regards the online phase, the average time to spot if a covert channel is present for the RBD method with the best values of the parameters is equal to 0.01 and 0.001 seconds, depending on whether neural networks or decision trees are used, respectively. As regards the CBD approach, such times when using the best values of the parameters are again equal to 0.01 and 0.001 seconds for neural networks and decision trees, respectively. It is worth noting that, in all cases, the required online computational effort is very small. Thus, the proposed methods appear to be well-suited to being implemented in an online detection framework directly running on a smartphone.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper we have presented a framework based on artificial intelligence tools such as neural networks and decision trees to detect the presence of malware using information hiding techniques based on power measurements. Specifically, we have focused on the colluding application scenario, which is characterized by two processes trying to communicate outside their sandboxes for malicious purposes, for instance, for sensitive data exfiltration. Two detection methods have been developed, requiring the solution of regression and classification problems, respectively. To verify the effectiveness of our approach, we have implemented on the Android platform seven local covert channels, five taken from the related literature and two new ones, and we have performed an experimental measurement and detection campaign. The obtained results indicate that both proposed detection methods are characterized by a good detection performance and can be used as an accurate IDS (Intrusion Detection System) software on a modern smartphone in order to reveal the presence of hazards exploiting information hiding.

Since the proposed detection methods are not designed to reveal the pair of processes exchanging data through the covert channel, developing mechanisms to detect the pair of colluding applications, after the covert communication is discovered, is part of our current ongoing research **Wojciech: here we should try harder to defend our approach - I tried to do it but more considerations are needed**. Moreover, we also aim at extending the considered approach to other threats based on information hiding. These extensions to this work should lead to the development of a detection application directly running on a smartphone to spot the presence of covert communications in real-time .

REFERENCES

[1] K. Allix, Q. Jerome, T. F. Bissyande, J. Klein, R. State, and Y. le Traon, "A forensic analysis of android malware-How is malware written and how it could be detected?" in *Computer Software and Applications Conf. (COMPSAC)*, 2014, pp. 384–393.

[2] V. Rastogi, Y. Chen, and X. Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 99–108, Jan. 2014.

[3] J.-F. Lalande and S. Wendzel, "Hiding privacy leaks in Android applications using low-attention raising covert channels," in *Int. Conf. on Availability, Reliability and Security (ARES)*, 2013, pp. 701–710.

[4] W. Mazurczyk and L. Caviglione, "Steganography in modern smartphones and mitigation techniques," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 334–357, 2014.

[5] ——, "Information hiding as a challenge for malware detection," *Security Privacy, IEEE*, vol. 13, no. 2, pp. 89–93, Mar 2015.

[6] M. L. T. Report, "Mcafee labs threat report," August 2014. [Online]. Available: http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q2-2014.pdf.

[7] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprintsa dead end?" in *Research in Attacks, Intrusions, and Defenses*. Springer, 2013, pp. 348–368.

[8] 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 *Proc. IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, 2010, pp. 105–114.

[9] L. Caviglione and A. Merlo, "The energy impact of security mechanisms in modern mobile devices," *Network Security*, vol. 2012, no. 2, pp. 11–14, 2012.

[10] A. Merlo, M. Migliardi, and P. Fontanelli, "On energy-based profiling of malware in Android," in *Int. Conf. on High Performance Computing & Simulation (HPCS)*, 2014, pp. 535–542.

[11] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the communication between colluding applications on modern smartphones," in *Proc. Annual Computer Security Applications Conf.*, 2012, pp. 51–60.

[12] S. Haykin, *Neural Networks, A comprehensive foundation*. Prentice Hall, 1999.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Chapman & Hall, 1984.

[14] L. Caviglione, "Enabling cooperation of consumer devices through peer-to-peer overlays," *IEEE Trans. Consumer Electronics*, vol. 55, no. 2, pp. 414–421, 2009.

[15] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in *Proc. Int. Conf. on Mobile Systems, Applications, and Services*, 2008, pp. 225–238.

[16] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16–29, 2009.

[17] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: a behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[18] L. Liu, G. Yan, X. Zhang, and S. Chen, "Virusmeter: Preventing your cellphone from spies," in *Recent Advances in Intrusion Detection*, 2009, pp. 244–264.

[19] G. Jacoby, N. J. Davis IV, *et al.*, "Battery-based intrusion detection," in *IEEE Global Telecommunications Conf. (GLOBECOM)*, vol. 4, 2004, pp. 2250–2255.

[20] D. C. Nash, T. L. Martin, D. S. Ha, and M. S. Hsiao, "Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices," in *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, 2005, pp. 141–145.

[21] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proc. Int. Conf. on Mobile systems, Applications, and Services*, 2008, pp. 239–252.

[22] B. Dixon, Y. Jiang, A. Jaiantilal, and S. Mishra, "Location based power analysis to detect malicious code in smartphones," in *Proc. Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 27–32.

[23] B. Dixon and S. Mishra, "Power based malicious code detection techniques for smartphones," in *IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2013, pp. 142–149.

[24] M. Curti, A. Merlo, M. Migliardi, and S. Schiappacasse, "Towards energy-aware intrusion detection systems on mobile devices," in *Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2013, pp. 289–296.

[25] T. K. Buennemeyer, G. Jacoby, W. G. Chiang, R. C. Marchany, J. G. Tront, *et al.*, "Battery-sensing intrusion protection system," in *Information Assurance Workshop*, 2006, pp. 176–183.

[26] T. K. Buennemeyer, T. M. Nelson, M. Gora, R. C. Marchany, J. G. Tront, *et al.*, "Battery polling and trace determination for bluetooth attack detection in mobile devices," in *Information Assurance and Security Workshop*, 2007, pp. 135–142.

[27] F.-E. Kioupakis and E. Serrelis, "Preparing for malware that uses covert communication channels: The case of tor-based Android malware," in *The International Conference in Information Security and Digital Forensics*, 2014, pp. 85–96.

[28] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: robust statistical feature signature for Android malware detection," in *Proc. Int. Conf. on Security of Information and Networks*, 2013, pp. 152–159.

[29] R. Andriatsimandefitra and V. V. T. Tong, "Detection and Identification of Android Malware Based on Information Flow Monitoring," in *The 2nd IEEE International Conference on Cyber Security and Cloud Computing*. To appear, 2015.

[30] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones." in *NDSS*, vol. 11, 2011, pp. 17–33.

[31] A. Merlo, M. Migliardi, and L. Caviglione, "A survey on energy-aware security mechanisms," *Pervasive and Mobile Computing*, 2015.

[32] M. Gaggero, G. Gnecco, and M. Sanguineti, "Dynamic programming and value-function approximation in sequential decision problems: error analysis and numerical results," *Journal of Optimization Theory and Applications*, vol. 156, no. 2, pp. 380–416, 2013.

[33] ——, "Approximate dynamic programming for stochastic N-stage optimization with application to optimal consumption under uncertainty," *Computational Optimization and Applications*, vol. 58, no. 1, pp. 31–85, 2014.

[34] A. Alessandri, C. Cervellera, and M. Gaggero, "Nonlinear predictive control of container flows in maritime intermodal terminals," *IEEE Trans. Contr. Syst. Technol.*, vol. 21, no. 4, pp. 1423–1431, 2013.

[35] A. Sen and M. Srivastava, *Regression Analysis - Theory, Methods, and Applications*. Springer-Verlag, 2011.

[36] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[37] T. Cover and T. Joy, *Elements of Information Theory*. Wiley, 1991.