Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?[*]

Hasan Faik Alan
Department of Computer Science
UNC - Chapel Hill, NC, USA
alan@cs.unc.edu

Jasleen Kaur
Department of Computer Science
UNC - Chapel Hill, NC, USA
jasleen@cs.unc.edu

ABSTRACT

The ability to identify mobile apps in network traffic has significant implications in many domains, including traffic management, malware detection, and maintaining user privacy. App identification methods in the literature typically use deep packet inspection (DPI) and analyze HTTP headers to extract app fingerprints. However, these methods cannot be used if HTTP traffic is encrypted. We investigate whether Android apps can be identified from their launch-time network traffic using only TCP/IP headers. We first capture network traffic of 86,109 app launches by repeatedly running 1,595 apps on 4 distinct Android devices. We then use supervised learning methods used previously in the web page identification literature, to identify the apps that generated the traffic. We find that: (i) popular Android apps can be identified with 88% accuracy, by using the packet sizes of the first 64 packets they generate, when the learning methods are trained and tested on the data collected from same device; (ii) when the data from an unseen device (but similar operating system/vendor) is used for testing, the apps can be identified with 67% accuracy; (iii) the app identification accuracy does not drop significantly even if the training data are stale by several days, and (iv) the accuracy does drop quite significantly if the operating system/vendor is very different. We discuss the implications of our findings as well as open issues.

Keywords

Network Traffic Analysis; Android Apps; Privacy

1. INTRODUCTION

The problem of identifying applications, by analyzing just the TCP/IP network traffic they generate, has received significant attention in the literature over the past decade and a half. This is for two compelling reasons. First, the ability to identify client applications from network traffic is tremendously useful in several domains—including malware detection, traffic management, network capacity planning, as well as, understanding the degree of user privacy leakage [1–4]. Second, with growing adoption of encryption and compression, as well as stronger Internet privacy legislation, HTTP payloads are increasingly becoming inaccessible to traffic monitors [5, 6]—consequently, analysis that relies on only TCP/IP headers is most useful.

The above trends have been well recognized in the literature on web page identification. Indeed, significant leaps have been made over the past several years in accurately identifying from TCP/IP headers, which web pages are being visited by a web client [4, 7, 8]. However, there has been little work in the domain of identifying mobile apps. While there has been some preliminary evidence collected using 40-70 apps [9, 10], there has been no comprehensive study of the identifiability of apps. Given the exponential rise of mobile app usage [11], we address this need in this paper. Specifically, we present the following key innovations:

- *Data Collection:* We set up 4 different Android devices (2 tablets and 2 phones) and select 1,595 most popular apps that use networking, as well as are compatible with all 4 devices. We then repeatedly run these apps over a duration of 25 days, and capture network traffic across 86,109 app launches. To the best of our knowledge, this is the largest data set considered to date.
- *Identifiability Evaluation:* Since mobile apps use mostly HTTP/HTTPS for networking [12], we next select learning algorithms and features that have worked well in the past for identification of web pages in web traffic. Specifically, we consider the packet sizes found in the initial launch time traffic of mobile apps, and evaluate three classifiers from the web page identification literature. We find that the mobile apps in our dataset do generate distinctive TCP/IP traffic, and just 32-64 initial packets from their launch time traffic can be used to identify them with up to 88% accuracy.
- *Robustness Evaluation:* The large number of app launches included in our data set also enables us to consider the impact on identification accuracy of several factors, including the training set size, frequency of training, app updates, and device differences. We find that the app identification accuracy does not drop significantly, even if the training data is stale by several days. However, when data from an unseen device is used for testing, the identification accuracy drops to around 67% (with similar operating system/vendor)

[*]This material is based upon work supported by the National Science Foundation under Grant No. CNS-1526268.

WiSec'16 , July 18-20, 2016, Darmstadt, Germany
© 2016 ACM. ISBN 978-1-4503-4270-4/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2939918.2939929

or to even around 28% (with very different operating system/vendor).

2. DATA COLLECTION

For data collection, we use 4 different Android devices (Table 1) and capture the network traffic generated by running 1595 apps on them. In the remaining of this section, we explain how we determined the 1595 apps that we experimented with and how we captured the network traffic of these apps. We also describe the dataset we collected.

Table 1: Android Devices

ID	Device	Type	Android Version
S7	Samsung Galaxy Tab 4 7.0	tablet	4.4.2
S4	Samsung Galaxy S4 mini	phone	4.4.4
N5	Nexus 5	phone	6.0.1
N7	Nexus 7	tablet	6.0.1

App Selection There are around 2 million apps on Google Play, the Android application distribution platform [13]. We rely on the GooglePlayAppsCrawler.py project [14] to identify the 2000 most popular (most installed) free apps as of November 2015. We next identify the subset of these apps that are compatible with all 4 of our devices. The Google Play page for a given app lists all compatible devices used by the same account—using this, we determined that 1655 of the above 2000 apps are compatible with all of our devices.

The goal of this paper is to study identifiability of apps from the network traffic they generate. We analyze the Android application package files of the above 1655 apps to find that 1595 of these explicitly request network access by using the "android.permission.INTERNET" permission—we consider all of these apps in our study. Please note that the above permission is necessary but not sufficient for an app to generate network traffic [15]—an app may simply not use networking even though it has the permission.

Network Traffic Capture We use a USB WiFi adapter in access point (AP) mode to provide Internet connectivity for multiple Android devices. We capture the network traffic on the AP interface using tcpdump [16], while running apps on the devices. We capture only the first 100 bytes of network packets, which contain TCP/IP headers.

Android Debug Bridge (ADB) is a command line tool that allows us to communicate with Android devices [17]. It provides commands for performing several tasks such as installing, starting and stopping, and uninstalling an app. We use ADB commands and automate the process of capturing the network traffic of an app while the app is running concurrently on multiple devices. First, we install the same app on each of the devices. We then start tcpdump for network traffic capture and run the app for 20 seconds on the devices. Finally, we stop tcpdump and uninstall the app from the devices. We repeat this process for each of the apps to be analyzed. We log the times when an app was started and stopped, and the IP address of the device that the app was running on.

We extract the network traffic generated by an app while it is running on a specific device using the information available in the logs. We consider all of the TCP connections that were initiated between start and stop times of the app and are associated with the IP address of the device that the app was running on.

Dataset We conducted 14 data collection sessions, which lasted over 25 days (Table 2). In each session, we ran each of the 1,595 apps for 20 seconds on all of our 4 devices using the methodology described above. To minimize interfering traffic, we had disabled the automatic updates of apps and Android OS—after the 7th session, we manually updated the 515 apps that had a new version and conducted 7 additional data collection sessions using the latest versions of the apps. In total 86,109 successful app launches were performed, and network traffic was generated in 83,606 of them.

3. TRAFFIC FEATURES & CLASSIFIERS

We model the app identification problem as a multi-class supervised machine learning problem. A supervised machine learning algorithm is first trained with samples each in the form of a pair (x, y) where x and y are the feature vector and the class of a sample, respectively. The algorithm is then expected to predict the class of an unseen sample given its feature vector. In our problem, app launches are the samples, and the app package names are used as the classes of the samples. The feature vector belonging to an app launch is extracted from the TCP/IP headers of the packets generated during the launch.

Why Do We Consider Packet Sizes of App Launch Time Traffic? Many apps use networking immediately upon their launch for several reasons. They retrieve new ads from ad-networks or send information to analytics services. They also perform app specific communication—for example, an email app may check for new emails when it is launched. Results established previously in the field of non-mobile traffic classification suggest that, packet sizes observed within such launch time traffic may yield good feature sets for app identification. Specifically, [18] shows that the first few packets of TCP flows can be used to classify Internet applications into categories such as Web, FTP, and Games. More relevantly, [8] shows that web page identification can be performed solely using packet sizes.[1]

Classifiers Considered We select three methods from the web page identification literature that have been shown to achieve high accuracies using packet sizes. These methods mainly differ in the way feature vectors are extracted from packet sizes and the classification algorithms used, and are briefly summarized below (in chronological order).[2]

Method 1 (Sun et al. [19]): This method first groups contiguous incoming or outgoing packets within a TCP connection into "bursts". Given a traffic sample, incoming burst sizes are extracted from TCP connections and each of them are rounded to the nearest 32 bytes. A traffic sample is considered as a multiset of burst sizes. Similarity between two samples is measured using Jaccard's coefficient ($Sim(X, Y) =$

[1]The server IP addresses may seem to be a tempting feature to rely on. However, in our pilot studies we attained a lower accuracy (60-65%) compared to the methods that we discuss later (88%), when we use a bag-of-words model with the frequencies of IP addresses found in a traffic sample. Furthermore, using the packet sizes and IP addresses together did not improve the accuracies we attain by using only the packet sizes. This may be because many apps use the same third party services such as advertisement and analytics services.

[2]We only consider TCP packets with a payload.

Table 2: Dataset Summary

					Nexus 5		Galaxy S4		Galaxy Tab 4		Nexus 7	
session	start	end	duration (days)	Days Since Beginning	S	N	S	N	S	N	S	N
1	01/30 06:13	01/31 12:26	1.3	0.0	1552	1547	1555	1486	1522	1486	1539	1533
2	01/31 13:50	02/01 20:21	1.3	1.3	1563	1558	1558	1505	1541	1533	1564	1561
3	02/02 13:22	02/03 19:17	1.2	3.3	1566	1560	1555	1496	1539	1532	1564	1560
4	02/03 20:57	02/05 11:12	1.6	4.6	1560	1557	1553	1495	1538	1528	1559	1555
5	02/06 16:15	02/07 20:23	1.2	7.4	1466	1348	1455	1315	1444	1329	1465	1350
6	02/08 14:13	02/09 21:14	1.3	9.3	1492	1488	1488	1429	1467	1454	1491	1489
7	02/10 13:44	02/12 00:57	1.5	11.3	1490	1475	1486	1395	1468	1445	1492	1476
8	02/13 20:18	02/15 03:41	1.3	14.6	1550	1543	1539	1473	1522	1519	1551	1544
9	02/15 08:40	02/17 00:23	1.7	16.1	1567	1558	1549	1442	1533	1523	1566	961
10	02/17 18:57	02/19 00:03	1.2	18.5	1566	1562	1558	1492	1549	1535	1564	1556
11	02/19 00:23	02/20 07:36	1.3	19.8	1565	1555	1558	1324	1550	1545	1565	1561
12	02/20 10:11	02/21 15:03	1.2	21.2	1567	1562	1556	1491	1552	1545	1567	1559
13	02/21 17:46	02/22 22:46	1.2	22.5	1567	1563	1558	1498	1551	1542	1567	1563
14	02/23 01:03	02/24 13:52	1.5	23.8	1565	1553	1559	1455	1550	1541	1566	1556

Notes. S: Successful app launches. N: App launches that generated network traffic.

$\frac{X\cap Y}{X\cup Y}$). A test sample is identified by finding the most similar sample among the training samples.

Method 2 (Liberatore et al. [8]): Given a traffic sample, packet sizes are extracted. Minus sign is used to represent incoming packet sizes. A traffic sample is considered as a multiset of packet sizes. The Gaussian Naive Bayes classifier is used for classification.

Method 3 (Herrmann et al. [7]): Given a traffic sample, multisets of packet sizes are extracted as in Method 2. Term frequency - inverse document frequency transformation and normalization are applied to feature vectors. The Multinomial Naive Bayes classifier is used for classification.

4. EVALUATION

In this section, we use the methods described in Section 3 to evaluate the identifiability of the apps in our dataset.

App Identification From Launch Time Traffic The number of packets generated by an app during launch time depends on the Internet speed and the application logic of the app. Two reasons motivate us to minimize the number of packets used for app identification. First, less number of packets to process means less computational resource requirements for an identification method. Second, and more importantly, a user may start performing actions in an app such as tapping buttons or entering text immediately after the user interface of the app is available. Such actions may generate additional network traffic which would add noise to the launch time traffic of the app. By minimizing the number of packets considered as launch time traffic, we ensure that there will be minimal interference from user actions.

To find the minimum number of packets to consider as launch time traffic, we train the methods multiple times, varying the number of packets used. We consider only those apps that generated network traffic every time when they were launched in the first 7 sessions (i.e., 28 times) to ensure that there are equal number of app launch samples from each app—there were 1046 such apps. We first use the 20920 samples from the first 5 sessions for training the methods. We then use the 4184 samples from the 6th session for validation and determine when the methods perform best. Figure 1a plots the validation accuracies of the methods. We find that the accuracies of the methods increase significantly as larger number of initial packets are used by them. However, the gain in accuracy seems to taper down considerably after 32-64 packets are considered.

Another consideration that should drive the choice of number of packets to use is whether the apps are likely to generate these many packets quickly. As explained before, the goal is to minimize the user action interference on the traffic. To investigate this issue, we consider the apps that generated network traffic every time when they were launched in the first session. There are 1384 such apps, yielding 5536 app launches. We find that more than 8 packets were generated in the 99% of these launches (see Fig. 1b) and the median time to generate 64 packets is under 5 seconds (see Fig.1c[3]). Based on these findings, in the rest of this paper, we decide to limit the launch time network traffic of apps to the first 64 packets they generate.

After determining the optimal number of packets to use as 64, we train the methods using the first 6 sessions (25104 app launches of 1046 apps) and test them on the 7th session (4184 app launches).[4] We obtain app identification accuracies of 87%, 82% and 74% for the methods Herrmann, Liberatore and Sun, respectively. An identification accuracy of 87% is significantly high, given that there are 1046 different apps (i.e., classes) that each of the 4184 app launch samples can be identified as.

How Much Training Data is Needed? We achieved a high identification accuracy above when we use the first 6 sessions for training and the 7th session for testing. We next study the impact of using smaller training sets on the accuracy. For this purpose, we train the methods using the most recent k sessions, where k ranges from 1 to 6, and test them on the 7th session. Figure 2a plots the identification accuracy versus the sessions used for training. We find that the methods do benefit from using multiple sessions for training. However, the contribution of each added session decreases. In particular, even when only the 6th session is used for training, Herrmann yields 80% identification accuracy. Furthermore, the performance of all three

[3]The boxes extend from the lower to upper quartile; median and mean are shown with line and square respectively (using matplotlib).

[4]Please note that in order to measure the generalization of the methods, we test them on samples unseen during training and validation.

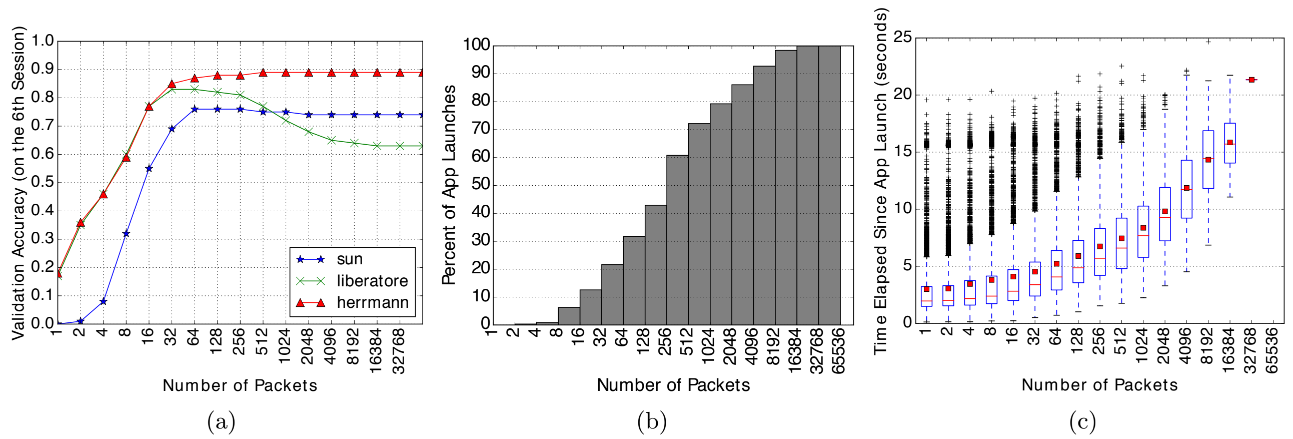


Figure 1: (a) Validation for Tuning Number of Packets; (b) Cumulative Histogram of Packets Per App Launch (Session 1); (c) Box Plot of Time Taken to Generate Certain Number of Packets (Session 1)

methods hardly improves once more than 4 sessions are used for training. These findings suggest that *app identification methods can sustain a high level of accuracy by keeping only a small number of most recent sessions for training, in this experiment 4, and discarding the older ones.*

How Frequently Do the Classifiers Need to Be Retrained? To answer this question, we use *one* of the first 6 sessions for training, and the 7th session for testing—Figure 2b plots the accuracy of each method. The best identification accuracy is achieved when the 6th session, the most recent session which contains samples just 2 days older than the test session, is used for training. The accuracies of the methods decrease gradually when older sessions are used for training—when samples that are 6 days older are used, the accuracy drops by only around 3%. The lowest accuracy (5-7% drop in accuracy for all methods) is attained when the first session, which contains samples 11 days older than the test samples, is used for training. These results indicate that launch time traffic of apps does change over time—however, the decrease in accuracy is not severe for samples collected within the past week. This allows us to collect training samples several days before testing.

How Do App Updates Impact Identification Accuracy? Mobile apps are regularly updated—app updates may impact the nature of launch time traffic as well. To study this issue, we updated the apps in our data set after the 7th session (515 of the apps had a new version). We then tested the methods using each of the sessions from 7 to 14 to investigate the effect of app updates—we trained the methods using the most recent 6 sessions before each test session. In this scenario, the 7th session is the only test session which contains samples from the original versions of the apps.

Since this evaluation involves all 14 sessions, we select the apps considered slightly differently—this is necessitated in part by the 9th session, in which Nexus 7 lost Internet connectivity and only 961 apps generated network traffic (Table 2). Specifically, we consider the apps that generated network traffic in at least 3 of the 4 launches in each of the 14 sessions—there are 1177 such apps. Figure 2c plots the accuracies of the methods, as a function of the test session.

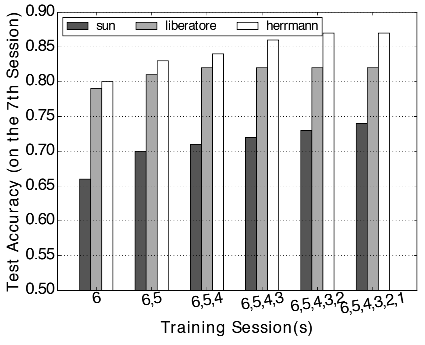
We find that the accuracies of the methods drop at the 8th session (by 5% for Herrmann) and then gradually increase on subsequent sessions. The reason is that the training set does not contain any launch samples from the new versions of the apps when we use the 8th session for testing. However, when we use the subsequent sessions for testing, the training set contains more and more samples from the new versions of the apps (since we use the most recent sessions for training). This result suggests that apps need to be updated regularly and *identification methods should be retrained with samples from the latest versions of the apps to keep them accurate over time.*

How Do Device Differences Impact Identification Accuracy? There are over 24000 distinct Android devices [20]. It is not feasible to run apps on each of these devices for the purpose of training an app identification method. Thus, we evaluate the methods with samples from a device that is unseen during training. For comparison, we also test them using samples from the same device. In all of these scenarios, we use the samples from the first 6 sessions for training and the 7th session for testing. Table 3 summarizes the experiments, and shows the respective accuracy of each method.

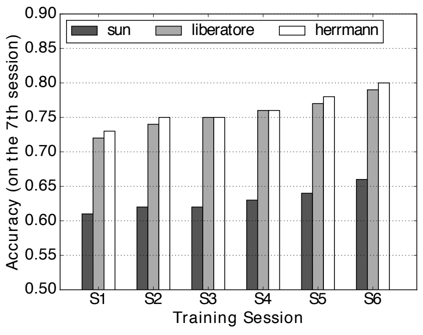
We find that the methods perform significantly better when they are trained and tested using the samples from the same device (see highlighted rows in Table 3)—Herrmann yields 88% identification accuracy, on average. The methods also perform better when both training and testing sets contain different devices but with a similar vendor/OS (Nexus/Android 6.0.1 or Samsung/Android 4.4.x).[5] However, when a similar vendor/OS is not included in the training set, the performance drops quite significantly (to around 28%)—compare the rows with a star.

To further investigate the vendor/OS differences, we downgrade the OS version of N7 (6.0.1) to the OS version of S7 (4.4.2). We collect new app launch samples. We use the samples from S7 for testing. The classification accuracies are 38%, 67% and 88%, when Herrmann is trained on samples from N5 (different OS & device), N7 (same OS & different

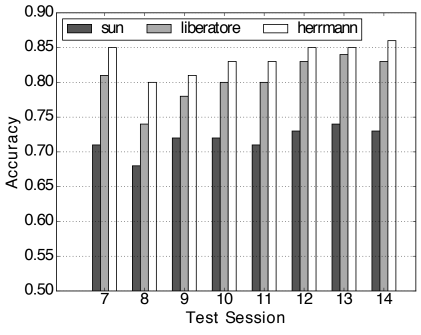
[5]The Nexus devices have the same Android version and the Samsung devices have very similar versions (see Table 1).



(a)



(b)



(c)

Figure 2: Evaluation of the methods, effect of: (a) training set size, (b) time, and (c) app updates on the accuracies

Table 3: Device Differences

test	training	herrmann	liberatore	sun
N5	N5	0.90	0.89	0.52
* N5	N7	0.66	0.59	0.47
* N5	S4	0.28	0.18	0.33
* N5	S7	0.28	0.14	0.33
N5	N7,S4,S7	0.70	0.61	0.49
N7	N5	0.67	0.58	0.45
N7	N7	0.88	0.88	0.49
N7	S4	0.26	0.16	0.28
N7	S7	0.29	0.15	0.30
N7	N5,S4,S7	0.68	0.59	0.47
S4	N5	0.29	0.13	0.28
S4	N7	0.27	0.13	0.28
S4	S4	0.84	0.81	0.39
S4	S7	0.66	0.59	0.39
S4	N5,N7,S7	0.71	0.64	0.42
S7	N5	0.27	0.13	0.34
S7	N7	0.29	0.15	0.35
S7	S4	0.67	0.62	0.45
S7	S7	0.88	0.88	0.52
S7	N5,N7,S4	0.71	0.61	0.50

Notes. N5, N7, S4, S7: see Table 1

device), and S7 (same OS & device), respectively. This result confirms that OS version and device have a significant impact on the accuracy. There are at least two approaches to address this issue. First, we can enumerate major Android OS versions and collect app launch samples from the devices running these OS versions to train an accurate classifier. Second, we can identify and use only those features that are robust across different devices.

It is critical to understand the significance of this observation—most of the past work on web page identification does not consider the impact of client diversity on the performance of identification methods. Our evaluation suggests that different devices (with different vendors/OS) do produce traffic features that may differ significantly. Consequently, *if an app identification method is biased towards a device or an Android version, it may perform poorly in the real world.*

5. DISCUSSION ON FUTURE WORK

When the app launch samples from same device is used for training and testing, we have found that apps can be identified with significantly high accuracy. Can this performance be improved further? It is likely that it can, especially since we have only evaluated with features and classification methods considered by others for web page identification. However, at 88% accuracy, there is only so much room for improvement of the accuracy itself. What is even more interesting is to investigate whether the identifiability of mobile apps is robust to other challenges. We identified one of these, robustness to differences across devices and operating systems, before. We identify some more below.

Web page identification experiments in the literature are conducted in laboratory conditions by logging start and end times of web page loads. Similarly, we log the exact start and stop times of apps during our data collection and consider only the network packets generated during this time interval for app identification. However, given a real world traffic trace, it is not easy to detect the start of web page loads or app launches in it. This problem is still being considered as unsolved and left as a future work in the web page identification literature as well as in this paper [4, 8]. In addition, an app may have already been launched before joining a network. In this case, it is not possible to capture the launch time network traffic of the app.

We show that the packet size based traffic analysis methods developed for webpage identification can also be used for app identification. Similarly, the countermeasures in the literature such as the ones that hide the packet sizes by padding can be used to preserve privacy of users. Nine such methods are analyzed in [21].

Our data collection methodology can be improved by using a VPN app that captures the traffic generated by other apps on the same device. Such an app is presented in [9]. In this context, there are VPN apps in Google Play that are used by millions of people. A VPN app developed with the explicit intent of collecting training data (or maliciously), can capture TCP/IP headers of network packets and label them with the apps generating them. A dataset collected in this way will contain app launch samples from many distinct devices. An app identification method trained using this dataset will be more robust to device/OS differences.

The ability to identify apps from launch time traffic can also help with classifying subsequent (non launch time) traffic—indeed, after identifying the launch of an app from the first 64 packets, subsequent packets are likely to belong to the same app. This heuristic can be used to extract labeled network traces of apps from a traffic trace collected in the wild—such datasets would be valuable for future traffic analysis research.

6. RELATED WORK

Dai et al. [12], Xu et al. [22], Yao et al. [23], and Miskovic et al. [24] analyze HTTP headers to extract string patterns that identify apps. Such strings are generally used by third party Ads or analytics services to assign unique identifiers to apps. However, as mentioned by the same studies [22, 23], such deep packet inspection methods cannot be used when an app encrypts is HTTP traffic using the TLS protocol.

Two recent studies have provided some preliminary evidence on the identifiability of mobile apps without using HTTP headers. Specifically, Le et al. [9] present a VPN app for capturing the network traffic of apps installed on a device and use 40 apps to conduct a pilot app identification study with the data collected using their VPN app. Qazi et al. [10] integrate an app identification method based on packet sizes to a Software Defined Networking (SDN) platform, and use it to study identifiability of 70 apps. In this paper, we collect a comprehensive data set by considering 1,595 mobile apps, multiple devices, and 86,109 app launches—we also evaluate three methods from the web page identification literature.

Stöber et al. [25] show that the identity of a smartphone that a combination of 14 specific apps are installed on, can be determined with 90% probability from the background 3G network traffic generated by the apps. In [26], network traffic generated when certain user actions are performed in 7 apps is studied. It is shown that a user action such as sending an email or opening chat can be identified with more than 95% accuracy when the app generating the traffic (e.g., Gmail) is known. This work can be considered as complementary to our paper. After identifying an app from its launch time network traffic, a method trained to differentiate between user actions performed within the app can be used on the subsequent network traffic.

7. CONCLUSIONS

In this paper, we study the identifiability of Android apps from the TCP/IP headers of their launch time network traffic. We first conduct a data collection study in which we capture launch time network traffic of 1595 apps, repeatedly over time and across distinct devices. We then formalize the concept of app launch time traffic by limiting the number of packets considered for app identification using validation. We use supervised learning methods from the web page identification literature to identify the apps that generated the launch time traffic samples. Our results show that when the methods are trained and tested using the samples collected on the same device, apps can be identified from their launch time network traffic with 88% accuracy. This finding has significant implications for domains that can benefit from app identification. However, there are several open issues that need to be addressed before such methods can be used in the real world.

References

[1] A. Callado et al. "A survey on internet traffic identification". In: *Communications Surveys & Tutorials, IEEE* 11.3 (2009), pp. 37–52.

[2] S. Chen et al. "Side-channel leaks in web applications: A reality today, a challenge tomorrow". In: *Security and Privacy (SP), 2010 IEEE Symposium on.* IEEE. 2010, pp. 191–206.

[3] W. Zhou et al. "Detecting repackaged smartphone applications in third-party android marketplaces". In: *Proceedings of the second ACM conference on Data and Application Security and Privacy.* ACM. 2012, pp. 317–326.

[4] B. Miller et al. "I know why you went to the clinic: Risks and realization of https traffic analysis". In: *Privacy Enhancing Technologies.* Springer. 2014, pp. 143–163.

[5] D. C. Sicker, P. Ohm, and D. Grunwald. "Legal issues surrounding monitoring during network research". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement.* ACM. 2007, pp. 141–148.

[6] A. M. White et al. "Clear and Present Data: Opaque Traffic and its Security Implications for the Future." In: *NDSS.* 2013.

[7] D. Herrmann, R. Wendolsky, and H. Federrath. "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier". In: *Proceedings of the 2009 ACM workshop on Cloud computing security.* ACM. 2009, pp. 31–42.

[8] M. Liberatore and B. N. Levine. "Inferring the source of encrypted HTTP connections". In: *Proceedings of the 13th ACM conference on Computer and communications security.* ACM. 2006, pp. 255–263.

[9] A. Le et al. "AntMonitor: A System for Monitoring from Mobile Devices". In: *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data.* ACM. 2015, pp. 15–20.

[10] Z. A. Qazi et al. "Application-awareness in SDN". In: *ACM SIGCOMM Computer Communication Review.* Vol. 43. 4. ACM. 2013, pp. 487–488.

[11] "Cisco Visual Networking Index: Global mobile data traffic forecast update, 2015-2020". In: *White Paper* (2015).

[12] S. Dai et al. "Networkprofiler: Towards automatic fingerprinting of android apps". In: *INFOCOM, 2013 Proceedings IEEE.* IEEE. 2013, pp. 809–817.

[13] *Number of available Android applications - AppBrain.* http://www.appbrain.com/stats/number-of-android-apps. (accessed February 13, 2016).

[14] M. Lins. *MarcelloLins/GooglePlayAppsCrawler.* https://github.com/MarcelloLins/GooglePlayAppsCrawler. (accessed February 7, 2016).

[15] *Connecting to the Network | Android Developers.* http://developer.android.com/training/basics/network-ops/connecting.html. (accessed February 13, 2016).

[16] *TCPDUMP/LIBPCAP public repository.* http://www.tcpdump.org/. (accessed February 13, 2016).

[17] *Android Debug Bridge | Android Developers.* http://developer.android.com/tools/help/adb.html. (accessed February 13, 2016).

[18] Y.-s. Lim et al. "Internet traffic classification demystified: on the sources of the discriminative power". In: *Proceedings of the 6th International COnference.* ACM. 2010, p. 9.

[19] Q. Sun et al. "Statistical identification of encrypted web browsing traffic". In: *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on.* IEEE. 2002, pp. 19–30.

[20] *Android Fragmentation Report August 2015 - OpenSignal.* http://opensignal.com/reports/2015/08/android-fragmentation/. (accessed February 13, 2016).

[21] K. P. Dyer et al. "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail". In: *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE. 2012, pp. 332–346.

[22] Q. Xu et al. "Automatic generation of mobile app signatures from traffic observations". In: *Computer Communications (INFOCOM), 2015 IEEE Conference on.* IEEE. 2015, pp. 1481–1489.

[23] H. Yao et al. "SAMPLES: Self Adaptive Mining of Persistent LExical Snippets for Classifying Mobile Application Traffic". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking.* ACM. 2015, pp. 439–451.

[24] S. Miskovic et al. "AppPrint: automatic fingerprinting of mobile applications in network traffic". In: *Passive and Active Measurement.* Springer. 2015, pp. 57–69.

[25] T. Stöber et al. "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.* ACM. 2013, pp. 7–12.

[26] M. Conti et al. "Analyzing Android Encrypted Network Traffic to Identify User Actions". In: *Information Forensics and Security, IEEE Transactions on* 11.1 (2016), pp. 114–125.