Analyzing Android Encrypted Network Traffic to Identify User Actions

Mauro Conti, Senior Member, IEEE, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde

Abstract—Mobile devices can be maliciously exploited to violate the privacy of people. In most attack scenarios, the adversary takes the local or remote control of the mobile device, by leveraging a vulnerability of the system, hence sending back the collected information to some remote web service. In this paper, we consider a different adversary, who does not interact actively with the mobile device, but he is able to eavesdrop the network traffic of the device from the network side (e.g., controlling a Wi-Fi access point). The fact that the network traffic is often encrypted makes the attack even more challenging. In this paper, we investigate to what extent such an external attacker can identify the specific actions that a user is performing on her mobile apps. We design a system that achieves this goal using advanced machine learning techniques. We built a complete implementation of this system, and we also run a thorough set of experiments. Our system has shown that our attack can achieve accuracy and precision higher than 95%, for most of the considered actions. We compared our solution with the three state-of-the-art algorithms, and confirming that our system outperforms all these direct competitors.

Index Terms—Cellular phones, information security, privacy.

I. INTRODUCTION

THE amount of sensitive data that users handle with their mobile devices is truly staggering. People continuously carry these devices with them and use them for daily communication activities, including not only voice calls and SMS, but also emails and social network interactions. A typical user gains access to her savings and checking account by using her smartphone. She installs and uses several apps to communicate with friends or acquaintances. Through her smartphone, she gets information about sensitive topics such as diseases, sexual or religious preferences, etc. As a consequence, several concerns have been raised about the capabilities of these portable devices to invade the privacy of users actually becoming “tracking devices”. In this context, an important aspect is related to the possibility of continuously spying and locating an individual [3], [32], [35]. Solutions to identify and isolate malicious code running on smartphones [31], [37], [42] as well as to protect against attacks coming from the network [4], [11] might significantly reduce current threats to user privacy. While people become more familiar with mobile technologies and their related privacy threats (also thanks to the attention raised by the media, e.g., see the recent attention on NSA for supposedly eavesdropping foreign governments leaders such as Angela Merkel [35]), users have started adopting good practices that better adapt to their privacy feeling and understanding. Unfortunately, we believe that even adopting such good practices would not close the door to malicious adversaries willing to trace people. Indeed, several attacks may violate the privacy of the user even when the adversary does not physically or remotely control the user device. In this paper, we consider a passive attacker that is able to sniff the network traffic of the devices from the network side. Obviously, if the network traffic is not encrypted, the task of such an attacker is simple: he can analyze the payload and read the content of each packet. However, many mobile apps use the Secure Sockets Layer (SSL) – and its successor Transport Layer Security (TLS) – as a building block for encrypted communications. Even when such solutions are in place, the adversary can still infer a significant amount of information from the analysis of the properly encrypted network traffic. For example, work leveraging analysis of encrypted traffic already highlighted the possibility of understanding the apps a user has installed on her device [36], or identify the presence of a specific user within a network [38].

This work focuses on understanding whether the user profiling made through analyzing encrypted traffic can be enhanced to understand exactly what actions the user is doing on her phone: as concrete examples, we aim at identifying actions such as the user sending an email, receiving an email, browsing someone profile on a social network, publishing a post or a tweet. The underlying issue we leverage in our work is that SSL and TLS protect the content of a packet, while they do not prevent the detection of networks packets patterns that instead may reveal some sensitive information about the user behavior. An adversary may use our approach in several practical ways to threaten the privacy of the user. In the following,
we report some possible scenarios:

- A censorship government may try to identify a dissident who spreads anti-government propaganda using an anonymous social network account. Comparing the time of the public posts with the time of the actions (inferred with our method), the government can guess the identity of that anonymous dissident.
- By tracing the actions performed by two users, and taking into account the communication latency, an adversary may guess (even if with some probability of error) whether there is a communication between them. Multiple observations could reduce the probability of errors.
- An adversary can build a behavioral profile of a target victim based on the habits of the latter one (e.g., wake up time, work time). For example, this could be used to improve user fingerprinting methods, to infer the presence of a particular user in a network [38], even when she accesses the network with different types of devices.

a) Contributions: In this paper (which is an extended version of the work in [12]), we propose a framework to infer which particular actions the user executes on some app installed on her mobile phone. In particular, we assume that the traffic is encrypted and the adversary eavesdrops (without modifying them) the messages exchanged between the user’s device and the web services that she uses.

Our framework analyzes the network communications and leverages information available in TCP/IP packets (like IP addresses and ports), together with other information like the size, the direction (incoming/outgoing), and the timing. By using an approach based on machine learning, each app that is of interest is analyzed independently. To set up our system, for each app we first pre-process a dataset of network packets labeled with the user actions that originated them, we cluster them in flow typologies that represent recurrent network flows, and finally we analyze them in order to create a training set that will be used to feed a classifier. The trained classifier will then be able to classify new traffic traces that have never been seen before. We run a thorough set of experiments to evaluate our solution considering seven popular apps: Facebook, Gmail, Twitter, Tumblr, Dropbox, Google+ and Evernote. The results show that it can achieve accuracy and precision higher than 95%, for most of the considered actions.

In the current version of the paper, we also add a discussion (not present in [12]) about the key idea underneath our traffic analysis approach. In particular, we examine in depth the concept of network flow and the metric to evaluate the similarity between them. We also report details of the machine learning techniques we leverage in our method. Furthermore, in addition to our previous work [12], we run a thorough comparison of our solution with three state of the art algorithms, showing that our solution outperforms them in all of the cases.

b) Organization: The rest of this paper is organized as follows. In Section II, we revise the state of the art around our research topic. In Section III, we introduce some background knowledge on machine learning and data mining tools used in our work. In Section IV, we present our framework describing all its different components. We present the evaluation of our solution for identifying user actions in Section V, where we compare with similar solutions as well. In Section VI, we discuss about possible countermeasures against the proposed attack. Finally, in Section VII we draw some conclusions and point out ways in which this work can be further extended.

II. RELATED WORK

Our main claim in this paper is that network traffic analysis and machine learning can be used to infer private information about the user, i.e., the actions that she executes with her mobile phone, even though the traffic is encrypted. To position our contribution with respect to the state of the art, we survey the works that belong to two main research areas that focus on similar issues: privacy attacks via traffic analysis (not necessarily focusing on mobile devices) and traffic analysis of mobile devices (not necessarily focusing on privacy).

c) Privacy attacks via traffic analysis: In the literature, several works proposed to track user activities on the web by analyzing unencrypted HTTP requests and responses [6], [7], [33]. With this analysis it was possible to understand user actions inferring interests and habits. More recently, Neasbitt et al. proposed ClickMiner [28], a tool that reconstructs user-browser interactions. However, in recent years, websites and social networks started to use SSL/TLS encryption protocol, both for web and mobile services. This means that communications between endpoints are encrypted and this type of analysis cannot be performed anymore.

Different works surveyed possible attacks that can be performed using traffic analysis assuming a very strong adversary (e.g., a national security agency) which is able to observe all communication links [30]. In [24], Liberatore and Levine evaluated the effectiveness of two traffic analysis techniques based on naive Bayes and on Jaccard’s coefficient for identifying encrypted HTTP streams. Such an attack was outperformed by [22], where the authors presented a method that applies common text mining techniques to the normalized frequency distribution of observable IP packet sizes, obtaining a classifier that correctly identifies up to 97% of requests. Similarly, in [29] the authors presented a support vector machine classifier that was able to correctly identify web pages, even when the victim used both encryption and anonymization networks such as Tor. Finally, Cai et al. [9] presented a web pages fingerprinting attack and proved its effectiveness despite traffic analysis countermeasures, such as HTTP/2 [25].

Unfortunately, none of the aforementioned works was designed for (or could easily be extended) to mobile devices. In fact, all of them focus on web pages identification in desktop environment (in particular, in desktop browsers), where the generated HTTP traffic strictly depends on how web pages are designed. Conversely, mobile users mostly access the contents through the apps installed on their devices [20]. These apps communicate with a service provider (e.g., Facebook) through a set of APIs. An example of such differences between desktop web browsers and mobile apps is the validation of SSL certificates [11], [19].

Traffic analysis has been applied not only to HTTP but also to other protocols. For example, Song et al. [34] prove that several versions of SSH are not secure. In particular, they show
that even very simple statistical techniques suffice to reveal sensitive information such as login passwords. More importantly, the authors show that by using more advanced statistical techniques on timing information collected from the network, the eavesdropper can also learn significant information about what users type in SSH sessions. SSH is not the only protocol that has been target of such attacks. Another example is Voice Over IP (VoIP). In [40], the authors show how the length of encrypted VoIP packets can be used to identify spoken phrases of a variable bit rate encoded call. Their work indicates that a profile Hidden Markov Model trained using speaker- and phrase-independent data can detect the presence of some phrases within encrypted VoIP calls with recall and precision exceeding 90%.

In [10], the authors show that despite encryption, web applications also suffer from side-channel leakages. The system model considered is different from ours. In particular, their focus is on web applications. On the contrary, we focus on mobile applications. More importantly, the authors leverage three fundamental features of web applications: stateful communication; low entropy input; significant traffic distinction.

We believe that in most mobile applications two of these features (stateful communication, low entropy input) are not very useful to characterize user actions. In contrast to the work in [10], we adopt a solution that only needs information about packet sizes and their order.

d) Traffic analysis of mobile devices: Focusing on mobile devices, traffic analysis has been successfully used to detect information leaks [17], to profile users by their set of installed apps [36], to find their position [5], and to generate network profiles to identify Android apps in the HTTP traffic [14]. Traffic analysis has also been used to understand network traffic characteristics, with particular attention to energy saving [18]. Stöber et al. [36] show that it is possible to identify the set of apps installed on an Android device, by eavesdropping the 3G/UMTS traffic that those apps generate. Similarly, Dai et al. [14] introduce an automatic app profiler that creates the network fingerprint of an Android app relying on packet payload inspection. Unfortunately, their solution is viable only for apps that do not use encrypted traffic.

In [43], Zhou et al. discovered three unexpected channels of information leaks on Android: per-app data-usage statistics, ARP information, and speaker status. In particular, the authors used a suite of inference techniques to reveal a phone user’s identity from the network-data consumption of Twitter app, by also leveraging online resources such as tweets published by Twitter. Unfortunately, the authors focused only on a specific user action (i.e., send a tweet) without distinguish that action from the other ones a user could perform. More recently, Coull and Dyer in [13] presented a work similar to ours. The authors inferred information analyzing payload lengths of network packets produced by Apple iMessage and other messaging apps on iOS and OSX. In particular, the purpose of their work is to infer the OS version, user actions and language used in instant messaging. The author focused on five actions strictly related to instant messaging apps: start writing, stop writing, message sending, attachment sending and read notification. In this paper, we consider social network and email service apps on Android. Those apps permit us to investigate a wider set of actions than the one offered by instant messaging apps. We believe that the interest from researchers to aim at different targets (i.e., OS version, actions, language) and the results obtained so far, underlines the feasibility of those attacks, the relevance of this issue and the importance to foster further research in this domain.

None of the work mentioned in this section aim at inferring the actions a user performs on Android apps via encrypted traffic analysis. The first and only work that achieved this goal is our preliminary work [12], that is extended in the current version of the manuscript.

III. Machine Learning and Data Mining Background

In this section, we briefly recall several machine learning and data mining concepts that we use in our paper, while we point the reader to appropriate references for a complete introduction on those topics.

A. Dynamic Time Warping

Dynamic Time Warping (DTW) [27] is a useful method to find alignments between two time-dependent sequences (also referred as time series) which may vary in time or speed. This method is also used to measure the distance or similarity between time series.

Let us consider two sequences that represent two discrete signals: \( X = (x_1, \ldots, x_N) \) of length \( N \in \mathbb{N} \); and \( Y = (y_1, \ldots, y_M) \) of length \( M \in \mathbb{N} \). DTW uses a local distance measure \( c : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+ \) to calculate a cost matrix \( C \in \mathbb{R}^{N \times M} \), s.t., each cell \( C_{i,j} \) reports the distance between \( x_i \) and \( y_j \). The goal is to find an alignment between \( X \) and \( Y \) having minimal overall distance. Intuitively, such an optimal alignment runs along a “valley” of low cost cells within the cost matrix \( C \). More formally, a warping path is defined as a sequence \( p = (p_1, \ldots, p_L) \) with \( p_l = (n_l, m_l) \in [1:N] \times [1:M], l \in [1:L] \) satisfying the following three conditions:

1) Boundary condition: \( p_1 = (1, 1) \) and \( p_L = (N, M) \);
2) Monotonicity condition: \( n_1 \leq n_2 \leq \ldots \leq n_M \) and \( m_1 \leq m_2 \leq \ldots \leq m_L \);
3) Step size condition: \( p_{l+1} - p_l = \{(0,1), (1,0), (1,1)\} \) for \( l \in [1:L-1] \).

The total cost of a warping path is calculated as the sum of all the local distances of its elements. An optimal warping path is a warping path \( p^* \) having minimal total cost among all possible working paths. The total cost of an optimal warping path is also used as a distance measure between two sequences \( X \) and \( Y \). In this paper, we will indicate the cost of an optimal warping path with \( DTW(X, Y) \).

Figure 1a shows an example of alignment between two signals (indicated in the figure with Flow A and Flow B). The arrows show the matched points which are given by the DTW algorithm. The same two flows have been used to calculate the heat matrix shown in Figure 1b. In this representation, the color of a cell \((i, j)\) represents the minimum distances to reach cell \((i, j)\) when starting from cell \((0,0)\). An optimal warping path is then highlighted with a line that runs from cell \((0,0)\) to cell \((12,13)\). It can be noticed that this warping
sets of observations as a function of the pairwise distances between observations, we will use the average distance, that is defined as:

\[ d(u, v) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} \frac{d(u[i], v[j])}{|u| \cdot |v|}, \]

where \( d() \) is a distance function, and \( u \) and \( v \) are two clusters of \( n \) and \( m \) elements, respectively. More details about Hierarchical clustering can be found in [21].

IV. OUR FRAMEWORK

Our framework is logically composed by two components: the “pre-processor” and the “traffic classifier”. The former has the task of executing all the pre-processing steps that allow us to model the network traffic into data that the traffic classifier can easily handle. The latter executes the actual classification task. Before using the traffic classifier, it has to be trained with labeled traffic data that we are able to generate by artificially stimulating the analyzed apps. We detail the steps executed by the pre-processor in Section IV-A, while in Section IV-B, we describe the methodology used to generate our training dataset, as well as the procedure used to classify user actions.

A. Network Traffic Pre-Processing Steps

Mobile apps generally rely on SSL/TLS to securely communicate with peers. These protocols are built on the top of the TCP/IP suite. The TCP layer receives encrypted data from the above layer, it divides data into chunks if the packets exceeds a give size. Then, for each chunk it adds a TCP header creating a TCP segment. Each TCP segment is encapsulated into an Internet Protocol (IP) datagram, and exchanged with peers. Since TCP packets do not include a session identifier, both endpoints identify a TCP session using the client’s IP address and the port number.

A fundamental entity considered in this paper is the traffic flow: with this term we indicate a time ordered sequence of TCP packets exchanged between two peers during a single TCP session. The pre-processor takes in input the network traffic, it builds the network flows that represent that network traffic, and it generates a set of time series: (i) a time series is obtained by considering the bytes transported by incoming packets only; (ii) another one is obtained by considering bytes transported by outgoing packets only; (iii) a third one is obtained by combining (ordered by time) bytes transported by both incoming and outgoing packets. Hence, we use this set of time series as an abstract representation of a connection between two peers. Note that additional time series may be added to this set for example by considering other parameters such as the time-gap between different packets. For the sake of simplicity, in the following we will only consider the first three types of time series mentioned above.

Table I reports an example of time series generated from three network flows, while Figure 2 graphically represents these flows through a cumulative chart. The lower side of the chart represents incoming traffic, while the upper side represents outgoing traffic. This is only one of the possible representations, and it shows that the “shapes” of these three
In the following, we will detail the three pre-processing steps component of the framework, which is the traffic classifier. For each flow, the result of these three pre-processing steps will be a set of time series that will be passed to the next packets); 3) we limit the length of the generated time series.

In particular: 1) we apply a domain filtering to select only flows belonging to the analyzed app; 2) we filter the remaining traffic generated from apps other than the considered one (or traffic generated by the OS) does not interfere with the analysis. Different configurations of packets intervals, showing that the best configuration is app dependent.

g) Timeout and packets interval: Two different techniques are used to limit the length of the generated time series: a timeout mechanism and the specification of a packets interval. The timeout mechanism is used to terminate the flows that did not receive any new packet since 4.5 seconds. Indeed, it has been proved experimentally that 95% of all packets arrive at most 4.43 seconds after their predecessors [36]. The packets interval specifies the first and the last packet to be considered. For example, considering a flow f composed by $l$ packets, and the interval $[x, y]$ with $x \leq y$ and $y \leq l$, the corresponding time series will be composed by $y - x + 1$ values that report the bytes of the $x^{th}$ to the $y^{th}$ packet. This simple mechanism allows us to focus on particular portions of the flow. The first part, for example, is often the more significant. In the experimental part, we report the results for different configurations of packets intervals, showing that the best configuration is app dependent.

B. Traffic Classification Details

Since we use a supervised learning approach, it is necessary to create a labeled dataset that describes the user actions that can be involved in the communication. The same happens when the back-end is composed of several components such as different web services, databases, etc. To overcome this problem we use another strategy: we take into consideration for further analysis only the flows which destination IP addresses owners have been clearly identified as related to the considered app. In the implementation of our framework, we leverage the WHOIS protocol for this purpose, but we want to highlight that this is only one of the possible ways. Business and other context information may be used in order to perform the domain filtering. We also take into consideration the traffic related to third party services (such as Akamai or Amazon) that are indeed used by several applications [39].

f) Packets filtering: Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, hence requesting retransmission of lost data, and reordering out-of-order data. As a result, several TCP packets that do not carry data, may hinder the analysis process. In the data exchange phase, for example, the receiver sends a packet with the ACK flag set to notify the correct reception of a chunk of data. These ACK packets are transmitted in asynchronous mode so they are affected by many factors related to round trip time of the connection link. The order of the received packets may hinder the evaluation of the similarity between two network flows. For this reason, we filter out all packets retransmissions, as well as packets marked with the ACK flag. Note that the metric that we will use in order to measure similarity between flows (see Section IV-B) will mitigate the consequences of missing packets. We also filter out other packets that do not bring any additional information helpful in characterizing flows. In particular, we filter out the three way handshake executed to open a TCP connection, and the packets exchanged to close it.

e) Domain filtering: The network traffic generated by an application is generally directed toward a back-end infrastructure. The back-end infrastructures might be composed of a single server, or a set of servers. The set of servers might even be behind a load balancer. Since we analyze each app independently, we need to make sure that traffic generated from apps other than the considered one (or traffic generated by the OS) does not interfere with the analysis. Different methods can be used in order to identify the app that generated each network flows. The destination IP address is a trivial discriminating parameter. However, in case of a load balanced back-end, we should know all the individual IP addresses that

### TABLE I
Example of Time Series Generated From Three Network Flows. Values Within Square Brackets Represent the Amount of Bytes Exchanged per Packet; Negative Values in Complete Time Series Indicate Incoming Bytes, While Positive Values Indicate Outgoing Bytes

<table>
<thead>
<tr>
<th>Flow</th>
<th>Type</th>
<th>Time series</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Incoming</td>
<td>[1514, 1514, 315, [13, 477]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[282, 188, 514, 96, 1514, 179, 603, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... 98, 801, 98]</td>
</tr>
<tr>
<td></td>
<td>Complete</td>
<td>[282, 1514, 1514, 315, 188, -113, 514, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... 96, 1514, 179, 603, 98, 801, 98, -477]</td>
</tr>
<tr>
<td>2</td>
<td>Incoming</td>
<td>[1514, 1514, 2666, 592, [113, 606]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[282, 188, 692, 423]</td>
</tr>
<tr>
<td></td>
<td>Complete</td>
<td>[282, 1514, 1514, 1266, 582, 188, -113, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>602, 423, -661]</td>
</tr>
<tr>
<td>3</td>
<td>Incoming</td>
<td>[1245, 1514, 107, 465, 172, 111]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[926, 655, 136, 913, 1514, 1514, 863]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[926, 655, 136, -1245, 913, 1514, 1514, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... -863, -1514, -107, -465, -172, -111]</td>
</tr>
</tbody>
</table>

Fig. 2. Representation of flows time series.
dataset, we simulate a series of user actions by interacting with the app to analyze. For each performed action we intercept and label the flows generated after the execution of the action itself. For each app that we analyze we focus on actions that are significant for that particular app.

In most cases, a single user action generates a set of different flows (i.e., not just a single one). Furthermore, different user actions may generate different sets of flows. Our classification method is based on the detection of the sets of flows that are distinctive of a particular user action. In order to elicit these distinctive sets of flows, we build clusters of flows by using the agglomerative clustering approach described in Section III-B. Similar flows will be grouped together in the same cluster, while dissimilar flows will be assigned to different clusters. The average distance is used as linkage criterion, while the distances of the corresponding time series. Supposing that each flow \( f_i \) is decomposed into a set of \( n \) time series \( \{T_i^1, \ldots, T_i^n\} \), the distance between \( f_i \) and \( f_j \) is defined as:

\[
dist(f_i, f_j) = \sum_{k=1}^{n} w_k \times DTW(T_k^i, T_k^j),
\]

where \( w_k \) is a weight assigned to the particular time series. Weights can be assigned in such a way as to give more importance to some type of time series with respect to others. For example, it is possible to give more weight to the time series that represent incoming packets, and less weight to those that represent outgoing packets.

In order to reduce the computational burden of the subsequent classification, a leader is elected for each cluster. Leaders will be the representative flows of their clusters. Given a cluster \( C \) containing the flows \( \{f_1, \ldots, f_n\} \), the leader is elected by selecting the flow \( f_i \) that has the minimum overall distance from the other members of the cluster, that is:

\[
\arg \min_{f_i \in C} \left( \sum_{j=1}^{n} \dist(f_i, f_j) \right).
\]

Clustering is executed on the set of flows that will be used to build the training dataset. In particular, after performing the clustering the training dataset will be composed as follows. The user actions will be the instances of the datasets, while the class of each instance is a label representing the action. We will have one integer feature for each cluster identified through the agglomerative clustering. The value of each feature is determined by analyzing the flows related to an action. Each flow \( f \) captured after the execution of an action will be assigned to the cluster that minimizes the distance between \( f \) and the leader of the cluster. The \( k \)th feature will therefore indicate the number of flows that have been assigned to the cluster \( C_k \) after the execution of that action. For example, for the action send mail, the \( k \)th feature will be equal to 2 if there are 2 flows labeled with send mail assigned to the cluster \( C_k \).

Finally, we execute the classification with the Random Forest algorithm. The main idea behind the overall approach is that different actions will “trigger” different sets of clusters.

The classification algorithm will therefore learn which are these sets, and will be able to correctly determine the class labels for unseen instances.

V. EXPERIMENTAL RESULTS

In order to assess the performance of our proposal, we considered several widespread apps that have different purposes: Gmail, Facebook, Twitter, Tumblr, Dropbox, Google+ and Evernote. We selected these apps because of their high popularity [1]. Indeed, Gmail is one of the largest email services and its Android app has over one billion downloads. On the other hand, Facebook and Twitter are not only the most popular Online Social Networks [2], but they also had a leading role in the Arab spring and the Istanbul’s Taksim Gezi Park protests (when Turkish government blocked Twitter). Tumblr is a widely used micro-blogging platform owned by Yahoo! Inc., while Dropbox is one of the most used cloud storage services. Google+ is the social network and social layer for Google services owned and operated by Google Inc. Finally, Evernote is an app designed for note-taking and archiving. Dropbox, Google+ and Evernote. Given the wide set of apps we considered, we believe that the results of our analysis also hold for any other app that generates network traffic as a consequence of a user action. Note that most of the apps make use of a back-end service to implement the logic of the service, and thus they must generate network traffic as a consequence of almost any user interaction. To collect the network traffic related to different user actions, we set up a controlled environment. In this section we present the elements that compose this environment (Section V-A), the methodology used to collect the data (Section V-B), and the results of the evaluation (Section V-C).

A. Hardware and Network Configuration

For the evaluation of our solution, we used a Galaxy Nexus (GT-I9250) smartphone, running the Android 4.1.2 (Jelly bean) operative system.

We enabled the “Android Debug” option in order to allow the usage of the ADB (Android Debug Bridge) interface via USB cable. We used a Wi-Fi access point (U.S. Robotics USR808054) to provide wireless connectivity to the mobile phone. Finally, we used a server (Intel Pentium Processor dual core E5400 2.7GHz with 4 GB DDR2 RAM) with two network cards running Ubuntu Server 11.04 LTS to route the traffic from the access point to the Internet, and vice versa.

To eavesdrop network packets flowing through the server, we used Wireshark software. From a Wireshark capture file, we created a comma separated file (csv), where each row describes a packet captured from the access point’s interface. For every packet we reported source and destination IP addresses, ports, size in bytes and time in seconds from Unix epoch, protocol type and TCP/IP flags. Since the payload is not relevant to our analysis, it has been omitted. This data has then been used to generate the time series as explained in Section IV-A.

1 \(00:00:00 \text{ UTC}, 01 \text{ January 1970.} \)
B. Dataset Collection and Analysis

For our study we considered seven apps installed from the official Android market: Gmail v4.7.2, Facebook v3.8, Twitter v4.1.10, Tumblr v3.8.6.08, Dropbox v2.4.9.00, Google+ v5.3.0.91034052 and Evernote v7.0.2. For the social apps, we created ten accounts that have been divided in two different categories of users: “active” and “passive” users. “Active” users simulated the behavior of users that actively use the app by sending posts, email, tweets, surfing the various menus, etc. “Passive” users simulated the behavior of users that passively use the app, just by receiving messages or posts. The accounts of both passive and active users have been configured in such a way as to have several friends/followers within the group. We avoided configuring the accounts with actual friends or followers, in order to avoid interference due to notifications of external users activities that were not under our control.

To reach a particular target, a user may have to perform several actions in a precise order. An action could be simple (e.g., a tap on a button, a swipe, or a selection of edit box), or complex (e.g., type a text, which is a sequence of keyboard inputs). For example, a user has to perform three actions in a precise sequence to post a message on her Facebook wall. He has to be sure that the Facebook app shows the “user’s wall”, then she has to tap on the “write a post” button (1), fill the edit box with some text (2), and finally tap on the “post” button (3). It is important to highlight that we do not use static text to fill in text boxes, but the text is randomly selected from a large set of sentences. A script submits the sequence of actions to the mobile phone through the ADB commands, and it captures the network traffic that is generated. The script also records the execution time of each action. By using the recorded execution time of each action, it is then possible to label the flows extracted from the network traffic with the user action that produced it. For each app, we choose a set of actions that are more sensitive than others from user privacy point of view (e.g., send an email or a message, for the reasons we report in Section I). For example, a user has to perform three actions or complex (e.g., type a text, which is a sequence of keyboard inputs). As explained in Section IV-A, each network flow is modeled as a set of time series. Table II reports the weights and the intervals for several configurations (“Conf.” in the table) used to limit the length of the time series generated by each app.

We used different weights configurations, and we selected the packets intervals by analyzing the statistical length of the flows. Figure 3 reports the statistical distribution of the length of the complete time series extracted from the network traffic. First and third quartile are represented as the left and right side of the notched box. The notch of the box represents the median value. Lines that extend horizontally from the boxes indicate the 2nd percentile (left) and the 98th percentile (right).
change for Gmail, while they are almost unvaried for Twitter. As a matter of fact, different Twitter actions just differ in their last packets. Nevertheless, our approach reaches very good performance for this app too. In our experiments, we used the Random forest classifier implemented by the Python library scikit-learn. The classifier is trained using 40 estimators (or weak learners). Each estimator consists of a decision tree without any restrictions on its depth limit. The number of features for each estimator is equal to the square root of the maximum number of available features.

C. Classification Performance

Before considering the classification of the user actions, it is worth discussing how to choose the number of clusters that should be used. In order to establish a reasonable value for this parameter, we used a validation dataset to study the accuracy of the classification when varying the number of clusters. Figure 5 reports the achieved results. For each app, we therefore considered the number of clusters that maximized the accuracy, in terms of averaged F-measure. In the following, we report the results of the classification app by app, and we discuss the average accuracy reached when detecting each sensitive user action. In Table III, we report detailed results for the precision, the recall and the F-measure metrics achieved by the best configuration of all the analyzed apps. Since we are space constrained, we report the corresponding confusion matrices only for some of the analyzed apps.

1) Facebook: We focused on seven different actions that may be sensitive when using the Facebook app. On average, the F-measure is equal to 99%, with a precision and a recall of 99% and 98% respectively. Performance reached with different configurations of weights and packets intervals constraints are reported in Figure 6a. For each action at least one of the configurations exceeds 94% of accuracy, while the worst performing is always higher than 74%.

Table III reports precision, recall and F-measure reached by using Configuration 3. We noticed that all the actions have a precision higher 96%. The recall is higher than 95% for all the actions apart from the open user profile, that reaches 91%. In fact, we realized that this particular action is classified as other in 9% of the examples, as we can see from the confusion matrix reported in Figure 7a.
2) Gmail: We analyzed four specific user actions of the Gmail app: send mail, reply button, open chats and send reply. Figure 6a shows the classification accuracy that has been reached for each configuration of weights and packet interval constraints. We observe that we are able to distinguish with high accuracy the action of sending of a new mail, from that of replying to a previously received message, as well as the tap on the reply button. The open chats action is instead more difficult to distinguish. Table III reports precision, recall and F-measure for Configuration 1. We can observe that the action open chats (that allows to read past chats) achieves a low precision but a high recall. Analyzing the confusion matrix depicted in Figure 7b it is possible to notice that 16% of other actions are wrongly classified as open chats. This is the reason of such a low precision.

3) Twitter: During the analysis we noticed that Twitter actions may be more difficult to classify than Gmail and Facebook actions. Indeed, different Twitter actions generate similar time series that have a large portion in common. Only the last three or four packets of each time series show some difference. Nevertheless, we have been able to reach outstanding results for this app as well. In particular, we focus on six specific user actions: refresh home, open contacts, tweet/message, open messages, open twitter, open tweets. Performance reached for all the analyzed configurations are reported in Figure 6c. For each action at least one of the configurations exceeds 96% of accuracy, while the worst configuration has an accuracy in any case higher than 91%. The best performing configuration is Configuration 1, that on average, reached an F-measure value equal to 97%, with a precision and a recall of 98% and 97% respectively (see Table III). The action open twitter has accuracy and recall equal to 100%, independently of the Configuration set used for the clustering phase. As a consequence, none of the examples of the test set have been wrongly classified. Figure 7c reports the confusion matrix obtained by considering the Twitter actions. Three of the six analyzed actions are correctly classified in more than the 99% of the cases, while the other three actions, that are open contacts, open messages and open tweets, are correctly classified in more than 95% of the cases.

4) Tumblr: We analyzed ten different user actions of the Tumblr app (see Table III). On average precision, recall and F-measure is equal to 99%. Precision is always greater than 97% for the individual actions, while recall is greater than 96% in all the cases but one: home page.

5) Dropbox: As for Dropbox, we analyzed eight different user actions. On average, we reached a precision of 95% and a recall of 92%. Only for two individual actions, we reached precision or recall lower than 80%. This is the case of folder creation or delete file. However, the average F-measure is still greater than 92%.

6) Google+: We analyzed ten different user actions of the Google+ app (see Table III). On average precision, recall and F-measure are equal to 90%, 94%, 92% respectively. Precision values range from 75% to 100% for the individual actions, while recall is greater than 84% in all the cases. The actions delete post and send comment have both precision and recall equal to 100%.

7) Evernote: We analyzed six different user actions of the Evernote app. Evernote is definitely the app that achieved better performance among those we analyzed. Indeed, we achieved an average precision, recall and F-measure equal to 100%.
D. Comparison With Other Methods

To confirm the validity of the proposed approach, we compared the results achieved by our solution with three traffic analysis techniques. The solutions we compare with have been proposed to face a problem similar to the one we consider, i.e., the identification of the websites the user is retrieving under the cover of an encrypted tunnel. Two of them are due to Liberatore and Levine [24]. They proposed two methods that are based on naive Bayes and Jaccard’s coefficient respectively. The third one is due to Herrmann et al. [22]. They applied common text mining techniques to the frequency distribution of observable IP packet sizes. Since all these algorithms require several parameters to be tuned, we analyzed different configuration and in the following we report only the results for the best configuration that we found. Figure 8 reports the results of the comparison. Because we are space constrained, we report the results of the comparison only for some of the apps we analyzed. The other apps do not show a significantly different behavior. In particular, the performance of our solution is always comparable or significantly better than the performance of the other proposed approaches.

In particular, Figure 8a shows the averaged F-measure for Facebook, Gmail and Twitter. The averaged F-measure is the average of the F-measures reached by classifying the actions considered for that specific app. It can be noticed that in all the cases our classifier outperforms the other approaches. Figures 8b, 8c and 8d show the results for each app more in depth. In particular, each figure compares the F-measures reached when classifying the individual actions of that app. As it turns out, our classifier significantly outperforms the other three approaches in the majority of cases, while the results are comparable in the remaining cases. This indicates a higher level of reliability with respect to the other approaches.

In contrast with the other algorithms, our solution uses more advanced machine learning techniques such as ensemble methods, Dynamic Time Warping, and hierarchical clustering. Furthermore, our solution uses information such as the packet order that is not considered in the other cases. Finally, our approach is resilient to packet retransmissions that might be significant in mobile apps. We believe that these features make our classifier more reliable than its competitors, especially for the mobile scenario. However, we want to highlight that our solution may also be competitive in desktop scenarios.

VI. POSSIBLE COUNTERMEASURES AND LIMITATIONS

Users and service providers might believe that their two parties communications are secure if they use the right encryption and authentication mechanisms. Unfortunately, current secure communication mechanisms limit their traffic encryption actions to the syntax of the transmitted data. The semantic of the communication is not protected in any way [23]. For this reason, it has been possible for example to develop classifiers for TLS/SSL encrypted traffic that are able to discriminate between applications.

The contribution of this paper was to investigate to which extent it is feasible to identify the specific actions that a user is doing on her mobile device, by simply eavesdropping the device’s network traffic. While it is out of the scope of the paper to investigate possible countermeasures to the proposed attack, we discuss in the following some related issues.

The common belief is that simple padding techniques may be effective against traffic analysis approaches. However, it has to be considered that padding countermeasures are already standardized in TLS, explicitly to “frustrate attacks on a protocol that are based on analysis of the lengths of exchanged messages” [15]. Nevertheless, our attack worked against TLS encrypted traffic. More advanced techniques have been proposed in the literature, such as traffic morphing and direct target sampling [40], [41]. However, a recent result showed that none of the existing countermeasures are effective [16].

The intuition is that coarse information is unlikely to be hidden efficiently, and the analysis of these features may still allow an accurate analysis. On the light of these results, we believe it is not trivial to propose effective countermeasures to the attack we showed in this paper. Indeed, it is the intention of the authors to highlight a problem that is becoming even more alarming after the revelation about the mass surveillance programs that are nowadays adopted by governments and nation states.

In our opinion, the main limitation of our approach is related the usage of supervised learning algorithms. It has to be considered that this technique is generally more efficient
VII. CONCLUSIONS

The framework proposed in this paper is able to analyze encrypted network traffic and to infer which particular actions the user executed on some apps installed on her mobile-phone. We demonstrated that despite the use of SSL/TLS, our traffic analysis approach is an effective tool that an eavesdropper can leverage to undermine the privacy of mobile users. With this tool an adversary may easily learn habits of the target users. The adversary may aggregate data of thousands of users in order to gain some commercial or intelligence advantage against some competitor. In addition, a powerful attacker such as a Government, could use these insights in order to de-anonymize user actions that may be of particular interest. We hope that this work will shed light on the possible attacks that may undermine the user privacy, and that it will stimulate researchers to work on efficient countermeasures that can also be adopted on mobile devices. These countermeasures may require a kind of trade-off between power efficiency and the required privacy level.

REFERENCES
