IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 3, 2020, accepted January 16, 2020, date of publication January 27, 2020, date of current version February 6, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2969626*

# HIDROID: Prototyping a Behavioral Host-Based Intrusion Detection and Prevention System for Android

**JOSÉ RIBEIRO[1,2], FIROOZ B. SAGHEZCHI[1], (Senior Member, IEEE), GEORGIOS MANTAS[1,3], JONATHAN RODRIGUEZ[1], (Senior Member, IEEE), AND RAED A. ABD-ALHAMEED[2], (Senior Member, IEEE)**

[1]Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
[2]Department of Engineering and Informatics, University of Bradford, Bradford BD7 1DP, U.K.
[3]Faculty of Engineering and Science, University of Greenwich at London, London SE10 9LS, U.K.

Corresponding author: José Ribeiro (jcarlosvgr@av.it.pt)

This work was supported in part by the European Regional Development Fund (FEDER), through the Competitiveness and Internationalization Operational Programme (COMPETE 2020), in part by the Regional Operational Program of the Algarve (2020), and in part by the Fundação para a Ciência e Tecnologia; i-Five.: Extensão do acesso de espectro dinâmico para rádio 5G, under Grant POCI-01-0145-FEDER-030500. The work of José Ribeiro was supported by the Fundação para a Ciência e Tecnologia (FCT-Portugal) under Grant SFRH/BD/112755/2015.

**ABSTRACT** Previous research efforts on developing an Intrusion Detection and Prevention Systems (IDPS) for Android mobile devices rely mostly on centralized data collection and processing on a cloud server. However, this trend is characterized by two major limitations. First, it requires a continuous connection between monitored devices and the server, which might be infeasible, due to mobile network's outage or partial coverage. Second, it increases the risk of sensitive information leakage and the violation of user's privacy. To help alleviate these problems, in this paper, we develop a novel Host-based IDPS for Android (HIDROID), which runs completely on a mobile device, with a minimal computation burden. It collects data in run-time, by periodically sampling features reflecting the utilization of scarce resources on a mobile device (e.g. CPU, memory, battery, bandwidth, etc.). The detection engine exploits statistical and machine learning algorithms to build a data-driven model for the *benign behavior*. Any observation failing to match this model triggers an alert, and the preventive agent takes proper countermeasure(s) to minimize the risk. HIDROID requires no malicious data for training or tuning, which makes it handy for day-to-day usage. Experimental test results, on a real-life device, show that HIDROID is well able to learn and discriminate normal from malicious behavior, with very promising accuracy of up to 0.9, while maintaining false positive rate by 0.03.

**INDEX TERMS** Android, security and privacy, intrusion detection and prevention system (IDPS), anomaly detection, malware detection, behavior analysis, machine learning, prototype development.

## I. INTRODUCTION

Smartphones play crucial role in modern life. They provide a wide range of appealing features enabling mobile users to access a plethora of high quality personalized services [1], which makes them attractive for cybercriminals. They enable ubiquitous connectivity by holding a portfolio of active radio interfaces (e.g. LTE, UMTS, WiFi, Bluetooth, etc.). With the deployment of Fifth-Generation (5G) mobile networks, this list will be proliferated even further, complementing them with revolutionary radio access technologies to support Giga-bit-per-second speeds on move [2], [3]. Nevertheless, this may also increase their attack surface since these interfaces could be leveraged as potential attack vectors.

Android is the most popular mobile operating system, capturing around 75% of global market share, which renders it a prime target for attackers [4]. In particular, its open operating system characteristic allows the user to install applications from not only trusted, but also untrusted sources (i.e. third-party markets). Consequently, malwares looking like an innocent software (e.g. games, utilities, etc.) might be downloaded and installed, which can pose serious security threats.

Apart from traditional Short Message Service (SMS)- or Multimedia Messaging Service (MMS)-based Denial of

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

Service (DoS) attacks, future smartphones can be exposed to more sophisticated attacks originated from mobile malwares (e.g. viruses, worms, Trojans, etc.). These attacks can target not only the device itself, but also the mobile infrastructure. Mobile malwares also enable attackers to exploit the stored personal data on the device or to launch attacks (e.g. DoS) against other entities, such as other user devices (e.g. smartphones, tablets, etc.), radio access networks, backhaul, or core network, or other external networks connected to the core [1], [5]. Therefore, security mechanisms such as Intrusion Detection and Prevention Systems (IDPSs) are of utmost importance to protect smartphones from a plethora of known and unknown threats and ensure the user's privacy.

Previous research efforts on designing an IDPS for Android mostly rely on rooting the device [6] or collecting data from remote devices and processing them in a command and control center within the cloud [7]–[9] However, these approaches have some severe limitations: i) they require a continuous link between a mobile device and a central IDPS server, that might not be always feasible due to the network's outage or partial coverage; and ii) they increase the risk of sensitive information leakage, which may lead to the violation of user's privacy. To mitigate these problems, in this paper, we extend our previous work in [10] and develop a prototype for a fully autonomous Host-based IDPS for Android (HIDROID). In particular, this paper provides following contributions:

1) we develop HIDROID as an IDPS application for Android, describing its full functional and implementation details;
2) we complement the detection engine of the proposed Intrusion Detection System (IDS), in [10], with an Intrusion Prevention System (IPS), paving the way towards a complete IDPS solution;
3) we evaluate the performance of HIDROID while running on a real-life mobile device and present our experimental results.

In order to keep the computation burden of HIDROID on a mobile device at minimum, without loss of generality, we implement a couple of lightweight Machine Learning (ML) and statistical algorithms, namely *one-class K-means* [11] – a variant of K-means clustering algorithm with only one cluster encompassing the benign data – and the *univariate Gaussian* algorithm for anomaly detection [12], [10]. The results published in our previous work, in [10], were based on simulations only, while this work presents experimental results from our implemented prototype. Furthermore, it is the first time that we employ *one-class K-means* algorithm for intrusion detection – in [10], we had studied the standard K-means algorithm, with two clusters, which requires both benign and malicious data for training. In contrast, the one-class variant proposed in this work relies only on benign data for training.

In fact, the merit of K-means algorithm, for an on-device IDPS, is its low computational cost. This allows the algorithm to be run on a resource-limited mobile device. However, the algorithm also demonstrates a very good detection performance according to our previous study in [10]. Nevertheless, HIDROID is modular in design and permits to plug and play other learning algorithms as well. Its extracted features reflect the total *resource utilization* on the host device. It runs wholly on the mobile device, a *host-based* approach, and requires no root permission. Upon detecting any security incident, the *Prevention Engine* launches adequate counteraction(s) to minimize the risk. These countermeasures can range from disconnecting the device from the Internet, switching off its WiFi and Bluetooth interfaces as well as the mobile data connection, to displaying a plain text alert on the screen, informing the user about the evidence and the likelihood of the detected intrusive event. This alert message pops up with key recommendations to the user to prevent cascading the risk. The Graphical User Interface (GUI) displays in run-time monitored features along with the latest intrusion detection results.

The rest of this paper is structured as follows. Section II reviews the related work. Section III presents HIDROID's architecture and its different components. Section IV describes its functional operation through flowcharts. Section V covers the implementation details including both hardware and software. Section VI describes experimental setup and discusses our real-life test results, evaluating the performance of HIDROID. Finally, Section VII concludes and draws guidelines for future work.

## II. RELATED WORK

Malware detection methods are divided into three main categories: 1) *static*, 2) *dynamic*, and 3) *hybrid* [13]. The static techniques (also known as *misuse-* or *signature-based*) maintain an updated database of malicious code patterns (i.e. attack signatures) and scan the code, without running it, for those signatures. Although being fast, they are susceptible to generate false negatives because any small variation in the code can easily evade the detection. In contrast, *dynamic* techniques (also known as *behavioral* or *anomaly-based*) monitor the behavior of an application in runtime. They construct a model for normal behavior. Any observation considerably deviating from this model is considered as an anomalous behavior. The main advantage of dynamic techniques is their capability to detect zero-day attacks though they may generate a large number of false positives, due to facing benign examples unseen in the training data. Finally, *hybrid* techniques combine both static and dynamic methods in order to exploit the high detection precision of the static approach and the zero-day attack detection by dynamic analysis.

Behavior-based IDPSs essentially construct a data-driven model for the benign behavior. The data can generally contain three major types of *features* [4], [14]: 1) *resource utilization* on a mobile device, i.e. consumed CPU/memory, drained battery, incoming/outgoing network traffic, etc.; 2) *system calls* of the mobile operating system's kernel invoked by different applications, e.g. the number of times functions such as *open*, *read*, *write*, *kill*, etc. are called by an

application; 3) *access permissions* requested by an application, e.g. to read/send SMS, accessing the camera, microphone, contact list, device's location, etc., that are registered in the manifest file [4], [15].

Based on the location where the detection algorithm is deployed, IDPSs are further divided into three major categories [16]:

1) *Host-based*: the whole system, including the detection engine, is deployed on the mobile device itself, an autonomous IDPS [10], [17], [18].
2) *Centralized*: a command and control center in the cloud monitors the mobile devices. In-depth analyses are performed on powerful servers, taking advantage of their abundant computation power and memory capacity [6], [19]–[23].
3) *Distributed*: the system is partly deployed on the mobile device and partly within the cloud. The data collection agent and some lightweight analyses are performed on the device, whereas computationally expensive analyses are carried out on a remote server computer [7], [24], [25].

There are several existing IDS solutions for Android [4], notably *Andromaly* [17], *Aurasium* [26], *Crowdroid* [6], *Drozer* [27] and *Kirin* [28]. In the following, we briefly review them.

*Andromaly* [17] is a behavioral host-based malware detection framework for Android that continuously monitors *resource consumption* on a mobile device. It employs supervised ML algorithms to classify a labelled dataset as benign or anomalous. The audit data is collected from two Android devices, each running four benign and four malicious applications. The detection process consists of run-time monitoring, data collection, pre-processing and analysis of features such as CPU consumption, number of sent packets through the WiFi interface, battery level, keyboard and touchscreen pressing rate, etc. An ensemble algorithm (e.g. majority voting, weighted average, etc.) is applied to combine the results of multiple learning algorithms in order to reliably assess the infection level of the device. A smoothing filter is then employed, combining the current detection outcome with the past history of alerts, in order to reduce the number of instantaneous false positives. Upon a detection, a notification message is displayed for the user and some automatic or manual countermeasures are considered to mitigate the risk. However, there are two main limitations with *Andromaly*. First, it relies on supervised learning which entails constructing a labelled dataset, which is a tedious task in practice since it requires infecting the device with a set of malwares and labelling the data by a human. Second, *Andromaly*'s capability for detecting *real malwares* still needs further investigation as the authors test it only against their own homemade simple malwares.

*Aurasium* [26] is a technology that enforces arbitrary run-time security policies in a simple and robust way to control the execution of apps, even the ones obtained from an untrusted place. The system is composed of two major components: 1) the repackaging mechanism inserting instrumentation code into arbitrary Android apps; and 2) the monitoring code intercepting an app's interactions with the system that enforces various security policies. Instead of installing the app directly on the mobile device, the app is pushed by the user to the *Aurasium* black box where a hardened version of the app is created. Then, the user installs this hardened version. It is assured that all of the app's interactions are closely monitored for malicious activities and adequate protection policies for the user's privacy and security are actively enforced. Nevertheless, the *Aurasium* process can be dangerous since it uses repackaging that is done by a third-party application. Furthermore, *Aurasium* modifies the original application – despite for a good cause. It also includes more policies and add its own code. This procedure forces to add a new signature and thus *Aurasium* itself can be treated as a malware by other IDSs.

*Crowdroid* [6] is a behavior-based (*dynamic*) malware detection system constituted by two major parts: 1) a crowdsourcing app (*Crowdroid*) running on the mobile device that is responsible for collecting the behavioral data (*system calls*) of Android applications; and 2) a remote server for the trace analysis and malware detection. It analyzes the behavior of Android applications and detects malwares in the form of Trojan horses. *Crowdroid* makes use of a crowdsourcing system to obtain the traces of application's behavior. A major constraint of this solution is that it uses the *Strace*, a system utility on the device to collect the *system calls* of the app, which requires the root access. Also, the collection of system calls is for a specific application and not for the entire system. So, it analyzes the behavior of one application at a time.

*Drozer* [27] is a comprehensive attack and security assessment framework for Android. It is available as an open-source software, under Berkeley Source Distribution (BSD) license, which is maintained by MWR *InfoSecurity*.[1] It is essentially a *hybrid* solution consisting of two main components: 1) the Agent app that is installed on the device and interacts with other apps through the Inter-Process Communication (IPC) mechanism; and 2) the server-side component, which is installed on a server and can remotely monitor the Android device. In addition, *Drozer* can interact with the *Dalvik* virtual machine to discover installed packages and related app components. Furthermore, it allows interaction with the app-components like services, content providers and broadcast receivers to identify vulnerabilities. It can also interact remotely with Android operating system through a shell. Although *Drozer* can detect some types of intrusions [27] and it is modular in design, it interacts with the user only through the command line, which can be a handicap for ordinary users that are less familiar with command line interfaces.

*Kirin* [28] is an *on-device* security policy enforcement mechanism. It verifies the *permissions* requested by an app against a set of rules. These rules are previously defined

[1] https://www.mwrinfosecurity.com/

**TABLE 1. Pros and cons of different malware detection systems for android.**

| Work | Advantage | Disadvantage |
|---|---|---|
| *Andromaly* [17] | Behavior-based IDS that analyzes *resource utilization* of the mobile device | Requires labeled data and only has been tested against malicious data collected from a few artificial malwares |
| *Aurasium* [22] | Enforces arbitrary runtime security policies | Uses repackaging – modifies the original application, can be treated as a malware by other IDSs |
| *Crowdroid* [6] | Uses crowdsourcing to acquire data from different sources and | Needs root access and analyzes one application at a time |
| *Drozer* [23] | IPC to monitor installed apps Easy to implement new models, | Uses a command line interface that is not user friendly |
| *Kirin* [24] | Verifies the apps' *permissions* against a set of predefined rules and provides a methodology for upgrading security requirements | Analyzes the application in install time and is not designed for monitoring the application behavior in runtime |

based on the combination of certain dangerous permissions that might be requested by an app. *Kirin* decides whether or not to permit the installation of new app, based on these rules, on behalf of the user. In addition, it provides a methodology for upgrading security requirements in Android and performs certification of the applications at install time. Although *Kirin* can detect some dangerous functionalities, it is far from being complete. Its set of rules can detect only specific types of malware and may reject legitimate applications. For instance, it cannot detect a malware that requests permissions that look like the ones requested by a legitimate application. *Kirin* analyzes the application on install time but is not designed for monitoring the application behavior on runtime. Table 1 summarizes the advantages and disadvantages of the reviewed works.

However, despite these existing solutions, there are still several limitations with IDPSs for Android. Towards that end, HIDROID brings several novelties to the state-of-the-art. For example, *Andromaly* [17] analyzes the data collected from the behavior of few artificially generated malwares by the authors and relies on a supervised dataset for training, which entails considerable human effort in practice. In contract, HIDROID analyzes data collected from real-world malwares and is trained using an unlabeled dataset. As another example, *Crowdroid* [6] is a cloud-based IDS that extracts features related to the frequency of *system calls* and performs computationally expensive data analysis tasks on a server. Nevertheless, HIDROID is a host-based IDPS that runs completely on the mobile device, therefore needing no connection to the server. Furthermore, HIDROID extracts and analyzes features representing the overall behavior of the device (by capturing the user's behavior), whereas *Crowdroid* relies on characterizing the behavior of each individual application, which limits its scalability to a larger number of malwares. Additionally, *Kirin* [28] is a static approach as it checks the permissions that an application requests at install-time (and not in run-time), and *Aurasium* [26] modifies the source code of the application under vigilance, using a tool that unpacks the application, embeds the IDS's own code, repacks the application, and resigns it, which could itself be considered as an intrusive behavior by third-party IDS applications. In contrast, HIDROID exploits a *dynamic* learning approach for detection and does not modify the applications' code at all. Last but not least, unlike the reviewed works, which perform only the detection task, HIDROID complements the detection with a prevention engine.

## III. HIDROID COMPONENTS

HIDROID app consists of the following main components, as illustrated in Fig. 1: *Run Time Data Acquisition*, *Run Time Dataset Generation*, *Feature Normalization*, *Detection Engine*, *Intrusion Probability Assessment*, *Alert Manager*, and *Prevention Engine*.

### A. RUN TIME DATA ACQUISITION

The *Run Time Dara Acquisition* component is the initiating building block of the system. It is responsible for collecting data on run-time for the set of features summarized by Table 2.

### B. RUN TIME DATASET GENERATION

The *Run-Time Dataset Generation* module succeeds the data acquisition and constructs training/test datasets in run-time. It saves the collected data in CSV (Comma-Separated Values) files where each file contains the data collected during one data acquisition interval which can be adjusted by the user from 1 min up to 1 h. Each entry (row), in the file, represents one training sample (example) and each column represents one feature.

### C. FEATURE NORMALIZATION

The *Feature Normalization* module receives the dataset from the *Run-Time Dataset Generation* component, in a CSV file, and normalizes the features (i.e. the columns of this file). The normalization is performed as follows. For each column (feature), it first subtracts the mean value of the column from every entry and then divides the results by the standard

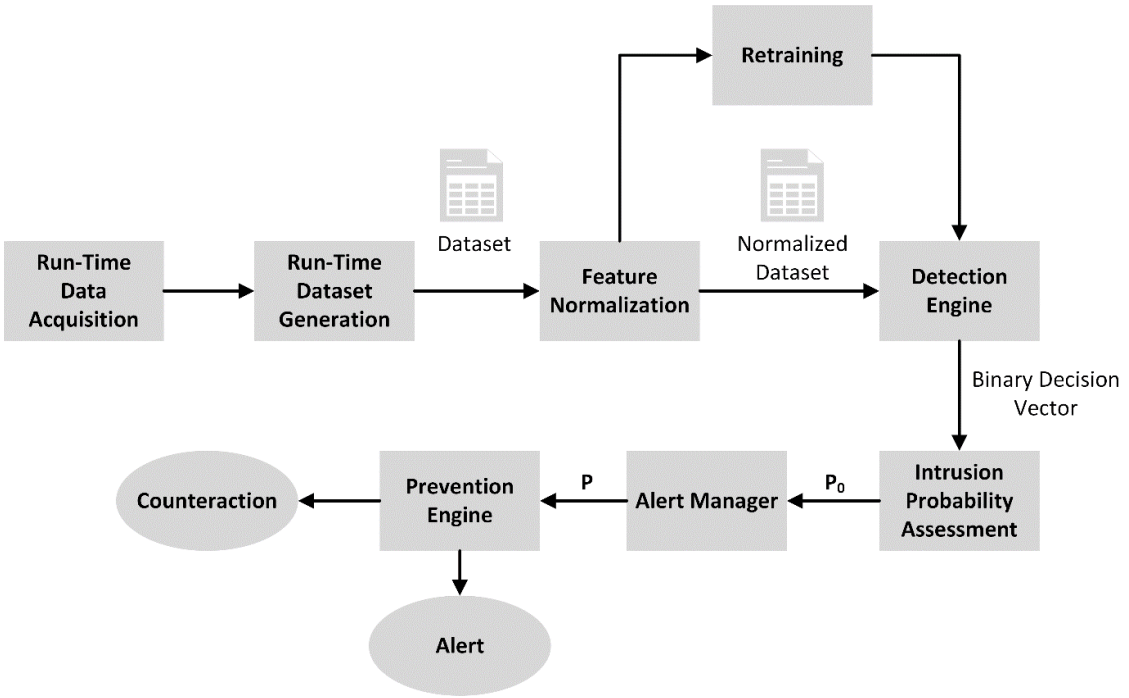



**FIGURE 1.** Architecture of HIDROID.

**TABLE 2.** Extracted features for Android malware detection.

| Index | Feature | Description |
|---|---|---|
| 1 | CPU usage | Overall CPU consumption (%) |
| 2 | Memory usage | Overall memory usage (kB) |
| 3 | Cached memory | Memory used as cache (kB) |
| 4 | Bit rate | Total sent/received bytes per second (kB/s) |
| 5 | Packet rate | Total sent/received packets per second (packet/s) |
| 6 | Battery drain | Drop in battery level (%) |
| 7 | Battery temperature | Battery temperature (ºC) |
| 8 | Running processes | Total number of running processes |
| 9 | Running services | Total number of running services |
| 10 | TCP open sockets | Total number of open TCP sockets (IPv4/v6) |
| 11 | SMS to unknown numbers | Total number of sent SMS to unknown numbers |
| 12 | Total sent SMS | Total SMS in outbox |
| 13 | Total outgoing calls | Total number of placed phone calls |
| 14 | Installed applications | Total number of installed applications |
| 15 | Screen status | Display on/off: on = 1 ; off = 0 |

deviation of the same column. This operation is repeated for every column and the output is saved in a new CSV file; each column of the new file has therefore zero mean and unit standard deviation.

### D. DETECTION ENGINE

The *Detection Engine* incorporates both ML and statistical algorithms. The ML algorithm is of unsupervised category, namely K-means clustering algorithm. More precisely, we use a variant of K-means with only one cluster, representing the benign data [29], [30]. During the training phase, we set a boundary separating benign region from the outlier region. To find this boundary, in the training phase, we assume that a certain majority of data (say 98%) is originated from benign behavior and the rest (2%) is due to outliers or noise. Hence, we set a spherical threshold (using Euclidian distance) in a way that encompasses 98% of the training data, while the rest 2% (outliers) lie outside the boundary.

As for the statistical algorithm, we use the univariate Gaussian algorithm, elaborated in [10]. The rationale behind choosing this algorithm is its low computational cost, but still a comparable performance to its multivariate counterpart since the latter requires computing the inverse of the covariance matrix, which is a computationally expensive operation, especially for a resource-constrained mobile device.

It is worth noting that the essential difference between univariate and multivariate Gaussian models is that the former assumes that all features are independent from one another, while the latter does not make such an assumption. Both algorithms principally estimate the parameters of the underlying (Gaussian) distribution, e.g. the mean and the variance or the covariance matrix (in case of a multivariate model), using a maximum likelihood estimator. Then, the algorithm sets a threshold for the probability distribution function, below which, is considered as anomaly. Upon any observation, its

probability is calculated, and if it falls below this threshold (i.e. in the tale of the distribution), the detection engine reports an anomaly.

### E. INTRUSION PROBABILITY ASSESSMENT

The *Intrusion Probability Assessment* analyzes the output (binary) decision vector of the *Detection Engine*. It calculates the probability of intrusion for a given data acquisition period based on the ratio of malicious examples observed in the test dataset generated in that period. This calculation is simply done by dividing the number of 1's in the input binary vector to its length.

### F. ALERT MANAGER

The *Alert Manager* calculates the *overall* probability of intrusion given the probability of intrusion for the current and past monitoring periods. Particularly, it receives the current probability of intrusion coming from the *Intrusion Probability Assessment* component and calculates the overall probability. This calculation takes into account the number of consecutive alerts and their associated *probability of intrusion* occurred in the recent past. The *overall probability of intrusion* at time instant k, P(k), is calculated as follows:

$$P(k) = 1 - \prod_{j=0}^{\alpha-1} (1 - P_0(k-j)) \quad (1)$$

Here $P_0(k)$ indicates the instantaneous probability of intrusion at time index k and $\alpha$ represents the number of consecutive intrusions that have been observed in row. When the overall probability of intrusion (P(k)) is calculated and it exceeds a predefined threshold (*security level*), the *Prevention Engine* is triggered.

### G. PREVENTION ENGINE

The *Prevention Engine* alerts the user and provides him/her with the probability that the system is compromised and the following recommended counteractions:

1) *Open the "Battery Management" in System Settings to verify what applications are consuming too much battery,*
2) *Open the "Application Management" in System Settings to verify the last application installed and if it was really installed by user,*
3) *Reboot the system.*

Moreover, the *Prevention Engine* is able to carry out the following countermeasures:

1) *disable the Wi-Fi interface,*
2) *disable 3G data connection,*
3) *disable the Bluetooth connection,*
4) *rename malware files (e.g. Trojans) that automatically send system information to a remote server controlled by an attacker.*

## IV. FUNCTIONAL DESCRIPTION

As discussed above, for the detection engine, we implemented the univariate Gaussian and the one-cluster K-Means algorithms. The aim is to study the performance of the overall IDPS and each of these detection algorithms in run-time operation. We install the IDPS app on the smartphone under test and keep it running while we use the device normally. The app begins by collecting the training examples, sampling 15 features every 2s. The sampled features are summarized by Table 2. The collected examples are appended to a CSV file until it reaches 12000 rows. The number of training samples (12000) was set based on the results of our previous study, where we achieved satisfactory results [10], [31].

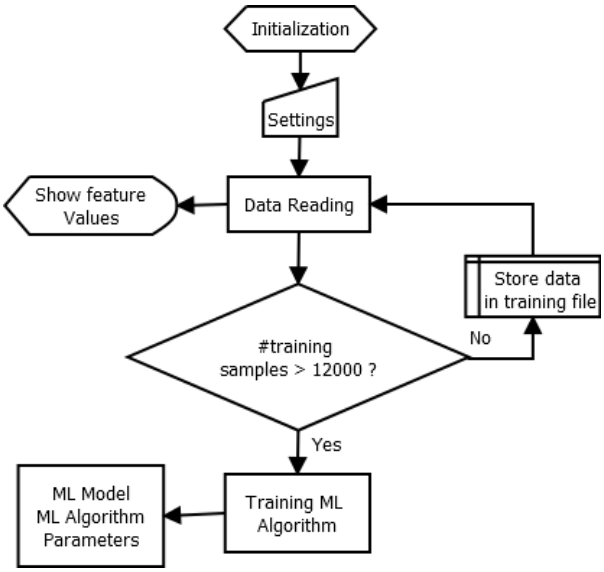



**FIGURE 2.** Initialization process of HIDROID.

Fig. 2 illustrates the initialization process of HIDROID. It starts with the training phase by reading the samples and running the procedures for each algorithm [10]. To determine the threshold for outliers, we vary the boundary between 80% and 100%, e.g. when the boundary is 95%, 95% of the whole training examples lie inside the boundary and the rest 5% lie outside. Then, we count the number of True Positives (TPs) and False Positives (FPs) for each malware in the dataset. We then choose the optimum value for the threshold by finding the point where FP is very low or zero, while the TP is as high as possible. This is indeed the point where the accuracy reaches its maximum (cf. Fig. 8).

It is worth noting that for the case of K-means algorithm, we perform both training (calculation of the centroid) and tuning (finding the optimal radius for the boundary) in a single step and so we use the whole 12000 training examples at once. However, for the case of Gaussian algorithm, we split the dataset and use 10,000 of them only for training (calculation of mean and variance vectors) and the rest 2000 for calibration (finding the optimum *threshold* on the probability distribution function that separates the tail region) [10]. Any data point with probability higher than this threshold is considered benign; i.e. the lower values for the probability imply a malicious event. For the case of K-means algorithm, for an observed sample, if the Euclidian distance to the centroid is

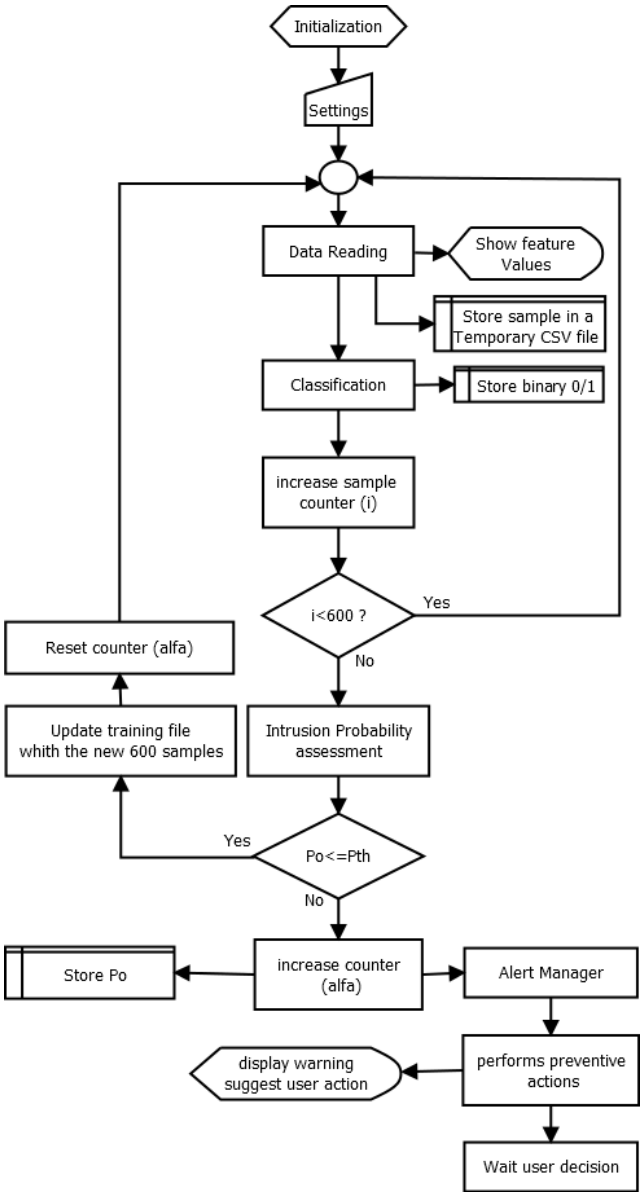



**FIGURE 3.** Run-time operation of HIDROID.

simply lower than the boundary, the sample is classified as benign, otherwise it is classified as malicious.

When the training is completed, all parameters are stored in the *SharedPreferences* and this ends the initialization process. *SharedPreferences* is indeed an Application Programming Interface (API) from Android Software Development Kit (SDK) to store and retrieve application preferences which are simply a set of data values that are stored persistently. Hence, even if the application stops running, or the device is switched off, the stored data in the *SharedPreferences* is still there [32], [33].

The IDPS is now ready to operate. Fig. 3 depicts its run-time operation. The shared preferences are useful in case the application is interrupted; when it restarts, it restores the values that were in use before the interruption. As we can see from the figure, HIDROID continually reads, classifies, and stores the classification result by appending it to a binary vector (0 stands for benign and 1 stands for malicious). It writes the sampled feature vector to a temporary CSV file, by appending it to the end of the file as a new entry until a predefined number of entries are collected. We consider this number to be 600, corresponding to 20 minutes of reading in intervals of 2s. It is worth noting that the classification is done at run time to reduce the memory usage; we only need to store a binary vector, with length 600, showing the results of classification for all entries.

When the IDPS collects 600 examples, it stops reading and classifying and then proceeds to the intrusion probability assessment. This probability is indeed calculated as the ratio of ones (malicious observations) in the output binary vector of the detection engine. If the calculated probability of intrusion at time instant k, $P_0$ (k), is less than or equal to the user's defined *security level* (see Fig. 6), it assumes that there is no intrusion and injects the collected 600 examples to the training CSV file, by replacing the oldest 600 entries in that file with the new ones. Otherwise, if the probability of intrusion for the actual processing time, $P_0(k)$, exceeds the *security level*, the IDPS considers this as a sign of intrusion and stores $P_0(k)$ in a vector. Note that the *security level* is indeed a threshold value for the probability of intrusion ($P_{th}$). It is set by the user and when the probability of intrusion, evaluated by the *Probability Assessment* module, exceeds that threshold, the IDPS reports an intrusion.

When an intrusion is detected, the *Prevention Engine* takes counteractions to minimize the risk and triggers an alert. Based on the severity of the condition, figured out from the overall probability of intrusion $P(k)$, it proposes an appropriate countermeasure that user should take to minimize the risk. User can choose to cancel this message – assuming the risk – or to apply the recommended prevention action. The default actions include: 1) disconnecting the device from the Internet by switching off 3G/4G, WiFi, and Bluetooth interfaces; 2) locating potentially malicious files or the ones with signs of malicious activity by looking for keywords such as "*name*", "*phone*", "*email*", or "*imei*" in the text. If any malicious file is found, the *Prevention Engine* renames the file and reports it to the user. The preventive actions can be escaladed according to the level of the overall probability of intrusion, i.e. the number of consecutive alerts in row. That is, the *Alert Manager* takes a decision based on the overall probability of intrusion, formulates an output and displays a warning message on the screen suggesting a counteraction.

There are some actions that the application can take (e.g. disconnecting the device from the Internet) without user's consent and without *system rights* in an unrooted device. However, in situations where it is not permitted to carry out a certain action, it can at least provide useful suggestions to instruct the user how to react. For example, it can: 1) launch the "Battery usage" in the Settings to let the user know if there is any application consuming too much battery; 2) launch the Application Manager in Settings and

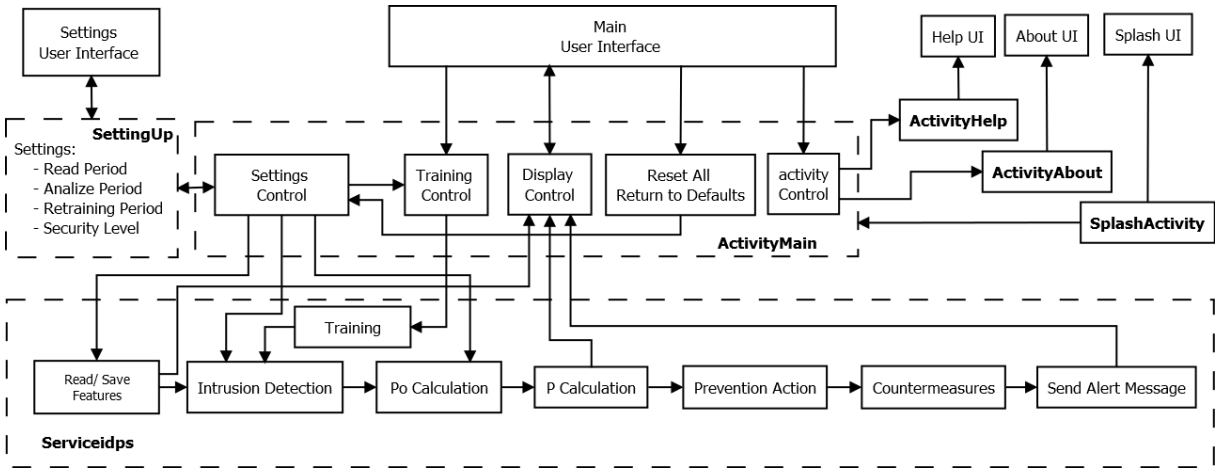



**FIGURE 4.** HIDROID structural diagram.

suggest the user to look out for the last installed applications. Nevertheless, in the end, it is the user's decision that overrides.

## V. HIDROID PROTOTYPE IMPLEMENTATION

### A. IMPLEMENTATION OVERVIEW

We developed HIDROID as a regular Android application using the latest version of the Android Studio platform (version 3.2.1) on a laptop running Windows$^{TM}$10 with latest updates. The Android Studio platform is dedicated for Android application development and uses all necessary tools for editing, compiling, testing, and installing Android applications. It also includes emulators that allow developers to test the applications on different hardware. We installed and tested HIDROID on an un-rooted Samsung Galaxy J100H smartphone (CPU ARM Cortex A7, 1200 MHz, 2 core, RAM 512 MB and 4GB storage) running Android KitKat (version 4.4.4). The Samsung Galaxy smartphone supported extended memory card and micro-sim card and was also equipped with camera, Bluetooth, WiFi, USB and positioning system (GPS, A-GPS, GLONASS).

### B. STRUCTURAL DIAGRAM

Fig. 4 shows the structural diagram of HIDROID where all its functional blocks along with their inter-connections are illustrated. The user can see the run-time measurements of the sampled features and adjust data collection parameters such as sampling period, data collection duration, retraining period, and the security level. Furthermore, the user can also force retraining the learning algorithms at any time, which is an important feature. For instance, after installing any new application on the mobile device, the IDPS needs to be retrained since this new installed application may considerably impact the device's behavior, resulting in a potentially high number of false positives. Finally, once an intrusive behavior is detected, the user is informed by an alert message displayed on the user interface.

### C. BACKGROUND RUNNING

Android apps are built as a combination of components that can be invoked individually. The *Activity* is the component that provides a user interface and invoke methods for various stages of its life cycle such as: *onCreate()*, used to create variables; user interface configurations; or *onDestroy()*, used when the application is terminated to delete variables and release the memory [34].

Other components such as *broadcast receivers* and *Services* allow the app to perform background tasks without interacting with the user. Our implementation uses the *service* to run all the main functions in the background, while the user is running other applications in the foreground. However, as Android applications run, the device's available memory may decrease. When the memory gets critically low, Android terminates processes so as to release the occupied memory [32], [33]. Our application uses the label "START_STICKY" at the "*onStartCommand()*" callback method that tells the system to create a fresh copy of the service after it recovers from low memory [32], [33]. Also, we implement a mechanism that receives a boot indication (*intent*) after system boots and starts the *Service* automatically. Intent is indeed an Android data structure to bind information between code objects such as Activities or Services or between different Android applications [33].

### D. USER INTERFACE

The user interface is organized in several functional pages allowing the user to configure and setup parameters and view the values of the principal features collected in run-time by tapping in the respective icon, as illustrated by Fig. 5. As shown in the figure, it displays the elapsed time from the beginning of the current data acquisition interval and the number of collected training samples for the current dataset (i.e. the number of rows in the dataset). The user can also access to the latest updates regarding the instantaneous and

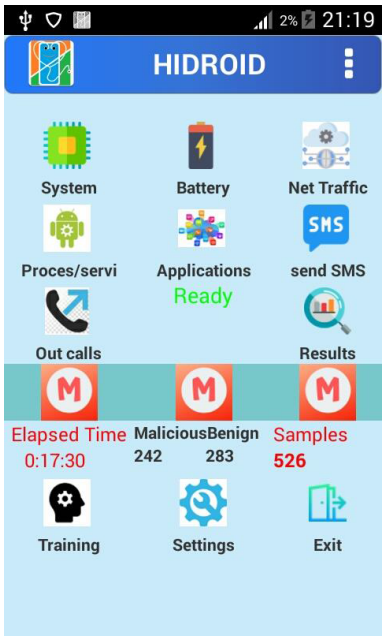



**FIGURE 5.** HIDROID's user interface, main page.


**FIGURE 6.** HIDROID's user interface, *Settings* page.

overall probabilities of intrusion, the number of generated alerts and the threshold value.

Furthermore, the acquisition interval, analysis period, training period and the *security level* parameters can be set up on the *Settings* page, as shown in Fig. 6. The value of the *acquisition interval* can vary from 1 to 5 seconds, and the *analysis period* varies between 5 and 50 minutes. The training is performed before any new analysis starts but the user can force it manually (Fig. 5) or scheduling it through the *Settings*. In addition, it is noteworthy that every time that HIDROID is re-started, it refreshes the detection algorithms by executing the retraining phase.

Moreover, on the *Settings* page, the user can set the *security level* to avoid false positives triggered by circumstantial occasions or noise. Although this parameter is essentially a probability and its value can be between 0 to 1, as discussed in Section VII, the recommended values lie between 0.05 (5%) and 0.15 (15%) as shown in Fig. 6.

Finally, other useful functionalities that user can find in the menu bar (see Fig. 5) include a *Clear All* button that resets all variables to default and cleans all the previous results.

### E. CLASS DIAGRAM

Fig. 7 shows the class diagram of HIDROID application, with the most representative classes. In the following, we briefly describe each of these classes [32], [33].

- The *ActivityMain* class implements the main interface. This class controls the entire application. It verifies the administration rights, it starts the *ServiceIdps* and sets the default variables values to start running. It gets the feature reading values such CPU, memory, battery, etc. and sends them to the user interface.
- *SplashActivity* class controls the *welcome* page that opens when the application is started and stays alive for a few seconds then closes before the home page is loaded.
- *CSVWriter* implements all the necessary functions to write in CSV files.
- *AlertDialod* implements the dialog interface. It is necessary to build the dialogs used in the *Alert Manager*.
- The *ServiceIdps* class implements a service. It implements all the main functionalities of the full design such as data acquisition, analysis and saving, *training*, *classification*, *Alert Manager*, etc. Moreover, it binds data with *ActivityMain*. It further implements functions to detect suspicious malicious text files. Finally, it is able to manage network connections, e.g. WiFi and Bluetooth.
- The *SettingUp* class is the activity that controls the graphical interface for user set up the parameters needed for reading, training, and other adjustments. At closure, this class returns to *ActivityMain*.
- The class *AdminReceiver* is used to activate the administration rights and gain access to several protected features. It implements methods to enable or disable administration rights.
- The class *SmsOutgoingObserver* is a service used to count the number of outgoing SMSs to unknown numbers. It needs permission to access the contact list.
- The *BootReceiver* class is a *BroadcastReceiver* used to capture the intent broadcasted by the system when finishes the boot process to start the application.
- The *Alarm* class is a *BroadcastReceiver* that captures the intent from the system alarm at a pre-set time to start the periodic retraining.
- The *SharedPreferences* class is responsible to store and retrieve application preferences which are simply sets of data values stored persistently. This means that the data

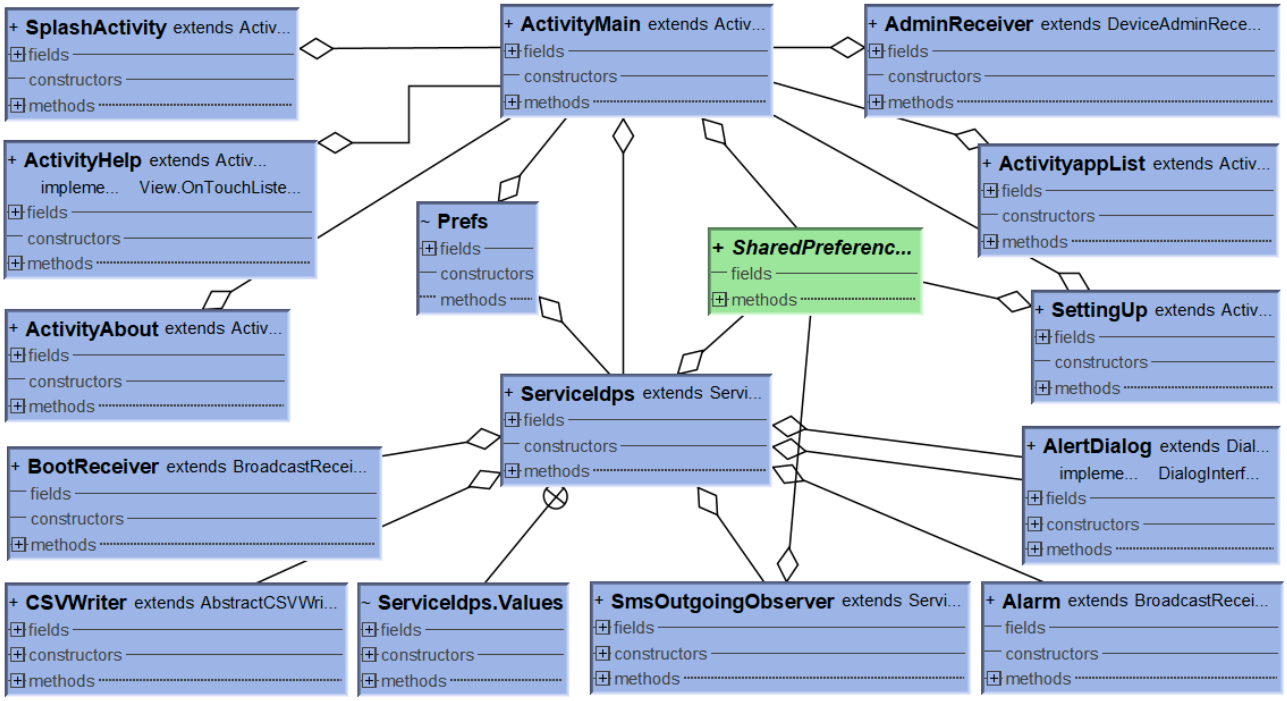



**FIGURE 7.** HIDROID Class diagram.

stored in the *SharedPreferences* still exists even if the application is closed or the device is switched off.

- The *Prefs* class is used to hold the variables' default values that will permit the application to start.
- The *ActivityHelp* controls the view interface that displays instructions to set the parameters along with a short description of how the application works.
- The *ActivityAbout* controls the view displaying application information.
- The class *ServiceIdps.Values* is used to return multiple variable values. When using *ServiceIdps.Values* class, the function returns a pointer to this class where it holds the variables' values.

## VI. PERFORMANCE EVALUATION

### A. EXPERIMENTAL SETUP

To evaluate the detection performance of HIDROID, in this section, we conduct several real-life experiments. We installed HIDROID on a Samsung mobile phone and used the device normally, i.e. sending SMS, placing and receiving phone calls, accessing the Internet, etc.

We firstly made sure that the device was clean from any malware and let the application collect 15 test datasets, each containing 600 examples, representing the benign behavior. We set the *data acquisition interval* to 20 min and the sampling period to 2 seconds; therefore, each dataset contains 600 entries. After collecting benign datasets, we installed one malware at a time from the list provided by Table 3 and continued using the device in the same way as we did before. The application collected 15 test datasets from each installed malware, again each with 600 entries. We repeated this data collection with every malware listed in the table and ended up with 60 malicious datasets ($4 \times 15$). The samples collected during one *data acquisition interval* were saved in a CSV file, each row representing one example and each column representing one feature.

The detection results of the two implemented algorithms (i.e. one-class K-Means and univariate Gaussian) were also saved in a CSV file, summarizing the total number of correct and erroneous decisions for each dataset. Note that for a benign dataset, the total number of correct and wrong decisions are reflected by True Negatives (TNs) and FPs, respectively. In contrast, for a malicious dataset, the total number of correct and wrong decisions are reflected by TPs and False Negatives (FNs), respectively.

We use Accuracy, True Positive Rate (TPR), and False Positive Rate (FPR) as performance metrics. These metrics are defined as follows [17].

*Accuracy* is the ratio of correct decisions out of the total number of decisions that the IDS takes:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

TPR, also known as *Detection Rate*, *Sensitivity*, or *Recall* is the ratio of positive examples that are correctly detected (recalled) by the IDS:

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

**TABLE 3.** Android malwares used in our trials.

| Malware | Type of misbehavior |
|---|---|
| Marcher | Uses the "Adobe Flash player" icon, causes high CPU consumption, gets administration rights, activates WiFi, installs a fake Google store, and sends SMS. |
| Secrettalk_Device | Gets administration rights and causes high CPU consumption. |
| AndroidXbot | Appears as the "Google Installer", gets administration rights, installs another app as "WhatsApp", and incurs high CPU consumption. |
| Radardroid2Map | Mines bitcoins and causes high CPU consumption. |

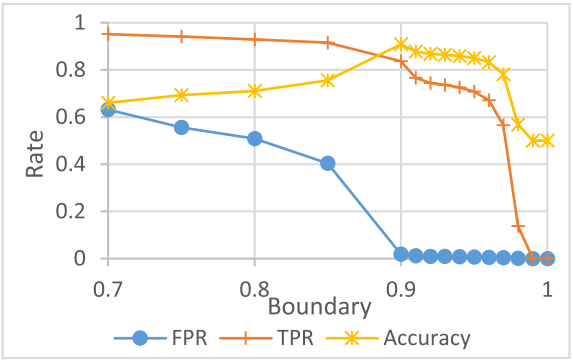



**FIGURE 8.** Accuracy, TPR and FPR of K-means algorithm when *boundary* varies from 0.7 to 1.

FPR is defined as the ration of negative examples that are wrongly classified as positive:

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

### B. DETECTION PERFORMANCE

Fig. 8 and Fig. 9 illustrate *Accuracy*, TPR and FPR for K-means and Gaussian algorithms, respectively, when we expand the parameter *boundary* from 0.7 to 1. The *boundary* is defined as the ratio of the training data that lies below the *threshold* separating benign from malicious data. For example, when *boundary* is 0.9, *threshold* is set to a value that encompasses 90 per cent of the training data as the *normality* region – the rest 10 per cent of the training data is assumed to be outliers, e.g. originated from noise or anomalies. Note that, in the case of K-means algorithm, *threshold* is a distance from the cluster's centroid, whereas in Gaussian algorithm, it is the probability (a fraction between 0 and 1) of the tail of the distribution where malicious examples occur. It can be noted from Fig. 8 that the maximum *Accuracy* of

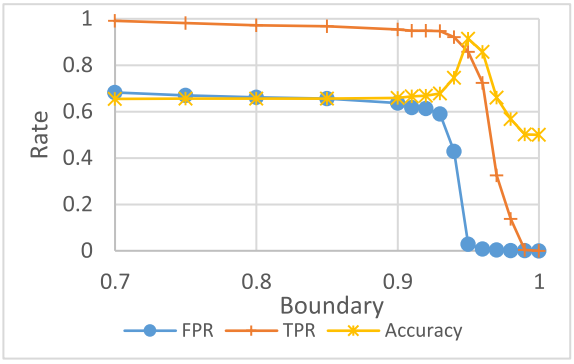



**FIGURE 9.** Accuracy, TPR and FPR of Gaussian algorithm when *boundary* varies from 0.7 to 1.

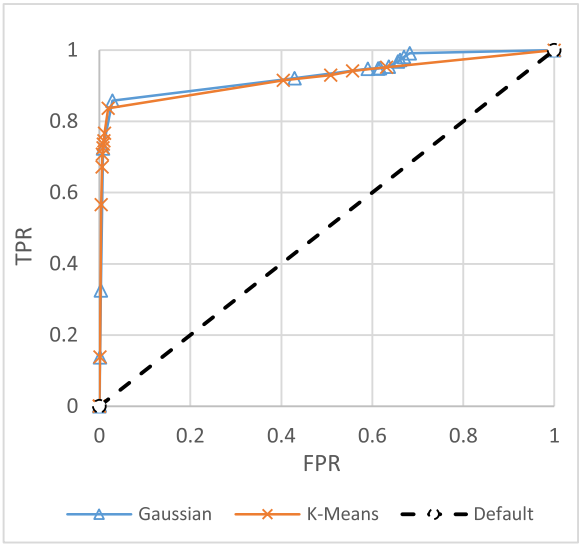



**FIGURE 10.** ROCs of K-means and Gaussian classifiers.

K-means is 0.909, which is attained when the *boundary* is 0.9. In contrast, the maximum *Accuracy* of the Gaussian algorithm, as Fig. 9 shows, is 0.914, which is attained when the *boundary* is 0.95. Overall, the two algorithms demonstrate quite similar *Accuracy*, which is around 0.91.

### C. RECEIVER OPERATING CHARACTERISTIC

Fig. 10 illustrates Receiver Operating Characteristic (ROC) for the two implemented learning algorithms, plotting TPR against FPR. We observe that the two algorithms have very similar ROC curves that are considerably far away from the ROC curve of a random classifier, illustrated by the dashed diagonal line in the figure. In general, having a higher Area Under Curve (AUC) is desirable because it allows the learning algorithms to operate in the knee point of their ROC, where they attain a very low FPR (between 0.02 and 0.03), while still maintaining a good TPR, around 0.8 to 0.85. It is worth noting that though the two algorithms demonstrate similar ROC curves, their knee points are attained with different *boundary* values: K-means reaches its knee point when *boundary* is set to 0.9, whereas the Gaussian algorithm reaches its knee point when the *boundary* is set to 0.95.

**TABLE 4. Optimal tuning and performance for both K-means and Gaussian algorithm.**

| Algorithm | Accuracy | TPR | FPR | Boundary |
|---|---|---|---|---|
| K-means | 0.9087 | 0.836667 | 0.019333 | 0.9 |
| Gaussian | 0.9143 | 0.857667 | 0.029 | 0.95 |

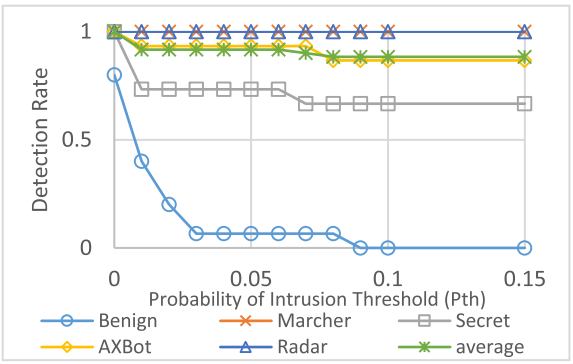



**FIGURE 11. Detection rate of K-means algorithm as a function of detection threshold for Probability of Intrusion ($P_{th}$).**

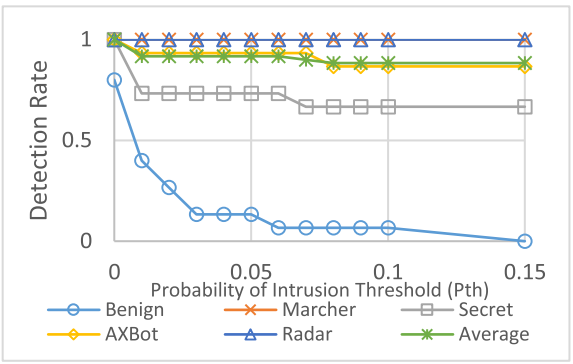



**FIGURE 12. Detection rate of Gaussian algorithm as a function of Probability of Intrusion *threshold* ($P_{th}$).**

This is indeed important when we deal with tuning the learning algorithms by identifying the *threshold* that separates the benign from the malicious region. Table 4 summarizes the results for the optimal tunings of the two learning algorithms.

### D. SECURITY LEVEL

Fig. 11 and Fig. 12 illustrate the detection rate of the K-means and the Gaussian algorithms, respectively, when the *threshold* for the probability of intrusion (cf. Fig. 6) varies from 0 to 0.15. In these figures. As mentioned above, for each malware, we constructed 15 test datasets, each with 600 examples collected from a device infected with that particular malware. We further collected 15 datasets, similarly each with 600 examples, when the device was clean from any malware (benign datasets). For each malware, we fed its 15 associated malicious datasets, one by one, to the IDPS. For each dataset, the IDPS calculates the probability of intrusion

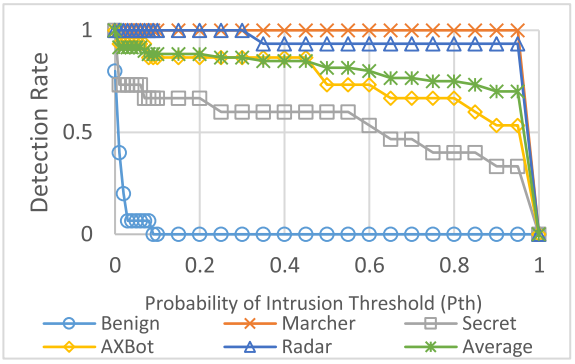



**FIGURE 13. Detection rate of K-means algorithm for large detection thresholds for Probability of Intrusion ($P_{th}$).**

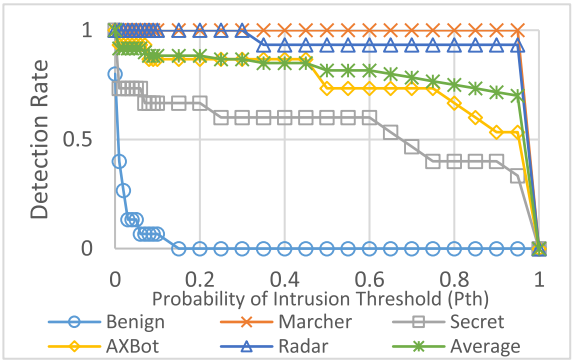



**FIGURE 14. Detection rate of Gaussian algorithm for large values of Probability of Intrusion *threshold* ($P_{th}$).**

and if it exceeds a certain *threshold* (*security level*), it reports a detection in its output.

In these figures, we calculate the *detection rate* or TPR (cf. (3)) as the ratio of the test datasets for a malware that the IDPS correctly detects, out of the 15 fed datasets for that particular malware. That is, the IDPS uses 600 examples in a test dataset to make a single decision; when the ratio of correctly detected examples in a dataset exceeds the probability of intrusion *threshold*, the IDPS considers the overall outcome of the dataset as one TP.

Note that, the *detection rate* for a benign dataset, illustrated by the light blue curve in the figures, is simply calculated as the ratio of the 600 examples in the dataset that are wrongly detected (i.e. FPs). Moreover, the *Average* curve, illustrated in green, is the average of detection rate curves of the four tested malwares.

Fig. 13 and Fig. 14 show the detection performance of K-Means and Gaussian algorithms, respectively, when the probability of intrusion detection extends beyond 0.15 (cf. to Fig. 11 and Fig. 12). As it can be noted from these figures, the best *threshold* for the probability of intrusion lies between 0.1 and 0.15. Lower values result in a too sensitive IDS, causing too many FPs, whereas larger values lead to a loose security, failing to detect too many positive incidents, both cases lead to a deteriorated detection rate. The best threshold is where we can reach a minimal FPR while still maintaining a very good TPR. Note that when $P_{th}$ exceeds 0.15,

**TABLE 5.** Application characteristics.

| Resource | HIDROID | Andromaly |
|---|---|---|
| CPU | 3.8% | 5.5% |
| Battery | 2% | 10% |
| Memory | 5.41MB | 16.38MB |

the detection performance starts deteriorating gradually and when $P_{th}$ approaches 0.5, the detection performance drops sharply. Hence, values larger than 0.5 are not recommended at all.

### E. APPLICATION CHARACTERISTICS

HIDROID occupies 5.41 Mega Bytes (MB) of memory space on the Secure Digital (SD) card. It consumes, in average, 3.8% of CPU, which occasionally peaks to 17%. HIDROID's impact on the battery consumption of the mobile device is only 2%. We measured the CPU consumption, for a duration of two hours, using the command line tool ''top''. We measure the memory occupancy using the android system application ''Application Information''. To measure HIDROID's impact on battery consumption, we developed an application that behaves like a screen saver by randomly display a set of pictures on the screen to prevent the device from sleeping. We let it run for two hours with HIDROID not running and battery level was registered for then, the battery was charged to maximum power (100%). The process was repeated with HIDROID running and we considered the HIDROID consumption as the difference between the first and second values. It should be noted that the measures were performed at least three times and averaged.

## VII. CONCLUSION

In this paper, we developed and tested a novel host-based IDPS application for Android (HIDROID) and presented its implementation details and run-time performance evaluation results. The application is autonomous and runs completely on a mobile device, without relying on a remote server. It monitors the device by regularly sampling features representing the overall resource utilization of the device (e.g. CPU, memory, battery, etc.). The detection engine adopts both ML and statistical algorithms and can detect not only known but also unprecedented (zero-day) attacks. When HIDROID observes a suspicious behavior, it provides the user with the probability of intrusion and takes necessary countermeasure(s) to minimize the risk. It further provides the user with some recommended counteractions. Test experiments with a real-life mobile device and several real malwares showed that HIDROID is well able to discriminate an infected device from a benign one. It demonstrates a very promising accuracy of up to 0.91 while limiting the FPR to below 0.03. HIDROID also incurs minimal computation burden to the host mobile device; it consumes 3.8% of the device's CPU and 2% of its battery. For future work, this study can be extended in several directions. Firstly, it is worthwhile to implement other learning algorithms, e.g. One-Class Support Vector Machine (OCSVM) [35] or combine the detection outcomes of multiple learning algorithms to improve the detection performance. Secondly, there is still lack of comprehensive training datasets for Android malware detection study. Therefore, we plan to create richer datasets, constructed from a wider spectrum of malwares and test HIDROID against them. Finally, for future work, we intend to use an appropriate design pattern, such as Model-View-Controller (MVC), to develop a more stable and efficient version of HIDROID.

## REFERENCES

[1] G. Mantas and N. Komninos, J. Rodriguez, E. Logota, and H. Marques, ''Security for 5G communications,'' in *Fundamentals of 5G Mobile Networks*. Hoboken, NJ, USA: Wiley, 2015, pp. 207–220, doi: 10.1002/9781118867464.ch9.

[2] T. Janevski, ''5G mobile phone concept,'' in *Proc. 6th IEEE Consum. Commun. Netw. Conf.*, vols. 1–2, Jan. 2009, pp. 823–824.

[3] F. B. Saghezchi, J. Rodriguez, S. Mumtaz, A. Radwan, W. C. Y. Lee, B. Ai, M. Islam, S. Akl, A.-E. M. Taha, ''Drivers for 5G: The 'pervasive connected world,'' *Fundamentals of 5G Mobile Networks*. Hoboken, NJ, USA: Wiley, 2015, pp. 1–27, doi: 10.1002/9781118867464.ch1.

[4] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, ''Android security: A survey of issues, malware penetration, and defenses,'' *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015, doi: 10.1109/comst.2014.2386139.

[5] M. La Polla, F. Martinelli, and D. Sgandurra, ''A survey on security for mobile devices,'' *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 446–471, 2013, doi: 10.1109/surv.2012.013012.00028.

[6] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, ''Crowdroid: Behavior-based malware detection system for Android,'' in *Proc. ACM Conf. Comput. Commun. Secur.*, 2011, pp. 15–26, doi: 10.1145/2046614.2046619.

[7] P. Borges, B. Sousa, L. Ferreira, F. B. Saghezchi, G. Mantas, J. Ribeiro, J. Rodriguez, L. Cordeiro, and P. Simoes, ''Towards a hybrid intrusion detection system for Android-based PPDR terminals,'' in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1034–1039, doi: 10.23919/inm.2017.7987434.

[8] A. D. Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Camtepe, and S. Albayrak, ''Monitoring smartphones for anomaly detection,'' *Mobile Netw. Appl.*, vol. 14, no. 1, pp. 92–106, Feb. 2009, doi: 10.1007/s11036-008-0113-x.

[9] A. Bose, X. Hu, K. G. Shin, and T. Park, ''Behavioral detection of malware on mobile handsets,'' in *Proc. 6th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2008, p. 225.

[10] J. Ribeiro, F. B. Saghezchi, G. Mantas, J. Rodriguez, S. J. Shepherd, and R. A. Abd-Alhameed, ''An autonomous host-based intrusion detection system for Android mobile devices,'' in *Mobile Networks and Applications—MONET*. New York, NY, USA: Springer, 2019, doi: 10.1007/s11036-019-01220-y.

[11] O. Mazhelis, ''One-class classifiers: A review and analysis of suitability in the context of mobile-masquerader detection,'' *South Afr. Comput. J.*, vol. 36, pp. 29–48, Jun. 2006.

[12] I. Irigoien, B. Sierra, and C. Arenas, ''Towards application of one-class classification methods to medical data,'' *Sci. World J.*, vol. 2014, Mar. 2014, Art. no. 730712, doi: 10.1155/2014/730712.

[13] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, ''A comparison of static, dynamic, and hybrid analysis for malware detection,'' *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 1, pp. 1–12, Feb. 2017, doi: 10.1007/s11416-015-0261-z.

[14] Y. Chen, Y. Li, X. Q. Cheng, and L. Guo, ''Survey and taxonomy of feature selection algorithms in intrusion detection system,'' in *Proc. Inf. Secur. Cryptol.*, vol. 4318, 2006, pp. 153–167.

[15] N. Peiravian and X. Zhu, ''Machine learning for Android malware detection using permission and API calls,'' in *Proc. IEEE 25th Int. Conf. Tools with Artif. Intell.*, Washington, DC, USA, Nov. 2013, doi: 10.1109/ictai.2013.53.

[16] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 961–987, 2nd Quart., 2014, doi: 10.1109/surv.2013.101613.00077.
[17] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, Feb. 2012, doi: 10.1007/s10844-010-0148-x.
[18] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, Dec. 2019, doi: 10.1186/s42400-019-0038-7.
[19] Y. Mehmood, M. A. Shibli, U. Habiba, and R. Masood, "Intrusion detection system in cloud computing: Challenges and opportunities," in *Proc. 2nd Nat. Conf. Inf. Assurance (NCIA)*, Dec. 2013, pp. 59–66.
[20] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 924–935, Sep. 2019, doi: 10.1109/tnsm.2019.2927886.
[21] S. Garg, K. Kaur, N. Kumar, S. Batra, and M. S. Obaidat, "HyClass: Hybrid classification model for anomaly detection in cloud environment," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–7, doi: 10.1109/icc.2018.8422481.
[22] S. Garg, K. Kaur, S. Batra, G. Kaddoum, N. Kumar, and A. Boukerche, "A multi-stage anomaly detection scheme for augmenting the security in IoT-enabled applications," *Future Gener. Comput. Syst.*, vol. 104, pp. 105–118, Mar. 2020, doi: 10.1016/j.future.2019.09.038.
[23] S. Garg, K. Kaur, S. Batra, G. S. Aujla, G. Morgan, N. Kumar, A. Y. Zomaya, and R. Ranjan, "En-ABC: An ensemble artificial bee colony based anomaly detection scheme for cloud environment," *J. Parallel Distrib. Comput.*, vol. 135, pp. 219–233, Jan. 2020, doi: 10.1016/j.jpdc.2019.09.013.
[24] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on Android," in *Proc. IEEE Int. Conf. Commun.*, Dresden, Germany, Jun. 2009, p. 631.
[25] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, *MADAM: A Multi-Level Anomaly Detector for Android Malware* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence: Lecture Notes in Bioinformatics), vol. 7531. Berlin, Germany: Springer, 2012, pp. 240–253, doi: 10.1007/978-3-642-33704-8_21.
[26] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for Android applications," in *Proc. 21st USENIX Secur. Symp.*, 2012, pp. 539–552.
[27] *Drozer User Guide*. Accessed: Sep. 13, 2019. [Online]. Available: https://labs.mwrinfosecurity.com/tools/drozer/
[28] W. Enck, M. Ongtang, and P. Mcdaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, 2009, pp. 235–245.
[29] S. S. Khan and M. G. Madden, "One-class classification: Taxonomy of study and review of techniques," *Knowl. Eng. Rev.*, vol. 29, no. 3, pp. 345–374, Jun. 2014, doi: 10.1017/s026988891300043x.
[30] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Artificial Intelligence and Cognitive Science*, vol. 6206. Berlin, Germany: Springer, 2010, pp. 188–197, doi: 10.1007/978-3-642-17080-5_21.
[31] J. Ribeiro, G. Mantas, F. B. Saghezchi, J. Rodriguez, S. J. Shepherd, and R. A. Abd-Alhameed, "Towards an autonomous host-based intrusion detection system for Android mobile devices," in *Social-Informatics and Telecommunications Engineering* (Lecture Notes of the Institute for Computer Sciences), vol. 263. Cham, Switzerland: Springer, 2019, pp. 139–148, doi: 10.1007/978-3-030-05195-2_14.
[32] *Stackoverflow*. Accessed: Sep. 13, 2019. [Online]. Available: https://stackoverflow.com/questions/
[33] *Android Studio*. Accessed: Sep. 13, 2019. [Online]. Available: https://developer.android.com/studio/index.html
[34] *Introduction to Android Activity Class*. Accessed: Oct. 14, 2019. [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities
[35] S. Ramyar, A. Homaifar, A. Karimoddini, and E. Tunstel, "Identification of anomalies in lane change behavior using one-class SVM," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Budapest, Hungary, Oct. 2016, pp. 4405–4410.

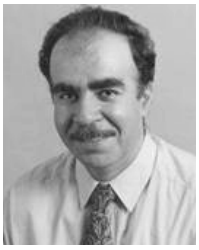
**JOSÉ RIBEIRO** received the B.Sc. degree (five year course) in telecommunications and electronic engineering and the M.Sc. degree in mobile telecommunications from the Polytechnic Institute of Castelo Branco, Portugal. He is currently pursuing the Ph.D. degree in an autonomous host-based intrusion detection and prevention system for android mobile devices. In 2007, he joined the Instituto de Telecomunicações—Aveiro, as a Researcher. He has been involved in European research projects, namely, FP6 ORACLE working on the integration and validation of the project's demonstrator, FP7 HURRICANE working on a platform for vertical handover simulation in heterogeneous networks environments and on FP7 COGEU working on a demonstrator. Worked on NewP@ss project promoted by national founding COMPETE and also had a small participation on the CarCoDe project as a Reviewer. He participated in Mobitrust CATRENE Project, reference: CA208. His main research interest includes PHY and MAC advanced techniques and intrusion detection systems for wireless mobile communications.

**FIROOZ B. SAGHEZCHI** (Senior Member, IEEE) received the B.Sc. degree from the University of Tabriz, Iran, in 2000, the M.Sc. degree from Shiraz University, Iran, in 2003, and the joint Ph.D. degrees from the Universities of Porto, Aveiro, and Minho, Portugal, in 2016, all in electrical engineering-telecommunications. From 2003 to 2010, he was a Lecturer with Islamic Azad University, Iran. Then, he joined Instituto de Telecomunicações, Portugal, as a Researcher, where he worked on several European research projects such as FP7 C2POWER, ENIAC E2SG, and HURRICANE. From 2016 to 2019, he held a postdoctoral position with the University of Aveiro, where he worked on two large-scale European research projects H2020-ECSEL SEMI40 and CATRENE MOBITRUST. Since 2019, he has been a Senior Researcher with the Instituto de Telecomunicações, Portugal, where he is involved in the European research project H2020 5GENESIS. He has coauthored several peer-reviewed journal and conference publications and book chapters. His research interests include 5G mobile networks, machine learning, intrusion detection, and cyber-physical systems.

**GEORGIOS MANTAS** received the Diploma degree in electrical and computer engineering from the University of Patras, Greece, in 2005, the M.Sc. degree in information networking from Carnegie Mellon University, Pittsburgh, PA, USA, in 2008, and the Ph.D. degree in electrical and computer engineering from the University of Patras, Greece, in 2012.

In 2014, he became a Postdoctoral Researcher with the Instituto de Telecomunicações, Aveiro, Portugal, where he has been involved in research projects such as ECSEL - SemI40, CATRENE - MobiTrust, CATRENE - NewP@ss, ARTEMIS - ACCUS, FP7 - CODELANCE, and FP7 - SEC-SALUS. Since 2018, he has been a Lecturer with the University of Greenwich, U.K. His main research interests include network and system security, authentication mechanisms, privacy-preserving mechanisms, intrusion detection systems, and secure network coding.

**JONATHAN RODRIGUEZ** (Senior Member, IEEE) received the masters degree in electronic and electrical engineering and the Ph.D. degree from the University of Surrey, U.K., in 1998 and 2004, respectively. In 2005, he became a Researcher with the Instituto de Telecomunicações (IT), Portugal, where he was a member of the Wireless Communications Scientific Area. In 2008, he became a Senior Researcher, where he established the 4TELL Research Group targeting next generation mobile systems. Since 2009, he has been serving as an Invited Assistant Professor with the University of Aveiro, Portugal, and attained Associate Level, in 2015. He is currently the Coordinator of the H2020-SECRET Innovative Training Network. In 2017, he was appointed as a Professor of mobile communications with the University of South Wales, U.K. He has served as a Project Coordinator for major international research projects, including Eureka LOOP and FP7 C2POWER whilst serving as a Technical Manager for FP7 COGEU and FP7 SALUS. He has authored more than 500 scientific works, including 10 book editorials. His professional affiliations include Chartered Engineer (CEng), since 2013, and Fellow of the IET, in 2015.



**RAED A. ABD-ALHAMEED** (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from Basrah University, Basrah, Iraq, in 1982 and 1985, respectively, and the Ph.D. degree from the University of Bradford, West Yorkshire, U.K., in 1997. He has also been a Research Visitor with Wrexham University, Wales, since September 2009 including the wireless and communications research areas. He is currently a Professor of electromagnetic and radio frequency engineering with the University of Bradford, U.K. He has long years' research experience in the areas of radio frequency, signal processing, propagations, antennas and electromagnetic computational techniques. He has published more than 600 academic journal and conference articles; in addition, he has coauthored four books and several book chapters. At the present, he is the Leader of radio frequency, propagation, sensor design, and signal processing; in addition to leading the Communications Research Group for years within the School of Engineering and Informatics, Bradford University, U.K. He is a Principal Investigator for several funded applications to EPSRCs and leader of several successful knowledge Transfer Programmes such as with Arris (previously known as Pace plc), Yorkshire Water plc, Harvard Engineering plc, IETG Ltd., Seven Technologies Group, Emkay Ltd., and Two World Ltd. including many Research Development Projects awards supported by Regional European funds. He has also been a Co-Investigator in several funded research projects including 1) H2020 MARIE Sklodowska-CURIE ACTIONS: Innovative Training Networks (ITN) Secure Network Coding for Next Generation Mobile Small Cells 5G-US, 2) Nonlinear and demodulation mechanisms in biological tissue (Department of Health, Mobile Telecommunications & Health Research Programme and 3) Assessment of the Potential Direct Effects of Cellular Phones on the Nervous System (EU: collaboration with 6 other major research organizations across Europe). He was awarded the Business Innovation Award for his successful KTP with Pace and Datong companies on the design and implementation of MIMO sensor systems and antenna array design for service localizations. He is the Chair of several successful workshops on Energy Efficient and Reconfigurable Transceivers (EERT): Approach towards Energy Conservation and CO2 Reduction that addresses the biggest challenges for future wireless systems. He has also appointed as a guest editor for the IET Science, Measurements and Technology Journal, since 2009, and 2012. His interests are in 5G green communications systems, computational methods and optimizations, wireless and mobile communications, sensor design, EMC, MIMO systems, beam steering antennas, energy efficient PAs, and RF predistorter design applications. He is the Fellow of the Institution of Engineering and Technology and of Higher Education Academy. He is a Chartered Engineer.