**QUEEN'S
UNIVERSITY
BELFAST**

DroidLight: Lightweight Anomaly-Based Intrusion Detection System for Smartphone Devices

Barbhuiya, S., Kilpatrick, P., & Nikolopoulos, D. (2020). DroidLight: Lightweight Anomaly-Based Intrusion Detection System for Smartphone Devices. In *Proceedings of the 21st International Conference on Distributed Computing and Networking: ICDCN 2020: 21st International Conference on Distributed Computing and Networking Kolkata India January, 2020* Association for Computing Machinery.
<https://doi.org/10.1145/3369740.3369796>

Published in:

Proceedings of the 21st International Conference on Distributed Computing and Networking

Document Version:

Peer reviewed version

Queen's University Belfast - Research Portal:

[Link to publication record in Queen's University Belfast Research Portal](#)

Publisher rights

Copyright 2020 ACM. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

DroidLight: Lightweight Anomaly-based Intrusion Detection System for Smartphone Devices

Sakil Barbhuiya
Queen's University Belfast
United Kingdom
sakil.barbhuiya@qub.ac.uk

Peter Kilpatrick
Queen's University Belfast
United Kingdom
p.kilpatrick@qub.ac.uk

Dimitrios S. Nikolopoulos
Queen's University Belfast
United Kingdom
d.nikolopoulos@qub.ac.uk

ABSTRACT

Smartphone malware attacks are increasing alongside the growth of smartphone applications in the market. Researchers have proposed techniques to detect malware attacks using various approaches, which broadly include signature and anomaly-based intrusion detection systems (IDSs). Anomaly-based IDSs usually require training machine learning models with datasets collected from running both benign and malware applications. This may result in low detection accuracy when detecting zero-day malwares, i.e. those not previously seen or recorded. In this paper, we propose DroidLight, a lightweight IDS which can detect zero-day malware efficiently and effectively. We designed an algorithm for DroidLight that is based on one class classification and probability distribution analysis. For each smartphone application, the classification model learns its normal CPU utilisation and network traffic pattern. The model flags an intrusion alert if there is any significant deviation from the normal pattern. By deploying three self-developed malwares we performed realistic evaluation of DroidLight, i.e. the evaluation was performed on a real device while a real user was interacting with it. Evaluation results demonstrate that DroidLight can detect smartphone malwares with accuracy ranging from 93.3% to 100% while imposing only 1.5% total overhead on device resources.

CCS CONCEPTS

• **Security and privacy** → **Malware and its mitigation; Intrusion detection systems; Denial-of-service attacks;** • **Human-centered computing** → **Smartphones;** • **Computing methodologies** → *Classification and regression trees.*

KEYWORDS

Smartphone, Intrusion Detection, Zero-day Malware, One Class Classification

ACM Reference Format:

Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. 2020. DroidLight: Lightweight Anomaly-based Intrusion Detection System for Smartphone Devices. In *21st International Conference on Distributed Computing and Networking (ICDCN 2020)*, January 4–7, 2020, Kolkata, India. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3369740.3369796>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDCN 2020, January 4–7, 2020, Kolkata, India

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7751-5/20/01...\$15.00

<https://doi.org/10.1145/3369740.3369796>

1 INTRODUCTION

Smartphones are becoming ever more popular. In 2019, the number of smartphone users reached 3.3 billion and it is expected that the number of smartphone users will reach 3.21 billion in 2020¹. This growth in smartphones has attracted an increasing number of security attacks on smartphone devices in recent years, mostly in the form of malicious applications (commonly termed malware). According to Kaspersky Lab² smartphone malware attacks doubled in 2018 (116.5 million) as compared to 2017 (66.4 million).

Researchers have proposed techniques to detect smartphone intrusions using various approaches, which broadly include signature and anomaly-based techniques such as [1, 5, 6, 8, 12, 13, 20, 21, 23, 26, 27]. However, the majority of these state-of-the-art techniques emphasise detection accuracy, while neglecting efficiency in terms of resource consumption, which manifests as performance overhead on the device (from here on we will use the terms device and smartphone interchangeably) resources and timeliness/detection latency, which is the time between arrival of an intrusion and its detection. Although efficiency is usually high for signature-based intrusion detection, efficiency always suffers in the case of anomaly-based intrusion detection. This is because anomaly-based intrusion detection usually incorporates computationally expensive data extraction and machine learning algorithms, which impose significant performance overhead on the device due to its constrained resources. The work in [7] experimentally demonstrates that running a classification algorithm on the smartphone device for anomaly-based intrusion detection can result in significant overhead on the device resources such as CPU, memory, and battery.

Furthermore, anomaly-based smartphone IDSs usually deploy machine learning techniques, which require training binary class classification models with the datasets collected from running both benign and malware samples inside sandboxes or emulators. Collecting samples of all the malware families is not an easy task as there are always new variants of malware. Also, there are new malware families which can sense if they are running on real devices or on any kind of emulator or sandbox in order to bypass model training³. Therefore, machine learning based IDSs may not be able to detect zero-day intrusions, i.e., those not recorded by the classification models during the training phases. To address this, researchers in [19] and [25] introduced one class classification (OCC) based IDSs, which can work with only the normal behavioural data collected from running benign applications and does not require training with the malware samples. However, such OCC based IDSs

¹<https://www.statista.com>

²<https://www.kaspersky.com/>

³<https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf>

usually suffer from high false positive rates due to the outliers present in the training datasets. These outliers may arise as a result of sudden behavioural changes in the smartphone, for example, a sudden CPU spike due to some Android OS maintenance activity, a sudden rise in the network traffic due to some background download or installation processes, etc. Also, OCC model training with the device-wide behavioural data may result in low detection accuracy rate. This is because device-wide behaviour usually covers a variety of behavioural patterns, which may bring challenge for the OCC classifier in identifying the malicious pattern due to the malware attack. To address the issues discussed above, we propose DroidLight, a lightweight smartphone IDS that detects zero-day malwares efficiently and effectively. In particular, DroidLight makes the following contributions to address the issues in the existing smartphone IDSs:

- (1) *Probability distribution analysis on the raw data to reduce false positive rate:* DroidLight detects zero-day intrusions using OCC models. These models learn the normal pattern of the device resource utilisation and classify a behaviour as malicious if it deviates significantly from its normal pattern. DroidLight addresses the high false positive rate issue in the OCC by performing window-based probability distribution analysis on the raw resource utilisation data as a pre-processing step for model training and testing.
- (2) *Application-specific modelling to achieve high detection accuracy:* We achieve high detection accuracy for DroidLight by building application-specific OCC models, which learn the normal resource utilisation pattern for each individual application available in the device. During the testing/detection phase, DroidLight chooses the OCC model that is specific to the application running in the foreground. This model can accurately perform the classification between normal and malicious application behaviour.
- (3) *Hybrid intrusion detection approach to improve DroidLight efficiency:* We build DroidLight with hybrid services, i.e. DroidLight-Host and DroidLight-Server. DroidLight-Server performs the heavyweight task of training the application-specific OCC models on a remote server, whereas DroidLight-Host performs only a small amount of computation on the device for detecting the intrusions. DroidLight-Host processes only the previous one minute of CPU utilisation and network traffic data (each sampled at 5 seconds) in order to detect the intrusions on the device. Firstly, it extracts $(60/5) = 12$ samples for each metric, i.e. a total of $(12 \times 2) = 24$ metric samples from the device and, secondly, it pre-processes these samples by using probability distribution analysis before performing the OCC for the intrusion detection.

We created a realistic evaluation environment for DroidLight, where we deployed three self-developed malwares on a real smartphone device. The malwares represent the behaviour of information theft, currency-mining bot, and DDoS attack. We evaluated DroidLight accuracy and efficiency in detecting these malwares while a real user was interacting with the device. This realistic evaluation process ensures that DroidLight can maintain its accuracy even in detecting the malwares on real devices under their realistic usage.

Experimental evaluation with three different device usage conditions (device idle, user using WhatsApp, user playing YouTube video) suggests that DroidLight can detect high intensive (in terms of device resource usage) smartphone malwares with accuracy ranging from 93.3% to 100%. This accuracy is achieved by pre-processing the raw utilisation data using probability distribution analysis and then performing application-specific OCC modelling using the pre-processed data. Experimental results reveal that this accuracy is better than the accuracy achieved by performing application-specific OCC modelling using raw data and by performing device-wide OCC modelling using the pre-processed data. Results further prove that false positive rates are reduced when we use probability distribution analysis as data pre-processing. Experiments also indicate that performing the intrusion detection by DroidLight on the device every one minute imposes a total 1.5% performance overhead on the device resources, which is comparable with MADAM [20].

The remainder of the paper is organised as follows. Section 2 presents related work in efficient smartphone IDSs, use of OCC for smartphone IDSs, and realistic evaluation of smartphone IDSs. Section 3 discusses the DroidLight approach for smartphone intrusion detection. Section 4 and Section 5 provide details of the DroidLight algorithm and architecture, respectively. Experimental results are presented and discussed in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2 RELATED WORK

We discuss the state of the art in smartphone IDSs under the following two categories: (i) efficient smartphone IDSs and (ii) one class classification for smartphone IDSs.

(i) *Efficient smartphone IDSs:* In general, there are two main approaches in smartphone IDSs: signature-based and anomaly-based. Signature-based systems [1], [12], [6], [27], [5] are based on matching target application signatures with known malware sample signatures, which are stored in a signature database. The signatures are obtained by using reverse engineering methods, which decompile application packages to extract the signatures. Since signature-based malware detection systems are based on known malwares, they cannot detect unknown malwares or variants of known malwares. Anomaly-based or behaviour-based systems [21], [8], [13], [20], [26], [23] monitor application or system behaviour in order to identify anomalous activities which may arise due to malware attacks. In general, they identify anomalous activities by deploying machine learning classification algorithms such as SVM, HMM, Naive Bayes, KNN, etc. The computational costs of these algorithms are generally high. Moreover, smartphone IDSs need to perform the detection in timely fashion with low detection latency, which is an important aspect when one considers the potential of modern malware to cause serious damage in a very short period of time.

Recently, researchers proposed host-based IDSs like Andromaly [21], MADAM [20], and Drebin [3], which can perform malware detection on the device in a lightweight manner. MADAM is efficient with CPU and memory overhead of 0.9% and 9.4%, respectively, whereas Andromaly shows CPU and memory overhead of 5.5% and 8.5%, respectively. On the other hand, Drebin is efficient in terms of its detection latency. However, the efficiency for these IDSs was explored only for the testing/detection phase without considering

the efficiency during the training phase. For machine learning based IDSs, it is important to have their efficiency in both the training and the detection phases. Researchers in [7], [4], [14], [18] put their effort in distributing the malware detection tasks between device and remote servers/clouds in order to achieve better efficiency by performing pre-processing of the data on the device and offloading execution of the heavyweight machine learning algorithms to the server. [4] runs static analysis on the host side before performing dynamic analysis on the server side for detailed malware analysis. [7] showcases the tradeoff between executing intrusion detection on device and on server, without proposing any particular scheduling algorithm to decide where to run the analysis; moreover, they do not provide a solution to distribute the analysis task between device and server. In [14] and [18] only architectures of the proposed hybrid solutions are proposed without any working proof of concept prototype. DroidLight uses the hybrid solution in a different way, where the heavyweight training of the classification models is performed on a remote server and the lightweight intrusion detection tasks are performed on the device. Importantly, the training is performed on a continuous basis until stable models are built, which produce accurate results in detecting the intrusions.

(ii) *One class classification for smartphone IDSs*: One class classification (OCC) is a form of classification, which can work with only a single class of data, without requiring a second class of data as it is the case in binary class classification. Researchers from different domains have used OCC algorithms in outlier/novelty detection and concept learning in scenarios where data from negative class is absent, poorly sampled or not defined well [11]. In the literature so far only a couple of works [19] and [25] propose OCC algorithm in detecting smartphone intrusions. In [19] a One-Class Support Vector Machine (SVM) algorithm is used. The algorithm trains the classification model using only the features (application permission and control flow graphs) extracted from the Android manifest and bytecode files of the benign applications. During the testing or detection phase, the classification model raises a negative alarm if the testing data are significantly different from the training data. Thus, [19] follows a static analysis approach for detecting the intrusions. The authors in [25] propose accurate detection of zero-day smartphone malware by using One-Class SVM, which identifies anomalous behaviour that deviates from normal behaviour of a large number of benign applications. They performed the analysis for intrusion detection on a remote cloud server in order to achieve better efficiency. DroidLight on the other hand uses OCC for performing dynamic analysis of the behavioural data to detect zero-day intrusions in smartphones under their realistic usage. Moreover, DroidLight performs the detection on the device efficiently and effectively.

3 DROIDLIGHT APPROACH TO INTRUSION DETECTION

DroidLight aims at detecting zero-day malwares, which have not been discovered by any IDS or anti-malware software before and do not have any samples available for training binary class classification based IDSs. DroidLight uses one class classification (OCC) for detecting zero-day malwares. For smartphone IDSs, OCC can learn the normal pattern of the device resource utilisation and classify

a pattern as malicious if it deviates significantly from its normal pattern. At this point, it is important to note our **first assumption**: malware attacks (as discussed in the previous section) generate significant deviation in the normal resource utilisation pattern of the device. We support this assumption with the observations from a preliminary experiment, where we considered playing a YouTube video on an Android device for a period of 10 minutes. In the second half of playing the video, i.e. in the minutes 6-10, we injected a self-developed DDoS attack that continuously sends thousands of network packets per second to a remote web service. The time-series graph in Figure 1 presents the CPU utilisation pattern of the YouTube application in the experimental period. The DDoS attack period (second half - minutes 6-10) is marked in colour. The graph supports our assumption by showing the significant deviation in the CPU utilisation pattern during the DDoS attack period. In the literature we note works such as [21], [22], [2], [15], [9], [16], [17] where deviations in smartphone resource utilisation are used as indicators of smartphone intrusions.

Figure 1: Time-series (top) and probability distribution (bottom) graphs of CPU utilisation for YouTube application under a self-developed DDoS attack in the second half (minutes - 6 to 10 in colour)

Table 1: Probability Distribution Analysis Results

Minutes	1	2	3	4	5	Avg	6	7	8	9	10	Avg
Mean	56.6	48.0	49.3	47.3	52.4	50.7	76.0	74.2	73.7	75.0	73.2	74.4
SD	15.1	4.1	3.1	5.4	8.4	5.5	5.7	3.1	3.6	3.6	2.3	3.60
MaxPD	0.02	0.10	0.13	0.07	0.05	0.07	0.07	0.13	0.12	0.12	0.17	0.12

For DroidLight, we use the OCC algorithm that is proposed by [10]. The algorithm first generates artificial data from a multi-variate normal distribution as estimated from the training data and secondly, uses these artificial data as a second class in the construction of a binary class classification model. The classification is based on the Bayes' Theorem⁴. Although the OCC algorithm

⁴<http://www.investopedia.com/terms/b/bayes-theorem.asp>

brings the benefit of detecting a malware without requiring its prior knowledge, it poses the challenge of reducing false malware alarms that may arise due to the outliers/spikes which are usually present in the training datasets. This challenge is rather common for all categories of classification algorithms. In the time-series graph in Figure 1, we observe two CPU utilisation spikes in minute 1, which are marked with green circles. These spikes match the malicious behaviour in the second half, and hence may be wrongly identified as malware and result in false alarms. However, we further observe that the utilisation spikes are instantaneous, meaning that they rise high, but drop back to their normal utilisation quickly, whereas the utilisation due to DDoS attack remains continuously high for a period of time. Based on this observation we make our **second assumption**: resource utilisation spikes in smartphones are usually instantaneous and high resource utilisation due to malware attacks is consistent for a period of time. Considering this assumption, we propose a solution for the false alarm issues which may arise due to the outliers/spikes in the training data. We pre-process the raw resource utilisation data using probability distribution analysis before they are used for training the OCC models. A window-based (per minute) probability distribution of the raw data gives us distribution curves for each minute in the experimental period. The distribution curves are presented in Figure 1. Each distribution curve can represent three distribution characteristics of the raw data: Mean, Standard Deviation (SD), and Maximum Probability Density (MaxPD). From the distribution curves we observe that the curves are shifting towards the right with slightly thinner and taller appearances in the second half when the DDoS attack is in action. These shifts are actually happening in the form of changes in the values of the distribution characteristics as seen in Table 1. Considering the average values in the first half (minutes 1-5) and in the second half (minutes 6-10), we observe the following changes in the second half: (i) the Mean value is increased from 50.7 to 74.4, (ii) the SD value is reduced from 5.5 to 3.6, and (iii) the MaxPD value is increased from 0.07 to 0.12. Importantly, we observe that the curve in minute 1 (where we noticed two spikes in the time-series graph) is flatter than the other curves in the first half, which is due to the fact that the CPU utilisation values are distributed between the high utilisation (spikes) and the normal utilisation. From Table 1, we see that the distribution characteristics in minute 1 (Mean=56.6, SD=15.1, MaxPD=0.02) do not match with the characteristics in the second half: rather they are close to the characteristics in the first half.

In summary, we can classify between normal behaviour and malicious behaviour as well as between a normal spike/outlier and malicious behaviour by performing pre-processing of the raw resource utilisation data with probability distribution analysis and using the distribution characteristic values (Mean, SD, MaxPD) as input for the OCC models. One may argue to use only window-based averaging of the raw data to solve the false alarm issues, but probability distribution characteristics (Mean, SD, MaxPD) give us more independent metrics to achieve better classification results.

Furthermore, we follow an application-specific modelling approach for DroidLight. This is based on our **third assumption**: the resource consumption on the device is due to the application running in the foreground and the background services or activities running in the device contribute negligible amount towards

this consumption. This assumption is valid for all the Android OS based smartphone devices as the Android OS always prioritises the foreground application over the background services and activities running in the device⁵.

Finally, while considering the evaluation of DroidLight, we found that researchers usually create malware datasets by running malware applications (downloaded from online malware repositories) individually on sandboxes or emulators and they do not consider any user activities during the malware application run. These datasets may not represent realistic malware behaviour as malwares are likely to perform attacks even when users are interacting with their devices and performing benign activities. This may result in degraded accuracy for anomaly-based IDSs while identifying intrusions on real devices. We address this issue by performing the testing of the classification models using the data collected from a real smartphone device under both benign and malicious conditions. To avoid running genuine malware applications on real devices, we deploy self-developed malwares that represent the malicious activities of the real world malware applications. In Section 6, we further support the three assumptions that we made in this section as well as validating our approach with rigorous experiments. In the next section, we describe the algorithm that we developed for DroidLight based on the OCC algorithm proposed by [10] and our approach as discussed above.

4 DROIDLIGHT ALGORITHM

The DroidLight algorithm comprises three phases: data pre-processing, model training, and intrusion detection. We present the pseudocode of the algorithm in Algorithm 1 and explain the phases as follows.

4.1 Data Pre-processing

During this phase the DroidLight algorithm performs probability distribution analysis on the raw resource utilisation metric data (CPU utilisation and network traffic). This analysis is window-based, where for each application the algorithm divides the time-series data into a number of data bins (pseudocode line 2), each with a fixed window size (1 minute) and does statistical computation to prepare the following vector that consists of distribution characteristics for each metric in each bin: [Mean, Standard Deviation (SD), Maximum Probability Density (MaxPD)] (pseudocode lines 3-9). These vectors created for all the metrics constitute the vector for each bin, for example [cpuMean, cpuSD, cpuMaxPD, networkMean, networkSD, networkMaxPD] (pseudocode line 10). Finally, in this phase, the algorithm returns an array (one for each application) that contains a number (equal to the number of data bins) of vectors (pseudocode lines 12 and 14).

4.2 Model Training

During this phase, the DroidLight algorithm builds OCC models for each application by taking the input as an array of vector data points (pseudocode lines 1-4), which are generated by the data pre-processing phase.

⁵<https://developer.android.com/guide/topics/processes/process-lifecycle.html>

4.3 Intrusion Detection

During this phase, the DroidLight algorithm identifies whether the testing data vector is normal or anomalous. The testing vector is generated by the data pre-processing phase by considering only the last 1 minute of data. The testing data period is decided experimentally, which gives enough samples to distinguish between normal and anomalous data samples. The algorithm uses the trained OCC model (specific to the application running in the foreground) in order to perform the classification of the testing vector point (pseudocode line 1). Finally, the algorithm flags an intrusion when the OCC model classifies the testing vector point as anomalous (pseudocode lines 2-6).

Algorithm 1 DroidLight Algorithm

Data Pre-processing

input: *data* - resource utilisation metric data set for all *N* applications in the smartphone with *L* metrics; *DW* - distribution window = 1 minute
output: *A* - array of *B* (total distribution bins) vectors (one for each application)
abbreviation: *SD* = Standard Deviation, *PD* = Probability Density

```

1: for each application  $a_i$  where  $i=1,...,N$  do
2:    $distributionBin(DB_{ij}) = data_i / DW$  where  $j=1,...,B$ 
3:   for each  $DB_{ij}$  do
4:     for each metric  $m_k$  where  $k=1,...,L$  do
5:        $Mean = DB_{ij}.getMean(m_k)$ 
6:        $SD = DB_{ij}.getSD(m_k)$ 
7:        $MaximumPD = DB_{ij}.getMaximumPD(m_k)$ 
8:        $vectorForMetric(VM_k) = [Mean, SD, MaximumPD]$ 
9:     end for
10:     $vectorForBin(VB_j) = [VM_1, ..., VM_L]$ 
11:   end for
12:    $arrayForApplication(A_i) = \{VB_1, ..., VB_B\}$ 
13: end for
14: return A

```

Model Training

input: *A* - array of *B* (total distribution bins) vectors with *L* metrics generated by pre-processing phase (one for each application)
output: *M* - array of *N* OCC models (one for each application)

```

1: for each application  $a_i$  where  $i=1,...,N$  do
2:    $A_i.setClassAttribute(a_i)$ 
3:    $M_i = buildOCCModel(A_i)$ 
4: end for
5: return M

```

Intrusion Detection

input: *X* - a single vector with *L* metrics generated by pre-processing phase (for last 1 minute of data samples collected from the currently running foreground application); *M_x* - trained OCC model for the currently running foreground application
output: 0 - normal; 1 - anomaly

```

1:  $intrusionIdentificationResult = M_x.classify(X)$ 
2: if (intrusionIdentificationResult is NORMAL) then
3:   return 0
4: else
5:   return 1
6: end if

```

5 DROIDLIGHT ARCHITECTURE

We built DroidLight with hybrid services, i.e. DroidLight-Host and DroidLight-Server. Fig 2 illustrates the components of both these services. DroidLight-Host performs the detection of intrusions on the smartphone device in a lightweight manner, whereas DroidLight-Server performs the training of the application-specific

OCC models on a remote server. Modules of both the DroidLight-Host and DroidLight-Server are described in the following sections.

Figure 2: DroidLight Architecture

5.1 DroidLight-Host

DroidLight-Host implements the following modules on the smartphone device:

5.1.1 Behavioural Metrics Collector. This module collects the behavioural metrics such as CPU utilisation and total number of network packets transmitted/received. The frequency of collecting these metrics is 5 seconds, which we experimentally found to be optimal on our experimental device. Any frequency higher than this increases overhead on device performance, whereas any frequency lower than this results in poor detection accuracy. However, this parameter may require tuning for some users and some devices with respect to their usage. The module uses top (linux-based system monitoring utility) for collecting CPU utilisation and checks proc files in the Android system for retrieving network packet information.

5.1.2 Local Storage. We use the Android SD card as the storage location for this module. The SD card stores the metrics collected by the Behavioural Metrics Collector module and, in addition, stores the OCC models trained by the DroidLight-Server. The metrics and the models are stored with labels specific to the applications, which are active in the device. The module sends the metrics data periodically to the DroidLight-Server. The DroidLight-Server stores the metrics data in its remote storage module and performs the training of the OCC models. The module retrieves the updated models once they are trained successfully. This communication is established via HTTP that communicates with a PHP web service running on the DroidLight-Server. We consider a time interval of 5 minutes for the communication, which we experimentally found to be optimal in terms of overhead on device performance. Again, users can tune this time interval based on their device and network strength to communicate with the remote server. It is important to note that the module flushes the metrics data every time it communicates and sends the data to the DroidLight-Server, and hence local storage needs to store a maximum of the last 5 minutes of metrics data only. Also, the retrieval of the updated

models from the DroidLight-Server for an individual application stops depending on its model performance. For example, training for WhatsApp may stop once it is trained with 60 minutes of data samples when the trained model starts performing accurately, i.e. no more updates are required for the WhatsApp model after 60 minutes of its run. Hence, the size of the trained models stored in the local storage may vary.

5.1.3 DroidLight Intrusion Detector. This module periodically executes the DroidLight algorithm (explained in section 4) for intrusion detection. As the algorithm requires the previous 1 minute of testing data samples, the module executes the algorithm every 1 minute. The module gets the metrics data and the OCC model (specific to the foreground application) from the storage module. We design and implement this module as an Android background application using Android Studio⁶, which further imports Apache Common Maths⁷ and Weka⁸ libraries for performing statistical operations and OCC. The application integrates with the other two modules discussed above to complete the functionality of DroidLight-Host in performing the intrusion detection.

5.2 DroidLight-Server

The DroidLight-Server implements the following modules on a remote server:

5.2.1 Remote Storage. This module receives the metrics data sent periodically by DroidLight-Host. The module stores the metrics as well as the OCC models trained by the DroidLight Model Trainer module, which is explained in the next section. The module flushes the metrics data for an application once it gets an accurate model and updates the models every time they are trained with newer datasets.

5.2.2 DroidLight Model Trainer. The DroidLight algorithm uses OCC models (specific to the applications active in the smartphone) in order to detect the intrusions. These models require training with the normal behavioural metrics data stored in the remote storage module in the DroidLight-Server. We perform the training of the models by implementing a Java application in the server that executes whenever there are updated metrics data in the remote storage. The Java application integrates Weka library to build the OCC models for each application that has training metrics data stored in the remote storage. Each Smartphone application has its own requirement of training data samples to produce an accurate model and hence, the training period for the models (which are application specific) varies application to application, device to device, and also, user to user.

6 EVALUATION

This section presents the evaluation of DroidLight in terms of its detection accuracy and efficiency.

6.1 Data Collection

To collect the training and testing data from the device we installed DroidLight-Host (section 5) on an Android smartphone (Samsung

Table 2: Self-developed Malware Attack Summary

Attack Size	DroidDDoS	DroidThief
Small	transmits 100-200 network packets per second	steals a file of 1KB of recent call logs
Medium	transmits 2000-2400 network packets per second	steals an image of 722KB
Large	transmits 5000-6000 network packets per second	steals an image of 1.4MB

Galaxy S2 duos running Android Lollipop version 5.0). We refer to this smartphone as *droid-device*. DroidLight-Host monitors the behavioural metrics (CPU utilisation and network traffic) every 5 seconds and stores them on the device's SD card with the names of the running foreground applications as labels for the data points. As the DroidLight architecture consists of hybrid (host and server-based) services, DroidLight-Host sends the stored data periodically every 5 minutes to the remote DroidLight-Server (section 5). DroidLight-Server stores the data in its remote storage for model training. DroidLight-Host flushes the metrics data every time it communicates and sends the data to the DroidLight-Server. Hence, local storage stores only a maximum of the last 5 minutes of metrics data, which occupies only 4KB of SD card space on our *droid-device*.

6.2 Self-developed malware applications

To perform our experiments with malicious application behaviour we developed three malware applications: DroidDDoS, DroidThief, and DroidHijack, which mimic the behaviour of real malwares as detailed in Table 3. The typical functionalities that we coded for these malware applications are as follows:

DroidDDoS: This malware application continuously transmits hundreds or thousands of network packets per second to the remote web server. Hence, the application converts the device into a bot to perform a DDoS attack on the web server. As presented in Table 2 a small DDoS attack per device will require a total 1.5 million devices to perform a Mirai DDoS⁹ attack on a web server, whereas a medium and large attack will require 75,000 and 30,000 smartphone devices respectively. This malware closely mimics the following real malware from Table 3: Loicdos, Ksapp, SeaWeth, and Tascudap.

DroidThief: This malware application retrieves a file from the SD card and sends it to the remote web server (a MacBook running PHP web service) via http. The application runs in the background periodically every 10 seconds to perform this task. In Table 2 we categorise the attack size of the malware based on the size of the file that it steals. This malware closely mimics the following real malwares from Table 3: DroidDream, DroidKungFu, Geinimi, Droid-Deluxe, and GingerMaster.

DroidHijack: This malware continuously runs in the background and consumes device CPU up to 90-100%. This malware closely mimics the CoinKrypt malware from Table 3.

⁶<https://developer.android.com/studio/index.html>

⁷<http://commons.apache.org/proper/commons-math/>

⁸<http://www.cs.waikato.ac.nz/ml/weka/>

⁹<https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks>

Table 3: Summary of Targeted Malware

Malware Family	Year Introduced	Behavioural Details	Category
DroidDream, DroidKungFu, Geinimi	2011	Collects user's device information (IMEI, OS version, device version etc.) and sends them to remote server for further instruction to propagate spams or to participate in distributed denial-of-service (DDoS) attacks	Mobile Botnet
DroidDeluxe, GingerMaster	2011	Gets root access to the device, collects sensitive information from the device like login credentials, banking credentials, etc., and sends them to remote server	Rootkit
Loicdos, Ksapp, SeaWeth, Tascudap	2012	Uses Smartphone devices as bots to perform denial of service (DoS) or distributed denial-of-service (DDoS) attacks on a targeted server or website	Mobile Botnet
CoinKrypt	2014	Converts smartphone devices into digital currency-mining bots and lends the computing time to the network. This malware mostly consumes device resources like CPU, I/O, etc. and occasionally network	Currency-mining Bot
HummingBad, Gooligan	2016	Gets root access to the device in order to generate fraudulent advertising revenue by downloading other applications and clicking on application advertisements provided by legitimate ad networks	Rootkit

6.3 DroidLight Intrusion Detection Performance

In this section we explore the performance of DroidLight both in terms of detection accuracy and false alarms.

6.3.1 Training and Testing Data Preparation. We considered detection of the self-developed malwares, i.e. DroidDDoS, DroidThief, and DroidHijack on *droid-device* under three realistic conditions: (i) the device is idle (running no user activities), (ii) the user is chatting on WhatsApp, and (iii) the user is playing a video on YouTube. We experimented with all the three sizes of DroidDDoS and DroidThief attack i.e small, medium, and large as described in Table 2.

During the training phase, we ran DroidLight-Host in the *droid-device* to collect the metric (CPU utilisation and network traffic) data samples under each of the aforementioned conditions. DroidLight-Host sends the metrics data to the DroidLight-Server every 5 minutes. Each time DroidLight-Server receives new data from DroidLight-Host, it executes data pre-processing to prepare the training datasets, and then builds the OCC models (one for each condition) with the training datasets. Thus, the training continues until stable OCC models are built. The decision on when to consider that the model for a particular application is stable, i.e. when to stop the training for a particular application depends on the model false alarm rates. For example, if a trained model does not raise many false alarms for a significant period of time then we may declare that the model is accurate or stable. The number of model false alarms further depends on a number of factors such as: device/application usage, metric pattern for the application, outliers in the metrics data, etc. Hence, we need to take decisions on when to stop the training by evaluating model false alarm rates at runtime. This is possible only with the help of a dynamic decision making that considers the aforementioned factors affecting the model false alarms and evaluates the false alarm rates periodically at runtime. While we keep adoption of such approach as our future work, we

Table 4: DroidLight training data samples and timeliness results

Device status or application running in the foreground	Training data samples	Testing/Classification time (milliseconds)
Device Idle	40	113
WhatsApp	80	158
Youtube	100	232

ran some static experiments to decide on when the OCC models in our experiments are stable. Based on our static experiments, we selected different numbers of training data samples (one sample per minute) for different conditions, which are given in Table 4.

During the testing phase, DroidLight-Host continuously monitors metric (CPU utilisation and network traffic) data under each of the realistic conditions while one of the self-developed malware performs its malicious activities in the background. The monitoring continues separately for 10 minutes for each condition. During the monitoring, in every 1 minute, DroidLight-Host executes its intrusion detection algorithm in order to detect the malicious activities by analysing the previous one minute of monitored metrics. Thus, DroidLight detection occurs for 10 attack samples for each attack under each condition.

6.3.2 Accuracy of Detection. Table 5 presents the DroidLight intrusion detection results when detecting the self-developed malware attacks with different data pre-processing approaches in the DroidLight algorithm. The approaches are: (i) application-specific raw data analysis: this does not perform any pre-processing on the raw metrics data collected from specific application; rather it simply

Table 5: DroidLight intrusion detection results (per 10 attack samples) with different data pre-processing approaches

Data pre-processing approach	Device status or foreground application	DDoS attack (DroidDDoS)			Information theft attack (DroidThief)			Hijack attack (DroidHijack)
		small	medium	large	small	medium	large	
Application-specific raw data analysis	Device Idle	10	10	10	0	10	10	10
	WhatsApp	9	10	10	0	5	10	0
	YouTube	2	8	1	0	0	0	0
	Accuracy	70%	93.3%	70%	0%	50%	66.6%	33.3%
Device-wide probability distribution analysis	Device Idle	0	10	10	0	0	0	0
	WhatsApp	0	10	10	0	0	10	0
	YouTube	9	10	10	0	0	3	5
	Accuracy	30%	100%	100%	0%	0%	43.3%	50%
Application-specific probability distribution analysis (chosen by DroidLight)	Device Idle	10	10	10	0	10	10	10
	WhatsApp	10	10	10	0	10	10	10
	YouTube	6	9	8	0	0	8	10
	Accuracy	86.6%	96.6%	93.3%	0%	66.6%	93.3%	100%

Table 6: DroidLight false positive results (per 60 minutes) with different data pre-processing approaches

Device status or application running in the foreground	Application-specific raw data analysis	Device-wide probability distribution analysis	Application-specific probability distribution analysis (chosen by DroidLight)
Device Idle	3	0	2
WhatsApp	1	1	0
Youtube	3	0	1
Total	7	1	3

feeds the raw data to the training phase in the algorithm, (ii) device-wide probability distribution analysis: this performs probability distribution analysis on the metrics data collected from running different applications in the device, and (iii) application-specific probability distribution analysis: this performs probability distribution analysis on the metrics data collected from running specific applications and this is the approach we employed in the DroidLight algorithm. From the results in Table 5 we observe the following:

- (1) Overall accuracy is improved with the application-specific probability distribution analysis (chosen by DroidLight) as the data pre-processing approach.
- (2) DroidLight fails to detect small DroidThief attacks. This is not surprising as we have already mentioned that DroidLight is intended to detect only those malware that affect device resource utilisation significantly and make a deviation in the utilisation pattern. Small DroidThief attacks represent the behaviour of small information theft attacks, which steal device information (IMEI, OS version, device version, etc.), recent call logs, or contact list, etc., in order to send them to a remote server for further instruction to propagate spam or to participate in DDoS attacks. That means that although DroidLight fails to detect small information theft attacks, it can successfully detect attacks carried out as a result of small information theft attacks.

- (3) If we consider high intensive attacks, i.e. the attacks that affect CPU utilisation or network traffic significantly like DDoS medium and large attacks, DroidThief large attack, and DroidHijack attack, then DroidLight accuracy lies in the range 93.3% to 100% for all three realistic experimental conditions that we have considered.

The accuracy achieved by DroidLight in this experiment may vary for the same or different applications run by different smartphone users on different devices. The accuracy results simply show that training DroidLight with the normal behavioural metrics data collected from running benign applications or while the device is idle can achieve accuracy up to 100% in detecting malware attacks similar to real-world information theft, currency-mining bot, or DDoS attack on smartphone devices. To adopt DroidLight to other realistic smartphone usage conditions, the training period for DroidLight needs to be adjusted, which as we have discussed will require dynamic decision making on how long to run the training in our continuous training process.

6.3.3 False Alarms. In order to identify the false alarms from DroidLight-Host, we carried out 60 minutes of experiment with each of the three realistic conditions that we considered for evaluating the accuracy. Table 6 presents the results of the false positives. From the table we find that the data pre-processing approach

adopted by DroidLight has fewer false positives (3 in total) than the raw data analysis approach (7 in total). On the other hand, the device-wide analysis approach has only 1 false positive; however, this is outweighed by its low accuracy as observed in the accuracy results.

6.4 DroidLight Efficiency

In this section we measure the overhead of DroidLight on the device resources and the timeliness/detection latency of DroidLight.

6.4.1 Overhead on Device Resources. We measured the overhead of DroidLight by running a standard Android benchmarking application, named *Quadrant Standard Edition*¹⁰ on droid-device. We present the benchmark result in Table 7, which reports the performance of the device with and without running DroidLight. Higher values of the results indicate better performance and each value is an average of 10 executions of the benchmark application. From the table we see that the total performance overhead of DroidLight is 1.5%. This overhead is mainly caused by two of the DroidLight-Host modules: the *Behavioural Metrics Collector* module, which collects the CPU and network metrics every 5 seconds and the *Intrusion Detector* module, which runs the DroidLight algorithm (pre-processing of metrics data by performing probability distribution analysis and one class classification of the pre-processed data) every 1 minute on the device. The total overhead of DroidLight (1.5%) is similar to the total overhead of MADAM [20] (1.4%); however, DroidLight has less impact on memory (4.6%) and I/O (0.32%) as compared to MADAM (memory - 9.4% and I/O - 4%). MADAM claims to have less overhead in comparison with the other smartphone IDSs.

In addition to identifying the overhead of DroidLight on CPU, memory, and I/O, we tried to identify the overhead of DroidLight on device network traffic as well. This overhead is important to consider as DroidLight establishes HTTP communication with a remote server every 5 minutes in order to send and receive the metrics and the model data, respectively. In Figure 3 we present the pattern of the CPU utilisation and the network traffic for 10 minutes while the device is Idle and DroidLight-Host is running in the background. From the figure we observe that the pattern of the network traffic show spikes with around 5 minute intervals (minute 5 and 10). This is due to the HTTP communication that happens every 5 minutes. However, these spikes do not generate any false alarms as DroidLight deals with such spikes/outliers using probability distribution analysis. This is evident from the distribution graph in Figure 3, where the curves for network at minutes 5 and 10 are shifted towards the left, which is similar to the curves at other minutes in the graph and different from the curves generated by malicious behaviour as observed earlier in the paper. Moreover, the HTTP communication for the data transfer is required only for a period of time, until the OCC models for each active application in the smartphone are trained properly.

6.4.2 Timeliness of Detection. Table 4 presents the testing/classification time for each experimental condition, which is in milliseconds. However, as DroidLight-Host executes its detection algorithm every 1

Table 7: DroidLight performance overhead results

Resource	Without DroidLight	With DroidLight	Overhead
Total	3310	3261	1.5%
CPU	5111	5067	0.86%
Memory	3370	3215	4.6%
I/O	5216	5199	0.32%

Figure 3: Time-series and probability distribution graphs of CPU utilisation and network traffic when device is idle and DroidLight-host is running in the background

Table 8: Summary of features in existing lightweight IDSs

IDS name	Accuracy	Detection Time	Performance Overhead	Zero-day Detection
Andromaly [21]	6-100%	NA	CPU: 5.5% memory: 8.5%	NO
Drebin [3]	94%	10s to 1 minute	NA	NO
Patronus [24]	87%	NA	CPU: 0.9% memory: 8.0%	NO
MADAM [20]	96.9%	NA	CPU: 0.9% memory: 9.4%	NO
DroidLight	93-100%	5s to 1 minute	CPU: 0.86% memory: 4.6%	YES

minute, the timeliness/detection latency of DroidLight may fluctuate from milliseconds (100-250 milliseconds) to 1 minute. Similar to the accuracy results, both the timeliness and false positive results

¹⁰<https://play.google.com/store/apps/details?id=com.aurorasoftworks.quadrant.ui.standard>

may also vary depending on the training period of the OCC models; a model trained with a short period of training data may detect an intrusion quickly but may have high false positives, whereas a model trained with a long period of training data may require a longer time for detection but may have low false positives.

Table 8 summarises the features of the existing lightweight anomaly-based smartphone IDSs in order to showcase where DroidLight stands. It is important to note that the results presented in the table from different IDSs are collected under different experimental setups and test environments, and hence their comparison cannot be truly justified: rather the results give us an understanding of what these IDSs are capable of in terms of the stated features.

7 CONCLUSION

This paper proposes DroidLight, a lightweight smartphone IDS, which can detect zero-day malwares effectively and efficiently without significant overhead. We evaluated DroidLight performance under realistic smartphone usage conditions. The evaluation results showed that DroidLight can accurately detect high intensive (in terms of device resource usage) self-developed malwares which represent the behaviour of information theft, currency-mining bot, and DDoS attack.

In future, we aim to extend DroidLight's functionality by incorporating a decision making algorithm which can dynamically decide on how long to run DroidLight's model training for individual applications and for individual users on a given device. This will help to achieve a high rate of detection accuracy and low false positive rates for DroidLight under realistic smartphone usage condition.

ACKNOWLEDGMENTS

This work has received funding from the European Commission under the European Union's Seventh Framework Programme (grant agreement 610811 - CACTOS project).

REFERENCES

- [1] A. J. Alzahrani and A. A. Ghorbani. 2015. Real-time signature-based detection approach for SMS botnet. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)*. 157–164. <https://doi.org/10.1109/PST.2015.7232968>
- [2] K. Ariyapala, H. G. Do, H. N. Anh, W. K. Ng, and M. Conti. 2016. A Host and Network Based Intrusion Detection for Android Smartphones. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 849–854. <https://doi.org/10.1109/WAINA.2016.35>
- [3] Daniel Arp, Michael Spreitzenbarth, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and explainable detection of android malware in your pocket.
- [4] Shweta Bhandari, Rishabh Gupta, Vijay Laxmi, Manoj Singh Gaur, Akka Zemmari, and Maxim Anikeev. 2015. DRACO: DROID Analyst Combo an Android Malware Analysis Framework. In *Proceedings of the 8th International Conference on Security of Information and Networks (SIN '15)*. ACM, New York, NY, USA, 283–289. <https://doi.org/10.1145/2799979.2800003>
- [5] Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. 2008. Behavioral Detection of Malware on Mobile Handsets. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*. ACM, New York, NY, USA, 225–238. <https://doi.org/10.1145/1378600.1378626>
- [6] Mihai Christodorescu and Somesh Jha. 2003. Static Analysis of Executables to Detect Malicious Patterns. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12 (SSYM '03)*. USENIX Association, Berkeley, CA, USA, 12–12. <http://dl.acm.org/citation.cfm?id=1251353.1251365>
- [7] Dimitrios Damopoulos, Georgios Kambourakis, and Georgios Portokalidis. 2014. The Best of Both Worlds: A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones. In *Proceedings of the Seventh European Workshop on System Security (EuroSec '14)*. ACM, New York, NY, USA, Article 6, 6 pages. <https://doi.org/10.1145/2592791.2592797>
- [8] Dimitrios Damopoulos, Sofia A. Menesidou, Georgios Kambourakis, Maria Papadaki, Nathan Clarke, and Stefanos Gritzalis. 2012. Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers. *Security and Communication Networks* 5, 1 (2012), 3–14. <https://doi.org/10.1002/sec.341>
- [9] Hyo-Sik Ham and Mi-Jung Choi. 2013. Analysis of Android malware detection performance using machine learning classifiers. In *2013 International Conference on ICT Convergence (ICTC)*. 490–495. <https://doi.org/10.1109/ICTC.2013.6675404>
- [10] Kathryn Hempstalk, Eibe Frank, and Ian H. Witten. 2008. One-Class Classification by Combining Density and Class Probability Estimation. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I (ECML PKDD '08)*. Springer-Verlag, Berlin, Heidelberg, 505–519. https://doi.org/10.1007/978-3-540-87479-9_51
- [11] Shehroz S. Khan and Michael G. Madden. 2010. A Survey of Recent Trends in One Class Classification. In *Proceedings of the 20th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'09)*. Springer-Verlag, Berlin, Heidelberg, 188–197. <http://dl.acm.org/citation.cfm?id=1939047.1939070>
- [12] Hahnsang Kim, Joshua Smith, and Kang G. Shin. 2008. Detecting Energy-greedy Anomalies and Mobile Malware Variants. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*. ACM, New York, NY, USA, 239–252. <https://doi.org/10.1145/1378600.1378627>
- [13] K. Majeed. 2015. *Behaviour Based Anomaly Detection System for Smartphones Using Machine Learning Algorithm*. London Metropolitan University. <https://books.google.co.uk/books?id=cvAdswEACA>
- [14] J. Milosevic, A. Dittich, A. Ferrante, and M. Malek. 2014. A Resource-Optimized Approach to Efficient Early Detection of Mobile Malware. In *2014 Ninth International Conference on Availability, Reliability and Security*. 333–340. <https://doi.org/10.1109/ARES.2014.51>
- [15] J. Milosevic, A. Ferrante, and M. Malek. 2016. MalAware: Effective and Efficient Run-Time Mobile Malware Detector. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. 270–277. <https://doi.org/10.1109/DASC-PiCom-DataCom-CyberSciTech.2016.65>
- [16] J. Milosevic, A. Ferrante, and M. Malek. 2016. What does the memory say? Towards the most indicative features for efficient malware detection. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 759–764. <https://doi.org/10.1109/CCNC.2016.7444874>
- [17] Jelena Milosevic, Miroslaw Malek, and Alberto Ferrante. 2016. A Friend or a Foe? Detecting Malware using Memory and CPU Features. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - Volume 4: SECRIPT, (ICETE 2016)*. 73–84. <https://doi.org/10.5220/0005964200730084>
- [18] A. Rodríguez-Mota, P. J. Escamilla-Ambrosio, S. Morales-Ortega, M. Salinas-Rosales, and E. Aguirre-Anaya. 2016. Towards a 2-hybrid Android malware detection test framework. In *2016 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. 54–61. <https://doi.org/10.1109/CONIELECOMP.2016.7438552>
- [19] J. Sahs and L. Khan. 2012. A Machine Learning Approach to Android Malware Detection. In *2012 European Intelligence and Security Informatics Conference*. 141–147. <https://doi.org/10.1109/EISIC.2012.34>
- [20] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli. 2016. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing* PP, 99 (2016), 1–1. <https://doi.org/10.1109/TDSC.2016.2536605>
- [21] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. 2012. "Andromaly": A Behavioral Malware Detection Framework for Android Devices. *J. Intell. Inf. Syst.* 38, 1 (Feb. 2012), 161–190. <https://doi.org/10.1007/s10844-010-0148-x>
- [22] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici. 2014. Mobile malware detection through analysis of deviations in application network behavior. *Computers and Security* 43 (2014), 1 – 18. <https://doi.org/10.1016/j.cose.2014.02.009>
- [23] B. Shrestha, D. Ma, Y. Zhu, H. Li, and N. Saxena. 2015. Tap-Wave-Rub: Lightweight Human Interaction Approach to Curb Emerging Smartphone Malware. *IEEE Transactions on Information Forensics and Security* 10, 11 (Nov 2015), 2270–2283. <https://doi.org/10.1109/TIFS.2015.2436364>
- [24] Mingshen Sun, Min Zheng, John C. S. Lui, and Xuxian Jiang. 2014. Design and Implementation of an Android Host-based Intrusion Prevention System. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*. ACM, New York, NY, USA, 226–235. <https://doi.org/10.1145/2664243.2664245>
- [25] Xiaolei Wang, Yuexiang Yang, and Yingzhi Zeng. 2015. Accurate mobile malware detection and classification in the cloud. *SpringerPlus* 4, 1 (2015), 583. <https://doi.org/10.1186/s40064-015-1356-1>
- [26] Liang Xie, Xinwen Zhang, Jean-Pierre Seifert, and Sencun Zhu. 2010. pBMDs: A Behavior-based Malware Detection System for Cellphone Devices. In *Proceedings of the Third ACM Conference on Wireless Network Security (WiSec '10)*. ACM, New York, NY, USA, 37–48. <https://doi.org/10.1145/1741866.1741874>
- [27] M. Zheng, M. Sun, and J. C. S. Lui. 2013. Droid Analytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 163–171. <https://doi.org/10.1109/TrustCom.2013.25>