IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Mind your wallet's privacy: Identifying bitcoin wallet apps and user's actions through network traffic analysis

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1870173 since 2022-07-20T07:54:32Z

Publisher:

Association for computing machinery

Published version:

DOI:10.1145/3297280.3297430

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

07 January 2023

ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/329220825

Mind Your Wallet's Privacy: Identifying Bitcoin Wallet Apps and User's Actions through Network Traffic Analysis.

Conference Paper · April 2019

DOI: 10.1145/3297280.3297430

CITATIONS
17

READS
6,743

4 authors:

Fabio Aiolli
University of Padova
107 PUBLICATIONS **1,094** CITATIONS
SEE PROFILE

Mauro Conti
University of Padova
475 PUBLICATIONS **12,583** CITATIONS
SEE PROFILE

Ankit Gangwal
20 PUBLICATIONS **213** CITATIONS
SEE PROFILE

Mirko Polato
Università degli Studi di Torino
44 PUBLICATIONS **424** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Safe transition towards the Internet of the future / Securing the Transition Toward the Future Internet View project

FIU Cyber-Physical Systems Security View project

All content following this page was uploaded by Ankit Gangwal on 07 May 2019.

The user has requested enhancement of the downloaded file.

Mind Your Wallet's Privacy: Identifying Bitcoin Wallet Apps and User's Actions through Network Traffic Analysis

Fabio Aiolli
aiolli@math.unipd.it

Mauro Conti
conti@math.unipd.it

Ankit Gangwal*
ankit.gangwal@phd.unipd.it

Mirko Polato
mpolato@math.unipd.it

ABSTRACT

With the surge in popularity of cryptocurrencies, Bitcoin has emerged as one of the most promising means for remittance, payments, and trading. Supplemented by the convenience offered by the smartphones, an increasing number of users are adopting Bitcoin wallet apps for different purposes.

In this paper, we focus on identifying user activities on smartphone-based Bitcoin wallet apps that are commonly used for sending, receiving, and trading Bitcoin. To accomplish our goal, we performed network traffic analysis using machine learning techniques. Since we focus on apps of the same type/functionality, it makes our classification problem even more difficult compared to classifying apps tailored for discrete purposes. Moreover, our goal is to identify user activities even in the presence of encryption. In our experiments, we considered the worldwide most downloaded Bitcoin wallet apps on both Google Play Store and Apple's App Store. For collecting network traffic traces, we used only physical hardware and omitted any emulator to build our experiment scenario as close to the real environment as possible. We process the traffic traces in several phases before extracting the features that are utilized to train our supervised learning algorithms. We deal with the classification problem in multiple stages in a hierarchical fashion. We ran a thorough set of experiments to assess the performance of our system and attained nearly 95% accuracy in user activity identification.

CCS CONCEPTS

• **Computing methodologies** → **Feature selection**; • **Applied computing** → **Network forensics**; • **Security and privacy** → *Web application security*;

KEYWORDS

Android, Bitcoin, iOS, Machine learning, Traffic analysis.

ACM Reference Format:
Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. 2019. Mind Your Wallet's Privacy: Identifying Bitcoin Wallet Apps and User's Actions through Network Traffic Analysis. In *the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297280.3297430

*Corresponding author

All authors are affiliated with the Department of Mathematics, University of Padua, 35121, Italy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC '19, April 8–12, 2019, Limassol, Cyprus
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297430

1 INTRODUCTION

Bitcoin – the first cryptocurrency – has gained enormous attention when its price raised from approximately $1,000 in March 2017 to nearly $19,500 in December 2017 [5]. The market capitalization of Bitcoin also grew dramatically and reached a level of $326 billion in Q4 2017. This growth in the value of Bitcoin attracted masses to trade for this cryptocurrency, and platforms such as Coinbase enabled users to trade for it conveniently. Moreover, several online shopping websites, supermarkets, *etc.* accept Bitcoin as a valid mode of payment. Consequently, an increasing number of users are adopting Bitcoin wallet apps for different purposes such as payment, remittance, and trading.

On another side, network traffic analysis has been extensively exploited by adversaries to create user profiles and deduce sensitive information such as which airline does the user fly with, which financial institution does the user banks with, or which company provides insurance to the user. User profiling is also used for other non-malicious activities such as network performance optimization in the culture of bring-your-own-device [20]. For traffic analysis, traditional TCP/IP traffic may be identified using port information because applications tend to use "well-known" port numbers that are typically reserved for standard services. Furthermore, to distinguish multiple sources of data-traffic from services that utilize the same port number (*e.g.,* Internet browsing), it may sometimes adequate to inspect the HTTP headers, in particular, IP addresses to identify the communicating peers. However, traffic analysis in the domain of the smartphones is complex because several apps exchange data using HTTP/HTTPS. If the developers choose to use HTTPS, the communication is encrypted, and thus, conventional techniques of inspecting the traffic cannot help in the task of traffic identification. Moreover, many apps as well as ad networks use Content Distribution Network (CDN) for scalable delivery of content and furnish APIs to their applications. Using CDNs and APIs may cause different apps to communicate via the same IP address (or IP range), which consequently hinders the identification techniques that solely depend on the IP addresses.

Motivation and research scope: We motivate our work by outlining one of the most critical situations arising in today's cyber-space: pseudo-anonymity offered by the Bitcoin system makes it difficult for law-enforcing agencies to trace the masterminds behind modern cyber-crimes, *e.g.,* ransomware campaigns, which are increasing day-by-day. We believe that our work can assist in the hunt of cyber-criminals by monitoring (or at least by filtering) the potential Bitcoin wallet users. Most importantly, our work can be extended to other categories of smartphone apps to improve user profiling even further.

Contributions: In this paper, we propose our approach to identify activities of Bitcoin wallet users via network traffic analysis. In particular, we recognize user actions within Bitcoin wallet apps. Our approach can identify user activities even in the presence of encryption.

Organization: The remainder of this paper is organized as follows. Section 2 discusses the previous works on traffic analysis. Section 3 elaborates our system's design. Section 4 covers the details of our classifier. We present and discuss our results in Section 5. Finally, Section 6 concludes the paper.

2 RELATED WORK

Since our work relies on network traffic analysis, we will primarily discuss the previous works related to it. Network traffic analysis using machine learning techniques has been an active area of research. Several attempts have been made to analyze network traffic from workstations, smartphones, *etc.* On the surface, smartphones' traffic analysis may appear as a sheer translation of existing works for workstations. However, several studies [11, 12, 17, 30] have concluded that despite having similarities (*e.g.,* end-to-end communication using via IP addresses and ports), there are nuances in the characteristics of the traffic generated by smartphones. Here, we will discuss the works related only to the smartphones; the main focus of our work. However, the interested readers may refer to the works [3, 13, 14, 18, 21, 23] to comprehend traffic analysis on workstations.

In the domain of the smartphones, network traffic analysis has been effectively utilized to leak sensitive user data [10], to find device's location [2], and to profile users based on the apps installed on their device [26]. Dai *et al.* [9] propose NetworkProfiler to automate profiling and identification of Android apps. It scrutinizes HTTP payload, and thus the approach is not adequate when the payload is encrypted. Wang *et al.* [29] present an approach for inspecting encrypted 802.11 frames to identify apps from App Store. Qazi *et al.* [22] propose a framework, called Atlas, to recognize Android apps using network flows obtained by leveraging SDN's data reporting. Mongkolluksamee *et al.* [19] use communication patterns and packet size distribution to distinguish among distinct Android apps. Alan and Kaur [1] use TCP/IP headers of the first 64 packets generated upon app launch to identify Android apps. Taylor *et al.* [27] showed that a passive eavesdropper can recognize Android apps by fingerprinting network traffic. Conti *et al.* [6] suggest eavesdropping encrypted network traffic to classify user actions within the scope of different Android apps. Similarly, Saltaformaggio *et al.* [24] propose NetScope, which also examines encrypted traffic to identify user activity. The work presented in [8] focuses on iMessage and three other third-party messaging apps for iOS to detect messaging activity. Zhou *et al.* [31] target a specific user action (*i.e.,* send a tweet) on the Twitter app installed on an Android smartphone.

To summarize, existing solutions either focus on apps from different categories that inherently generate distinct network traffic or consider a particular smartphone platform. Our work is different from the state-of-the-art on two dimensions: (1) we focus on apps of same type/functionality, which makes classification problem even more difficult; (2) we consider both Android and iOS operating system, which collectively covers the majority of smartphone users.

3 SYSTEM DESIGN

We elucidate our decision for selecting the smartphones, apps, and their actions in Section 3.1. We also elaborate our equipment setup in Section 3.2.

3.1 Smartphone, app, and action selection

According to the report[1] by Gartner, Android and iOS devices together accounted for 99.9% of all smartphone sales by the end of the year 2017. Hence, in our study, we used both Android (Samsung Galaxy S5 running Android 6.0.1) and iOS (iPhone 5 running iOS 10.3.3) smartphones. Table 1 lists the worldwide most downloaded[2] Bitcoin wallet apps on both Google Play Store and Apple's App Store in the year 2017.

Table 1: Top 10 most downloaded Bitcoin wallet apps

N.	Google Play Store	App Store
1	Coinbase	Coinbase
2	Zebpay	Blockchain
3	Bitcoin Wallet (Bitcoin.com)	Bread
4	Luno	Bitcoin Wallet (Bitcoin.com)
5	Xapo	Xapo
6	Unocoin	BitPay
7	Mycelium	Zebpay
8	Wirex	Wirex
9	Bitcoin Wallet (Bitcoin Wallet Devs)	BTC.com
10	BTC.com	Copay

From the apps listed in Table 1, we considered nine apps. These apps are listed in Table 2. The apps we omitted in our experiments are either country restricted (*e.g.,* Unocoin requires Indian phone number and tax code to register) or available for both the platforms with identical features (*e.g.,* Wirex). However, as the representative of the later class of apps, we included Bitcoin Wallet (Bitcoin.com) app for both the platforms. For non-Bitcoin apps, we chose the top-10 apps along with additional 20 Internet-dependent apps from the respective official application store of each platform. It is important

[1] gartner.com/newsroom/id/3859963
[2] sensortower.com/blog/bitcoin-wallet-app-growth

to mention that these numbers do not include system apps and the apps that do not require the Internet, *e.g.,* calculator app.

Table 2: Apps used for classification

	Android	iOS
Wallet apps	BTC.com	BitPay
	Bitcoin Wallet (Bitcoin.com)	
	Coinbase	Blockchain
	Luno	Bread
	Mycelium	Copay
Non wallet apps	Top-10 apps along with additional 20 Internet-dependent apps.	

We inspected each app and identified the actions available on it. For Bitcoin wallet apps, we found seven classes of actions relevant to Bitcoin transactions: open the app, receive Bitcoin, send Bitcoin, generate a new Bitcoin address, buy/sell (trade) Bitcoin, see transaction history, and check available balance. We omitted other actions available on the wallet apps because they do not necessarily elicit network traffic, *e.g.,* exit/close the app and share the app via SMS/Bluetooth. Table 3 lists the actions available on the wallet apps mentioned in Table 1.

Table 3: Actions available on Bitcoin wallet apps

App	Action						
	Open app	Receive Bitcoin	Send Bitcoin	Generate addresses	In-app buy/sell	Transaction history	Check balance
BTC.com	✓	✓	✓	✗	✠	❋	❋
BitPay	✓	✓	✓	✓	✓	♣	❋
Bitcoin Wallet (Bitcoin Wallet Devs)	✓	✓	✓	✗	✗	❋	❋
Bitcoin Wallet (Bitcoin.com)	✓	✓	✓	✓	✠	♣	❋
Blockchain	✓	✓	✓	✗	✓	♣	❋
Bread	✓	✓	✓	✗	✓	♣	❋
Coinbase	✓	✓	✓	✗	✓	♣	❋
Copay	✓	✓	✓	✓	✓	♣	❋
Luno	✓	✓	✓	✗	✓	❖	✙
Mycelium	✓	✓	✓	✗	✠	❖	❋
Unocoin	✓	✓	✓	✗	✓	❖	❋
Wirex	✓	✓	✓	✗	✓	❋	❋
Xapo	✓	✓	✓	✗	✓	❖	❋
Zebpay	✓	✓	✓	✗	✓	❖	❋

✓ Available ✗ Not available ❋ On app's home ♣ Under individual wallet/currency
✙ Under dedicated menu for wallets' summary ❖ Under dedicated menu for transaction history
✠ Redirects to an external website, leaving the app

From Table 3, it is clear that only three actions, *i.e.,* open the app, receive Bitcoin, and send Bitcoin are available across all the wallet apps. Hence, we choose these three actions for classification, which indeed are the most important actions for Bitcoin transactions. It is also important to mention that opening the app may also be seen as the user's intent to inquire about the available balance or to synchronize transaction history. Given the wide-variety of distinct apps in the non-Bitcoin app category, we collected traffic traces for such apps while using each device normally for 8 hours. Next, we explain our equipment setup for experiments and data collection.

3.2 Equipment setup

Figure 1 shows our equipment setup for collecting network traffic generated from the apps. The workstation was equipped with two Ethernet-based Network Interface Controllers (NICs); one for connecting it to the Internet and the other one for connecting it to the Wi-Fi Access Point (AP). The workstation was configured to forward traffic between Wi-Fi AP and the Internet. The smartphones were provided access to the Internet over a wireless connection via Wi-Fi AP. It is important to mention that only one smartphone was connected to the Wi-Fi AP at a time. To simulate user actions on the smartphones and thus evoke network traffic, scripted-commands were sent via USB. For the Android device, we used Android Debug Bridge (adb[3]), while for iOS device, we used Alloy 2.1.1[4] app that allows to automate the device without jailbreaking it. The generated network traffic was captured on the workstation using Wireshark 2.2.6[5].

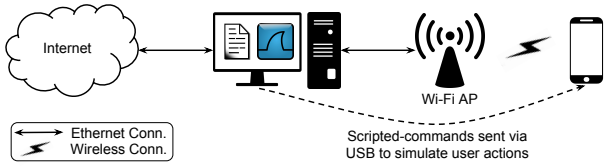


Figure 1: Equipment setup

The captured traces were exported to Comma Separated Value (CSV) files; each row holding the details of one captured packet. For each packet, we collected time, source IP address, destination IP address, ports, packet length, protocol, and TCP/IP flags. Although the packet's payload was gathered, it was discarded since it may or may not be encrypted. Finally, to make the experiment scenario as close to the real environment as possible, we used only physical hardware and omitted any device emulator or virtual machine. We used Actiona 3.9.1[6] tool to coordinate the entire process of data collection.

4 CLASSIFIER DESIGN

In this section, we present the design of our classification procedure. At first, we describe the data preprocessing phase necessary to get suitable training instances for the classification algorithm. Then, we briefly define the machine learning methods we used and how they have been trained and finally used for the classification.

4.1 Data preprocessing

To handle network traffic traces via machine learning models, we need to perform a preprocessing step. In this work, we employ

[3] developer.android.com/studio/command-line/adb
[4] alloylauncher.com
[5] wireshark.org
[6] wiki.actiona.tools/doku.php?id=:en:start

a procedure inspired by the one proposed in [27]. The complete procedure is composed of the following steps:

Network trace capture The network trace capturing process aims to collect traffic data from a network in which simulated users are using apps connected to such network. The full equipment setup has been described previously in Section 3.2.

Traffic *burstification* After the data collection phase, the network traffic needs to be parsed. The parsing aims to obtain chunks of traffic that can be directly converted into training instances suitable for the learning models. The first part of the parsing step is the so-called traffic *burstification*: the network traffic is divided into macro-chunks called bursts. A burst is defined as a sequence of traffic packets where each packet is either received or transmitted within a threshold of time. In our experiments, such threshold has been fixed to one second, as previously done in [27].

Flows separation The next part of the parsing step further divides the bursts into chuncks, called flows, corresponding to traffic between pairs of IP addresses/port. Anytime the port information was not available the corresponding packet was discarded. Similarly, flows with less than three packets has also been discarded.

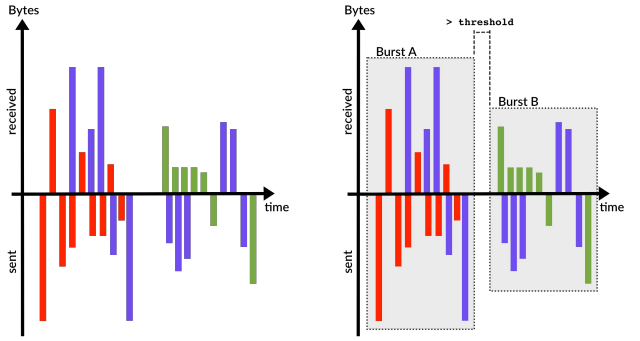
4.2 Feature selection

After the data preprocessing stage, we obtain a set of flows; each of them corresponding to a particular action of a specific app. The final step consists of converting the flows into training instances. It is important to notice that a single action of an application can produce more than one flow, and hence it can produce more than one training instance.

For each flow, the following feature selection procedure has been performed:

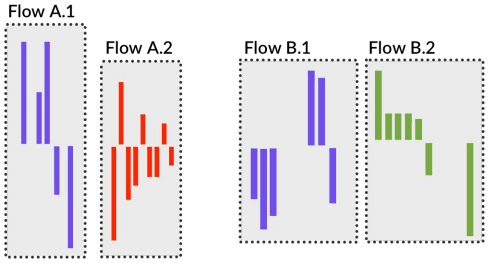
- Assuming the local IP address as the target endpoint of the flow, we convert each packet into a number corresponding to its length in bytes. If the packet has been sent by the target, such number is negated. At the end of this step, the flow is converted into a sequence (time series) of integer numbers. It is worth to note that the lengths of such sequences of numbers are not uniform and hence, in general, are not directly suitable as training instances. Machine learning models usually require fixed length input instances.
- The extracted sequence is finally converted into a training instance via a statistical feature extraction procedure, which simply compute some statistics over the time series. The used statistics are: length of the series, minimum, maximum, mean, median, mode, variance, skewness, kurtosis, and percentile at 25%, 50% and 75%. These statistics are collected for the entire sequence, for the incoming packets only, and for the outgoing packets only. Hence, the resulting training instance has a dimension of 36 (12×3). Finally, if we called $\mathbf{x} \in \mathbb{R}^{36}$ the vector containing the statistics, the complete training instance is described by the pair $(\mathbf{x}, y)$, where y is the target classification value.

It is worth to mention that while source and destination IP addresses have been used for flows separation, they were not leveraged in any way during the classification. The full procedure, including the data preprocessing steps, is depicted in Figure 2.

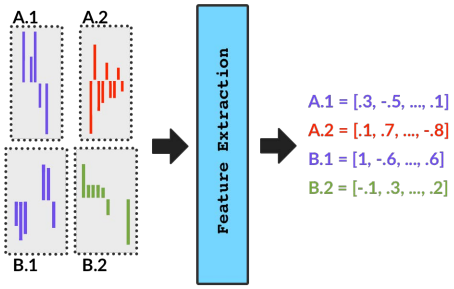


(a) Network traces capture: different colors represent different applications.

(b) Traffic *burstification*: traces are split into bursts.



(c) Flows separation: for each burst, different flows are separated by means of the pairs of source-destination IPs.



(d) Training set creation: flows are converted into vectors of statistical features about the packets length.

Figure 2: Network traffic preprocessing

4.3 Machine learning

Given the modularity of the proposed framework, any classification method can be plugged to perform the task. For experimental purposes, we employed two of the most successful machine learning methods for classification, namely Support Vector Machine (SVM) and Random Forest (RF).

SVM is one of the most used kernel methods in machine learning [7, 25, 28]. Besides its good reputation in terms of classification accuracy, SVM's popularity is also given by its strong theoretical foundation. SVM aims to find a hyperplane that separates the instances with different labels. Such hyperplane is guaranteed to maximize the minimum distance between two points with different label. This property entail very good generalization capabilities to unseen examples, which is a desired feature for any machine learning model.

RF also known as random decision forests [15], are learning algorithm of the family of ensemble learning methods. RF for classification operate by constructing a set of decision trees at training time, and at the prediction time, they output the class that is the mode of the classes (classification) of the individual decision trees. One of the strengths of RF are their efficiency and their simplicity. Since they are based on decision trees, it is very easy to grasp what they do under the hood. Moreover, RF have achieved state-of-the-art performances in many classification tasks.

4.4 Training

The training phase consists of learning the model parameters using the training set. In our scenario, in which we want to identify whether a Bitcoin app is being used and also which specific action of such app is being performed, we need to tackle the problem at different levels. We can identify the following layers of classification:

(1) Classify whether the instance represents a flow of a Bitcoin app or not;
(2) If so, classify whether it belongs to an Android app or an iOS app;
(3) If it has been categorized as Android (or iOS) app, classify the specific app;
(4) Given the app from the previous step, finally, classify the specific action.

The full stack of classification layers is depicted in Figure 3.

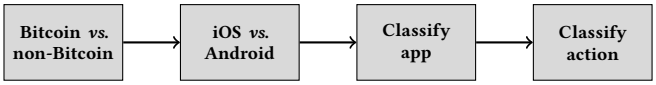


Figure 3: Classification hierarchy: *(i)* Bitcoin apps are isolated from the non-Bitcoin ones; *(ii)* Bitcoin apps are classified on the basis of the operating system; *(iii)* For the target operating system, the Bitcoin app is identified; *(iv)* given the app, the related action is identified.

Hence, the training phase requires to learn one model for each of the classification tasks described above. Before starting the training, to normalize the input data, we apply to the training instances a scaling function. In particular, we used both MinMax and Standard scaling techniques. See Appendix A for further details on these scaling techniques.

4.5 Prediction

Given a new instance to classify, the prediction is performed using the same steps as in the training phase. Clearly, if a wrong prediction is made in one step, all the following steps will also be wrong except for user action classification, which can be still correct. This is due to the fact that same actions are shared between the applications, and it can be correctly identified even if the app identification is not correct.

5 EVALUATION

In this section, we show the evaluation procedure used to assess the quality of the proposed approach. For each of the classification step identified in Section 4.4, we first trained a classifier, and then we tested it on an hold out set of instances. We performed two different experiments:

(1) Single classifier assessment: in this setting, each single classifier is tested independently of the others.
(2) Full stack classification: in this setting, the classification is performed following the full-sequence of classification as described in Figure 3 (Section 4.4).

We also argue about the obtained results.

5.1 Evaluation settings

All the experiments have been conducted using a stratified 5-fold cross validation. In order to increase the statistical significance of the result, we repeated each experiment 10 times with different 90-10% training and test splits. The validation procedure is used to do model selection and the validated hyper-parameters for SVM and RF are shown in Table 4 and Table 5, respectively. We chose standard range of values for the hyper-parameters [16]. We also validated the scaling techniques shown previously.

Table 4: Hyper-parameters validated for Support Vector Machine

Parameter	Validated values	Effect on the model
γ	$\{10^{-6}, \ldots, 10^{3}\}$	Shape parameter of the RBF kernel which defines how an example influence in the final classification.
C	$\{10^{-3}, \ldots, 10^{3}\}$	Regularization parameter that controls the trade-off between the achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data.

Table 5: Hyper-parameters validated for Random Forest

Parameter	Validated values	Effect on the model
# of trees	{10, 50, 100}	Number of trees use in the ensemble.
Max depth	3, ∞	Maximum depth of the trees.
Bootstrap	yes / no	Bootstrap Aggregation (a.k.a. bagging) is a technique that reduces model variances (overfitting) and improves the outcome of learning on limited sample or unstable datasets.
Split criterion	gini, entropy	Criterion used to split a node in a decision tree.

It is worth to mention that, even though the dataset has been collected in a controlled setup, the full hierarchical classification well simulate a real-world scenario in which instances are gathered in real time. Table 6 describes the instances distribution over the apps

and over the actions for both iOS and Android. The total number of instances for non-Bitcoin app are 4662. After the preprocessing step, Luno app did not produce any meaningful flow. Hence, we had to discard Luno app. One of the possible reasons for such behavior of the app could be that the app mostly processes the data off-line.

Table 6: Dataset description: app name; operating system; number of instances for open app, receive Bitcoin, and send Bitcoin actions; and the total number of instances

App	OS	Open App	Receive Bitcoin	Send Bitcoin	Total
BTC.com	Android	149	22	20	191
Bitcoin Wallet (Bitcoin.com)	Android	51	21	46	118
Coinbase	Android	286	20	19	325
Mycelium	Android	251	20	38	309
BitPay	iOS	20	54	20	94
Bitcoin Wallet (Bitcoin.com)	iOS	49	40	137	226
Blockchain	iOS	346	40	167	553
Bread	iOS	29	208	217	454
Copay	iOS	20	52	20	92
Total		1101	477	684	2362

It is worth to notice that the real proportion of Bitcoin and non-Bitcoin apps is actually much more imbalanced. However, in machine learning one of the standard approach to deal with highly imbalanced datasets is to use over-sampling of the minority class [4]. Thus, in our setting, the almost balanced dataset follows the same direction of the over-sampling technique.

All methods have been evaluated using standard classification metrics: Accuracy, Precision, Recall, and F1 measure. See Appendix A for details on these metrics.

5.2 Results

In this section, we present and discuss the results obtained by our proposal on user activity identification task.

5.2.1 Single classifier assessment. The following battery of experiments aim to assess each classification layer individually. In these cases, every layer works with a controlled training set and independently from the others. The goal of this preliminary assessment is to check whether some of the classifications are harder than others. Tables 7 - 12 present the results for single classifier for different tasks mentioned in Section 4.4. Here, we present the classification performances of RF and SVM over 10 runs of a stratified 5-fold cross validation. We report the average results with their standard deviations, and (•) indicates the best result for the metric.

For Bitcoin *vs.* non-Bitcoin app classification, we achieved an accuracy of 97.7% using RF, see Table 7. Next, as shown in Table 8, we attained an accuracy of 98.4% using RF in correctly identifying the OS to which an app belongs to. The performance metrics for Bitcoin app classification on Android platform are listed in Table 9. Here, we reached an accuracy of 96.6% using RF. The accuracy in identification of user actions in Bitcoin apps on Android platform is listed in Table 10. The performance metrics for Bitcoin app classification on iOS platform are listed in Table 11. Here, we attained an accuracy of 96.2% using RF. The accuracy in identification of user actions in Bitcoin apps on iOS platform is listed in Table 12.

Table 7: Bitcoin *vs.* non-Bitcoin app classification

Method	Accuracy	Precision	Recall	F1
RF	0.977 ± 0.005•	0.977 ± 0.005•	0.973 ± 0.005•	0.975 ± 0.005•
SVM	0.930 ± 0.01	0.922 ± 0.01	0.923 ± 0.01	0.922 ± 0.02

Table 8: App's OS classification

Method	Accuracy	Precision	Recall	F1
RF	0.984 ± 0.01•	0.984 ± 0.01•	0.983 ± 0.01•	0.983 ± 0.01•
SVM	0.956 ± 0.01	0.955 ± 0.02	0.955 ± 0.02	0.955 ± 0.02

Table 9: Bitcoin app classification on Android

Method	Accuracy	Precision	Recall	F1
RF	0.966 ± 0.01•	0.968 ± 0.01•	0.968 ± 0.01•	0.968 ± 0.01•
SVM	0.945 ± 0.02	0.948 ± 0.02	0.948 ± 0.02	0.948 ± 0.02

Table 10: Classification of user actions in Bitcoin apps on Android

Method	Bitcoin Wallet (Bitcoin.com)	BTC.com	Coinbase	Mycelium
RF	0.8 ± 0.15	0.98 ± 0.03•	0.991 ± 0.01•	0.971 ± 0.03•
SVM	0.85 ± 0.1•	0.975 ± 0.03	0.988 ± 0.02	0.958 ± 0.05

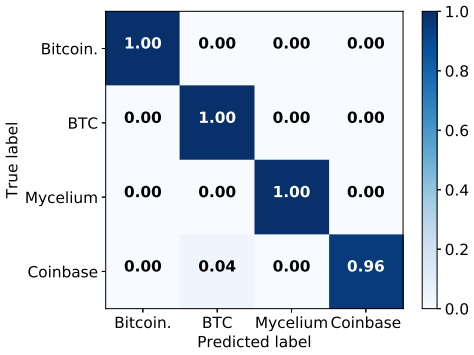
Table 11: Bitcoin app classification on iOS

Method	Accuracy	Precision	Recall	F1
RF	0.962 ± 0.02•	0.964 ± 0.02•	0.963 ± 0.02•	0.963 ± 0.02•
SVM	0.935 ± 0.02	0.938 ± 0.02	0.935 ± 0.02	0.935 ± 0.02

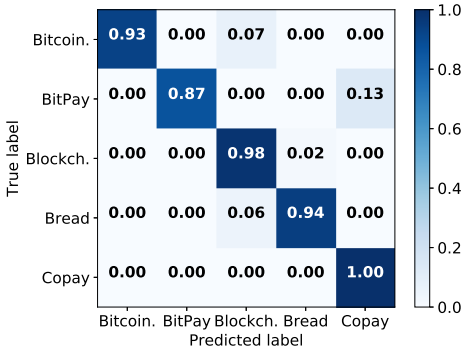
Table 12: Classification of user actions in Bitcoin apps on iOS

Method	Bitcoin Wallet (Bitcoin.com)	BitPay	Blockchain	Bread	Copay
RF	1.0 ± 0.0•	1.0 ± 0.0•	0.920 ± 0.02•	0.943 ± 0.03	1.0 ± 0.0•
SVM	1.0 ± 0.0•	1.0 ± 0.0•	0.911 ± 0.03	0.958 ± 0.04•	1.0 ± 0.0•

As discussed above, we obtained slightly better results on Android for Bitcoin app identification while user action identification was better on iOS. Moreover, RF performed better over SVM for most of the task.

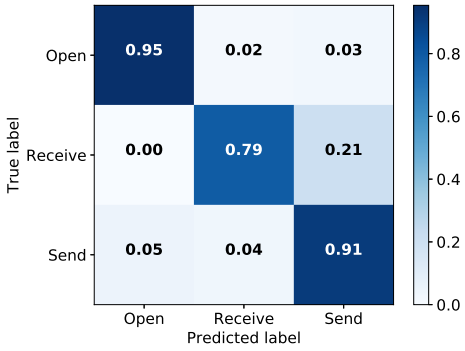


(a) Android

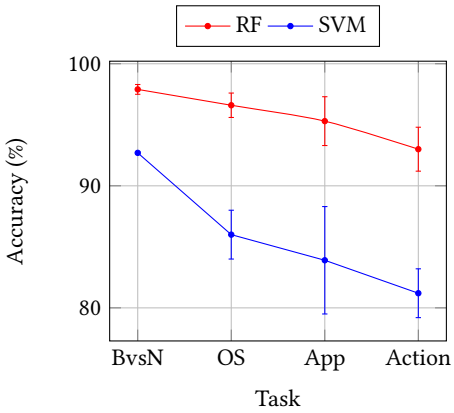


(b) iOS

Figure 4: Confusion matrix for Bitcoin app classification using RF for both (a) Android and (b) iOS. The confusion matrices are taken from one out of ten runs performed.



(a) Confusion Matrix



(b) Classification accuracy

Figure 5: (a) Confusion matrix for user action classification in Bitcoin apps using RF. The confusion matrices are taken from one out of ten runs performed. (b) Accuracy of both RF and SVM along the classification stack. Performance are reported as the average accuracy (%) over 10 runs.

5.2.2 Full stack classification. This experiment represents a simulation of a real-world scenario. Hence, the identification of both Bitcoin app and Bitcoin-related user operations have to be done assuming that the classifications made in the previous layers of the classification stack are correct. A single error in one stage of the classification stack influences all the subsequent ones.

Figure 4 depicts the confusion matrix for identification of Bitcoin apps using RF for Android (Figure 4(a)) and iOS (Figure 4(b)). Similar to the previous experiment setting, we got better results on Android in Bitcoin app identification. Figure 5(a) shows the confusion matrix for classification of user actions in Bitcoin apps using RF. Figure 5(b) gives a comparison of accuracy achieved by RF and SVM along the full stack classification.

It is clear from Figure 5(b) that misclassification errors in one stage are propagated to the subsequent stages. Hence, the results for a stage of full stack classification are limited by the performance of the previous stage(s); except for user action classification, which can be still correct (see Section 4.5). Nevertheless, we obtained an accuracy of nearly 95% in user action identification using RF in the full stack classification.

6 CONCLUSIONS AND FUTURE WORK

The popularity of cryptocurrencies, especially Bitcoin, is increasing day-by-day. Bitcoin is now recognized as a regular mode of payment. The convenience of smartphones has also driven people to adopt and use this new currency. In this paper, we have focused on identification of user actions within Bitcoin wallet apps. By analyzing network traffic using machine learning techniques, we have identified the most crucial user actions related to Bitcoin transactions with a very high accuracy of nearly 95%. In the future, we will investigate the security and privacy implication of transacting on such apps by considering a stronger adversary model. We will also explore the possibility to de-anonymize financial transaction placed via wallet apps.

ACKNOWLEDGMENTS

Ankit Gangwal is pursuing his Ph.D. with a fellowship for international students funded by Fondazione Cassa di Risparmio di Padova e Rovigo (CARIPARO). This work is partially supported by the

EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896), the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and by the grant "Scalable IoT Management and Key security aspects in 5G systems" from Intel.

REFERENCES

[1] Hasan Faik Alan and Jasleen Kaur. 2016. Can Android Applications be Identified using only TCP/IP Headers of their Launch Time Traffic?. In *9th ACM conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 61–66.

[2] Giuseppe Ateniese, Briland Hitaj, Luigi Vincenzo Mancini, Nino Vincenzo Verde, and Antonio Villani. 2015. No Place to Hide that Bytes won't Reveal: Sniffing Location-Based Encrypted Traffic to Track a User's Position. In *Springer Network and System Security (NSS), LNCS*, Vol. 9408. 46–59.

[3] Xiang Cai, Xin Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching From a Distance: Website Fingerprinting Attacks and Defenses. In *19th ACM Computer and Communications Security (CCS)*. 605–616.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16, 1 (2002), 321–357.

[5] Mauro Conti, Ankit Gangwal, and Sushmita Ruj. 2018. On the Economic Significance of Ransomware Campaigns: A Bitcoin Transactions Perspective. *Elsevier Computers & Security* 79 (2018), 162–189.

[6] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2016. Analyzing Android Encrypted Network Traffic to Identify User Actions. *IEEE Transactions on Information Forensics and Security* 11, 1 (2016), 114–125.

[7] Corinna Cortes and Vladimir Vapnik. 1995. Support Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[8] Scott E Coull and Kevin P Dyer. 2014. Traffic Analysis of Encrypted Messaging Services: Apple iMessage and Beyond. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 5–11.

[9] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. NetworkProfiler: Towards Automatic Fingerprinting of Android Apps. In *32nd IEEE International Conference on Computer Communications (INFOCOM)*. 809–817.

[10] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems* 32, 2 (2014), 1–29.

[11] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. 2010. A First Look at Traffic on Smartphones. In *10th ACM SIGCOMM Internet Measurement Conference (IMC)*. 281–287.

[12] Hyo Ham and Mi Choi. 2012. Applicaion-level Traffic Analysis of Smartphone Users using Embedded Agents. In *14th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1–4.

[13] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *1st ACM Cloud Computing Security Workshop (CCSW)*. 31–42.

[14] Andrew Hintz. 2003. Fingerprinting Websites using Traffic Analysis. In *Springer Privacy Enhancing Technologies (PET), LNCS*, Vol. 2482. 171–178.

[15] Tin Kam Ho. 1995. Random Decision Forests. In *3rd International Conference on Document Analysis and Recognition (ICDAR)*. 278–282.

[16] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2003. *A Practical Guide to Support Vector Classification*. Technical Report.

[17] Sang-Woo Lee, Jun-Sang Park, Hyun-Shin Lee, and Myung-Sup Kim. 2011. A Study on Smartphone Traffic Analysis. In *13th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1–7.

[18] Marc Liberatore and Brian Neil Levine. 2006. Inferring the Source of Encrypted HTTP Connections. In *13th ACM Computer and Communications Security (CCS)*. 255–263.

[19] Sophon Mongkolluksamee, Vasaka Visoottiviseth, and Kensuke Fukuda. 2016. Combining Communication Patterns & Traffic Patterns to Enhance Mobile Traffic Identification Performance. *Journal of Information Processing* 24, 2 (2016), 247–254.

[20] Thuy TT Nguyen and Grenville Armitage. 2008. A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE Communications Surveys & Tutorials* 10, 4 (2008), 56–76.

[21] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *10th ACM Workshop on Privacy in the Electronic Society (WPES)*. 103–114.

[22] Zafar Ayyub Qazi, Jeongkeun Lee, Tao Jin, Gowtham Bellala, Manfred Arndt, and Guevara Noubir. 2013. Application-awareness in SDN. In *ACM SIGCOMM conference*. 487–488.

[23] Jean-François Raymond. 2001. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Springer Designing Privacy Enhancing Technologies, LNCS*, Vol. 2009. 10–29.

[24] Brendan Saltaformaggio, Hongjun Choi, Kristen Johnson, Yonghwi Kwon, Qi Zhang, Xiangyu Zhang, Dongyan Xu, and John Qian. 2016. Eavesdropping on Fine-Grained User Activities within Smartphone Apps over Encrypted Network Traffic. In *10th USENIX Workshop on Offensive Technologies (WOOT)*. 1–10.

[25] John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.

[26] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. 2013. Who do you sync you are? Smartphone Fingerprinting via Application Behaviour. In *6th ACM conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 7–12.

[27] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2018. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Transactions on Information Forensics and Security* 13, 1 (2018), 63–78.

[28] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[29] Qinglong Wang, Amir Yahyavi, Bettina Kemme, and Wenbo He. 2015. I Know What You Did on Your Smartphone: Inferring App Usage over Encrypted Data Traffic. In *3rd IEEE Communications and Network Security (CNS)*. 433–441.

[30] Jie Yang, Shuo Zhang, Xinyu Zhang, Jun Liu, and Gang Cheng. 2013. Analysis of Smartphone Traffic with MapReduce. In *22nd IEEE Wireless and Optical Communication Conference (WOCC)*. 394–398.

[31] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. 2013. Identity, location, disease and more: Inferring your secrets from android public resources. In *20th ACM Computer and Communications Security (CCS)*. 1017–1028.

Appendix A STANDARD DEFINITIONS

MinMax scaler scales each feature in the range [0,1]. Specifically, given a feature x and one of its assumed value x_i the following formula is applied:

$$minmax(x_i) = \frac{x_i - min(x)}{max(x) - min(x)},$$

where $min(x)$ and $max(x)$ are the minimum and maximum value of the feature x in the dataset.

Standard scaler each feature is transformed in such a way that the mean becomes zero and standard deviation becomes one. Specifically, given a feature x and one of its value x_i, the following formula is applied:

$$standard(x_i) = \frac{x_i - \mu(x)}{\sigma(x)},$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the variable x.

Accuracy measures how often the classifier makes the right prediction defined as the ratio between the number of hit and the number of predictions.

Precision quantifies the ability of a classifier to not label a negative example as positive. It is computed as the ratio between the number of true positives and the total number of instances labeled as positives.

Recall defines the probability that a positive prediction made by the classifier is actually positive. It is computed as the fraction between the number of true positives and the total number of positives in the set.

F1 is a single metric that combines both precision and recall via their harmonic mean:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

View publication stats