Classifier System for Application-Level Mobile Traffic

Yair Ivgi 308275601 Johan Thuillier 336104120
Advisor: Dr. Amit Dvir
Department of computer science, Ariel University, Israel

October 1, 2019

Abstract

abstract
With the development of Internet and computer science, computer network is changing peoples lives. The use in encrypted traffic in those networks has become a necessity, To prevent network attacks and increasing network privacy. Nowadays, cell phones have become a computer for all intents and purposes and The main way of communication for most people. Meanwhile, Network traffic prediction model itself becomes more and more complex.
In this paper, we investigate and offer an automated classifier system for application-level mobile traffic identification, That can be identified using only the features that remain intact after encryption.

I INTRODUCTION

Today there are nearly 4.5 billion Internet users in the world (June 2019) and the number continues to grow. Users access their bank account, exchange confidential information's or controls their connected devices (IOT). As a result, ensure the security of the internet becomes evermore important.
We notices two broad ways to attack a personal device the Active attack and the Passive attack. The major difference between active and passive attacks is that in active attacks the attacker intercepts the connection and modifies the information. Whereas, in a passive attack, the attacker intercepts the transit information with the intention of reading and analyzing the information.
Network traffic that isn't encrypted becomes a target. For instance, HTTP vulnerability let attackers gain access to website accounts and sensitive information. fortunately, today most of internet traffic is encrypted using HTTPS, designed to withstand such attacks and considered secure against them. Nevertheless, numerous studies and experiments especially using internet traffic classification methods have shown that encrypted traffic is not totally safe of potential attack. What is interesting is that most of the studies deals with the classification of computer networks rather than mobile networks.
In 2018, more than 52.2 percent of all website traffic worldwide is generated through mobile phones. Due to the strong growth of the smartphone market in recent years, this is no surprise. Network operators and mobile carriers are facing serious security challenges caused by an increasing number of services provided by smartphone Apps. For example, Android OS has more than 1 million Apps in the store. Thus, traffic classification is increasingly challenged by the massive diffusion of handheld devices. Moreover, their diversity make classification even more challenging, defeating established approaches.
In this paper, we aim to identify application-level mobile traffic using machine learning techniques. The objectives is to identify the application at 70% accuracy.
To achieve this goal, we do not specify one traffic identification method for all mobile application. Our proposed classifier includes an adaptive algorithm that automatically find the low cost accurate classifier by analysing the available classifier candidates to a specific mobile. As implied by our feasibility tests, the proposed classifier is both fast and accurate. Furthermore The experimental results indicate that our classifier is extremely robust to encryption.
The remainder of this paper is organized as follows. In Section II we review the state of the art on this topic and the work that was done. Section III presents our solution to identifying the user's mobile operating system and application. In Section IV we show our system results archived by the machine learning algorithms. Finally in Section V, we discuss the future lines of work and the developments we wish to achieve.

II RELATED WORK

Traffic Classification of mobile apps has seen huge interest by several recent works, mainly based on Encrypted Traffic assumption. This section first describes the literature on Traffic Classification in general and app classification applied to mobile context.
Network traffic analysis and machine learning can be used to infer private information about the user, the actions that he executes with his computer and mo-

bile phone, even though the traffic is encrypted.

II.1 Early Classifiers.

The basic approach dealt with HTTP (that is non-encrypted traffic), Neasbitt et al. proposed Click-Miner [1], a tool that reconstructs user-browser interactions. However, in recent years, websites and social networks started to use SSL/TLS encryption protocol, both for web and mobile services. This means that communications between endpoints are encrypted and this type of analysis cannot be performed anymore. So, a tool for encrypted traffic was needed.
Different works surveyed possible attacks that can be performed using analysis of encrypted traffic.
Liberatore and Levine [2] showed the effectiveness of two traffic analysis techniques for the identification of encrypted HTTP streams. One is based on a na'ıve Bayes classifier and the other on Jaccard's coefficient similarity measure.

II.2 Classifiers in Existing Approaches.

Traffic identification approaches are by general categorized into two groups content-based approaches, and statistics-based approaches. Content-based traffic identification deals with signatures of applications.
The signature matching method maps signatures to applications or application groups. The signature matching operation, whose comparison cost is high, guarantees high accuracy if there is no flaw in the manual creation of the signatures. Automated methods for creation of signatures have been proposed to reduce the amount of effort extended in signature creation process. In [3] they were able to identify individual apps, even in presence of noise, with great accuracy.
This classification is done by using packet-level traffic analysis in which side channel information leaks is used to identify specific patterns in packets regardless of whether they are encrypted or not.

II.3 statistics-based approach.

The statistics-based approach uses classifiers as a group of non deterministic features. The classification techniques used in this approach convert flows to a predetermined number of clusters and determine classifiers according to the following constraints: port number, number of packets, flow duration, average packet size of a flow, packet inter-arrival time, etc.
AppScanner is proposed in [4] as a framework for fingerprinting and identification of mobile apps using Statistical features. The fingerprints in AppScanner are collected by running apps automatically on an Android device, the network traces are pre-processed (to remove background traffic and extract features) to train an SVC and an RF. Statistical features are collected on sets of packets defined through timing criteria and destination IP address/port. Experimental results report 99% average accuracy in app identification, and up to 86.9% in classifying them.
The downside of this solution is the limitation in handling ambiguous traffic - traffic that is common among more than one app, that will cause poorer performance, add to that AppScanner does not provide an understanding of the variability and longevity of app fingerprint.
In [5] Which is a direct continuation of the previous study, they measure how different devices, app versions, or the passage of time affect the fingerprint.

III SYSTEM OVERVIEW

The system has to classify the mobile applications regardless of the OS of the mobile device or the network. In order to achieve this goal, we build the system in two different approaches, The first is by building from scratch and using java. The second is by using well-known libraries in python.
In both approaches the system consists of four main steps: the network trace capture. The dissection of the recorded data to sessions. The analyzations of the data and the creation of the vectors for the ML. The actual machine learning algorithms.

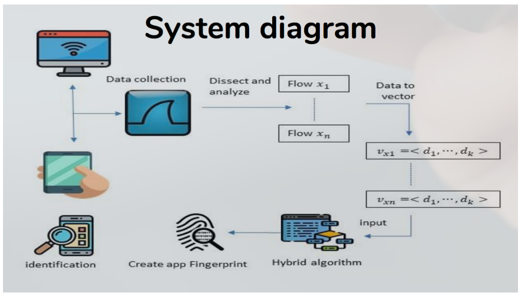


Fig. 1: The architecture of the System

III.1 Network trace capture

The main feature that distinct our project from previous works is the recording method. We used iPhone 7 and OnePlus 6 to see the results also for IOS and Android devices. We recorded around 3 GB of traffic data as Pcap files. The recording was made by using the support of Flash Networks Inc.
Flash Networks has the largest market share in the mobile Internet optimization and monitoring market with hundreds of deployments, and they provided us their servers and software for research. Both of these devices was only connected using 3G or 4G.
We used 3 popular apps as our main classification goal to simplify the recording phase, the recording of more apps can be done as needed. The mobile appli-

cation we used are Amazon, CNN and Facebook.
The recording was made in a period of 4 month in both devices and resulted several PCAP files.

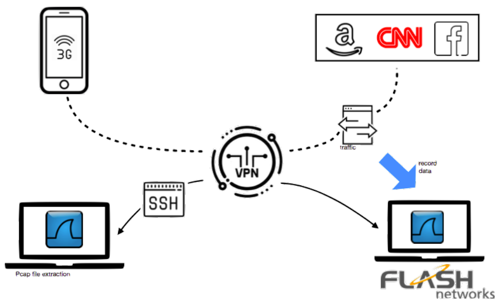


Fig. 2 : network trace capture diagram

III.2 Data dissection

We used open source Pcap file splitter named SplitCap. Each Pcap file was dissected to TCP and UDP sessions, the packets was grouped by five parameters and it's bi-directional (both direction are considered part of the same session): Source IP, Destination IP , source port , destination port and protocol. Each session was saved as a separate Pcap file.
Then, we had to Associate the recording with the correct application. To achieve this goal, we extracted the SNI - server name indicator of each session and manually associated it with the known servers of each application.
In the case of not valid SNI, the session was removed automatically.

III.3 Analyzing and features extraction

For each separated session of encrypted traffic we extracted features. All features arise from TCP layer of the packets.
Features extraction was made by two different methods. First, we used a python code [6] working with Tshark (a Wireshark Network analyser). The feature extraction takes the session network traffic as input and extracts in total 25 features. (figure 3.a)

1. Total number of packets
2. Mean of packet size
3. Variance of packet size
4. Max packet size
5. Min packet size
6. Packets in forward direction
7. Packets in backward direction
8. Bytes in forward direction
9. Bytes in backward direction
10. Min forward inter-arrival time
11. Min backward inter-arrival time
12. Max forward inter-arrival time
13. Max backward inter-arrival time
14. Standard deviation of forward inter- arrival times
15. Standard deviation of backward inter- arrival times
16. Mean forward inter-arrival time
17. Mean backward inter-arrival time
18. Min forward packet length
19. Min backward packet length
20. Max forward packet length
21. Max backward packet length
22. Std deviation of forward packet length
23. Std deviation of backward packet length
24. Mean forward packet length
25. Mean backward packet length

fig. 3 a : Features set 1

For the second features extraction method, we implemented a Java code using Pcap4J, an open source java library for crafting packets. We extracted 12 features per session. Then, we computed for each feature 5 different statistics that gave us in total 60 parameters for session. (figure 3.b.)
For the Operating system label, we parsed Pcap file's name previously defined in term of the used device.

Session features

1. session_id_length
2. comp_methods_length
3. extension_len
4. cipher_suites_length
5. handshake_version
6. ip_ttl
7. window_size
8. mss_val
9. wscale_shift
10. flags_ack
11. flags syn
12. flags_reset

Statistics computed for each one

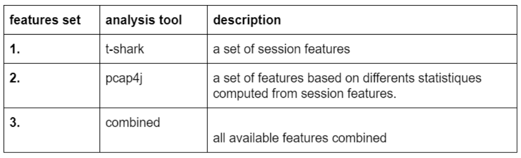
1. Mean
2. Variance
3. Standard Deviation
4. Skew
5. Kurtosis

Total features by session: 60

fig. 3. b : Features set 2

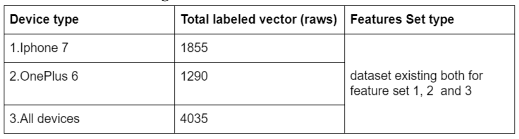
In total we distinguish three features set. One resulting from the first extraction method, one from the second method and the last from the association of the two features sets.
Once we extracted features and assigned label for each session, we created CSV data-sets compatible to supervised machine learning techniques we used.
In total we got 9 data-sets : 1.1, 1.2, 1.3, 2.1, 2.2,

2.3, 3.1, 3.2, 3.3 when 1.3 for example is the data-set with device type 1 and feature set type 3 (detailed in figure 4a, 4b).



features set	analysis tool	description
1.	t-shark	a set of session features
2.	pcap4j	a set of features based on differents statistiques computed from session features.
3.	combined	all available features combined

fig. 4.a : Features sets table.



Device type	Total labeled vector (raws)	Features Set type
1.Iphone 7	1855	dataset existing both for feature set 1, 2 and 3
2.OnePlus 6	1290	
3.All devices	4035	

fig. 4.b : Data-sets table

e.g: 1.3 is the data-set with device type 1 and feature set type 3

III.4 Machine learning Algorithm

To find the most appropriate algorithm for the classification we tested and compared both the java approach and the python approach we present the results in each of those and the conclusion.
Our motivation was to find the most accurate machine learning algorithm for final use in our classifier system.
We succeeded to reach our goals both in the java version and in the python version. The python version yielded batter results then java version by using Decision Tree algorithm.

III.4.1 JAVA ML approach

After we created the CSV file containing the vectors, we used the JAVA-ML open source library.
The algorithm we used is the k-Nearest Neighbors algorithm. "The Knn algorithm output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors" (Wikipedia).
Another way to validate our results is using Cross-validation also provided by the JAVA-ML library. We tested both Features sets in fig. 3.b .
classifier tuned on default parameters

III.4.2 python approach

After we created the CSV file containing the vectors, we used the Global classifiers comparison (scikit-learn) library.
The algorithm we used are:

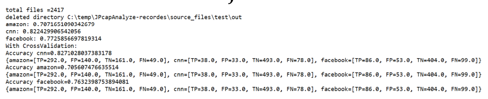
- k-Nearest Neighbors
- Decision Tree
- Random Forest

We tested both Features sets in fig. 3.a and fig. 3.b .
classifier tuned on default parameters

IV RESULT

These results simulate laboratory conditions since they are taken from data-sets that have been filtered of noise, that is filtered from the SNI that are not in our data set. (we can't relate them to any app server although they can be related)
The results are a baseline evaluation of the system performance using training and testing sets derived from single data-sets.

Result achieved with java-ML

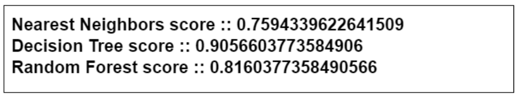


```
total files =2417
deleted directory C:\temp\JPcapAnalyze-recordes\source_files\test\out
amazon: 0.7071651090342679
cnn: 0.822429906542056
facebook: 0.7725856697819314
With CrossValidation:
Accuracy cnn=0.8271028037383178
{amazon=[TP=292.0, FP=140.0, TN=161.0, FN=49.0], cnn=[TP=38.0, FP=33.0, TN=493.0, FN=78.0], facebook=[TP=86.0, FP=53.0, TN=404.0, FN=99.0]}
Accuracy amazon=0.705607476635514
{amazon=[TP=292.0, FP=140.0, TN=161.0, FN=49.0], cnn=[TP=38.0, FP=33.0, TN=493.0, FN=78.0], facebook=[TP=86.0, FP=53.0, TN=404.0, FN=99.0]}
Accuracy facebook=0.7632398753894081
{amazon=[TP=292.0, FP=140.0, TN=161.0, FN=49.0], cnn=[TP=38.0, FP=33.0, TN=493.0, FN=78.0], facebook=[TP=86.0, FP=53.0, TN=404.0, FN=99.0]}
```

fig. 5.c : java-ML Results

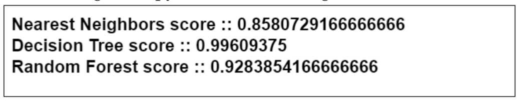
As we can see in fig 5.a the best result achieved when using java-ML library with the Knn algorithm is with an accuracy of more than 70%.

Result achieved with python scikit-learn



```
Nearest Neighbors score :: 0.7594339622641509
Decision Tree score :: 0.9056603773584906
Random Forest score :: 0.8160377358490566
```

fig. 5.b : python Results using features set 1



```
Nearest Neighbors score :: 0.8580729166666666
Decision Tree score :: 0.99609375
Random Forest score :: 0.9283854166666666
```

fig. 5.c : python Results using features set 2

As we can see in fig 5.a the best result achieved when using python scikit-learn library using the Decision Tree algorithm is with an accuracy of 99.6%.

The objectives were to identify the application at 70% accuracy. We succeeded to reach our goals both in the java version and in the python version.
These results are fairly good but may overestimate the performance of the system. This is because the training and testing sets in each case were generated from one original data-set.
The python version yielded batter results then java version by using Decision Tree algorithm that gave us very high accuracy. moreover by the results we can see that the 60 statistical features per session approach is much more accurate than the 25 values. as we can see from fig 5.b and 5.c .
By searching for best parameter and playing with different data-set we finally obtained an accuracy of 99.6 %.

V FUTURE DEVELOPMENTS

Although we are quite satisfied with our results, our proposed system is far from being perfect.
We have many things to improve the main improvements and further development are mentioned below:

- Increase the data set at least by a factor of 10.
- Record and analyze 20 more apps.
- Examine the system after a few months, see if the time is a critical factor, and what we can improve in the features selection so the Change of time will make less impact on the system.
- Improve the run time of the system so it will work in real time.
- Improve the SNI association with the correct application.
- Create a connection from the recording and the classification system.
- Automate the recordings of the mobile traffic.

References

[1] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms. Clickminer. *Towards forensic reconstruction of user-browser interactions from network traces* . In ACM CCS, 2014.

[2] M. Liberatore and B. N. Levine. *Inferring the source of encrypted http connections*. [*On the electrodynamics of moving bodies*]. In Proceedings of the 13th ACM Conference on Computer and Communications Security ,255–263, 2006.

[3] Q. Wang, A. Yahyavi, B. Kemme and W. He, `I know what you did on your smartphone: Inferring app usage over encrypted data traffic` IEEE Conference on Communications and Network Security (CNS), Florence, 2015, pp. 433-441.

[4] V. F. Taylor, R. Spolaor, M. Conti and I. Martinovic, `Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic` IEEE European Symposium on Security and Privacy , Saarbrucken, 2016, pp. 439-454.

[5] V. F. Taylor, R. Spolaor, M. Conti and I. Martinovic, `Robust Smartphone App Identification via Encrypted Network Traffic Analysis` IEEE Transactions on Information Forensics and Security, vol. 13, no. 1, pp. 63-78, Jan. 2018.

[6] Y. ivgi, `https://github.com/YairIvgi/JPcapAnalyze2` Java complete system and feature extraction code.

[7] j. Thuillier , `https://github.com/johant93/Mobile_App_classifier` Python machine learning code.