# ASAINT: a spy App identification system based on network traffic

**Conference Paper** · August 2020

DOI: 10.1145/3407023.3407076



**3 authors:**

Mauro Conti
University of Padova
**748** PUBLICATIONS **21,398** CITATIONS
SEE PROFILE

Giulio Rigoni
Sapienza University of Rome
**12** PUBLICATIONS **86** CITATIONS
SEE PROFILE

Flavio Toffalini
Singapore University of Technology and Design
**23** PUBLICATIONS **353** CITATIONS
SEE PROFILE

# ASAINT: A Spy App Identification System based on Network Traffic

Mauro Conti
University of Padua, Italy
conti@math.unipd.it

Giulio Rigoni*†‡
University of Firenze, Italy
giulio.rigoni@unifi.it

Flavio Toffalini
Singapore University of Technology and Design, Singapore
flavio_toffalini@mymail.sutd.edu.sg

## ABSTRACT

*Spy app* is a class of malware for mobile devices that allows an adversary to steal sensitive information. Detecting *spy apps* is challenging because they do not rely on classic malware techniques, for instance, they use standard services to store stolen data, and do not perform privileges escalation on the victim phone. Thus, their behavior is generally closer to the benign apps and poses new challenges for their detection.

In this paper, we propose ASAINT: A *Spy App* Identification System based on Network Traffic. To the best of our knowledge, ASAINT is the first system capable of detecting *spy apps* in a network without any physical or software control of the victim mobile device. Core of our approach is a wide range of non-intrusive network detection methods designed by studying several popular *spy apps*.

We test ASAINT on a self-collected dataset containing network traffic from both spy and benign applications, either on Android and iOS. Our result is an F1-score of 0.85 on average, that confirms the effectiveness of ASAINT. Moreover, our analysis provides a methodological classification of the exfiltration strategies used by *spy apps* in different operating systems. In sum, our work gives new and practical insights about the detection of modern *spy apps*, paving the way for future research in detecting this class of malware.

## CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; • **Networks → Network experimentation**; • **Computing methodologies** → *Machine learning*; Cross-validation.

## KEYWORDS

machine learning, detection system, networking analysis, mobile

**ACM Reference Format:**
Mauro Conti, Giulio Rigoni, and Flavio Toffalini. 2022. ASAINT: A Spy App Identification System based on Network Traffic. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. 10.1145/nnnnnnn.nnnnnnn

*University of Perugia, Italy
†University of Padua, Italy
‡Corresponding author

*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

*Spy app* is a threat that affects mobile devices and that allows an adversary to exfiltrate sensitive information.[1] Compared to other classes of mobile malware, such as such as virus, trojan, spyware [27], *spy apps* received relative less attentions [3].

Peculiarity of *spy apps* is the ability of pursuing their goal without relying on classic exploitation techniques. Thus, they behaviorally appear close to legit and harmless apps. In fact, modern solutions are unable to detected them [28]. *Spy apps* elusive behavior, combined with an easy deployment from standard mobile markets (*e.g.,* Google Play [7]) or through installation file (*e.g.,* APK), alerted big antivirus player about the seriousness of such threats [10, 11].

To understand the success of *spy apps*, we analyzing the mitigation strategies of modern mobile devices in terms of privacy, that we classify in three macro-area: (i) operating system privacy restrictions, (ii) run-time monitor agents, (iii) network analysis. Privacy restriction is a common feature of smartphone operating systems (*e.g.,* Android and iOS) that limits apps' actions. However, previous works showed practical multi-stage attacks able to bypass these policies [31]. In addition, malicious apps may mislead users to grant dangerous permissions (*e.g.,* a casual game that asks for a geographical position). To strengthen privacy policies, we can validate run-time mobile apps behavior [16, 23, 29, 32] through built-in monitoring agents [22]. Or else, we can rely on network detection to seek malware traces [30]. Unfortunately, current network analysis works focus only on *spy apps* toy examples. On the contrary, to the best of our knowledge, we are the first who studies real *spy apps* network behavior.

In this work, we propose ASAINT: A Spy App Identification System based on Network Traffic. The main contribution of ASAINT is the ability of detecting information leakage by solely analyzing the network traffic. Core to our approach, an exhaustive study of machine learning (ML) algorithms that allows us to identify the best classification strategy for *spy apps*. Furthermore, we stressed our classifiers against legit apps traffic to study the robustness of ASAINT. Our solution overcomes previous works because it recognizes real *spy apps* without installing invasive monitors or re-thinking privacy policies. We do not intend our work as merely academic. On the contrary, we designed ASAINT as a strong solution able to increase the privacy guarantees in those contexts where is not possible to physically monitor devices, *e.g.,* a corporate internal network that contains either employees' and guests' mobile devices.

To validate ASAINT, we redact a list of popular *spy apps* from the market. As a result, we obtained an average F1-score of 0.85, with

[1] We refer interchangeably at smartphone, tablet, and mobile devices

only one class at 0.58, another one at 0.76, and all the others over 0.82. This means that, overall, we are able to detect the data exfiltrate by *spy apps* with good confidence. In addition to the classification performances, we observed a detection time that ranges from 0.1s to 0.002s. This enables ASAINT to be deployed in a real environment.

***Contribution.*** The contribution of this paper are:

- We propose ASAINT: a Spy App Identification System based on Network Traffic for detecting information leak solely through network inspection.
- We provide an investigation and classification of real *spy apps* for Android and iOS.
- We discuss insight about different exfiltration techniques used to implement *spy apps*.
- We measure ASAINT performances in terms of classification quality and detection speed.

## 2 RELATED WORKS

Malware detection, especially for spyware, is an important research topic on mobile device panorama. In the literature, either researchers and companies spent a lot of effort on designing countermeasures. We summarize their results in two main categories: (i) embedded applications (Section 2.1), (ii) external mechanisms (Section 2.2). ASAINT shares similarities with either classes that we discuss in the rest of the section.

### 2.1 Embedded applications

Embedded applications refer to any system that needs to be installed on the target device for the full functionality.

In the work [30], the authors studied how spyware works by creating one their self, called *Chameleon*; with the knowledge acquired from this step, they designed an application able to find *Chameleon*, called *DriodSmartFuzzer*. The system basically scans the privileges granted to all installed applications and the internet traffic to identify potential *spy apps*. Differently, Carlsson et al. [19] describe *KAUDroid*, which consists of an application that collects permission usage on phones. Information about permissions are stored, elaborated, and presented to the public through a web user interface.
Ali-Gombe et al. [17] propose *Aspectdroid* which is an application-level system designed to investigate Android applications for possible unwanted activities.

All the previous works rely on a dedicated monitoring agent on the target phone. Our approach, instead, does not consider the installation of a dedicated application. ASAINT can detect a *spy app* by observing network communications. Our approach can, therefore, monitor multiple mobile devices by using a WiFi connection and without any intervention on the target device.

### 2.2 External mechanisms

External mechanisms are systems that work outside the target device but interacting with it through some means of communication.

Malik et al. [25] introduce *CREDROID*, a system that can identify malicious applications based on: (i) their Domain Name Server (DNS) queries, (ii) the data it transmits to a remote server, by performing the in-depth analysis of network traffic logs in offline mode. Even though they leverage on network traffic, we achieve better performances by adopting machine learning algorithms. Moreover, they deal with a class of *spy apps* which differs from us.

Anshul et al. proposed a network traffic analyzer for malicious activity [18]. Their detector mainly focuses on Android malware that relies on background connections. On the contrary, our work

faces a broader scenario that encompasses either Android an iOS. Moreover, as we will describe in Section 5.2, *syp apps* employs different exfiltration strategies that were not considered in Anshul's work.

Taylor et al. [20, 33] focus on the identification of user activities on a target device, while our work moves the attention toward network activities un-related with user applications. Moreover, they collect and analyze multiple flows of data to identify with more precision the user activity, instead we focus on a single flow to minimize the *spy app* exfiltration. Thus, they face a different scenario.

## 3 BACKGROUND AND PRELIMINARIES

In this section, we introduce the *spy app* definition that we adopted (Section 3.1) and the properties of our threat model (Section 3.2).

### 3.1 Definition of a Spy App

The term "*spy app*" can be used in different contests and with different meanings. In this paper, we consider the definition introduced by Kaspersky in [3]. Those applications are defined as *not-a-virus* and they are installed in a mobile device without abusing any flow, as malware/spyware does. This definition traces a well-defined line between classic malware/spyware and *spy apps* detection. Differently from classic malicious software, *spy apps* may be considered benign [28] at a first inspection, and they are considered legal if installed by the device owner.

Usually, *spy apps* can be installed by any person who had physical access to the mobile device for enough time, *e.g.,* a developer, an employer, a wife, a parent. The reasons for installing a *spy app* may be various, *e.g.,* for monitoring a child, excessive control or spying the work of an employee. A possible scenario is an employer who presents free phone as gifts to his employees for monitoring their activity. Even though these apps cannot be considered traditional malware, they can steal private data by using HSO (Hidden Sensitive Operations), for example upon sending or receiving text messages.

### 3.2 Threat Model

We list the threat model assumptions that we consider for ASAINT.

***Assumptions with respect to the mobile device:***

- The victim devices does not contain any specific monitor installed (*e.g.,* antivirus).
- The *spy app* relies on a WiFi connection to exfiltrate data. This assumption comes from the fact that:
  (i) The mobile device's owner does not know of being spied, so he does not disconnect the WiFi intentionally. Even if the WiFi were disconnected, the *spy app* could detect the event and alters its behavior.
  (ii) WiFi connection gives more power to the *spy app* in terms of upload capability since it does not consume mobile traffic (as most mobile connections do) thus being more stealthy,
  (iii) Mobile apps usually rely on WiFi connection if both, WiFi and mobile, are available.

***Assumptions with respect to the user:***

- The user is not aware of being spied.
- The user is a casual entity of a network (*e.g.,* a guest who visits a company).

## 4 OUR PROPOSAL: ASAINT

In this Section, we describe an overview of the transformation process from raw traffic to the final samples and the infrastructure adopted (Section 4.1, Section 4.2 and Section 4.3). Then, we illustrate the network model we employed (Section 4.4) and the features extracted (Section 4.5). In the end, we describe the adopted machine learning strategies (Section 4.6).

### 4.1 Overview

The aim of ASAINT is to detect *spy apps* in a corporate network. Those apps follow the threat model described in Section 3.2. Roughly speaking, we aim at identifying compromised devices that use the network for exfiltrating private data without user interaction. Moreover, we assume we don't have control over such devices (*i.e.,* we cannot install/remove applications). We opted for this solution for two reasons: (i) any guest who visits occasionally a company might introduce a new device, (ii) a malware may bypass an in-device monitoring agent.

Keeping this in mind, we modeled ASAINT to infer *spy app* activities by only observing TCP/IP network behaviors without any further monitor installed in the device.

### 4.2 Transformation Process

To infer spy apps activities, we apply a network transformation process that builds machine learning samples starting from raw network traffic data. Figure 1 shows the procedure adopted for ASAINT. At first, we collect raw network traffic from a controlled infrastructure (Section 4.3). In our implementation, we organize the traffic into *pcap* files, but ASAINT can be also deployed to perform an on-line analysis, as we discuss later (Section 5.5). After recording the traffic, we process the raw data to create objects called *flows* (Section 4.4). Then, we process the flows to produce a dataset of samples (Section 4.5). Finally, we build a machine learning model on top of our dataset to classify the samples (Section 4.6).

### 4.3 Network Analysis Infrastructure

The infrastructure simulates a simple corporate network which is composed by a gateway, an access point, and a set of devices connected to the access point. Those devices might be compromised or healthy, but all of them use the internal network for communicating. In our network, the devices are handled by the access point that

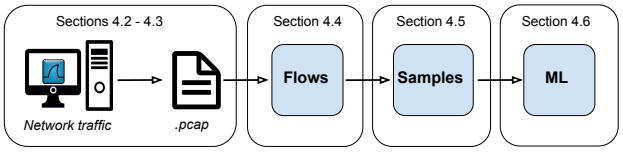



**Figure 1: Transformation process from raw traffic to classified element.**

routes the traffic to the gateway. The latter manages the communication between the local network and the Internet, moreover, the gateway employs a sniffing software to record the network traffic. As a sniffing software, we opted for Wireshark [1], which is a well know network analyzer. This infrastructure allows us to inspect communications toward and from the Internet, however, the traffic recorded is generally encrypted (*e.g.,* HTTS, FPTS). Therefore, we cannot just inspect the traffic to identify its nature. Furthermore, since we do not know a priori which server is going to be contacted, we cannot rely on destination IP addresses as well.

Notice that, the local network can contemporaneously contain healthy and compromised devices, and both communicate with remote servers for different purposes. Since both communications are encrypted, it is crucial to discern between these two behaviors.

## 4.4 Flow Representation

Taking inspiration from Taylor et al. [34], we model the network traffic as *flows*. Intuitively, a *flow* is any stream of packets between two given IPs. Note that we include in the same *flow* all the packets exchanged between two given IPs, regardless of the port used or their direction. That is, a *flow* may contain packets with different source and destination port, but those packets must be exchanged between the same IP pair. Same for the direction, a *flow* may contain both ongoing and incoming packets as long as those packets are exchanged between the same IP pair. This choice comes from this observation: an application may communicate with a server by using different services (*i.e.,* different destination ports) to perform an action. Therefore, to fully catch an application action (*e.g.,* uploading a file), we must accumulate in a single object (*i.e.,* the *flow*) all the traffic generated for performing that action.

For extracting *flows*, we rely on the concept of *burst* [24]. Intuitively, a *burst* is a network traffic partition which is identified by a high quantity of data transferred over a delta time with no relevant pause in the transmission. To identify a burst, we borrowed a simple heuristic from [34]: if the inter-arrival time between two consecutive packets is larger than a fixed delta time, then we consider the packets belonging to two bursts.

To sum up, the algorithm we used to identify a *flow* is as follow:

(1) we extract *bursts* from the network traffic by grouping packets with a time gap lower than one second (as done in [34]).
(2) we split each *burst* in *flows* by grouping all packets exchanged between two given IP address[2].

Since a *flow* represents an application action, our goal is now to discern between those *flows* which are generated by *spy apps* and legitimate apps. To achieve it, we will employ a machine learning approach. Starting from a well-defined set of *flows*, we first make a dataset by extracting several features from each *flow* (Section 4.5). Then, we apply different machine learning algorithms to identify the better strategy (Section 4.6).

## 4.5 Samples and Features Extraction

Our dataset is composed of samples which are made from the analysis of *flows* (*i.e.,* a sample for each *flow*). Since the samples are generated by different apps and actions, we define multiple labels, shown in Table 1, each label represents an action we aim to identify. We also define a generic label for the remaining traffic. Each sample can belong only to one of those labels. As we shall discuss in Section 5, we collect all the traffic in a controlled environment; therefore, we manually labeled each sample.

**Table 1: Labels and samples constructing the dataset.**

| Label | Description | # of samples |
|---|---|---|
| cImg | Cerberus spy app for Img shot | 65 |
| cSms | Cerberus spy app for SMS log | 97 |
| mSpy | mSpy spy app back-up | 180 |
| tSpy | The TruthSpy spy app back-up | 279 |
| | *Total spy app samples* | **621** |
| DB | Dropbox photo upload | 99 |
| GF | Google Foto photo upload | 184 |
| | *Total legitimate app samples* | **283** |
| GEN | Generic traffic/noise | 2461 |
| | *Total* | **3365** |

As features list, we took inspiration from [33]. For each sample, we define 39 different features. In a nutshell, these features are statistical measurements such as number of packets, amount of data, average delta time between packets, skews and kurtosis on packet dimension, 90th percentile on packets dimension. We calculate a set of features for each *flow* by considering three cases: (i) considering only ongoing packets, (ii) considering only incoming packets, (iii) and considering both; The idea is that an application will generate similar *flows* that should share similar properties. Therefore, we aim to catch those similarities through statistical measurements of incoming and ongoing packets.

## 4.6 Machine Learning

In this work, we employ supervised machine learning algorithms to identify those *flows* that belong to *spy apps*.

Our intention is to distinguish which (and implicitly if) a device is being watched by a *spy app*. Moreover, we aim at identifying which type of data the spy app is sending (*e.g.,* photo or text messages). In this section, we explain the machine learning challenges introduced by ASAINT.

***Data Preprocessing.*** Due to the unbalanced number of samples between *spy apps* and general traffic, we opt for an oversampling algorithm to adjust the class distribution. In this work, we opt for the SMOTE (Synthetic Minority Over-sampling Technique) algorithm from the imbalanced-learn API of sci-kit library [26].

After these operations, we obtain an enriched dataset.

***Machine Learning Algorithm.*** We test three fast and low-complexity ML algorithms, for a future run-time detection, on the enriched dataset: Random Forest (RM), Logistic Regression (LR) and k-NN, and we evaluate which of them provides better performances.

Having few samples might bring over-fitting problems caused by an imbalanced dataset. We cope with this issue by using a nested

[2]Since we group both ongoing and incoming packets. We group in a single flow all packets that have the same IP pair as source and destination address, *i.e.,* ($IP_1$,$IP_2$) and ($IP_2$,$IP_1$)

cross-validation (CV) as explained in [13]. Figure 2 shows the nested CV principle, the outer cross-validation divides the dataset in training and test set, applying oversampling only to the training set. Instead, the inner cross-validation does a fine-tuning for the hyper-parameters of the ML algorithm solely on the over-sampled training set. Using this technique, with every outer CV iteration, the best ML model from the inner CV is applied to the outer test set. The outcomes are the best results obtained from the outer CV, each computed with the best hyper-parameters configuration from the inner CV.

# 5 EVALUATION

In our evaluation, we explore the commercial word of *spy app* and we tune our experiments in order to answer the three specific research questions:

- **RQ1** What are the different mechanisms adopted by *spy apps*?
- **RQ2** Is there any different *spy app* strategy between Android and iPhone?
- **RQ3** Is it possible to detect *spy apps* activities by using network analysis?

The rest of the section is organized as follows. First, we describe a qualitative analysis of *spy apps* behaviors (Section 5.1) and the use cases chosen (Section 5.2). Then, we describe the data collection methodology adopted (Section 5.3) and the database generated (Section 5.4). Finally, the performance of ASAINT (Section 5.5) and a comparison between Android and iOS (Section 5.6).

## 5.1 Exfiltration Techniques

Designing ASAINT, we study and select *spy apps* for two mobile operating systems: Android [2] and iOS [9]. We pick three *spy apps* for Android and one for iOS according to their popularity and their exfiltration strategies.

As first, we perform a qualitative analysis of these applications to understand their behaviors. After this analysis, we noticed that the *spy apps* of Android significantly differ from the ones of iOS. We found out that iOS *spy apps* do not actually exfiltrate data from the network, but instead relies on iCloud services [8]. Therefore, ASAINT cannot be used to detect such apps (details in Section 5.6). On the other hand, Android *spy apps* interact with the network. Therefore, we apply ASAINT to those cases.

In Android, the three *spy apps* we study rely on two typical programming patterns: *polling* and *push notification*. These two strategies upload sensitive data over the network in different ways.

***Polling***. In this strategy, the *spy app* periodically uploads private data (*e.g.,* image or messages) from the phone into a remote server. The *spy apps* analyzed provide a Web portal where the attacker can re-arranged the scheduling.

***Push Notification***. In this strategy, through push notification, the *spy app* uploads data on-demand. The attacker can use a dashboard to query the device that will perform an action. For instance, the attacker can remotely shot a photo and send it through email, or else the phone can upload the call logs.

This analysis allows us to answer to **RQ1**: in Android, the commercial *spy apps* rely on classic programming patterns such as *polling* and *push notification*.

## 5.2 Use Cases

As use cases for ASAINT, we choose one *spy app* based on push notification (*i.e., Cerberus* [4]) and two *spy apps* based on polling (*i.e., mSpy* [12] and *TheTruthSpy* [14]). Then, we design the following tests for collecting data and create the dataset.

***Cerberus***. Since Cerberus employs a push notification mechanism, we manage to record network traffic specific to two different actions: (i) SMS log file, (ii) instant photo. With the first action, we ask to retrieve all the SMS sent and received by the phone, while the second action allows us to silently take a photo from the phone. In particular, we collected (i) 30 instances of SMS log and (ii) 10 instances of instant photo.

***mSpy***. mSpy employs a polling mechanism. Moreover, it does not let the user of the *spy app* choose a single action. It, basically, creates and uploads a back-up of the entire phone. Therefore, we re-arranged the tests as follow: (i) 5 recordings of 30 minutes of network traffic, with a polling rate of 5 min; this means that the application will retrieve the back-up from the phone every 5 min. (ii) 5 recordings of 30 minutes of network traffic, with a polling rate of 10 min; this means that the application will retrieve the back-up from the phone every 10 min.

***TheTruthSpy***. Similar to mSpy, TheTruthSpy spy app uses a polling mechanism and so we did: (i) 10 recordings of 30 minutes of network traffic, with a polling rate of 5 min; this means that the application will retrieve the back-up from the phone every 5 min.

We applied this approach to three different device configurations: (i) Samsung Galaxy Nexus with Android 4.3, (ii) Samsung Galaxy S5 with Android 4.4.2, and (iii) Samsung Galaxy S5 with Android 6.0.1.

Common mobile apps (*e.g.,* Facebook, Instagram, a Web browser) mainly download data instead of uploading, while *spy apps* mainly upload data. This fact let us wonder whether *spy apps*' network behavior might be confused with other apps that mainly upload data. Therefore, we design some special use case to investigate possible false positives. In particular, we record network traffic of common file-sharing apps: *Google Foto* [6] [3] and *Dropbox* [5].

***Google Foto and Dropbox***. We select these two common applications and test for each one: 50 instances of uploading a photo. We label these two use cases separately because we are interested in understanding whether they can fake our detection.

***Generic Traffic***. During the aforementioned tests, we record also casual and random network traffic generated from the phone (*e.g.,* traffic from background applications and system synchronization) and Web-browser traffic. Since we consider this network traffic as physiological, we label the relative samples as *GEN* (Generic traffic).

[3] We refer to the italian version of Google Photos

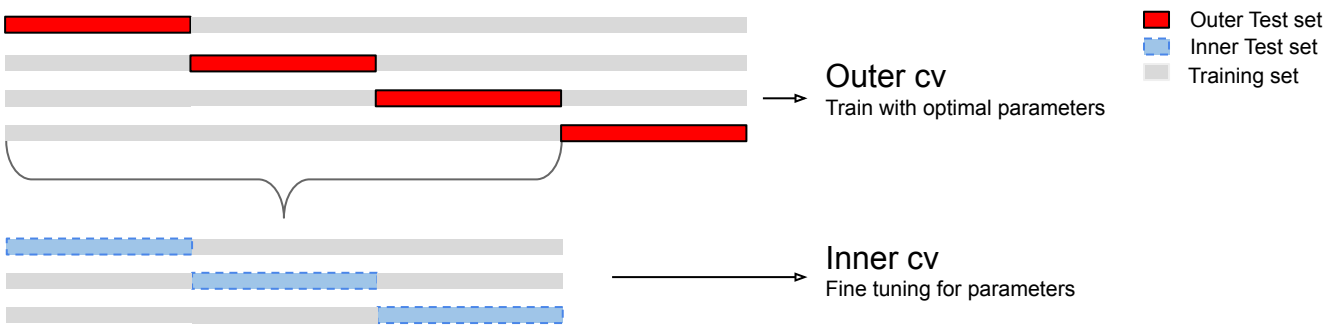



**Figure 2: Nested cross-validation process. The whole dataset is split in chunks for training, letting one out every iteration as test set. In each iteration, the training set is further split in training and test set as represented, repeating the same process for fine-tuning the algorithm.**

**Table 2: Description of every tests for each applications.**

| App Type | Label | # of tests | Test description |
|---|---|---|---|
| Spy | cImg | 10 | Instances of instant photo |
| Spy | cSms | 30 | Instances of SMS log |
| Spy | mSpy | 5 | 5 min. polling rate, 30 min. rec. |
| Spy | mSpy | 5 | 10 min. polling rate, 30 min. rec. |
| Spy | tSpy | 10 | 10 min. polling rate, 30 min rec. |
| Legitimate | DB | 50 | Instances of photo upload |
| Legitimate | GF | 50 | Instances of photo upload |

## 5.3 Data Collection Methodology

To collect data from the use cases previously listed, we use the network infrastructure described in Section 4. We connect the phones to a controlled access point and we sniff the network traffic at a controlled gateway. It is important to notice that only one smartphone was connected to the WiFi access point at a time, granting clean readings. We describe the whole dataset in Table 2. After the tests, we manually inspected the traffic for labeling the *flows*. This methodology is useful to reverse *spy app* behavior and spot significant peculiarity otherwise hidden. For instance, we identify *spy app* exfiltration strategies for iOS devices (see Section 5.6).

To distinguish between each *flow*, we first list the IP addresses, then, we use a so-called *whois* service [15] (based on WhoIs protocol [21]) to understand which app generated the traffic. For some apps, these operations are quite straightforward (*e.g.,* Cerberus). In other cases, such as mSpy, we analyze whether the *burst* generated toward the same remote IP address match the polling schedule set in the portal. This approach was fundamental since those *spy apps* use VPS stored in clusters (*e.g.,* AWS). Therefore, solely server IP address is not enough to recognize the nature of the service.

## 5.4 Dataset Specification

From our experiments (Section 5.2), we gather 245, 7 MB of data for $\sim$ 73, 33 hours of recorded internet traffic. We, then, convert the raw traffic in a dataset that is summarized in Table 1. Each sample of our dataset represents a *flow*.

The table recalls the labels described in Section 5.2. Since we aim to distinguish the actions for each *spy app*, we defined a label for each app. For Cerberus, we manage to label single actions. We also define labels for some legitimate apps and the generic traffic.

The number of samples generated is 621 for all *spy apps*, 283 for legitimate apps (*i.e.,* Google Foto and Dropbox) and 2461 for the general traffic.

## 5.5 ASAINT Performances

Following the procedures and methodologies described in Section 4.6, we obtain the results in Figure 3. To evaluate ASAINT performances, we choose F1-measure because of the unbalanced nature of our dataset. Figure 3a illustrates the F1-score distributions obtained by using three machine learning algorithms. The best F1-score values derive from the training with the RF algorithm. It is also possible to observe similar results in Figure 3b that shows the mean F1-score for the three algorithms. Even in this case, we obtain the best result with RF, an F1-score of 0.857.

The distributions in Figure 3a underline a more compact distribution for RF. Furthermore, some of its results out-stands the average reaching 0.92. This indicates that RF is more stable with respect to k-NN and LR. Overall, RF algorithm seems to perform better than the others since its worst value is better than the best results from k-NN and LR.

For the tests, we use a 10-fold nested cross-validation in which the inner 10-fold cross-validation search through some hyper-parameters such as the depth of the tree (for RF), the dimension of the neighborhood (for k-NN) and the C parameter (for LR). Also, the best k-features are selected during the process of fine-tuning. Figure 3b depicts the mean F1-score values from 10 different outcomes of the 10-fold cross-validation process.

Figure 3c shows an example of a confusion matrix obtained predicting the outcomes with our approach based on RF. This representation gives a glance at the overall performances. A dark-blue color means high numbers, while a light color means numbers close to zero. Furthermore, confusion matrix is used to represent true/false

positive and true/false negative just by looking at each entry on the matrix. In the image, we notice how the cImg labels are the worst classified. More precisely, cImg are heavily miss-classified as GEN. However, we also observe that the number of miss-classification inter-labels, excluding GEN, are really low. This behavior can be explained by the fact that generic applications upload/download images (*e.g.,* a post on Facebook) in a similar manner than Cerberus. Therefore, the relative traffic is similar to cImg samples. However, according to the results obtained, we can assert that it is possible to detect *spy apps*' activities in a network. This answers to **RQ3**.

***Detection Time.*** Figure 3d shows the average training and classification times. We perform the experiments on a general-purpose machine equipped with 8GB of memory and an Intel i5. The picture shows that RF is the slower algorithm (0.1s), while the fastest is LR (0.002s). Even though RF expresses the slowest classification time, the performance is fast enough to alarm and block ongoing attacks. The algorithms show a different pattern for the training time. In this case, LR requires more than 400s to compose the model, while the k-NN needs around 15s. In a scenario where a defender needs to update ASAINT as soon as possible, we can still adopt RF that requires a relative fast training phase (34s).

### 5.6 Differences between Operating Systems

Initially, our intention was to test the most used Operating System for mobile devices: iOS and Android. Unfortunately, the systems are too different and the *spy app* we choose works in completely distinct ways. For example for using mSpy on Android, is necessary to install an application on the target device, and through this application, the mSpy system can retrieve information. Instead, on a not-jailbreak iOS device, there is no application to be installed. Moreover, we collect more evidence on the matter using two simple strategies:

***Sideways content modification.*** Accessing iCloud from a desktop environment, we made some changes and added new content to the cloud store. The logical outcome is that the *spy app* should not be able to retrieve these changes and new contents, because they are not on the spied device. Instead, we observe new elements updated from the *spy app* dashboard.

***Recordings.*** We record the target device, in idle mode, for a whole day with and without setting up the *spy app* for the device. The two network traces were identical, hence no sign of any *spy app* presence.

During the configuration procedure, mSpy asks for iCloud information *i.e.,* userID and password; once a day, the target mobile phone creates a back-up and stores it on iCloud; at that moment we assume that mSpy will download the back-up from the iCloud server using the credentials provided at the configuration time. This system makes our testing solution completely inappropriate and incompatible with the Android testing scenario. Furthermore, to our knowledge, all the *spy apps* designed for iOS, rely on this same mechanism.

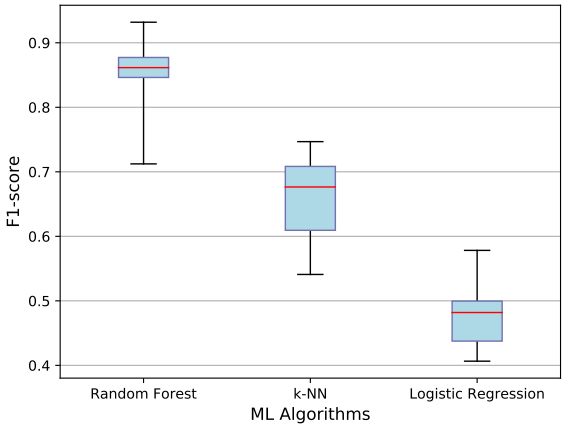
This analysis enables us to answer to **RQ2**: Android's *spy apps* generate network traffic for exfiltrating data, while iOS's *spy apps* interact with iCloud. Moreover, according to our results, iOS *spy apps* cannot be detected through network analysis nor built-in monitors.

## 6 DISCUSSION AND CONCLUSIONS

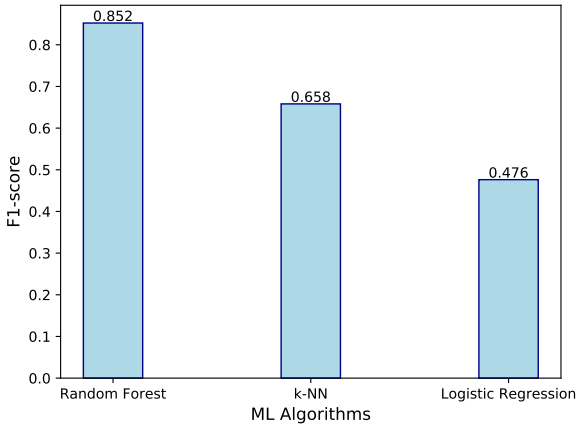
There is a growing number of threats for mobile devices, one of them are *spy apps*. In particular, this kind of malware can steal personal data using HSO remaining undetected. We propose a wide-range non-intrusive method for the identification of *spy apps* at work in a controlled environment using a WiFi internet connection as a means of communication. Through the use of Machine Learning algorithms, we achieve an average F1-score of over 0.85; we were able to not only identify the *spy app* but also the specific action carried out. Moreover, we test our approach on real *spy apps* that are available on the market, and against benign applications with similar behavior like Google Play and Dropbox. Finally, we show that ASAINT provides fast detection and thus it is suitable for run-time detection systems. In the future, we will focus on high-level tuning for even better results, and we will pose stricter constraints on the environment (*e.g.,* different connection type).

## REFERENCES

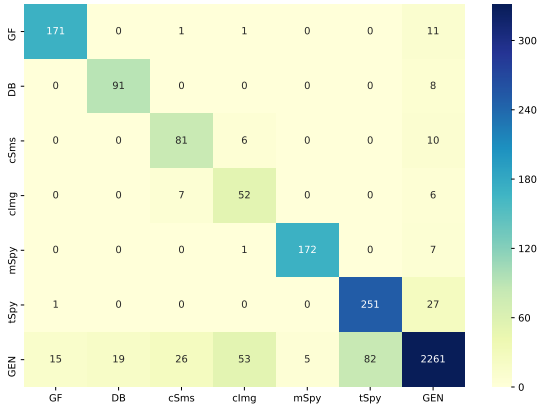

[1] Wireshark, 1998. Last access Oct 2018.
[2] Android, 2018. Last access Dic 18 2018.
[3] Android commercial spyware, 2018. Last access Dic 18 2018.
[4] Cerberus, 2018. Last access Dic 18 2018.
[5] Dropbox, 2018. Last access Dic 18 2018.
[6] Google foto, 2018. Last access Dic 18 2018.
[7] Google play, 2018. Last access Dic 18 2018.
[8] icloud, 2018. Last access Dic 18 2018.
[9] ios, 2018. Last access Dic 18 2018.
[10] Mcafee labs threats report, 2018. Last access Dic 18 2018.
[11] Mobile malware evolution 2017, 2018. Last access Dic 18 2018.
[12] mspy, 2018. Last access Dic 18 2018.
[13] Nested versus non-nested cross-validatio, 2018. Last access Oct 2018.
[14] Thetruthspy, 2018. Last access Dic 18 2018.
[15] Whois, 2018. Last access Dic 18 2018.
[16] Afridi, M. W., Ali, T., Alghamdi, T., Ali, T., and Yasar, M. Android Application Behavioral Analysis through Intent Monitoring. In *Digital Forensic and Security (ISDFS), 2018 6th International Symposium on*, IEEE, pp. 1–8.
[17] Ali-Gombe, A., Ahmed, I., Richard III, G. G., and Roussev, V. Aspectdroid: Android App Analysis System. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (2016), ACM, pp. 145–147.
[18] Arora, A., Garg, S., and Peddoju, S. K. Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies* (2014), IEEE, pp. 66–71.
[19] Carlsson, A., Pedersen, C., Persson, F., and Söderlund, G. *KAUDroid: A Tool that Will Spy on Applications and How They Spy on Their Users.* Karlstads universitet, 2018.
[20] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Can't You Hear Me Knocking: Identification of User Actions on Android Apps Via Traffic Analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (2015), ACM, pp. 297–304.
[21] Daigle, L. WHOIS Protocol Specification. Tech. rep., 2004.
[22] Jimenez, L. M., Ochoa, M., and Rueda, S. J. Jif-Based Verification of Information Flow Policies for Android Apps. *IJSSE 8*, 1 (2017), 28–42.
[23] Kirda, E., Kruegel, C., Banks, G., Vigna, G., and Kemmerer, R. Behavior-based Spyware Detection. In *Usenix Security Symposium* (2006), p. 694.
[24] Kurose, J. F., and Ross, K. W. *Computer Networking: A Top-Down Approach (6th Edition).* Pearson, 2012.
[25] Malik, J., and Kaushal, R. CREDROID: Android Malware Detection by Network Traffic Analysis. In *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing* (2016), ACM, pp. 28–36.
[26] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.
[27] Qamar, A., Karim, A., and Chang, V. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems 97* (2019), 887 – 909.



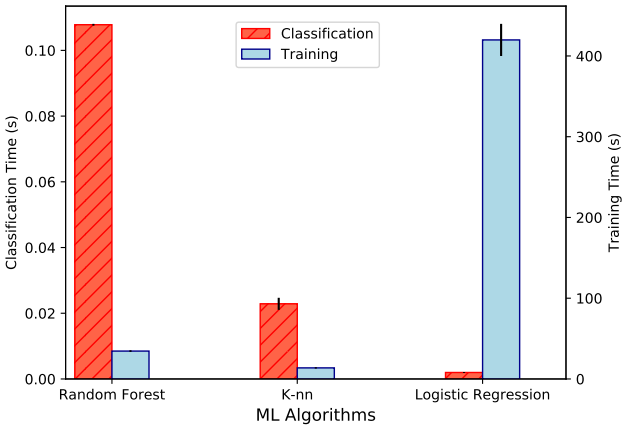

**(a) F1-score distribution.**




**(b) Mean F1-score.**




**(c) Heatmap representation.**




**(d) Time performances.**

**Figure 3: Results of our experiments.**


[28] R. Chatterjee and P. Doerfler and H. Orgad and S. Havron and J. Palmer and D. Freed and K. Levy and N. Dell and D. McCoy and T. Ristenpart. The spyware used in intimate partner violence. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), pp. 441–458.

[29] Rathi, D., and Jindal, R. DroidMark: A Tool for Android Malware Detection using Taint Analysis and Bayesian Network. *arXiv preprint arXiv:1805.06620* (2018).

[30] Saad, M. H., Serageldin, A., and Salama, G. I. Android Spyware Disease and Medication. In *Proceedings of the Second International Conference on Information Security and Cyber Forensics (InfoSec)* (2015), IEEE, pp. 118–125.

[31] Salvia, R., Ferrara, P., Spoto, F., and Cortesi, A. SDLI: Static Detection of Leaks Across Intents. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)* (2018), pp. 1002–1007.

[32] Saracino, A., Sgandurra, D., Dini, G., and Martinelli, F. Madam: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing 15*, 1 (2018), 83–97.

[33] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Appscanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic. In *In 2016 IEEE European Symposium on Security and Privacy (Euro S&P)* (2016), IEEE, pp. 439–454.

[34] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Transactions on Information Forensics and Security 13*, 1 (2018), 63–78.