Peel the onion: Recognition of Android apps behind the Tor Network

Authors hidden for double-blind review process

No Institute Given

Abstract. According to Freedom on the Net 2017 report [14] more than 60% of World's Internet users are not completely free from censorship. Solutions like Tor allow users to gain more freedom, bypassing these restrictions. For this reason they are continuously under deep observation to detect vulnerabilities that would compromise users anonymity. The aim of this work is showing that Tor is vulnerable to app deanonymization attacks on Android devices through network traffic analysis. While attacks against Tor anonymity have already gained considerable attention in the context of website fingerprinting in desktop environments, to the best of our knowledge this is the first work that addresses a similar problem on Android devices. For this purpose, we describe a general methodology for performing an attack that allows to deanonymize the apps running on a target smartphone using Tor. Then, we discuss a Proof-of-Concept, implementing the methodology, that shows how the attack can be performed in practice and allows to assess the deanonymization accuracy that it is possible to achieve. Moreover, we made the software of the Proof-of-Concept available, as well as the datasets used to evaluate it. In our extensive experimental evaluation, we achieved an accuracy of 97%.

Keywords: TOR, De-anonimization, Android, Traffic Analysis

1 Introduction

Tor is a very popular anonymization network, currently counting more than two million daily users [22]. While Tor is mainly associated with preserving anonymity during Web navigation, its protection capabilities are not limited to such application. In general, Tor can be used to protect any TCP-based traffic, being it generated by a desktop or mobile application. Nowadays, smartphone apps are replacing web browsers for interacting with many online services, such as social networks, chat services and video/audio streaming. The usage of anonymization mechanisms, such as Tor, on mobile devices is gaining momentum and is motivated by the increasing interest of several actors in profiling mobile users, e.g., for marketing purposes, government surveillance, detection and exploitation

of vulnerabilities and other activities that may be harmful for users' privacy and security, or perceived as such by them. Several works in the past studied the privacy guarantees offered by Tor, focussing, in particular, on the Desktop PC scenario where a large fraction of the anonymized traffic is web data or file sharing services. Conversely, less attention has been devoted to the usage of Tor on mobile devices, and the level of anonymity it can provide. The aim of this work is to show that Tor is vulnerable to app deanonymization attacks on Android devices through network traffic analysis. For this purpose, we describe a general methodology for performing an attack that allows to deanonymize the apps running on a target smartphone using Tor, which is the victim of the attack. Then, we discuss a proof-of-concept, implementing the methodology, that shows how the attack can be performed in practice and allows to assess the deanonymization accuracy that it is possible to achieve.

Summarizing, this work provides the following contributions:

- a methodology for deanonymizing apps on Android-based smartphones that use Tor;
- a Proof-of-Concept that implements the deanonymization methodology, which can be used to verify Tor's vulnerability to app deanonymization and assess the level of accuracy that can be achieved;
- a dataset[1] of generated Android Tor traffic traces that can be used to check the validity of our Proof-of-Concept and compare alternative methodologies.

The remainder of the paper is organized as follows. Section 2 reports the related works. Section 3 presents the fundamental concepts related to Tor and the machine learning algorithms employed in this work. Section 4 introduces the threat model that we consider. Section 5 discusses the methodology for deanonymizing Android apps behind the Tor network. Section 6 describes the Proof-of-Concept. Section 7 reports the experiment performed to evaluate the accuracy of the methodology and discusses the obtained results. Finally, in section 8 we draw some conclusions and discuss possible future directions for this work.

2 Related Works

Many works have been published in the broad area of traffic analysis both in the context of *desktop environments* and *smartphone environ-*

[1] Both the software necessary to reproduce the Proof-of-Concept and the dataset can be downloaded from the following repository: *[URL hidden for double-blind review process]*

ments (mostly assuming the Android operating system). While, there are some works in the context of desktop environments that has focused on deanonymizing Tor traffic, to the best of our knowledge, there is no work assuming both a smartphone environment and that traffic is anonymized through Tor. Therefore, there is no work we can directly compare to.

In this section we report the most related works considering a desktop environment, with or without Tor anonymized traffic, and an Android environment without Tor.

Desktop environment without Tor: In the context of website fingerprinting, Hintz [16] proposes an attack against SafeWeb, an encrypting web proxy, that allows to determine the webpages visited by the users. The attack exploits the fact that, even if traffic is encrypted, many browser open separate TCP connection for downloading resources from visited pages, allowing an attacker to monitor their sizes. Such sizes can be used to fingerprinting webpages. The author proposes some protections based on the addition of noise or on multiplexing data on a single connection.

Bissias *et al.* [10] propose a statistical website fingerprinting attack. The attacker creates a profile of the target website by monitoring the distribution of packet sizes and inter-arrival times. These data are then compared to user traffic.

Liberatore *et al.* [18] describe a website fingerprinting attack against HTTPS connections. They use unique packet lengths to build profiles of HTTPS connections and compare them against a dataset of known profiles using a naive Bayes classifier.

Desktop environment with Tor: In the context of Website fingerprinting, Wang *et al.* [28] propose an attack that uses a $k-$Nearest Neighbor Classifier to effectively fingerprint web pages behind Tor. They employ several types of features, including general statistics about total traffic, unique packet lengths, packet orderings, bursts and inter-packet times. They show that their attack has significantly higher accuracy than previous attacks in the same field.

AlSabah *et al.* [9] propose a machine learning based approach for Tor's traffic classification. The aim of the work is to recognize different classes of workloads that, in combination with QoS policies, can significantly improve the experience of Tor clients. However, since Tor's traffic is encrypted, it is not possible to rely on classical QoS to discriminate applications traffic. The proposed technique achieves an accuracy higher than 95%.

Juarez *et al.* [17] analyze the known website fingerprinting attacks on Tor. Known attacks claim to be effective under precise assumptions about threat model and user settings, which often do not hold in practical scenarios. The authors conduct a critical evaluation of these attacks and show their weaknesses when performed in real scenarios.

Chakravarty *et al.* [11] evaluate the feasibility and effectiveness of practical traffic analysis attacks on the Tor network using NetFlow data. It is not a passive attack as authors deliberately alter traffic characteristics at the server side and observe how this alteration affects client side through a statistical correlation. They achieve 100% accuracy in laboratory tests, and 81.4% accuracy in real world tests.

Ling *et al.* [19] propose TorWard, a system that attempts to recognize malicious traffic over Tor. In their experiments they found that a considerable portion of the Tor traffic is malicious (around 10%) with 8.99% of the alerts generated due to malware and 78.03% of the alerts generated due to malicious P2P traffic.

Mittal *et al.* [20] exploit throughput information to gain information about the user. The attack can identify the Guard Node (entry point to Tor network) and identify if two concurrent TCP connections belong to the same user.

Habibi Lashkari *et al.* [15] focus on recognition of traffic types instead of websites. They consider 8 application traffic types: browsing, email, chat, audio streaming, video streaming, file transfer, VoIP and P2P. They perform network traffic analysis by splitting the traffic traces in flows of a given duration. For each flow they compute several features based on inter-arrival times, active and idle periods, packet rates and byte rates. They employ a supervised machine learning approach to classify the traffic type of each flow. In particular they explored $k-$Nearest Neighbor, Random Forest and C4.5 classifiers.

Android environment without Tor: A number of authors have proposed various approaches to identify smartphone apps through network traffic analysis. Some of these solutions focus on examining IP addresses and packet payloads. However, relying on IP addresses is less effective because a lot of applications exploit Content Delivery Networks (CDN) for scalability. AppScanner [27] targets mobile environments and uses traffic features to fingerprint mobile apps. They rely on a supervised machine learning approach using only features that do not require the inspection packet payloads, thus working also on encrypted traffic. They perform experiments with SVM and Random Forest classifiers achieving 99% of

accuracy in their dataset with 110 of the most popular apps in the Google Play Store.

Dai *et al.* [13] propose a technique for app fingerprinting based on building network traffic profiles of apps. They run each app in an emulator, exercising different execution paths through a novel UI fuzzing technique, and collect the corresponding network traces. They compute a fingerprint of the app by identifying invariants in the generated network traces. Using the generated fingerprint they were able to detect the presence of apps in real-world network traffic logs from a cellular provider.

Conti *et al.* [12] describe a machine learning based network traffic analysis approach to identify user actions on specific apps (facebook, gmail and twitter). They achieve more than 95% of accuracy and precision for most of the considered actions.

Stöber *et al.* [26] focus on identifying smartphones from 3G/UMTS data capture. Even if 3G/UMTS data is encrypted an attacker could reliably identify a smartphone using only the information extracted from periodic traffic patterns leak side-channel information like timing and data volume. They show that they can identify smartphones with only 15 minutes of traffic monitoring and fingerprints computed on 6 hours of sniffed background traffic, obtaining an accuracy of 90%.

Saltaformaggio *et al.* [24] develop a tool called NetScope which is able to detect user activities on both Android and iOS smartphones. They compute features by only inspecting the IP headers, and use a SVM multi-class classifier to detect activities. NetScope achieves a precision of 78.04% and a recall of 76.04% on average on a set of 35 widely used apps.

3 Background on Tor

In this section we briefly summarize the basic concepts about the Tor network. Tor [23] is a distributed overlay network that anonymizes TCP-based applications (web browsers, secure shells, mail clients) while trying to keep the latency low. The network consists of a set of interconnected entities called *Onion Routers* (ORs). Tor clients, also known as *Onion Proxies* (OPs), periodically connect to directory servers to download the list of available ORs. OPs use this information to establish *circuits* in the Tor network, to connect to a destination node (which is often outside the Tor network). A circuit is a path of ORs in which each OR knows only its predecessor and its successor ORs. A Tor circuit has three types of nodes:

– *Entry* or *Guard Node*: this represents the entry point to the Tor network for the Tor client.
– *Relay Nodes*: these are the intermediate ORs of the circuit.
– *Exit Node*: this is the last OR in the Tor circuit. That is, the one that connects to the destination.

Each Tor circuit must have one entry node, at least one relay node (but there may be multiple) and one exit node. The entry node is the only node in the circuit that knows the Tor client, while the exit node is the only one that knows the destination.

Messages exchanged between the Tor client and the destination are split into *cells* when they traverse the Tor network. Cells are the basic unit of communication among Tor nodes. Tor cells used to have a 512 bytes fixed size in earlier Tor versions. Though this choice provided some resistance against traffic analysis, it was inefficient and made Tor traffic easier to discover due to packet-size distribution [23]. Therefore, variable length cells have been introduced in newer Tor versions.

When establishing a circuit, the Tor client shares a symmetric key with each node of the circuit. When the Tor client sends a packet to the destination it encrypts the corresponding cells' payloads with all the shared keys, in reverse order from the exit node to the entry node. Each node along the path unwraps its layer using its key. Only the exit node can reconstruct the message to be sent to the destination in clear. The same happens in the opposite direction, with each node that instead encrypts with its own key.

3.1 Padding

Internet service providers and surveillance infrastructures are known to store metadata about connections. Collecting and analyzing such data is useful for characterizing traffic, but may also represent a threat to anonymity.

Per-flow records are emitted by routers on a periodic basis depending on two configurable timeouts: *active flow timeout* and the *inactive flow timeout*. The expiration of the active flow timeout causes routers to emit a new record for each active connection. The inactive flow timeout causes the emission of a new record when a connection is inactive for a certain amount of time. The value of such timeouts is configurable and the range depends on routers vendors, but active flow timeout is typically in the order of minutes, while the inactive flow timeout in the order of tens of seconds. Therefore, the aggregation level of records data (on a temporal

basis) is at least the active flow timeout, but may be finer when there are inactive periods longer than the inactive flow timeout.

Thus, to reduce the granularity level of records' data (with the aim of hindering deanonymization techniques based on traffic analysis), long inactive periods should be avoided. For this reason, the Tor protocol introduced *connection padding*. With connection padding, special purpose cells (PADDING cells) are sent if the connection is inactive for a given amount of time, so as to reduce the duration of inactive periods.

Connection Padding Connection padding cells are exchanged only between the Tor client and entry node. To determine when to send a connection padding cell, both the Tor client and the entry node maintain a timer. These timers are set up with a timeout value between 1.5 and 9.5 seconds. The exact value depends on a function that samples a distribution described in [21]. After the establishment of the Tor circuit the timers start on both sides, if any of the two timers expires, a padding cell is sent to the other endpoint. Exchanging any cell different from a padding cell resets the timers.

Reduced Connection Padding Connection padding introduces an overhead in terms of exchanged data. Especially in mobile environments, this overhead may become excessive. Therefore, *reduced connection padding* has been introduced to lower the overhead due to connection padding. With reduced connection padding the timeout is sampled from a different range, between 9 seconds to 14 seconds.

4 Threat Model

In our threat model an *attacker* wants to deanonimize the apps on a *target smartphone* that uses Tor. That is, he/she wants to recognize which apps are being used by the target smartphone at any given time. We assume that the target is connected to the Internet through a wireless access point, either via a Wi-Fi LAN or via the cellular WAN, and that the attacker is able to passively capture the traffic between the target and the access point. We assume that the Tor client (i.e., an Onion Proxy) is installed in the smartphone itself and all apps' traffic passes through the Tor client.

5 Deanonymization Methodology

Figure 1 shows an overview of our methodology for deanonymizing Android apps behind Tor. The assumption at the basis of the methodology

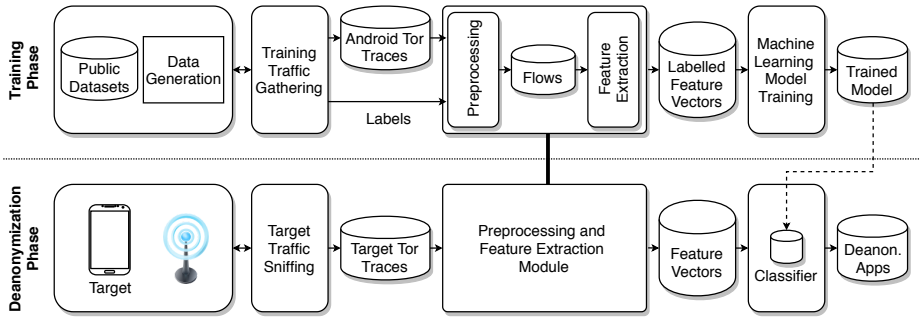

Fig. 1. Overview of the deanonymization methodology.

is that different apps produce different network traffic patterns, which are discernible, through proper network traffic analysis, even when the traffic is anonymized through Tor.

The methodology relies on a machine learning based network traffic analysis and consists of two distinct phases:

- *Training Phase*: during which we build a machine learning model of the distinctive characteristics of apps' Tor traffic. This is the preparation phase of the attack.
- *Deanonymization Phase*: during which we conduct the actual attack against the target, by monitoring the target's traffic and using the model built in the previous phase to recognize which apps the victim is using.

During the training phase we build a machine learning model of how different apps produce Tor traffic. We assume that the attacker is interested in recognizing a predefined set of apps $\mathcal{C} = \{app_1, \ldots, app_n\}$. If the target is using an app which is not included in $\mathcal{C}$, our methodology will not be able to recognize that app. Both the phases of our methodology include a *Traffic Gathering* and a *Preprocessing and Feature Exctraction* modules, followed by the building of the Machine Learning model for the Training Phase and Classifier module for the Deanonymization one. In the following sections we describe each logical block in details.

Traffic Gathering - Since we assume a supervised learning process, for the training phase, the first step is collecting a training dataset. In particular our methodology requires to gather, for each app in $\mathcal{C}$, raw Tor traffic traces. These traces can be picked from public datasets, if available (such as the one that we made available with this work), or can be generated

synthetically, as described in section 6.1. For the Deanonymization Phase instead, our methodology requires the attacker to passively capture the target's network traffic.

Preprocessing and Feature Extraction Module - This module processes the network traces gathered at the previous step and extracts the features that will be fed to the machine learning algorithm. For each network trace, we sort all TCP sessions (note that Tor only supports TCP) and we split sessions into flows. A *flow* is a portion of a TCP session of a predefined fixed duration T_F, the *flow timeout*. We split each TCP session into flows of T_F seconds. When, we find a TCP packet with the FIN flag set, we stop splitting. Thus, the last flow of each TCP connection may actually last less than T_F seconds. The flow timeout is a configurable parameter of our methodology that has an impact on the deanonymization accuracy. As detailed later, in section 7, we performed experiments with $T_F = 10$ and $T_F = 15$. The experiments with $T_F = 10$ yielded slightly better results. For the Training Phase, once we have split all traces into flows, we label each flow with the app in $\mathcal{C}$ that has generated the corresponding traffic.

For each flow x_i we compute a vector of features $v_i = (f_1(x_i), \ldots, f_m(x_i))$. Section 6.2 reports the set of features that we considered in our Proof-of-Concept. The general methodology does not rely on a particular set of features. However, as always, the choice of such set strongly impacts accuracy. Our set of features has been derived from an experimental analysis involving various feature sets. Since many machine learning algorithms (e.g., SVM and $k-$NN) work best with standardized features, for each component $y_{i,k} = f_k(x_i)$ (of each feature vector) we compute its standard score.

Machine Learning Model Training - During this step we feed the machine learning training algorithm with the training set built by the other modules. Our methodology does not rely on a particular machine learning model, but assumes a generic multi-class classifier whose set of classes is the set of apps $\mathcal{C}$. In our experiments we tested three different classifiers based on, respectively, Random Forest, $k-$Nearest Neighbors and SVM.

Classifier - In this step, each feature vector coming from the previous step is directly fed to the classifier that has been trained during the training stage. For each feature vector the classifier outputs a class, namely one of the apps in $\mathcal{C}$. The output of the classifier is also the output of the methodology, i.e., the deanonymized apps. In our Proof-of-Concept we adopt an *offline* approach. That is, the two phases are not concurrent,

they are performed subsequently. We first perform monitoring, collecting enough traces, and then we perform the classification. However, our methodology is general enough to allow for an *online* implementation, in which the two stages are actually executed simultaneously, and a new processing and classification step is performed as soon as the corresponding data is available.

6 Proof-of-Concept

This section presents details about our Proof-of-Concept implementing the methodology described in the previous section. We use a simple *architecture* made by a workstation, a wireless router connected to internet and two *target* smartphones connected to the router. On the targets we install *Orbot* [1], a proxy app that allows to use Tor on Android. The workstation is in charge of collecting the raw TCP traces, preprocessing them and extract feature vectors. We also use it to train the machine learning models and use them to deanonymize the network traffic.

6.1 Dataset

Since no public datasets collecting Android Tor's traces were available at the time of this writing, we generate our own datasets. To build them, we used AndroidViewClient [3], Culebra GUI and CulebraTester [4]. With these tools we developed different simulation scripts for each app, in order to reproduce a typical human user. We reported the details about simulated stimulation of the various apps in appendix A. In this way we can create, for each app in $\mathcal{C}$, a synthetic, yet as realistic as possible, network trace. To sniff the traffic and perform basic network analysis, we execute *Tcpdump* [2] on the router and *Wireshark* [8] on the workstation. We collected two datasets of network traces: 11.24 GB of traces with default configuration, that we call *Reduced Connection Padding Dataset* and 9.84 GB with the (full) connection padding activated, that we call *Full Connection Padding Dataset*, see section 3.1. In both datasets we collected about 4 hours of network traffic for each of the following apps: Dailymotion, Facebook, Instagram, Replaio Radio, Skype, Spotify, TorBrowser Alpha, Twitch, uTorrent, YouTube.

6.2 Features

In our Proof-of-Concept we employed three types of features.

Time-based Features - Since Tor's relay cells (those that transport the actual payload) are fixed sized, initially we concentrated on time-based, rather than size-based features. In particular, we employed the following features, given that they led to good results in the context of recognition of traffic classes in desktop environments [15]:

- *FIAT* (Forward Inter Arrival Time): time between two outgoing packets;
- *BIAT* (Backward Inter Arrival Time): time between two incoming packets;
- *FLOWIAT* (Flow Inter Arrival Time): time between two packets, no matter the direction;
- *Active time*: amount of time a flow is active;
- *Idle time*: amount of time a flow is idle;
- *Flow bytes per second*: number of bytes per second;
- *Flow packets per second*: number of packets per second;
- *Duration*: duration of the flow in seconds.

For all the above features except the last three, we actually compute 4 statistical values: minimum, maximum, mean and standard deviation. Moreover, the active and idle time depends on a configurable threshold, the *activity timeout* T_A. We performed experiments with $T_A = 2$ and $T_A = 5$ seconds.

Packet Direction and Burst Features - Packet direction and burst features have also been proven to be effective in the context of website fingerprinting on desktop environments [28]. Packet direction indicates whether a packet is going forward, from the source (the Tor client) to the destination, or backward, i.e., in the opposite direction. A burst instead is an uninterrupted sequence of packets in the same direction. After a preliminary analysis, we decided to enrich our feature set with the following features:

- Direction of the first 10 packets (of the flow);
- Incoming Bursts: number of bursts, bursts mean length, length of the longest burst;
- Outgoing Bursts: number of bursts, bursts mean length, length of the longest burst;
- Lengths of the first 10 incoming bursts;
- Lengths of the first 10 outgoing bursts.

Size-based Features - Event though relay cells are fixed sized, Tor uses variable-length cells for traffic control. As a preliminary analysis, we counted the number of packets for each packet size and we observed that, while there is a large variability in packet sizes, there is a relatively small set of possible packet sizes. Thus, we decided to introduce a feature for each of the ten most frequent packet sizes. These, were (in order of higher frequency) 1500, 595, 583, 2960, 1097, 1384, 151, 1126, 1109 and 233 bytes. We soon decided to discard size 2960, as this exceeds the MTU (1500 byes) and thus represents a reassembled packet. Each feature is a counter of the number of packets of that size observed in the flow.

7 Experimental Evaluation

We performed several experiments using the prototype implementation of our methodology described in the Proof-of-Concept section (see section 6). For each experiment we vary the following settings:

- *Tor's connection padding*: *Reduced* or *Full*, depending on whether we use the dataset with reduced connection padding or full connection padding (see section 6.1);
- *Flow Timeout* (T_F): either 10 or 15 seconds (see section 5);
- *Activity Timeout* (T_A): either 2 or 5 seconds (see section 6.2);
- *Presence of the Web Browser app*: Yes/No.

In particular, the last setting indicates whether the traces related to the usage of the web browser app are included in the experiment's dataset or not. The choice of performing experiments for both cases is motivated by the fact that, according to our experiments, the web browser app seems to be the most difficult to recognize among those due to the fact that each class of webpage can potentially have its own pattern that can be similar to apps of the same type. Thus its inclusion significantly reduces the accuracy of the methodology. Due to space constraints in this paper we discuss only the four most significant experiments (see Table 1). The results of the other experiments are available in the extended version of this work[2] (a brief summary is also reported in this paper in Appendix B). However, from these we drew the same general conclusions drawn from the first four experiments.

[2] *Citation removed for double-blind review process*

Table 1. Experiments discussed in this article (Flow Timeout and Activity Timeout are in seconds).

Experiment	Connection Padding	Flow Timeout	Activity Timeout	Web Browser
Experiment 1	Reduced	10	2	Yes
Experiment 2	Reduced	10	2	No
Experiment 3	Full	10	2	Yes
Experiment 4	Full	10	2	No

7.1 Evaluation Methodology

For each experiment we evaluate the performance achieved by our Proof-of-Concept, namely the performance of the classifier. We asses both the overall performance of the classifier and the performance achieved on a per-class basis, so as to highlight whether some apps are more easily recognized than others. The per-class performance are computed in terms of precision, recall, F1 score and accuracy computed for each class in $\mathcal{C}$. The overall classifier performance are computed by averaging the per-class metrics. Note that precision, recall and F1 score are averaged according to two criteria: micro and macro. The two criteria account differently for imbalances in the dataset (i.e., uneven proportion of samples per classes). The micro criteria biases the corresponding metrics towards the most populated classes, while the macro criteria treats all classes equally [25]. Note, that micro precision and micro recall (and thus micro F1 score), are mathematically equivalent. Thus, when presenting the results of the experiments we will only report the micro F1 score.

7.2 Results

In this section we present the results of the experimental evaluation.

Global evaluation - Table 2 shows a comparison of the results obtained

Table 2. Summary of the results of Experiments 1-4.

Experiment	Avg. Accuracy	Micro F1	Macro Precision	Macro Recall	Macro F1
Experiment 1	0.968	0.840	0.834	0.830	0.832
Experiment 2	0.969	0.859	0.859	0.852	0.855
Experiment 3	0.972	0.861	0.857	0.849	0.853
Experiment 4	0.973	0.880	0.877	0.872	0.875

in each experiment through this classifier. In all experiments we achieved

Table 3. Per-class performance of each classifier for Experiment 1.

	Random Forest				$k-$NN				SVC			
APP	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**
dailymotion	0.83	0.77	0.8	0.96	0.56	0.58	0.57	0.91	0.74	0.72	0.73	0.95
facebook	0.9	0.84	0.87	0.98	0.62	0.7	0.66	0.94	0.86	0.85	0.86	0.97
instagram	0.79	0.86	0.82	0.94	0.58	0.67	0.62	0.88	0.77	0.83	0.8	0.94
replaio_radio	0.99	0.98	0.98	1.0	0.98	0.96	0.97	0.99	0.98	0.98	0.98	0.99
skype	0.99	0.96	0.97	1.0	0.97	0.94	0.95	0.99	0.98	0.95	0.97	0.99
spotify	0.67	0.65	0.66	0.94	0.56	0.48	0.52	0.92	0.63	0.66	0.65	0.93
torbrowser	0.68	0.77	0.72	0.97	0.6	0.47	0.53	0.95	0.67	0.71	0.69	0.96
twitch	0.83	0.87	0.85	0.97	0.68	0.76	0.71	0.93	0.83	0.83	0.83	0.96
utorrent	0.9	0.91	0.9	0.98	0.82	0.69	0.75	0.96	0.85	0.84	0.85	0.97
youtube	0.76	0.69	0.72	0.95	0.61	0.57	0.59	0.93	0.72	0.63	0.67	0.95

Table 4. Per-class performance of each classifier for Experiment 2.

	Random Forest				$k-$NN				SVC			
APP	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**
dailymotion	0.83	0.78	0.8	0.96	0.56	0.58	0.57	0.9	0.74	0.72	0.73	0.94
facebook	0.9	0.84	0.87	0.98	0.65	0.71	0.68	0.94	0.86	0.85	0.85	0.97
instagram	0.79	0.87	0.83	0.94	0.6	0.67	0.63	0.88	0.77	0.82	0.79	0.93
replaio_radio	0.99	0.98	0.99	1.0	0.98	0.96	0.97	0.99	0.98	0.98	0.98	0.99
skype	0.99	0.96	0.98	1.0	0.98	0.95	0.96	0.99	0.98	0.96	0.97	0.99
spotify	0.72	0.74	0.73	0.95	0.6	0.5	0.55	0.92	0.67	0.72	0.69	0.94
twitch	0.84	0.87	0.85	0.97	0.68	0.76	0.72	0.93	0.84	0.83	0.83	0.96
utorrent	0.9	0.93	0.91	0.98	0.83	0.71	0.77	0.96	0.87	0.87	0.87	0.97
youtube	0.77	0.71	0.74	0.95	0.63	0.56	0.59	0.93	0.73	0.66	0.69	0.95

the best results with the Random Forest classifier. In all experiments we obtained comparable accuracy (~ 0.97).

Per-app evaluation - Tables 3-6 show the per-app result of each experiment. For all classifiers, we observe a certain variability in how accurate the classifier is in recognizing the various apps. Looking at the F1 score, Spotify, Tor Browser and YouTube appear to be the most difficult apps to recognize. Indeed, by looking directly at the data, we observed that these three apps are often confused, one for another. Since both Spotify and YouTube provide streaming contents, they probably generate strongly similar traffic patterns, that mislead the classifiers. The same reasoning probably applies to Tor Browser. Indeed, webpages may embed streaming content, including YouTube videos themselves. Moreover, in experiments 3 and 4, by looking at the F1 score, we observe that the apps that mislead the classifiers the most are Facebook, Instagram and Tor Browser. This is not surprising. Indeed, if we think of the typical usage patterns of the apps that we considered in our experiments, Facebook, Instagram and Tor Browser are the ones with the largest idle periods (the user "think time"), as opposed to the other apps, that mainly provide streaming content (typ-

Table 5. Per-class performance of each classifier for Experiment 3.

	Random Forest				$k-$NN				SVC			
APP	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**
dailymotion	0.87	0.81	0.84	0.97	0.66	0.72	0.69	0.95	0.8	0.77	0.78	0.97
facebook	0.78	0.72	0.75	0.96	0.48	0.52	0.5	0.91	0.71	0.72	0.72	0.95
instagram	0.72	0.7	0.71	0.94	0.47	0.47	0.47	0.89	0.67	0.7	0.68	0.93
replaio_radio	0.95	0.97	0.96	0.99	0.88	0.94	0.91	0.98	0.96	0.96	0.96	0.99
skype	0.98	0.95	0.97	0.99	0.93	0.92	0.93	0.98	0.98	0.94	0.96	0.99
spotify	0.82	0.84	0.83	0.96	0.69	0.68	0.68	0.93	0.76	0.78	0.77	0.95
torbrowser	0.81	0.69	0.75	0.97	0.64	0.35	0.45	0.94	0.78	0.67	0.72	0.97
twitch	0.86	0.92	0.89	0.98	0.68	0.77	0.72	0.94	0.86	0.87	0.87	0.97
utorrent	0.99	0.99	0.99	1.0	0.94	0.98	0.96	0.99	0.98	0.99	0.98	1.0
youtube	0.79	0.88	0.83	0.96	0.68	0.65	0.66	0.92	0.79	0.82	0.8	0.95

Table 6. Per-class performance of each classifier for Experiment 4.

	Random Forest				$k-$NN				SVC			
APP	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**	**PR.**	**REC.**	**F1**	**ACC.**
dailymotion	0.88	0.82	0.85	0.97	0.67	0.71	0.69	0.94	0.81	0.77	0.79	0.96
facebook	0.82	0.73	0.77	0.96	0.51	0.52	0.52	0.91	0.73	0.73	0.73	0.95
instagram	0.75	0.71	0.73	0.94	0.51	0.49	0.5	0.89	0.69	0.7	0.69	0.93
replaio_radio	0.95	0.97	0.96	0.99	0.88	0.94	0.91	0.98	0.96	0.96	0.96	0.99
skype	0.98	0.96	0.97	0.99	0.96	0.92	0.94	0.98	0.97	0.95	0.96	0.99
spotify	0.84	0.86	0.85	0.97	0.73	0.69	0.71	0.94	0.8	0.81	0.8	0.95
twitch	0.87	0.92	0.9	0.98	0.71	0.77	0.74	0.94	0.88	0.87	0.87	0.97
utorrent	0.99	0.99	0.99	1.0	0.96	0.98	0.97	0.99	0.98	0.99	0.99	1.0
youtube	0.81	0.88	0.84	0.96	0.71	0.65	0.68	0.92	0.8	0.83	0.81	0.95

ically, with less frequent and shorter idle periods). Since the connection padding mechanism is activated by idle periods, it is normal to observe a performance degradation when using full connection padding rather than the reduced one.

Result Summary - As expected, all performance metrics slightly improve when we do not consider the Tor Browser app (see Table 2). Indeed, the type of the visited website strongly impacts on the characteristics of the generated traffic, which makes this app sometimes be confused with other apps. For example, when visiting a webpage with streaming content the Tor Browser app might be confused with a streaming app (such as Spotify or YouTube). A counterintuitive result that we obtained is that apparently the use of Tor's (full) connection padding actually improved the accuracy over the used reduced connection padding. If we look at the per-class results (Tables 3-6) we notice that the performance on Facebook and Instagram apps actually worsen significantly. Also the recall of the Tor Browser app worsen significantly, though its precision improves, which means that the proportion of false negatives increases (the app is more often confused with others), while the number of false positives

decreases (other apps are less frequently confused with Tor Browser). The fact that these three apps are more often misclassified when using full padding is what we expected. Indeed, as already pointed out, their typical use patterns involve more frequent "think times" and, thus, idle periods, which trigger the connection padding mechanism. On the other hand, the other apps are mainly characterized by a "streaming" pattern, thus involving extremely less frequent idle periods, which explains why for the majority of them the performance does not worsen. However, it does not explain why they improve. Clearly, the padding mechanism has a strong impact on the time-based features (see section 6.2), especially the active/idle time. Our guess is that the full padding mechanism is actually activated statistically more often for some of these streaming apps and less often for others, which actually results in a better separation of the corresponding classes. We plan to better investigate this aspect as future work.

8 Conclusion

In this work we have shown that Tor when used on Android devices is vulnerable to app deanonymization. We described a general methodology to perform an attack against a target smartphone which allows to unveil which apps the victim is using. The proposed methodology performs network traffic analysis based on a supervised machine learning approach. It leverages the fact that different apps produce different recognizable traffic patterns even when protected by Tor. We also provided a Proof-of-Concept that implements the methodology, that we employed to assess the accuracy that it can achieve in deanonymizing apps. We performed several experiments achieving an accuracy of 97.3% and a F1 score of 87.5%. We made the software of the Proof-of-Concept, as well as the datasets that we built during the experiments, publicly available, so that it can be used to assess Tor's vulnerability to this attack, compare alternative methodologies and test possible countermeasures.

As future work we plan to experiment with additional machine learning algorithms. Moreover, in this work we adopted a multi-class classifier approach. That is, we trained a single classifier on all possible classes. We plan to extend our experimental evaluation by testing alternative binary-class approaches (such as *one-vs-all* and *one-vs-one*), in which we employ several binary classifiers in place of a single multi-class classifier. Another improvement to this work may be to enlarge the datasets with a richer set of apps.

References

1. Orbot: Tor for android (2018), https://guardianproject.info/apps/orbot/
2. Tcpdump (2018), https://www.tcpdump.org/
3. Androidviewclient (2019), https://github.com/dtmilano/AndroidViewClient
4. Culebra (2019), http://culebra.dtmilano.com/
5. The majestic million (2019), https://majestic.com/reports/majestic-million
6. Socialblade.com top 500 most followed profiles (sorted by followers count) (2019), https://socialblade.com/instagram/top/500/followers
7. Socialblade.com top 500 most liked facebook pages (sorted by count) (2019), https://socialblade.com/facebook/top/500/likes
8. Wireshark (2019), https://www.wireshark.org/
9. AlSabah, M., Bauer, K., Goldberg, I.: Enhancing tor's performance using real-time traffic classification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 73–84. CCS '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2382196.2382208, http://doi.acm.org/10.1145/2382196.2382208
10. Bissias, G.D., Liberatore, M., Jensen, D., Levine, B.N.: Privacy vulnerabilities in encrypted http streams. In: Proceedings of the 5th International Conference on Privacy Enhancing Technologies. pp. 1–11. PET'05, Springer-Verlag, Berlin, Heidelberg (2006). https://doi.org/10.1007/11767831(_)1
11. Chakravarty, S., Barbera, M.V., Portokalidis, G., Polychronakis, M., Keromytis, A.D.: On the effectiveness of traffic analysis against anonymity networks using flow records. In: Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362. pp. 247–257. PAM 2014, Springer-Verlag, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-319-04918-2(_)24, https://doi.org/10.1007/978-3-319-04918-2(_)24
12. Conti, M., Mancini, L.V., Spolaor, R., Verde, N.V.: Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. pp. 297–304. CODASPY '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2699026.2699119, http://doi.acm.org/10.1145/2699026.2699119
13. Dai, S., Tongaonkar, A., Wang, X., Nucci, A., Song, D.: Networkprofiler: Towards automatic fingerprinting of android apps. pp. 809–817 (04 2013). https://doi.org/10.1109/INFCOM.2013.6566868
14. Freedom on the Net: 2017 report (2017), https://freedomhouse.org/report/freedom-net/freedom-net-2017
15. Habibi Lashkari, A., Draper Gil, G., Mamun, M.S.I., Ghorbani, A.A.: Characterization of tor traffic using time based features. In: Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,. pp. 253–262. INSTICC, SciTePress (2017). https://doi.org/10.5220/0006105602530262
16. Hintz, A.: Fingerprinting websites using traffic analysis. In: Proceedings of the 2Nd International Conference on Privacy Enhancing Technologies. pp. 171–178. PET'02, Springer-Verlag, Berlin, Heidelberg (2003), http://dl.acm.org/citation.cfm?id=1765299.1765312
17. Juarez, M., Afroz, S., Acar, G., Diaz, C., Greenstadt, R.: A critical evaluation of website fingerprinting attacks. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 263–274. CCS '14,

ACM, New York, NY, USA (2014). https://doi.org/10.1145/2660267.2660368, http://doi.acm.org/10.1145/2660267.2660368

18. Liberatore, M., Levine, B.N.: Inferring the source of encrypted http connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 255–263. CCS '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1180405.1180437, http://doi.acm.org/10.1145/1180405.1180437
19. Ling, Z., Luo, J., Wu, K., Yu, W., Fu, X.: Torward: Discovery of malicious traffic over tor. IEEE INFOCOM 2014 - IEEE Conference on Computer Communications pp. 1402–1410 (2014)
20. Mittal, P., Khurshid, A., Juen, J., Caesar, M., Borisov, N.: Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. pp. 215–226. CCS '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2046707.2046732, http://doi.acm.org/10.1145/2046707.2046732
21. Perry, M.: Tor padding specification (2019), https://gitweb.torproject.org/torspec.git/tree/padding-spec.txt
22. Project, T.: Tor metrics, https://metrics.torproject.org/, accessed: Jan. 2019
23. Roger Dinledine, Nick Mathewson, S.M., Syverson, P.: Tor: The second-generation onion router (2014 draft v1) (2014), \url{"https://murdoch.is/papers/tor14design.pdf"}
24. Saltaformaggio, B., Choi, H., Johnson, K., Kwon, Y., Zhang, Q., Zhang, X., Xu, D., Qian, J.: Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In: Proceedings of the 10th USENIX Conference on Offensive Technologies. pp. 69–78. WOOT'16, USENIX Association, Berkeley, CA, USA (2016), http://dl.acm.org/citation.cfm?id=3027019.3027026
25. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Inf. Process. Manage. **45**(4), 427–437 (Jul 2009). https://doi.org/10.1016/j.ipm.2009.03.002, http://dx.doi.org/10.1016/j.ipm.2009.03.002
26. Stöber, T., Frank, M., Schmitt, J., Martinovic, I.: Who do you sync you are?: Smartphone fingerprinting via application behaviour. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 7–12. WiSec '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2462096.2462099, http://doi.acm.org/10.1145/2462096.2462099
27. Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I.: Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P). pp. 439–454 (March 2016). https://doi.org/10.1109/EuroSP.2016.40
28. Wang, T., Cai, X., Nithyanand, R., Johnson, R., Goldberg, I.: Effective attacks and provable defenses for website fingerprinting. In: Proceedings of the 23rd USENIX Conference on Security Symposium. pp. 143–157. SEC'14, USENIX Association, Berkeley, CA, USA (2014), http://dl.acm.org/citation.cfm?id=2671225.2671235

A User Simulation

This section describes how we simulated the user interaction in our Proof-of-Concept.

Tor Browser The user activity on the Tor Browser app has been simulated through a python script that visits webpages randomly sampled from a list of the top 10,000 sites extracted from the Majestic Million dataset [5]. The script spend a randomly drawn amount of time on each webpage, before navigating to the next one.

Instagram To simulate the user interaction with Instagram, we created a new account and added the Socialblade's top 500 most followed profiles [6]. The simulation script generates random swipe inputs on the Instagram app to scroll the main page up and down with random delays. Swipe down inputs are generated with higher probability than swipe up inputs, as a user browsing Instagram posts would typically scroll the page from top to bottom. After a random number of swipes there is a 30% probability that the user decides to visit another random profile, or otherwise a 30% probability that the user will push the like button on the current Instagram post.

Facebook The simulation of the user interaction with the Facebook app is very similar to that of Instagram. First we create a Facebook account for the user and we add a list of followed pages derived from Socialblade's top 500 most liked Facebook Pages [7]. Similarly to that of Instagram, the simulation script scrolls the posts in the main page of the Facebook app, by generating random swipe inputs with random delays. After a random number of swipes there is a 30% probability that the user pushes the like button on the post showing on the screen.

Skype Skype calls have been generated by starting calls with an audio source near the smartphone microphone.

uTorrent The uTorrent app is a Torrent client and, therefore, it does not require a complex user interaction. We simply add some torrent file to the app, and it starts the download.

Dailymotion, Replaio Radio, Spotify, Twitch, YouTube Also this apps do not require a very complex interaction with the user. We start each app on some streaming content and leave the app in execution.

B Experiments Result Summary

Table 7 shows the settings of all the experiments that we performed and a summary of the results obtained.

Table 7. Complete set of experiments with results (Flow Timeout and Activity Timeout are in seconds).

Experiment	Connection Padding	Flow Timeout	Activity Timeout	Web Browser	Avg. Accuracy	Micro F1	Macro Precision	Macro Recall	Macro F1
Experiment 1	Reduced	10	2	Yes	0.968	0.840	0.834	0.830	0.832
Experiment 2	Reduced	10	2	No	0.969	0.859	0.859	0.852	0.855
Experiment 3	Full	10	2	Yes	0.972	0.861	0.857	0.849	0.853
Experiment 4	Full	10	2	No	0.973	0.880	0.877	0.872	0.875
Experiment 5	Reduced	10	5	Yes	0.969	0.844	0.836	0.833	0.835
Experiment 6	Reduced	10	5	No	0.969	0.860	0.860	0.854	0.857
Experiment 7	Full	10	5	Yes	0.972	0.862	0.858	0.849	0.854
Experiment 8	Full	10	5	No	0.973	0.878	0.876	0.871	0.873
Experiment 9	Reduced	15	2	Yes	0.970	0.852	0.851	0.844	0.847
Experiment 10	Reduced	15	2	No	0.970	0.866	0.871	0.861	0.866
Experiment 11	Full	15	2	Yes	0.975	0.876	0.874	0.859	0.867
Experiment 12	Full	15	2	No	0.976	0.890	0.888	0.878	0.883
Experiment 13	Reduced	15	5	Yes	0.971	0.853	0.851	0.844	0.847
Experiment 14	Reduced	15	5	No	0.972	0.873	0.877	0.868	0.873
Experiment 15	Full	15	5	Yes	0.976	0.878	0.875	0.861	0.868
Experiment 16	Full	15	5	No	0.976	0.893	0.891	0.881	0.886