Can't You Hear Me Knocking: Identification of User Actions on Android Apps via Traffic Analysis

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

Luigi V. Mancini
Sapienza University of Rome
Rome, Italy
lv.mancini@di.uniroma1.it

Riccardo Spolaor
University of Padua
Padua, Italy
rspolaor@math.unipd.it

Nino V. Verde
Sapienza University of Rome
Rome, Italy
verde@di.uniroma1.it

ABSTRACT

While smartphone usage become more and more pervasive, people start also asking to which extent such devices can be maliciously exploited as "tracking devices". The concern is not only related to an adversary taking physical or remote control of the device, but also to what a passive adversary without the above capabilities can observe from the device communications. Work in this latter direction aimed, for example, at inferring the apps a user has installed on his device, or identifying the presence of a specific user within a network.

In this paper, we move a step forward: we investigate to which extent it is feasible to identify the specific actions that a user is doing on mobile apps, by eavesdropping their encrypted network traffic. We design a system that achieves this goal by using advanced machine learning techniques. We did a complete implementation of this system and run a thorough set of experiments, which show that it can achieve accuracy and precision higher than 95% for most of the considered actions.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General - *Security and protection*

Keywords

Network traffic analysis; Machine learning; Privacy; Mobile security.

CODASPY'15, March 2–4, 2015, San Antonio, Texas, USA.

http://dx.doi.org/10.1145/2699026.2699119.

1. INTRODUCTION

People continuously carry smartphone devices with them and use them for daily communication activities, including not only voice calls and SMS but also emails and social network interaction. In the last years, several concerns have been raised about the capabilities of those portable devices to invade the privacy of the users and to become actual "tracking devices". Even when the adversary has no actual control of the phone (either physical or remote control via malicious apps) several attacks may violate the privacy of the communications. Indeed, if the network traffic is not encrypted, the task of an eavesdropper is simple: he can analyze the payload reading 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. Unfortunately there is often a gap between theory and practice, e.g., leveraging the SSL vulnerabilities of smartphone apps [15, 16] one might run an SSL man-in-the-middle attack to compromise the confidentiality of communications.

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 [29]), users start adopting some good practices that better adapt to their privacy feeling and understanding. For examples, solutions to identify and isolate malware running on smartphones [27, 31, 36] as well as to protect against attacks coming from the network [3, 10] might significantly reduce current threats to user privacy. Unfortunately, we believe that even adopting such good practices would not close the door to malicious adversaries willing to trace people. In fact, the wireless and pervasive nature of mobile devices would still leave many practical options for adversarial tracing. In particular, even when such solutions are in place, the adversary can still infer a significant amount of information from the properly encrypted traffic. For example, work leveraging analysis of encrypted traffic already highlighted the possibility of understanding the apps a user has installed on his device [30], or identify the presence of a specific user within a network [32].

This work focuses on understanding whether the user profiling made through analyzing encrypted traffic can be pushed up to understand exactly what actions the user is doing on his phone: as concrete examples, we aim at identifying actions such as the user sending an email, receiving an email, browsing someone profile in a social network, rather than 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 [32], even when he accesses the network with different type of devices.

Contributions.

In this paper, we propose a framework to infer which particular actions the user executed on some app installed on his mobile-phone, by only looking at the network traffic that the phone generates. In particular, we assume the traffic is encrypted and the adversary eavesdrops (without modifying them) the messages exchanged between the user's device and the web services that he 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, direction (incoming/outgoing), and 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 be then able to classify new traffic traces that have never been seen before. We fully implemented our system, and we run a thorough set of experiments to evaluate our solution considering three very popular apps: Facebook, Gmail, and Twitter. The results shows that it can achieve accuracy and precision higher than 95%, for most of the considered actions.

Organization.

The rest of this paper is organized as follows. In Section 2, we revise the state of the art. In Section 3, we present our framework describing all its different components. We present the evaluation of our solution for identifying user actions in Section 4. We discuss about possible countermeasures against the proposed attack in Section 5. Finally, in Section 6 we draw some conclusions and point out ways in which this work can be further extended.

2. 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 he executes by using his mobile phone, even though the traffic is encrypted.

In the literature, several works proposed to track user activities on the web by analyzing unencrypted HTTP requests and responses [4, 5, 28]. With this analysis it was possible to understand user actions inferring interests and habits. However, in recent years, websites and social networks started to use SSL/TLS encryption protocol, both for web and mobile services. As a consequence, communications between endpoints are often 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 [6, 26]. In [21], Liberatore et al. 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 [19], 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 [25] 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. [8] presented a web pages fingerprinting attack and proved its effectiveness despite traffic analysis countermeasures, such as HTTPOS [22].

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 [17]. 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 [10, 16].

In [9], the authors show that despite encryption, also web applications suffer from side-channel leakages. The system model considered is different from our. In particular, their focus is on web applications. On the contrary, we focus on mobile applications.

Focusing on mobile devices, traffic analysis has been successfully used to detect information leaks [14], to profile users by their set of installed apps [30], and to automatically generating network profiles for identifying Android apps in the HTTP traffic [11]. Stober et al. [30] show that it is possi-

ble to identify the set of apps installed on an Android device, by eavesdropping the 3G/UMTS traffic that those apps generate. Similarly, Tongaonkar et al. in [11] 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 [37], Zhou et al. discovered three unexpected channels of information leaks on Android: per-app data-usage statistics, ARP information, and speaker status. 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.

None of the works mentioned in this section aim at inferring and distinguish the potential user actions actions that a user can perform on mobile apps that rely on encrypted network traffic, which is the goal of our paper.

In the following, we briefly recall several machine learning and data mining concepts that we use in our paper: *Dynamic Time Warping*, *Hierarchical Clustering* and the *Random Forest classifier*. Furthermore, we point the reader to appropriate references for a complete introduction on those topics. *Dynamic Time Warping* (DTW) [24] 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. In particular, this method aims to find the alignment with minimum cost between two sequences X and Y. This cost is also called *optimal warping path* and it can be consider as a distance metric. In this paper, we will indicate the cost of an *optimal warping path* with $DTW(X, Y)$. *Hierarchical Clustering* is a cluster analysis method which seeks to build a hierarchy of clusters [18]. This clustering method has the distinct advantage that any valid measure of distance can be used. In fact, the observations themselves are not required: all that is used is a matrix of distances. *Random Forest* is an ensemble classifier [7] that combines a group of weak learners called "decision trees" to form a strong learner. In practice, it combines together the results of several decision trees trained with different portions of the training dataset and different subsets of features.

3. OUR FRAMEWORK

In this section we describe our framework. In particular, Section 3.1 introduces the pre-processing steps that allow us to model the network traffic. Section 3.2 describes the methodology used to build training and test datasets, and the procedure used to classify user actions.

3.1 Network Traffic Pre-Processing

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 given 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. 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. We model each network flow as 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.

Before generating for each flow the corresponding set of time series, a few pre-processing steps have to be performed. In particular: 1) we apply a domain filtering to select only flows belonging to the analyzed app; 2) we filter the remaining flows, in order to delete packets that may degrade the precision of our approach (i.e., we filter out ACK and retransmitted packets); 3) we limit the length of the generated time-series. In the following, we will detail these three pre-processing steps.

Domain filtering.

The network traffic generated by an app is generally directed toward a back-end infrastructure. The backend infrastructures might be composed by 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) do 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 can be involved in the communication. The same happens when the back-end is composed by 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 way. 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 parties services (such as Akamai or Amazon) that are indeed used by several applications [33].

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. It comes out that 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 3.2) will mitigate the consequences of missing packets. We filter out also other packets that do not bring any

additional information helpful in characterize flows. In particular, we filter out the three way handshake executed to open a TCP connection.

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 [30]. 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 portion of a flow, 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.

3.2 Classification of User Action

Since we use a supervised learning approach, it is necessary to create a labeled dataset that describes the user actions that we want to classify. In order to build this dataset, we simulate a series of user actions, and for each one we identify 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 an agglomerative Hierarchical Clustering method. 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 computation of the distance between two flows combines the distances of the corresponding time series.

Supposing that each flow f_i is decomposed into a set of n time series $\{T_1^i, \ldots, T_n^i\}$, 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 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 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 follow. 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 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.

4. EXPERIMENTAL RESULTS

In order to assess the performance of our proposal, we considered three widespread apps: Gmail, Facebook and Twitter. We select 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). We believe that the results of our analysis also hold for other apps that provide similar functionalities (e.g., Yahoo mail, WhatsApp or LinkedIn), while a thorough evaluation of this claim is left as future work. 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 4.1), the methodology used to collect the data (Section 4.2), and the results of the evaluation (Section 4.3).

4.1 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 de-

scribes 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[1], protocol type and TCP/IP flags. Since the payload is not relevant to our analysis, it has been omitted. This data have been then used to generate the time series as explained in Section 3.1.

4.2 Dataset Collection and Analysis

For our study we considered three apps installed from the official Android market: Gmail v4.7.2 , Facebook v3.8, and Twitter v4.1.10. For each app, 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 to have several friends/followers within the group. We avoided to configure 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.

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 records also 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 1). The list of these actions is reported in Table 1. We underline that we do not ignore other user actions, but we label them as *other*. In such a way we have several benefits [23]: we obtain a greater representation of data in terms of variety and variance of examples; we reduce the chances of overfitting; we improve the performance of the classifier on relevant user actions.

We collected and labeled the traffic generated by 220 sequences of actions for each app, where a sequence is composed by 50 types of actions (for a total of 11660 examples of actions for Gmail, 6600 for Twitter, and 10120 for Facebook). The user action examples in the dataset was divided in a training set and a test set. We use the training set to train the classifier, while we use the test set to evaluate its accuracy. We underline that to build the test set we used accounts that have not been used to create the training set. By using different accounts to generate the training and the test set, it is possible to assure that the results of the classification do not depend on the specific accounts that have been analyzed.

As explained in Section 3.1, each network flow is modeled as a set of time series. Table 2 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. In particular, the median value and the third quartile have been used as thresholds to limit the maximum length of the generated time series.

[1]00:00:00 UTC, 01 January 1970

Facebook

Action	Description
send message	send a direct message to a friend
post user status	post a status on the user's wall
open user profile	select user profile page from menu
open message	select a conversation on messages page
status button	select "*write a post*" on user's wall
post on wall	post a message on a friend's wall
open facebook	open the Facebook app

Gmail

Action	Description
send mail	send a new mail
reply button	tap on the reply button
open chats	select chats page from menu
send reply	send a reply to a received mail

Twitter

Action	Description
refresh home	Refresh the home page
open contacts	select contacts page on menu
tweet/message	publish a tweet or send a message
open messages	select direct messages page
open twitter	open the Twitter app
open tweets	select tweets page

Table 1: Description of the relevant actions for each app.

Apps	Sets	Weights	In	Out	Complete
Gmail	Conf. 1	0.80	[1,4]	[1,2]	[1,6]
		0.20	[1,6]	[1,3]	[1,9]
	Conf. 2	0.66	[1,4]	[1,2]	[1,6]
		0.33	[1,6]	[1,3]	[1,9]
	Conf. 3	0.33	[1,4]	[1,2]	[1,6]
		0.66	[1,6]	[1,3]	[1,9]
Facebook	Conf. 1	0.66	[1,3]	[1,5]	[1,7]
		0.33	[1,6]	[1,7]	[1,12]
	Conf. 2	0.33	[1,3]	[1,5]	[1,7]
		0.66	[1,6]	[1,7]	[1,12]
	Conf. 3	0.20	[1,3]	[1,5]	[1,7]
		0.80	[1,6]	[1,7]	[1,12]
Twitter	Conf. 1	0.95	-	-	[7,10]
		0.05	-	-	[1,10]
	Conf. 2	0.95	-	-	[8,11]
		0.05	-	-	[1,11]
	Conf. 3	0.95	-	-	[8,10]
		0.05	-	-	[1,10]

Table 2: Weights set configurations and packets intervals for Gmail, Facebook and Twitter apps.

In our experiments, we used the Random forest classifier implemented by the Python library *scikit-learn*[2]. The classifier is trained using 40 estimators (or weak learners). Each estimator consists in a decision tree without any restrictions on its depth limit. The number of features for each estimator is equal to the the square root of the maximum number of available features.

4.3 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 1 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

[2]`http://scikit-learn.org/`

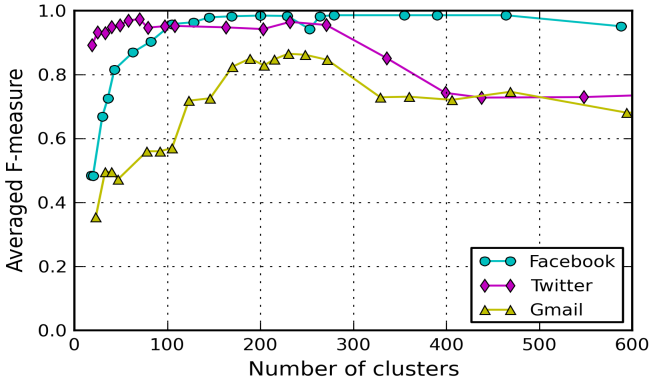


Figure 1: Classification accuracy over number of clusters.

Actions	Precision	Recall	F-measure
send message	1.00	1.00	1.00
post user status	1.00	0.95	0.97
open user profile	0.96	0.91	0.94
open message	0.98	1.00	0.99
status button	1.00	1.00	1.00
post on wall	1.00	0.98	0.99
open facebook	1.00	1.00	1.00
other	0.99	1.00	0.99
Average	0.99	0.98	0.99

Table 3: Classification results of Facebook actions by using Configuration 3.

app by app. In particular, we discuss the average accuracy reached when detecting each sensitive user action, we report detailed results for the precision, the recall and the F-measure metrics.

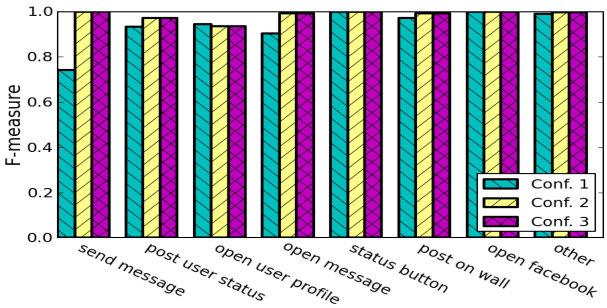
4.3.1 Facebook

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 2a. For each action at least one of the configurations exceeds 94% of accuracy, while the worst performing is always higher than 74%.

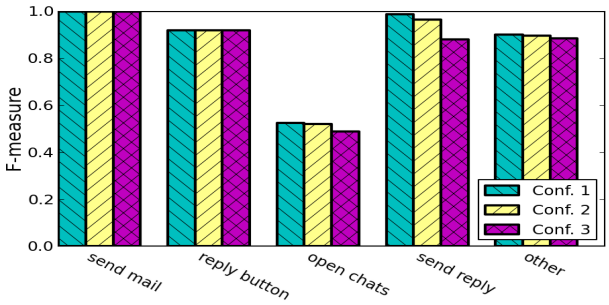
Table 3 reports precision, recall and F-measure reached by using Configuration 3. We noticed that all the actions have a precision higher 96%. The recall is higher than 95% for all the actions but the *open user profile*, that reaches 91%.

4.3.2 Gmail

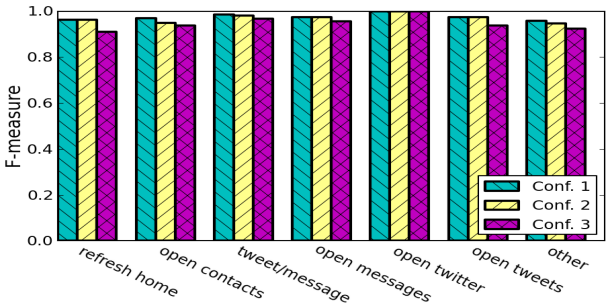
We analyzed four specific user actions of the Gmail app: *send mail*, *reply button*, *open chats* and *send reply*. Figure 2b shows the classification accuracy that has been reached. We observe that we are able to distinguish with high accuracy the action of sending of a new mail, from that of replying to a previously received message, as well as the tap over the reply button. The *open chats* action is instead more difficult to distinguish. Table 4 reports precision, recall and F-measure for different configurations of weights and packets intervals constraints. We can observe that the action *open chats* (that allows to read past chats) achieves a low precision but a high recall.



(a) Classification accuracy of the Facebook actions.



(b) Classification accuracy of the Gmail actions.



(c) Classification accuracy of the Twitter actions.

Figure 2: Classification accuracy of Facebook, Gmail and Twitter actions.

4.3.3 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 in common a large portion. Only the last three or four packets of each time series show some difference. Nevertheless, we have been able to reach outstanding results also for this app. In particular, we focus on six specific user actions: *refresh home*, *open contacts*, *tweet/message*, *open messages*, *open twitter*, *open tweets*.

On average, the F-measure is equal to 97%, with a precision and a recall of 98% and 97% respectively (see Table 5). Performance reached are reported in Figure 2c. 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 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 examples of the test set have been wrongly classified. Three of the six analyzed actions are correctly classified in more than the 99% of the cases. However, the other three actions,

Actions	Precision	Recall	F-measure
send mail	1.00	1.00	1.00
reply button	0.85	1.00	0.92
open chats	0.36	0.94	0.52
send reply	0.98	1.00	0.99
other	0.99	0.82	0.90
Average	0.83	0.85	0.86

Table 4: Classification results of Gmail actions reached by using Configuration 1.

Actions	Precision	Recall	F-measure
refresh home	0.94	0.99	0.96
open contacts	0.97	0.96	0.97
tweet/message	0.97	1.00	0.98
open messages	1.00	0.95	0.97
open twitter	1.00	1.00	1.00
open tweets	1.00	0.95	0.97
other	0.96	0.96	0.96
Average	0.98	0.97	0.97

Table 5: Classification results of Twitter actions reached by using the Configuration 1.

open contacts, *open messages* and *open tweets* are correctly classified in more than 95% of the cases.

5. POSSIBLE COUNTERMEASURES

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 [20]. 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 his mobile device, by simply eavesdropping the device's network traffic. While it is out of the scope of the paper to investigate possible countermeasure to the proposed attack, we discuss in the following some related issues.

One 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" [12]. 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 [34, 35]. However, a recent result showed that none of the existing countermeasures are effective [13]. 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 shown in this paper. Indeed, it is 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.

6. CONCLUSIONS

We proposed a framework to analyze encrypted network traffic and to infer which particular actions the user executed on some apps installed on his 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 thousand 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-anonimize 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 be adopted also on mobile devices.

7. ACKNOWLEDGMENTS

Mauro Conti was supported by Marie Curie Fellowship PCIG11-GA-2012-321980, funded by the European Commission for the PRISM-CODE project. This work has been partially supported by the TENACE PRIN Project number 20103P34XC funded by the Italian MIUR, by the Project "Tackling Mobile Malware with Innovative Machine Learning Techniques" funded by the University of Padua, and by the European Commission Directorate General Home Affairs, under the GAINS project, HOME/2013/CIPS/AG/4000005057. We would like to thank Fabio Aiolli, Michele Donini, and Mauro Scanagatta for their insightful comments.

References

[1] Androidrank. `http://www.androidrank.org/`.

[2] Top 15 most popular social networking sites, May 2014. `http://www.ebizmba.com/articles/social-networking-websites`, May 2014.

[3] C. A. Ardagna, M. Conti, M. Leone, and J. Stefa. An anonymous end-to-end communication protocol for mobile cloud environments. *Services Computing, IEEE Transactions on*, 2014.

[4] R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of ACM WWW*, 2006.

[5] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user navigation and interactions in online social networks. *Inf. Sci.*, July 2012.

[6] O. Berthold, H. Federrath, and M. Köhntopp. Project anonymity and unobservability in the internet. In *Proceedings of ACM CFP*, 2000.

[7] L. Breiman. Random forests. *Machine Learning*, 45, 2001.

[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of ACM CCS*, 2012.

[9] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of IEEE S&P*, 2010.

[10] M. Conti, N. Dragoni, and S. Gottardo. Mithys: Mind the hand you shake-protecting mobile devices from ssl usage vulnerabilities. In *Security and Trust Management*. Springer, 2013.

[11] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. Networkprofiler: Towards automatic fingerprinting of android apps. In *Proceedings of IEEE INFOCOM*, 2013.

[12] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[13] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of IEEE S&P*, 2012.

[14] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of USENIX OSDI*, 2010.

[15] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of ACM CCS*, 2012.

[16] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of ACM CCS*, 2012.

[17] Y. Go, D. F. Kune, S. Woo, K. Park, and Y. Kim. Towards accurate accounting of cellular data for tcp retransmission. In *Proceedings of ACM HotMobile*, 2013.

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning (2nd ed.)*. Springer, 2009.

[19] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *Proceedings of ACM CCSW*, 2009.

[20] B. Krishnamurthy. Privacy and online social networks: Can colorless green ideas sleep furiously? *IEEE Security and Privacy*, 2013.

[21] M. Liberatore and B. N. Levine. Inferring the source of encrypted http connections. In *Proceedings of ACM CCS*, 2006.

[22] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Proceedings of NDSS*, 2011.

[23] T. M. Mitchell. *Machine learning*. 1997.

[24] M. Müller. *Information Retrieval for Music and Motion*. Springer, 2007.

[25] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of ACM WPES*, 2011.

[26] J.-F. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*. Springer, 2001.

[27] B. P. Rocha, M. Conti, S. Etalle, and B. Crispo. Hybrid static-runtime information flow and declassification enforcement. *Information Forensics and Security, IEEE Transactions on*, 2013.

[28] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding online social network usage from a network perspective. In *Proceedings of ACM IMC*, 2009.

[29] C. Staff. Germany: U.s. might have monitored merkel's phone. `http://edition.cnn.com/2013/10/23/world/europe/germany-us-merkel-phone-monitoring/`, Oct. 2014.

[30] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of ACM WiSec*, 2013.

[31] G. Suarez-Tangil, J. E. Tapiador, P. Peris, and A. Ribagorda. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, 2013.

[32] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. In *Proceedings of IEEE ICDCS*, 2014.

[33] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of ACM Mobicom*, 2012.

[34] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of IEEE S&P*, 2008.

[35] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of NDSS*, 2009.

[36] Y. Zhauniarovich, G. Russello, M. Conti, B. Crispo, and E. Fernandes. Moses: Supporting and enforcing security profiles on smartphones. *Dependable and Secure Computing, IEEE Transactions on*, 2014.

[37] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of ACM CCS*, 2013.