Mobile Device Fingeprinting

Technical Report, FIT BUT

Authors: Petr Matoušek, Ivana Burgetová, Malombe Victor

Technical Report no. FIT-TR-2020-05

Faculty of Information Technology, Brno University of Technology

Last modified: September 25, 2024

Contents

Abstract

Network communication of mobile devices provides valuable information about installed apps, user activities and mobile device usage which can be interesting for network management and cyber security. Based on meta data obtained from mobile device communication, we can select specific features that can identify a device or app and form a mobile device fingerprinting.

This report presents several mobile device fingerprinting techniques developed by the FIT BUT research team and discusses their usability, reliability and deployment for network monitoring and digital forensics. The presented techniques include mobile traffic profiling based on meta data obtained from common network protocols like HTTP, DNS, SSL, QUIC and DHCP. The report also shows how TLS fingerprinting method called JA3 can be applied on mobile communication. The obtained results demonstrate that combination of various features from TLS handshake together with DNS data can identify a mobile app with high precision.

Introduction

This technical report summarizes results of the research focused on identification of mobile devices and mobile applications from network communication. This process called fingerprinting is based on extracting specific features from mobile device communication. The idea behind this method says that each device or app has a specific setting that is unique and can be used to distinguish two device even if having the same hardware and operating system. The main task of mobile fingerprinting is to find out these features. Features should be obtained from protocol headers and preserve stability, uniqueness and reliability.

In the past, there were various mobile device fingerprinting methods based on active or passive approach that focused mostly on data from HTTP headers. However, with a rapid move to encrypted communication around 2017, most of that technique became unusable. Unless an Internet Provider has access to unencrypted data through a web proxy or an end system (like HTTPs server) we deal mostly with SSL/TLS traffic.

This report includes both methods that work with unencrypted traffic like HTTP, DNS, or DHCP but also with encrypted connections. We show that even when a mobile traffic is mostly encrypted, we are still able to identify an application that sends this traffic using features obtained from TLS headers and DNS traffic.

The reports include our experiments, observations but also techniques and tools used for mobile device fingerprinting.

Structure of the Report

Text of the report is structured as follows. In chapter 1 we give an overview of various network device fingerprinting methods and observe their pros and cons. Then we focus on passive mobile device fingerprinting using values from common network protocols, more specifically from HTTP headers, DNS traffic, SSL handshake, DHCP request and QUIC. We discuss reliability and stability of this multi-level profiling of mobile devices.

Chapter 3 describes our experiments with Luman App developed by University of California. This app is able to collect data from mobile app com-

munication. Its primary goal is to detect privacy data leakage. Using this app we observed obtained data and evaluate their usage for mobile apps fingerprinting.

In chapter 4 we move our focus to a specific method for TLS fingerprinting called JA3. Based on previous research we test JA3 and JA3S hashes on our datasets and observe if this approach gives more reliable results then the previous method. As shown in Section 4.5, combination of JA3, JA3s and SNI features provides a unique fingerprint for a mobile app and can be used for mobile devices fingerprinting as discussed in Section 4.6.

Acknowledgement

This work presented in this document was supported by project "Integrated platform for analysis of digital data from security incidents", 2017-2020, No.VI20172020062, granted by Ministry of Interior of the Czech Republic. The authors would also like to express their thanks to students Matej Meluš, Radovan Babic and Alberts Saulitis who participated in generation of datasets that were used for research experiments.

Chapter 1

Mobile Device Fingerprinting

1.1 About Device Fingerprinting

A mobile phone has become an essential part of personal belongings today, similarly to a wallet, ID card, or house key. Unlike desktops or laptops, we take our mobile phone always with us. A mobile phone is usually not shared with our best friends, colleagues, or family members. Its model, setting, usage, and applications reflect the person who uses it. Mobile communication, variability, and frequency of usage of mobile apps, the volume of transmitted data, a list of connected sites, etc. reveal a lot about the mobile device owner.

Based on these observations, we can search for traits that are related to specific hardware, operating system, and preferable applications. Having a snapshot of the network communication within a specified period, we can create a communication profile of the device. The profile can be used to identify the given mobile device in another communication trace.

Unlike traditional mobile fingerprinting approaches, the proposed mobile device profiling takes into account multiple identification techniques based on communication data. Our method does not require active interaction with the device or installation of the specific app like current mobile fingerprinting techniques. The communication snapshot can be obtained by simple passive data capturing using, e.g., `tcpdump`.

The approach is based on the assumption, that each operating system, device drivers, system and user applications, etc. differ in versions, settings, and implementations that keep traces in network traffic. By profiling, we extract relevant data from the captured network traffic and build a profile of the device. Captured communication usually includes both user-initiated communication, e.g., sending an email, web browsing, chatting, and also system/application-initiated communication, e.g., connectivity tests, regular updates, service synchronization, etc. Both types of communication are valuable sources of features for building a device profile.

In this paper, we describe a structure of mobile device traffic. We show what protocols can be exploited to obtain fingerprinting data. Using the combination of different identification techniques we create *a multi-level mobile device profile* that can be used for identification of a given mobile device in the network traffic. Unlike other approaches, we restrict ourselves to passive data capturing so that the result can be applied by LEAs for operational purposes or as a part of lawful interception.

Definitions.

In this part, we define two important terms related to our work: profiling and fingerprinting.

Profiling is *the process of "discovering" correlations between data in data bases that can be used to identify and represent a human or nonhuman subject (individual or group), and/or the application of profiles (sets of correlated data) to individuate and represent a subject or to identify a member of a group or category. Data mining technology is generally considered as a means by which relevant patterns are discovered and profiles are generated from larger quantities of data.*[1]

Fingerprinting is a method for collecting publicly available information called attributes or features about a remote computing device for the purpose of identification. The data forms a digital fingerprint of the remote device. Fingerprints can be used to fully or partially identify individual users or devices. Active fingerprinting requests a specific fingerprinting data from a remote device using querying, e.g., obtaining web browser parameters or network settings. Passive fingerprinting relies on data obtained by monitoring the communication of a remote device without interfering to it.

Since there is a noticeable overlap of these terms, we need to clarify how these terms are used in our research.

The term *fingerprinting* describes a method for creating a fingerprint based on the specific data, e.g., a DHCP fingerprint is derived from DHCP communication, an HTTP fingerprint uses HTTP headers, etc. Common fingerprinting methods have a limited scope of accuracy, and mostly they are not able to distinguish two individual devices with the same hardware and OS. Rather, these methods identify a group of similar devices based on the same operating system, local settings, installed applications, etc. Furthermore, computing the fingerprint requires that the device communicates with peers using the corresponding network protocol, e.g., to obtain a DHCP

[1] See Geradts, Zeno; Sommer, Peter (2008), "D6.7c: Forensic Profiling", FIDIS Deliverables, 6 (7c), part 3.2. Available at http://www.fidis.net/fileadmin/fidis/deliverables/ [March 2018].

fingerprint we need to capture DHCP communication of the tested device. If such communication is missing, this fingerprinting method fails.

By *profiling* we denote the process of device identification that is based on various data sources, e.g., a TCP/IP fingerprint, a DHCP fingerprint, etc. Profiling can also be called *multi-level fingerprinting*. The advantage of profiling is that uses various data sources. If one source is missing, there is still a possibility to create a device profile using another data source. As the definition above states, a profile represents a searched object using a set of correlated data.

We do not say that the profiling is more precise than the fingerprinting in object identification. Instead, we use the profiling as the more general term for object identification than fingerprinting.

1.2 State of the Art

Identification of mobile devices based on the captured network communication has been researched from different angles in the past. One of the viable approaches is fingerprinting based on *mobile device hardware*. This method evaluates physical characteristics of the device: the image sensor, frequency response of the speaker-microphone system, an accuracy of the accelerometer, clock skew of GPS, touch screen misalignment, etc.[6]. By this approach, we can identify a group of devices that have the same or similar hardware. Obtaining such fingerprint requires active communication with the device which is usually provided via a specifically-tailored application that extracts all necessary data from the device. Passive network monitoring cannot easily obtain hardware features.

Another popular fingerprinting approach is *browser fingerprinting* which searches for web browser features, e.g., *version, installed plugins, system fonts, screen size, color depth, touch support, time zone, installed plugins, language support* [8, 14, 21]. Some of these features transmitted within HTTP headers, e.g., *Accept-Encoding*, *Accept-Language*, *User-Agent*, can be extracted directly from the captured network traffic. Application of browser fingerprinting on mobile devices is, however, limited. [21] shows that contrary to web browsers on desktops or laptops, the fingerprints taken from mobile devices are far from unique. This is due to the application isolation mode, where mobile applications run in sandbox. This means, that in case of installing a new app, the font list available in the phone's web browser does not change. For the same reason, mobile phone browser usually does not feature a browser plug-in model. This limits a set of data transmitted in HTTP headers and reduces a list of features for creating the fingerprint.

In addition, most of browser fingerprinting features, e.g., *charset, language, time zone, plugin versions, screen resolution, font list*, can be obtained using active communication only, e.g., by running a JavaScript or

Flash Applet in the browser, which does not work well for passive network monitoring.

Today, most of the Internet services, that uses browser fingerprinting for mobile users' identification, implement cookies-based identification or active fingerprinting. Surely, cookies are excellent tool for device identification, however, their persistency is limited. Also, cookies are generated on per target base. This means that connection to a new web side imposes generation of new cookies which limits application of cookies for fingerprinting.

Another interesting approach in mobile user profiling observes *personal traits*. As researched by [17, 25, 24], mobile phones are a subject of tracking user applications via advertising or tracking libraries. Using these libraries, we can obtain a list of installed applications at the mobile device [25]. This list can disclose age of the person (child, teenager, adult), family status (single, married, parent), hobbies, preferred activities, etc. Similarly, by tracking the user activity, we can analyze the big-five personality traits, e.g., extroversion, agreeableness, conscientiousness, emotional stability, and openness to experience, see [10]. Such traits are deducted from a list of installed applications, number of Bluetooth connections, number of incoming/outgoing text messages and their length, number of incoming and outgoing calls, their average duration, number of unique phone numbers, etc. Although the results obtained by this approach are remarkable, an active access to the the device is required.

Since our approach is restricted on captured network communication only, we focus on fingerprinting methods that extract features from it. Table 1.1 gives an overview of available communication protocols that are typically used for mobile device fingerprinting.

Layer	**Protocol**	**Features**
L7	DHCP	DHCP options, vendor
L7	SSL/TLS	SSL/TLS version, cipher-suite list, TLS extensions
L7	HTTP	User-Agent, Accept, Accept-Language, Accept-Encoding, Accept-Charset
L7	DNS	Query patterns, time interval
L4	TCP	Window Size, Window scale, MSS, TCP options, TCP flags, Timestamp increment)
L3	IPv4, IPv6	Initial TTL, IP options, Don't fragment flag
L2	Ethernet	MAC address, MTU

Table 1.1: Fingerprinting Network Communication

Very popular fingerprinting method is *OS fingerprinting* that utilizes values from L2-L4 headers, mostly IP and TCP headers. Well-known OS fin-

gerprinting tools are `nmap`[2] for active fingerprinting, and `p0f`[3] for passive fingerprinting. Although these tools are focused on general OSes, they also identify mobile device OSes. Since using L3 and L4 headers, OS fingerprinting does not work well for communication that is subject to NAT translation, tunneling, proxy, or other techniques that break end-to-end connection on L3 or L4 layers.

When L3 and L4 layers do not provide reasonable results, we can move on to application layer (L7). One of the popular identification technique is DHCP fingerprinting. DHCP fingerprinting is built on assumption, that each DHCP client implementation uses different configuration, especially DHCP options *hostname*, *requested-parameters*, *vendor-id*, *client-id*, *list of options*, etc. This data together with a MAC address of the sending device can be used for unique identification [29]. For example, the fingerprinting database Fingerbank[4] contains around 4,900 DHCPv4 and DHCPv6 fingerprints.

The scope of usage of DHCP fingerprints is limited to LAN only since DHCP communication is broadcasted and the first router on the network filters DHCP messages. Thus, DHCP fingerprinting does not work outside the LAN of the sender device.

DNS fingerprinting observes DNS communication and analysis characteristics of DNS queries specific to each OS, e.g., unique domain names, query patterns, time intervals [28]. From point of view of mobile device identification, it seems more reasonable to observe frequency and distribution of DNS queries rather than time interval patterns which often depend on DNS client configuration, TTL value of DNS records and DNS cache setting. In addition, the authors of the above cited paper were able to identify an OS of the device, but not the specific device. Nevertheless, this direction seems to be promising since DNS traffic is not encrypted and provides interesting data concerning DNS resolver configuration, local system setting, and user communication.

Utilization of HTTP communication for fingerprinting has been already discussed in the paragraph about *browser fingerprinting*. Because of security reasons, most of HTTP communication is encrypted today. This limits HTTP fingerprinting methods to application gateways that decapsulate the encrypted HTTP traffic. Without encryption, HTTP fingerprinting cannot be applied.

With the increase of SSL/TLS encryption, a new type of fingerprinting methods emerged. *SSL/TLS fingerprinting* uses *Client Hello* packets to extract *SSL/TLS version*, *cipher suite list* and *TLS extensions* that are used to create a SSL/TLS fingerprint. Experiments with SSL/TLS fingerprinting [22] proves viability of this approach. Unlike DHCP fingerprinting, this

[2] See `http://nmap.org` [April 2018].
[3] See `http://lcamtuf.coredump.cx/p0f3/` [April 2018].
[4] See `https://fingerbank.org` [April 2018].

method is not limited to LAN. Moreover, SSL/TLS fingerprints are not the subject of NAT translation or tunneling.

In our approach, we create mobile device profiles from the captured network communication obtained by passive monitoring. Thus, active approaches like *browser* or *hardware fingerprinting* are excluded. The point of traffic monitoring is very important in the approach. If the traffic is captured on the local network, ARP and DHCP communication can be analyzed, and MAC addresses and DHCP fingerprints can be extracted. If the traffic is captured outside the LAN, we are limited to IP/TCP traffic, DNS and SSL/TLS fingerprinting.

The following text describes the structure of the mobile traffic. We will see what protocols and features can be used to build a mobile device profile.

1.3 Network Multi-level Profiling

Network profiling is a technique that creates a unique profile of the device based on the available network. Having a captured network communication of the device, we (i) analyze selected protocols, (ii) extract a set of features, (iii) apply data mining methods on the features, and (iv) create the profile.

Given unknown network traffic associated with the specific user, we aim at profiling common behavior in this traffic. This behavior includes typical communication patterns with Internet services. The patterns can be expressed using entropy, traffic volume, feature distribution, temporal properties, and so on.

Common network traffic profiling techniques work with a limited amount of information acquired from the network traffic data such as information from TCP/IP headers. In this work, we consider the offline profiling method that uses information available in (almost) complete packet traces. In-depth analysis of its entire communication determines the user's device profile. To cope with the possible enormous amount of information we need to identify and prioritize different sources of information. Answers to the following questions shape the solution:

- *What communication protocols provide suitable sources of information to compose the profile?*

  We assume to profile mobile devices. It is essential to identify the communication protocols that can be observed in mobile device traffic.

- *What protocol features can identify dominant patterns / fingerprints best?*

  Each protocol has many possible features that can be considered. However, only some of them are relevant in pattern identification. A collection of identified patterns represents the fingerprint of the device.

- *How to compose a device profile from a given set of fingerprints?*

  By protocol analysis, we can identify various patterns in the device traffic. The combination of the patterns creates a fingerprint. To create a device profile, we collect the fingerprints. The profile is more than a simple enumeration of fingerprints. Some fingerprints can be quite general and shared by several devices. Moreover, weighting the contribution of fingerprints to the profile is important. Also, specific fingerprints can be unstable having temporal validity only.

- *How to efficiently match the profile in the database?*

  A compact representation of the device profile is required. An efficient algorithm has to exist for matching extracted behavior with stored profiles. Because the matching algorithm can be computationally intensive, the distributed environment can be employed to improve the performance of large databases.

1.3.1 Structure of Mobile Communication

Now, let us look at the structure of typical mobile communication. What communication protocols are common, which of them can be exploited for fingerprinting, what features can be employed for building the communication profile?

For our experiments, we created several datasets with full packet captured traffic, see Table 1.2. These datasets were created using different sets of communicating devices over a given period of time.

In our experiments, we observed communication of twelve different mobile devices within an hour. Captured data contained both user-initiated and system-initiated communication. The structure of the protocols is showed in Table 1.2. Totally, the captured traffic included 542,725 packets and 434 MB of data.

As expected, the majority of network traffic is encrypted. The number of transmission protocols is limited; we found about 25 different protocols, some of them transmitted only a few packets. The most frequent protocols are mentioned in the Table. The encrypted traffic includes HTTPS, IMAPS, SMTPS and general SSL/TLS traffic. Observable is also FB Zero protocol (used by Facebook), QUIC (QUIC UDP Internet Connection) developed by Google and OpenVPN.

Concerning unencrypted traffic which varies between 5 to 45 %, useful data includes HTTP traffic, DNS and multicast DNS packets, DHCP version 4 and 6, and L2 system traffic (ICMP, IGMP, ARP). We observed that some mobile devices tried to open IPv6 connection which was not supported in our environment, so we did not work accurately with IPv6. However, IPv6 DNS requests are included in the datasets.

	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**
Time	70 min	12 min	37 min	21 min
Size	452 MB	35 MB	423 MB	18 MB
Packets	542 725	44 699	424 922	25 525
Encrypted Traffic				
SSL/TLS	44,26 %	86,03 %	30,52 %	80,1 %
IMAPS	0,65 %	1,67 %		
FB Zero	0,65%	0,12 %	0,01 %	0,13 %
QUIC	6,15 %	3,9 %	4,74 %	8,07 %
OpenVPN			54,94 %	
Total	51,7%	91,7%	90,21 %	88,3 %
Unencrypted Traffic				
HTTP	41,47 %	1,65 %	7,55 %	2,12 %
DNS,mDNS	0,93 %	1,78 %	0,64 %	2,41 %
DHCP	0,05 %	0,13%	0,04 %	0,26 %
ICMP,IGMP	0,36 %	0,53 %	0,14 %	0,60 %
ARP	2,11 %	1,01 %	1,62 %	3,17 %
Total	47,05 %	5,12%	9,99 %	8,56 %

Table 1.2: Structure of mobile device traffic

As seen from the list of the protocols, the traffic was captured on LAN. Protocols like ARP, ICMP, IGMP or DHCP would not be seen outside the LAN. However, by observing the traffic, we can state that mobile devices communicate uses a limited set of network protocols and the majority of the traffic is encrypted. This is observation is important for further steps of device profiling.

In the next section, we give an overview of identified network protocols and their features considered in fingerprinting methods.

1.3.2 Passive Protocol Fingerprinting

Protocol fingerprinting techniques reveal differences in the use and implementation of protocols by different software implementations. Because of this, the most common Internet protocols can be a target of fingerprinting.

To get the protocol fingerprint, we identify candidate attributes. Attributes can equal to protocol fields or can be constructed from the protocol structure. For instance, *User-Agent* can be a suitable protocol attribute.

Let a be a protocol attribute. We define a set of possible values for this attribute, denoted as R_a. In case of *User-Agent* attribute this set contains all possible strings that can appear as the value of this field. This gives us the vocabulary, which can be used to create an attribute vector $\vec{t_a}$ using one-hot encoding:

- An element in the vector corresponding to the given string value is set to 1.
- All other elements are set to 0.

The length of this vector corresponds to the size of the vocabulary.

Fingerprint is represented as a (sparse) vector $\vec{t}$, which is the result of concatenation of attribute vectors:

$$\vec{t} = \vec{t}_{a_1} \cdot \cdots \cdot \vec{t}_{a_k}$$

The resulting fingerprint vector can be large, containing hundred thousand elements. Feature hashing [43] can be applied providing a suitable representation of high dimensional vectors. The main idea of feature hashing is to map the high dimensional input vectors into a lower dimensional feature space. A hash function determines the location of a feature in the more moderate dimensional vector. Although feature hashing does not provide one-to-one mapping, the collision rate is acceptable for sparse vectors.

In the next subsections, we identify a set of attributes suitable for fingerprinting of each selected protocol. The selection of attributes was made based on published literature and our experiments. Also, it is possible to compute the amount of information provided by each attribute to confirm the attribute selection.

TCP/IP communication.

Passive TCP/IP fingerprinting can determine operating system of the device based on the information in TCP and IP headers. This is possible because of subtle differences in network stack implementations that uniquely identify each operating system. Passive method is non-intrusive and relies on the observation of the normal traffic rather than actively probing the target system. The widely used passive fingerprinting tool `f0p` is rule-based and relies on the manually created database of identified signatures. The OS fingerprinting includes the following attributes:

- Initial Time to Live
- IPv4/IPv6 Option Length
- Maximum Segment Size
- TCP Window Size
- TCP Window Scaling Factor
- TCP Options and their order
- Selective Acknowledgement Option
- Content of SYN packet

DHCP communication.

DHCP fingerprinting allows identification of a device and operating system installed. DHCP supports various options that help to identify the client. In particular, DHCP header data transmitted in Discover and Request packets sent by the client is used to build a fingerprint. The most interesting is the options section where it is possible to find the operating system name, device name, vendor id, and other values. Besides, each DHCP client offers a specific set of options which depends on the operating system and its version.

For DHCP fingerprinting we use the following DHCP attributes:

- Client identifier (option 61): client MAC address
- Host name (option 12), e.g., `Galaxy-A5-2016`
- Vendor class identifier (option 60), e.g., `android-dhcp-7.0`
- Parameter Request List (option 55), e.g., `1,3,6,15,26,28,51,58,59`

There are additional parameters like domain name server, renewal time, netmask, DHCP server identifier, etc. However, these additional parameters depend on the ISP provider, so they change when connected to a different operator. The attributes mentioned above are stable. In addition, these system data are generated by the client software and cannot be easily forged by the mobile phone user. As mentioned before, the downside of DHCP fingerprinting is that it is limited to the LAN only.

DNS communication.

Most of the Internet applications rely on DNS data. By analyzing DNS data we can identify the operating system, installed applications, and user activity. Devices do not encrypt DNS communication. Therefore necessary information can be easily extracted from DNS header.

For fingerprinting, we extract attribute values from standard unicast and multicast DNS requests. The following data are considered for building DNS fingerprint:

- DNS request type: most common are A, AAAA, PTR, and ANY, i.e., values `1, 12, 28`, and `255`.
- DNS server IP address: some of DNS clients use DHCP-assigned DNS name servers, many mobile users prefer to set there own IP address, e.g., `147.229. 9.43`, `8.8.4.4`, or `8.8.8.8`.
- DNS resolved name: resolutions include not only user initiated names like `facebook.com`, but also system-initiated resolutions, e.g., `android.google`, `apis.com, appchat.xioami.net, _ipps._tcp.local` or application-initiated domains, e.g., `shop.oebb.at, media.novinky.cz`.

In addition to collecting DNS queries, it is possible to identify operating system and some applications observing DNS query patterns. Operating systems tend to send specific DNS queries regularly. It can be used as an additional technique for detecting the OS.

On the other hand, DNS fingerprint depends on user activity and time of connection. When capturing DNS data of a mobile device for an hour, we receive hundreds of unique DNS domain names. Thus, it is necessary to filter these data and find a stable set of DNS names that can be used for creating the fingerprint. Filtering can be based on the number of occurrences of unique domain names where domain names under the given threshold are eliminated from the fingerprint.

HTTP communication.

HTTP headers provide a rich source of interesting data that can be used for forensic purposes. Even a small number of HTTP packets yield variety of different values. For example, in our smallest dataset #2 (12 minutes of communication) HTTP traffic formed only 1,65% of all communication. Despite this, we detected 16 unique HTTP headers with 61 unique values. For HTTP fingerprinting the most promising headers are those that appeared on more devices and have the largest entropy of information. From our observations, we noticed that the most promising headers are *User-Agent* string, *Accept-Encoding*, *Cookie* and *Content-type*. For our purposes we limited ourselves to *User-Agent* string an *Cookie* only.

- *User-Agent* string defines name and type of the web browser, operating system, and even smart home name and version, e.g., `Microsoft-CryptoAPI/10.0`
- *Cookie* is a small piece of information stored at web browser and sent as a part of the request. It keeps relation between the server and the client even if the client reconnects. Time validity of cookies is limited and depends on the the service.

In our experiments, we notice that a single device uses a reasonably large set of different User-Agent strings. This is obvious when we realize that HTTP communication is initiated not only by web browsers but also using any app communicating over HTTP. Thus, *User-Agent* fingerprinting can quite precisely characterize communication of a particular device.

In our device profiling database, we consider a set of *User-Agent* strings to form an HTTP fingerprint of one device. For example, in our dataset #1, each identified device contained 1 to 6 unique user-agent strings. Thus, fingerprinting is based on matching as many as the possible string in the profile database. Also, user-agent string is considered as a stable feature that is not easy to change on mobile devices.

Web browser fingerprinting is one of the most common ways to track users in WWW environment.It is used primarily for marketing purposes (targeted ad for returning users), but also for checking whether the credentials have not been compromised. Most web based fingerprinting tools, e.g., Panopticlick[5], use an active approach. The user is forced to load a page that contains a JavaScript code that collects the local browser settings and forwards it to the server where data are stored in the fingerprint database.

In our approach we can rely on the information found in the HTTP communication only. Most of these attributes characterize web browser instead of a particular user:

- Accept: enumerates supported document types.
- Accept-Encoding: lists supported encoding of the documents.
- Accept-Charset: states preferred languages of the browser/agent.

In addition to the previous header fields that serve to browser identification, we also extract the following variable fields. These fields depend on the HTTP session and corresponds to user activity:

- Hostname: domain name of the server specified as a part of the HTTP request.
- Cookies: a small piece of information stored at web browser send as a part of the request.

SSL/TLS communication.

A significant amount of network communication is encrypted with SSL/TLS. This reduces fingerprint capabilities based on the communication content. However, by observing information exchanged during secure connection establishment, it is possible to identify the implementation of SSL/TLS library and its version. Operating systems usually contain variety of different SSL/TLS implementations with unique identifiers and cipher suite sets. For SSL fingerprinting, the following attributes can be exploited from *Client Hello* opening message:

- SSL/TLS Version. Most common versions are `SSL 3.0`, `TLS 1.0`, `TLS 1.1` and `TLS 1.2`.
- *Cipher Suite List* is a list of available combination of cryptographic algorithms for encryption and message protecting. Each cipher suite is identified by a standard value defined by IANA[6]. The content and

[5] See https://panopticlick.eff.org/about [April 2018].
[6] See https://www.iana.org/assignments/tls-parameters/ [May 2018].

order of available cipher suite items depend on the SSL/TLS library and its version. Example of a cipher suite list follows: `49195, 49199, 49162, 49171, 49172, 156, 47, 53`.

- *Supported Extensions*: SSL/TLS library also provides a list of available extensions that can be also used for fingerprinting, e.g., `0, 11, 10, 35, 13, 13172, 16`

Each mobile device contains several SSL/TLS-based applications with different SSL/TLS libraries. Thus, SSL fingerprint includes a list of versions, cipher suite lists, and extension lists. Our profile database created using dataset1 contained 1 to tens different SSL fingerprints per device. Thus, when providing matching, the threshold must be defined for proper analysis.

1.3.3 Device Profiling

The device profile is a collection of vectors produced by the protocol fingerprinting modules. Each fingerprinting module F (e.g., DHCP, SSL, DNS, HTTP) is used to compute a fingerprint vector $\vec{t}_F$ for the known traffic d. The device profile P is an ordered collection of computed vectors:

$$P = \langle \vec{t}_{F_i} | \textsf{forall fingerprinting modules } F_1 \dots F_n : \vec{t}_{F_i} = T_{F_i}(d) \rangle,$$

where d is a traffic known to be generated by the target device and T_F is a fingerprinting function of module F. The profile is stored in the profile database D, and it is used for device matching as described in the next section.

1.3.4 Device Matching

The created device profile is applied to unknown network traffic to compute the probability that the network traffic was generated by a known device. The evaluation whether the known device generated the captured traffic is done by computing the similarity of the vectors that are the result of processing fingerprinting modules for the captured traffic. Computed fingerprints are stored in the device profile database.

The similarity is expressed as the distance between vectors. A distance between vectors can be computed by various methods, e.g., Mahalanobis distance, Hamming distance, or Euclidean distance. For two arbitrary vectors $\vec{v}_1, \vec{v}_2$ we define $dist_{F_i}(\vec{v}_1, \vec{v}_2) \in (0 \dots 1)$ be a distance function for fingerprinting module F_i. Given a profile $P = \langle \vec{t}_{F_1}, \dots, \vec{t}_{F_n} \rangle$ and an ordered set of vectors $Q = \langle T_{F_1}(d), \dots, T_{F_n}(d) \rangle$ computed by fingerprinting modules for observed traffic d,we compute the *distance vector* $\vec{s}$ as follows:

$$\vec{s} = \langle dist(\vec{t}_{F_i}, T_{F_i}(d)) | \textsf{forall i} : 1 \leq i \leq n \rangle$$

Let $\vec{w}$ is a weight vector, we compute the *similarity score* from the distance vector as follows:

$$s = \sum_{i=1}^{n} \vec{s}[i] \cdot \vec{w}[i]$$

The weight vector allows us to specify the contribution of the individual fingerprints to the similarity score. The similarity score is used to find a most likely match for the target device within a database of stored profiles.

1.4 Case Study

In this section, we provide demonstration of profile computation and device matching using the previously described approach. We present rather a simplistic system that uses only two fingerprinting modules, namely, HTTP and SSL. The example presents the profile for a smartphone running Android 7.0. The captured communication is processed by `tshark` tool that extracts HTTP, and SSL information, respectively. Next, this data is passed to fingerprint modules.

1.4.1 HTTP Fingerprinting

Various HTTP headers provide the main source of information for HTTP fingerprinting. *User-Agent* field gives a very detailed description of the client's platform. In this example, we combine *User-Agent* and *Accept-Language* attributes. Both fields have a string representation, and their value can be directly used for representing elements of the vector. An example is shown in Fig. 1.1.

1.4.2 TLS Fingerprinting

TLS Cipher Suites attribute is represented as a string that enumerates all supported cipher suites. For instance, the following cipher suite list

```
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA   (0xc014)
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA (0xc022)
TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA (0xc021)
```

is represented as `"49172,49162,49186,49185"`. The string representation is suitable for both the occurrences and the order of cipher suite items that provide valuable information for fingerprinting.

TLS extensions enable to enrich the functionality provided by TLS message format. A list of extensions can be extracted from the TLS communication. We are only interested in the list of extensions offered/required by the TLS client. Again, the list is represented as a single string, for instance, `0,11,10,35`, where each number corresponds to an TLS extension.

By extracting TLS Cipher Suite and TLS Extension strings from the captured communication belonging to the device, we compose its TLS fingerprint. An example of the fingerprint is shown in Fig. 1.1.

1.4.3 Device Profile Matching

The profile of a device considered in this example is represented as an collection of computed fingerprints:

$$P_1 = \{(http, \langle 1,0,0,0,1,0,0,1,1,1\rangle), (tls, \langle 1,0,1,1,1,0,0,0,1,0\rangle\}\rangle$$

We use the same distance function that computes how many elements that occurs in $\vec{y}$ also occurs in $\vec{x}$:

$$dist(\vec{x}, \vec{y}) = \frac{\text{number of matching ones in x and y}}{\text{total number of ones in y}}$$

To demonstrate the profile matching methods, we consider two unknown devices for which we captured the communication. The captured communication of these devices is analyzed and profiles d_2 and d_3 are computed:

$$d_2 = \{(http, \langle 0,0,0,0,1,0,0,0,1,0\rangle), (tls, \langle 1,0,1,0,0,1,0,0,1,0\rangle)\}$$
$$d_3 = \{(http, \langle 0,0,0,0,0,1,1,0,1,1\rangle), (tls, \langle 0,1,0,1,1,0,0,1,1,0\rangle)\}$$

Using defined distance function, we compute similarity vectors:

$$\vec{s_2} = \langle 1, 0.75\rangle$$
$$\vec{s_3} = \langle 0.5, 0.6\rangle$$

Given weight vector, $\vec{w} = \langle 0.6, 0.4\rangle$, which slightly prioritizes HTTP fingerprints, we compute the final similarity scores as follows:

$$s_2 = 0.9 \qquad s_3 = 0.54$$

The first device has a better match with the profile than the other device. Indeed, the device is also Android smartphone running the same version of the operating system as the known device. The other device is a Windows Phone which has smaller similarity considering the given fingerprints.

The presented example was radically simplified to demonstrate the principles of the presented profiling method. In the example, it is only possible to detect different groups of devices, e.g., different operating systems and versions. The full profiling method considers much more attributes as inputs to differentiate the devices also by observing activities of users and applications.

$\vec{t}_{http} =$

u_1	u_2	u_3	u_4	u_5	u_6	u_7	e_1	e_2	e_3
1	0	0	0	1	0	0	1	1	1

```
u_1 = "AndroidDownloadManager/7.0 (Linux; U; Android 7.0; SM-A510F Build/NRD90M)"
u_2 = "AndroidDownloadManager/7.1.1 (Linux; U; Android 7.1.1; E5823 Build/32.4.A.1.54)"
u_3 = "Dalvik/1.6.0 (Linux; U; Android 4.3; GT-I9301I Build/JSS15J)"
u_4 = "Dalvik/2.1.0 (Linux; U; Android 5.1; Amazfit Sports Watch Build/LMY47D)"
u_5 = "Dalvik/2.1.0 (Linux; U; Android 7.0; Redmi Note 4 MIUI/V8.5.8.0.NCFMIED)"
u_6 = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/61.0.3163.100 Safari/537.36"
u_7 = "Windows-Update-Agent/10.0.10011.16384 Client-Protocol/1.58"
e_1 = ""
e_2 = "cs"
e_3 = "en"
```

a) HTTP Fingerprint Vector

$\vec{t}_{tls} =:$

c_1	c_2	c_3	c_4	x_1	x_2	x_3	x_4	x_5	x_6
1	0	1	1	1	0	0	0	1	0

```
c_1 = "10,5,4,255"
c_2 = "49196,49195,49200,49199,159,158,156,61,60,53,47,10,5,4"
c_3 = "49196,49195,49200,49199,159,158,61,60,53,47,10"
c_4 = "49196,49195,49200,49199,157,156,61,60,53,47,10"
x_1 = ""
x_2 = "0,10,11,13,35,23,65281"
x_3 = "0,5,10,11,13,35,23,65281"
x_4 = "0,5,10,11,13,35,16,23,24,6528"
x_5 = "0,5,10,11,13,35,16,23,65281"
x_6 = "0,5,10,11,13,35,23,24,65281"
```

b) TLS Fingerprint Vector

Figure 1.1: An example of HTTP and TLS Fingerprint vectors

1.4.4 Experiments

The above written approach was implemented using a set of scripts for analysis of HTTP, DNS, SSL, QUIC, and DHCP communication together with profile matching[7].

We performed experiments with several mobile devices obtaining preliminary results to evaluate the capabilities of the presented method. We captured the communication of about dozen devices at different periods. For the experiment we used four input datasets (see Table 1.2). We took the first dataset for creating protocol fingerprints of these devices. The rest of datasets were used to evaluate the matching algorithm. In experiments, we

[7]The implementation is available at https://github.com/matousp/mobile-profiling [June 2020]

DHCP	TLS	DNS	HTTP
1.00	0.33	0.54	0.90

Table 1.3: Accuracy of protocol fingerprint

considered four fingerprint modules, namely, DHCP, TLS, DNS, and HTTP.

We compute the accuracy of the individual protocol fingerprints. The overall results are presented in Table 1.3.

While DHCP fingerprint can be used to identify the device with the absolute accuracy, the disadvantage is that DHCP communication is not usually available. The trivial TLS and DNS fingerprint functions give unacceptable results. The problem of DNS is that it relies on actual resolution performed by the device. Some data preprocessing and more sophisticated statistical methods would be necessary to create a more characteristic fingerprint of the device. Finally, HTTP fingerprint that only considers *User-Agent* attribute provides more exciting results. However, we are aware that this is because the set of device used in this experiment was rather small.

1.5 Summary

Digital device profiling aims at assist forensics investigator in identifying a possible user digital profile. In this chapter, we presented a digital device profiling method specifically focused to mobile devices, although this type of analysis is suitable to all Internet enabled devices.

The profile is computed by analysis of network communication only. Because of this, the method is passive, which enables to use it in various scenarios. Similarly to other methods that deal with full packet capture, there are strong privacy concerns.

Although most of the communication is encrypted to protect user data, by creating a user profile it is possible to reveal sensitive information about the user. When necessary for profiling, it is possible to analyze only communication metadata. For most protocols, we consider only header fields.

During experiments, we did the following observation:

- It is necessary to combine the fingerprints to create a profile. A single fingerprint is not enough to reliably discriminate different devices.
- The SSL fingerprint is not as unique as stated in [22]; It may be because our communication samples were rather short, and most of the applications use the SSL library provided by the platform.
- Short communication snapshots (about 30 min) can provide enough data for matching the profile. However, for some protocol fingerprinting modules, the date needs to be filtered or preprocessed to remove noise and infrequent patterns, e.g., DNS.

The presented work provides preliminary results on the possibility to determine a reliable profile of a mobile device user. We prepared dataset containing the frequent smartphone communication and identified the set of the most used Internet protocols. Then, based on the literature review we recognized the possible attributes to form a collection of features for fingerprinting. The individual protocol fingerprints were evaluated with respect to accuracy. It was shown that no fingerprint gives accurate results by itself. We also demonstrated the method to create a profile and compute the best match for the unknown device.

Chapter 2

Mobile Fingerprinting Using DNS

2.1 Identifying a Mobile Device Using DNS Fingerprinting

Network communication of mobile devices is a valuable source of information about the device. From captured communication, it is possible to infer what applications are installed on the system [24], or we can observe user activity [10]. The goal of this study is to observe Domain Name System (DNS) communication and extract specific features from DNS packets that can be used to identify a device. For data processing we employ frequency analysis of DNS queries.

2.1.1 Background

DNS system typically translates a domain name, e.g., www.vutbr.cz, to an IP address, e.g., 147.229.2.90. Domain name translation is required for any communication over IP. Before an IP datagram is sent, a DNS client sends a DNS query to the DNS server for domain name resolution. The client forwards the query to its primary DNS server that is either manually or dynamically configured. By observing DNS queries, their types, e.g., A, AAAA, PTR, or requested domains, we can learn much about the device. For instance, if the system supports IPv6, it requests AAAA records in DNS. Also, most of installed applications contact vendor servers for updates or synchronization, which yields in resolution of DNS names like spotify.com, github.com, facebook.net. Similarly to applications, we can identify the operating system by observing queries to DNS domains like android.google.com, xioami.net, etc. Based on the frequency and uniqueness of requested domain names we can create a DNS fingerprint that identifies a unique device.

In our scenario, we will analyze captured DNS communication sent by a

set of mobile devices. First, we process a training dataset where we extract selected fields from DNS communication. Based on frequencies of requested domain name look-ups we create a DNS fingerprint for each device detected in the dataset. In the second phase this DNS fingerprint is compared with fingerprints obtained from other datasets. If similarity of fingerprints from two datasets is high enough, we can deduce that DNS communication comes from the same source device.

The hypothesis behind the research is that almost every software installed on the device leaves a unique trace in DNS communication. By collecting all these traces we can describe an individual device and distinguish one device from others.

2.1.2 Input Data

As source data, we use captured DNS communication. From DNS communication we extract specific values using `tshark` command:

```
tshark -r dns.pcap -T fields -E separator=";"
  -e ip.src -e ip.dst -e dns.qry.type -e dns.qry.name
  "dns.flags.response eq 0 and ip"
```

Using this command, we obtain the following values:

- Source IP address (DNS client's address).
- Destination IP address (DNS server's address).
- DNS query type, e.g., A, AAAA, PTR, etc.
- Requested domain name, e.g., www.googleapis.com.

The source IP address is used to classify DNS requests that belong to the same device. The destination address identifies a primary DNS server that is configured on the device.

Tshark provides a list of tuples *{srcIP, serverIP, query_type, requested_domain}* which are later transformed into a table where each device identified by an IP address will be assigned a vector describing frequency of all DNS queries found in the dataset, see Figure 2.1. Query type 1 means A record. The frequency vector forms a DNS fingerprint of a device.

2.1.3 Method

The naïve solution uses raw frequency vector to build a DNS fingerprint. Then a DNS fingerprint of an unknown device can be compared to a set of DNS fingerprints of known devices. Based on similarities of two DNS fingerprints we can say how similar these two devices are.

Figure 2.1: Creating a frequency table for DNS queries

However, raw frequency matching without normalization does not work well with real DNS data. Thus, we applied TF-IDF (Term Frequency–Inverse Term Frequency) method [27]. TF-IDF method includes normalization and weighting. The method is typically used to assign a document to the most similar document taken from a set of documents based on frequency of terms that are found in documents. In case of DNS fingerprinting, we will use domain names to represent TF-IDF terms. For each term in the individual document, Term Frequency value $tf_{t,d}$ is computed using frequency of term t in document d. Normalized Term Frequency is given by the following formula:

$$tf_{t,d} = 0.5 + \frac{0.5 * f_{t,d}}{MaxFreq(d)} \tag{2.1}$$

where $MaxFreq(d)$ is maximal frequency of any term in document d.

Term frequency suffers from a serious problem: all terms are considered equally important when it comes to assessing relevancy on a query. This means, when a term is present in almost every document, it distorts the ability to distinguish documents based on term frequency. Using *Inverse Document Frequency* (IDF) we can weight frequency by a number of documents in the collection that contains term t. Thus, terms that are present in the small number of documents will be given the higher weight. IDF can be computed using the following formula [27]:

$$idf_t = \log \frac{N}{df_t} \tag{2.2}$$

where N is a number of all documents and df_t is a number of documents that contain term t. Combination of TF and IDF creates the TF-IDF weighting scheme that assigns a weight to term t in document d by the following equation:

$$tfidf_{t,d} = tf_{t,d} \times idf_t \tag{2.3}$$

2.1.4 Evaluation

In our case, the DNS fingerprint is expressed as a TF-IDF vector that contains weighted and normalized frequencies of domain names found in the dataset. Table 2.1 displays frequencies for selected domain names of devices F0 to F3. Column `F0` denotes an unknown device that we will be matched against fingerprints of devices F1 to F3. For example, the first domain name

	F0	F1	F2	F3
clients4.google.com	1	1	2	16
graph.facebook.com	2	0	9	0
mtalk.google.com	4	0	2	1
pxl.jivox.com	1	1	0	0
www.google.com	6	3	5	8
www.googleapis.com	4	6	8	6

Table 2.1: Frequency of domain name look-ups

`clients4.google.com` has been found in communication of all devices with frequencies 1, 1, 2, or 16 respectively. Domain `px1.jivox.com` appeared only one time in communication of device F0 and F1.

Based on raw frequencies, we can compute $tf_{t,d}$ and idf_t values for each domain name using equations (2.1) and (2.2), see Table 2.2. Each column

	TF_{F0}	TF_{F1}	TF_{F2}	TF_{F3}	IDF
clients4.google.com	0,583	0,583	0,611	1,0	0
graph.facebook.com	0,667	0	1,0	0	0,477
mtalk.google.com	0,833	0	0,611	0,531	0,176
pxl.jivox.com	0,583	0,583	0	0	0,477
www.google.com	1,0	0,75	0,778	0,75	0
www.googleapis.com	0,833	1,0	0,944	0,688	0

Table 2.2: TF and IDF values for found domains

TF_{xx} includes a list of TF frequencies for all domain names requested by a given device. By applying IDF frequency using equation (2.3), we get an TF-IDF value that represents a DNS fingerprint of the device.

Comparison of DNS fingerprints can be evaluated using cosine similarity [18]. The resulting score describes a level of similarity between documents q and d, in our case between two DNS fingerprints:

$$score(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|} \tag{2.4}$$

$\vec{V}(q) \cdot \vec{V}(d)$ is a dot product of weighted DNS fingerprints $tfidf_{t,d}$. For our input data, similarity score of DNS fingerprint $F0$ towards DNS fingerprints

$F1$, $F2$ and $F3$ is in Table 2.3. Based on the score, the most similar fingerprint to F0 seems to be F2.

	F1	F2	F3
F0	0.621	0.767	0.333

Table 2.3: Score values for dataset F0 towards F1 to F3.

2.1.5 Experimental results

In this section, we present the results achieved with the proposed method on real datasets described in section 1.3.1. We present the results achieved for the datasets no. 1 and 2 as this pair of datasets contains the highest number of identical devices (see table 2.4).

	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**
Dataset1	x	6	2	2
Dataset2	6	x	4	3
Dataset3	2	4	x	3
Dataset4	2	3	3	x

Table 2.4: Number of identical devices for each pair of datasets

We conducted two types of experiments. In the first one, we utilize all DNS queries captured in our datasets. For this experiment, the terms for TF-IDF method are composed of the type of DNS query and requested domain name. In the second type, we focused only on the unicast queries that constitute the majority of the DNS data. Here, the terms correspond solely to reguested domain name. In both types of experiments, we matched the devices from the dataset no. 2 against the devices from the dataset no. 1. The individual devices are denoted by special identifiers derived from their MAC addresses.

Table 2.5 shows the results of the first experiment, where all DNS queries were utilized. We can see, that 4 devices (10bb, c87c, b4c0, c072) from the dataset no. 2 were correctly identified. On the other hand, the method is not capable to correctly identify the remaining two devices (4073, 70a3). This problem can be caused by several reasons. As the DNS traffic depends on the user activity, presumably the activity on these devices differ so much in the individual datasets, that we cannot match the correct device.

In order to evaluate TF-IDF method, we also compare the results with the method based on the simple measure of the similarity - the number of identical terms (see table 2.6) between the given pair of devices. The results of both methods are similar, hovewer TF-IDF method seems to be able distinguish identical devices slightly better (e.g. device c072).

	10bb	c87c	b4c0	4073	c072	70a3	8c2a	1019	dc9f	10e5	a4c0
10bb	***0,463***	0,128	0,001	0,313	0,088	0,057	0,026	0,039	0,233	0,204	0,312
c87c	0,125	***0,574***	0,433	0,188	0,013	0,163	0,243	0,350	0,068	0,170	0,091
b4c0	0,001	0,000	***0.527***	0,001	0,004	0,004	0,024	0,213	0,001	0,001	0
4073	0,240	0,366	*0,628*	**0.120**	0,059	0,029	0,452	0,073	0,069	0,107	0,017
c072	0	0,018	0	0,040	***0,883***	0,084	0,022	0	0,045	0	0,448
70a3	0,292	0,130	0,018	0,043	0,004	**0,074**	0,153	0,168	*0,502*	0,023	0,056

Table 2.5: TF-IDF score for each pair of mobile devices from datasets no. 1 and 2. Columns represent the devices from the dataset no. 1 and rows denote the devices from the dataset no. 2. Scores for matching devices are highlighted by bold. Maximal score in each row is highlighted by italic.

	10bb	c87c	b4c0	4073	c072	70a3	8c2a	1019	dc9f	10e5	a4c0
10bb	***11***	10	1	10	2	5	6	4	8	5	7
c87c	12	***35***	28	16	2	11	25	18	12	10	8
b4c0	1	1	***12***	1	1	1	2	5	1	1	0
4073	13	26	*31*	**9**	3	5	19	8	7	6	3
c072	0	1	0	1	***2***	1	1	0	1	0	2
70a3	8	*10*	2	5	1	**5**	8	7	6	3	4

Table 2.6: The number of identical terms for each pair of mobile devices from datasets no. 1 and 2. Columns represent the devices from the dataset no. 1 and rows denote the devices from the dataset no. 2. The numbers of identical terms for matching devices are highlighted by bold. Maximal value in each row is highlighted by italic.

	10bb	c87c	b4c0	4073	c072	70a3	8c2a	1019	dc9f	10e5	a4c0
10bb	***0,228***	0,637	0	0,155	0,034	0,045	0,033	0,011	0,094	0,087	0,165
c87c	0,073	**0,201**	*0,222*	0,105	0,006	0,170	0,102	0,093	0,033	0,081	0,059
b4c0	0,001	0	***0.244***	0,001	0,001	0,004	0,009	0,051	0,001	0,001	0
4073	0,140	0,121	*0,322*	**0.071**	0,022	0,030	0,075	0,010	0,012	0,051	0,011
c072	0	0,024	0	0,050	**0,103**	*0,192*	0,021	0	0,059	0	0,087
70a3	*0,207*	0,068	0,012	0,035	0,002	**0,110**	0,048	0,034	0,007	0,016	0,048

Table 2.7: TF-IDF score for each pair of devices from datasets no. 1 and 2 derived from unicast data only. Columns represent the devices from the dataset no. 1 and rows denote the devices from the dataset no. 2. Scores for matching devices are highlighted by bold. Maximal score in each row is highlighted by italic.

The results of the second experiment (unicast data) are presented in table 2.7. In this experiment, requested domain names directly constitute the terms for TF-IDF method. We ommited type of DNS query and also multicast data. This choice significantly decrease the utility of this method. Only 2 devices out of 6 were correctly identified in this case. This experiment shows that it does not make sense to reduce the volume of the data. The more data, the better results can be achieved.

2.1.6 Summary

The presented case study shows how TF-IDF method can be applied on DNS traffic and used for devices identification using fingerprinting. Experimental results show that the TF-IDF method is useful tool for mobile device identifying. On the other hand, the usability of the DNS traffic is questionable. Identification process requires the stability of the user's behavior. The DNS traffic depends on the actual needs and interests of the user. Reliable DNS fingeprinting would require additional data and methods for their preprocessing.

Chapter 3

Observing Mobile Privacy Using Lumen

3.1 Motivation

For mobile app fingerprinting, we need to understand what traffic is sent by a specific application. To capture mobile apps traffic we decided to evaluated tool *Lumen* that was created by Int. computer Science Institute in University of California, Berkeley and IMDEA Networks Institute, Madrid, Spain. Lumen App is available through Google store[1].

Lumen[2] is a tool that helps you to keep user personal data under control and obtain network traffic logs. It analyzes the app's traffic to identify personal information leaks and the organizations collecting such sensitive data.

This chapter brings results of our experiments with Lumen. The goal of these experiments is to evaluate what information about mobile app traffic can be obtained by Lumen and if this app can be useful for automated creation of mobil apps profiles.

The authors state that With Lumen, completely anonymous traffic traces in the wild can be obtained. Lumen collects aggregated and anonymized information about how your mobile apps connect with online services. In addition, Lumen does not export any private information from the phone: all personal data remains with the user.

3.2 Lumen App

Lumen allows a user to select and block flows to gain control over user personal data and the traffic emanating from Android apps. Lumen gives for

[1] See https://play.google.com/store/apps/details?id=edu.berkeley.icsi.haystack [June 2020]

[2] See https://haystack.mobi/ [June 2020]

each app installed on the phone a list of domains that the app communicates with. Those domains associated with advertising or tracking services will be indicated with an eye icon. The user is able to select the domains he/she wants to block.

Some apps may use encrypted channels to upload user personal information to their online servers. Lumen can exploit apps that do not use correctly TLS to perform TLS interception (also known as man-in-the-middle) locally on the device. This allows a user to understand what data your apps leak over encrypted channels.

3.2.1 Testing Environment

In our experiments, we used Lumen version 2.2.2 which supports Android version 4.2 and higher.

The experiments were provided on the mobile device TECNO-J8 with CPU core-count 4 and CPU frequency 1 3GHz. Internal storage had 16 GB, RAM 2GB. The screen resolution was 1280X720. The device was equipped with Android version 5.1.

3.3 Experiments

For monitoring, we used 25 mobile apps: Boomplay, Chrome, Downloads, Google, Play Store, Duolingo, Facebook, Messenger, FMWhatsApp 2, Drive, Maps, Photos, Tasks, Google Backup Transport, Gmail, Google, Google Calendar Sync, YouTube, Weather, TikTok, Equa bank, KB Klic, Nextbike, Mobilni banka, Telegram and Android system (root).

During experiments we noticed that 88.7% traffic was transmitted by HTTPs, 4,1% was XMPP, 3,1% was HTTP, and 4.1% other traffic.

We analyzed communication of 8.951 connections to 288 unique IP addresses. Analyzed traffic had 811 MB.

3.3.1 Results

The following table brings results obtained by Lumen App. It lists all detected privacy leaks with their description and mobile apps that were involved in these leaks.

Table 3.1: Detected Privacy Leaks

Type	Risk Level	Description	Leaked Values	Apps
Android ID	High Risk	This value allows ad networks and online trackers to identify you uniquely as a unique Google user for tracking, surveillance or advertising purposes. This allows them to track you uniquely across platforms, as when you are surfing the web.	8ac0ea9c54a508ca	Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 14:41:37 TikTok Organization: musical.ly Number of Times: 1 Last time: 04/28/2020 13:03:52 Equa bank Organization: equamobile.cz Number of Times: 1 Last time: 04/28/2020 10:57:23
Device Serial	High Risk	This value allows ad networks and online trackers to identify you uniquely for tracking, surveillance or advertising purposes	352770081284400	hiOS Android system (root) OS Organization: reallytek.com Number of Times: 1 Last time: 04/28/2020 10:46:09
IMSI	High Risk	The IMSI (International Mobile Station Equipment Identity) is a value stored in your SIM card that identifies your device uniquely. The receiving organization can use this information to track your traffic and your online behavior. Applications leaking the IMSI can cause serious privacy violations and ease online tracking and surveillance when they send this data over insecure protocols like HTTP. If you do not want a given app to access this information, you can disable their access in your system settings under the apps category	639031628116217	hiOS Android system (root) OS Organization: reallytek.com Number of Times: 1 Last time: 04/28/2020 10:46:09
Account (com. facebook. auth. Iogin)	High Risk	Many applications request permissions to access your online service accounts. This ranges from your name to your Google account and any social media service accounts. If Lumen identifies an app leaking this information, you may need to consider whether this is legitimate (required by the app itself) or not. If you do not want a given app to access this information, you can disable their access in your system settings under the apps category	Facebook	Boomplay Organization: facebook.com Number of Times: 35 Last time: 04/28/2020 14:46:18 Boomplay Organization. fbcdn.net Number of Times: 3 Last time: 04/28/2020 12:41:36 Duolingo Organization: facebook.com Number of Times: 5 Last time: 04/28/2020 11:16:14 Duolingo Organization: duolingo.com Number of Times: 3 Last time: 04/28/2020 10:58:01 Duolingo Organization: fbcdn.net Number of Times: 1 Last time: 04/28/2020 10:56:12 Messenger Organization: facebook com Number of Times: 1 Last time: 04/28/2020 14:41:37 Messenger Organization: fb com Number of Times: Last time: 04/28/2020 13:07:35

				PHX Browser Organization: facebook com Number of Times: 6 Last time: 04/28/2020 16:01:03 TikTok Organization: facebook.com Number of Times: 13 Last time: 04/28/2020 16:00:47 nextbike Organization: facebook com Number of Times: 1 Last time: 04/28/2020 09:18:45
Serial Number	High Risk	The serial number identifies uniquely your device. The receiving organization can use this information to track your traffic and your online behavior. Applications leaking the IMSI can cause serious privacy violations and ease online tracking and surveillance when they send this data over insecure protocols like HTTP. Applications do not require any permission to read and leak this unique identifier.	0153801720700690	Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 14:41:37 Android system (root) Organization: reallytek.com Number of Times: 1 Last time: 04/28/2020 10:46:09
Installed Apps	Mid Risk	Apps may monitor which other apps you have installed and run on your device. This allows them to find out many things about your personality, taste, and demographics. It also allows tracers and advertisers to perform demographic and marketing studies and campaigns	com.android.hios.launcher3	Duolingo Organization: duolingo.com Number of Times: 3 Last time: 04/28/2020 10:58:01 Messenger Organization: facebook.com Number of Times: 2 Last time: 04/28/2020 14:41:37
Board info	Low Risk	This value identifies your hardware and the phone model you use. Applications typically use this information to adapt content to your display or for improving advertising efficiency. However, this information can also reveal things about your personality, taste, wealthiness and your demographics	unknown	Duolingo Organization: duolingo.com 0 Number of Times: Last time: 04/28/2020 10:56:10 Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 14:41:37

Device Model	Low Risk	This value identifies your device model and manufacturer. Applications typically use this information to adapt content to your display or for improving advertising efficiency. However, this information can also reveal things about your personality, taste, wealthiness and your demographics.	TECNO-J8	Boomplay Organization: boomplaymusic.com Number of Times: 8 Last time: 04/28/2020 14:38:29 Boomplay Organization: facebook.com Number of Times: Last time: 29 04/28/2020 13:26:09 Boomplay Organization: appsflyer.com Number of Times: 16 Last time: 04/28/2020 12:51:41 Boomplay Organization: fbcdn.net Number of Times: 5 Last time: 04/28/2020 12:41:36 Boomplay Organization: shalltry.com Number of Times: 2 Last time: 04/28/202011:19:33 Boomplay Organization: mobadvent.com Number of Times: 2 Last time: 04/28/2020 09:19:34 Duolingo Organization: facebook.com Number of Times: 5 Last time: 04/28/202011:16:14 Duolingo Organization: duolingo.com Number of Times: 6 Last time: 04/28/2020 11:06:01 Duolingo Organization: fbcdn.net Number of Times: 1 Last time: 04/28/2020 10:56:11 Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 14:41:37 FMWhatsApp 2 Organization: google.com Number of Times: 19 Last time: 04/28/2020 16:00:40 FMWhatsApp 2 Organization: whatsapp.net Number of Times: 7 Last time: 04/28/2020 14:49:08 Weather •F Organization: accuweather.com Number of Times: 3 Last time: 04/28/2020 14:21:00 Equa bank Organization: equamobile.cz Number of Times: 7 Last time: 04/28/2020 11:42:15 Equa bank Organization: equa.cz Number of Times: 1 Last time: 04/28/2020 10:57:12 KB Klic Organization: morebanka.cz Number of Times: 1 Last time: 04/28/2020 12:50:19 nextbike Organization: nextbike.net Number of Times: 2 Last time: 04/28/2020 09:18:48 nextbike Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 09:18:47 nextbike Organization: crashlytics.com 0 Number of Times: 1 Last time: 04/28/2020 09:18:46 Mobilni banka Organization: trusteer.com Number of Times: 3 Last time: 04/28/2020 12:49:01 hiOS Android system (root) OS Organization: reallytek.com Number of Times: 1 Last time: 04/28/202010:46:09

Hardware info	Low Risk	This value identifies your hardware and the phone model you use. Applications typically use this information to adapt content to your display or for improving advertising efficiency. However, this information can also reveal things about your personality, taste, wealthiness and your demographics	mt6735	AI Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/2020 14:41:37 hiOS Android system (root) OS Organization: reallytek.com Number of Times: 1 Last time: 04/28/2020 10:46:09
Build Host	Low Risk	This value identifies the host used for building your Android version	rlk-buildsrv37	Messenger Organization: facebook.com Number of Times: 1 Last time: 04/28/202014:41:37

Build Fingerprint	Low Risk	It is a value that identifies uniquely your Android OS and the version you run. Applications typically use this information to adapt content to your display or for improving advertising efficiency. However, this information, specially when combined with other leaks, can reveal things about your personality, taste, wealthiness and your demographics	LMY47D	Boomplay Organization: boomplaymusic.com Number of Times: 8 Last time: 04/28/2020 14:38:29 Boomplay Organization: facebook.com Number of Times: 26 Last time: 04/28/2020 13:26:09 Boomplay Organization: appsflyer.coM Number of Times: 16 Last time: 04/28/2020 12:51:41 Boomplay Organization: fbcdn.net Number of Times: 5 Last time: 04/28/2020 12:41:36 Boomplay Organization: shalltry.com Number of Times: 2 Last time: 04/28/202011:19:33 Boomplay Organization: 34.255.134.237 Number of Times: 1 Last time: 04/28/2020 09:19:51 Duolingo Organization: facebook.com Number of Times: 5 Last time: 04/28/2020 11:16:14 Duolingo Organization: duolingo.com 0 Number of Times: 6 Last time: 04/28/2020 11:06:01 Duolingo Organization: fbcdn.net Number of Times Last time: 1 04/28/2020 10:56:11 Af Messenger Organization: facebook.com Number of Times: Last time: 04/28/2020 14:41:37 FMWhatsApp 2 Organization: google.com Number of Times: 19 Last time: 04/28/202016:00:40 Weather Organization: accuweather.com Number of Times: 3 Last time: 04/28/202014:21:00 PHX Browser Organization: instagram.com Number of Times: 2 Last time: 04/28/2020 09:17:34 PHX Browser Organization: google.com Number of Times: 1 Last time: 04/28/2020 09:17:33 TikTok Organization: musically Number of Times: 13 Last time: 04/28/2020 16:00:41 Equa bank Organization: equamobile.cz Number of Times: 6 Last time: 04/28/202.011:42:15 KB Klic Organization: mojebanka.cz Number of Times: 1 Last time: 04/28/2020 12:50:19 Mobilni banka Organization: trusteer.com Number of Times: 3 Last time: 04/28/2020 12:49:01

3.4 Summary

From the the experimental results that can be seen in Table 3.1 we can notice various leaked values caused by installed applications. However, in many cases, the same value was detected in multiple apps. This limits the ability of Lumen to use leaked values as unique features for fingerprinting.

Chapter 4

JA3 Fingerprinting

4.1 Motivation

In the previous chapter we observed various methods for mobile device fingerprinting. The presented approach based on weighted score, however, depends on user activity. When a user actively uses selected application, its score is higher. Our approach also expected that when a mobile device is running without explicit user activity, there will be traces of network communication due to the app synchronization, checking update, etc. However, our experiments proved that when a real device is not used actively, the mobile operating system starts reducing network communication and mobile apps change their operating state to sleeping.

Thus, we decided to shift our focus from mobile device fingerprinting to mobile apps fingerprinting. Thus, our goal is to identify communication of a mobile app in the network traffic. Due to the rise of encrypted communication, we focus on TLS and DNS traffic only.

Table 4.1 shows the structure of network protocols involved in mobile communication. Datasets 1 to 4 created by the authors of this study in 2018 (see also Table 1.2) show that the ratio of encrypted communication to unencrypted varied from 51,7 to 91,7%. You can also notice a presence of non-encrypted HTTP traffic. Especially HTTP headers, e.g., *User-Agent*, *Accept-Language*, *Accept-Charset*, were an important source of data for various fingerprinting techniques [14, 21].

A year after our first experiments, we noticed that ratio of encrypted communication increased to 99% which prevented utilization of traditional fingerprinting methods. As seen in Figure 4.1, besides encrypted TLS traffic transmitted over port 443, only Domain Name System (DNS) communication remained unencrypted. It is a question for how long because of various attempts to encrypt DNS traffic using DNS over TLS (DoT) or DNS over HTTP (DoH) [20, 19].

As reaction to the encryption of communication, researchers focused their

	2018				2019
	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5
Time	70 min	12 min	37 min	21 min	96 min
Size	452 MB	35 MB	423 MB	18 MB	610 MB
Packets	542.725	44.699	424.922	25.525	597.097
Encrypted Traffic					
SSL/TLS	44,26%	86,03%	30,52%	80,10%	98,06%
UDP over 443					1,12%
IMAPS	0,65%	1,67%			
FB Zero	0,65%	0,12%	0,01%	0,13%	
QUIC	6,15%	3,90%	4,74%	8,07%	
OpenVPN			54,94%		
Total	*51,71%*	*91,72%*	*90,21%*	*88,30%*	*99,18%*
Unencrypted Traffic					
HTTP	41,47%	1,65%	7,55%	2,12%	0,32%
DNS, mDNS	0,93%	1,78%	0,64%	2,41%	0,31%
DHCP	0,05%	0,13%	0,04%	0,26%	0,01%
ICMP/IGMP	0,36%	0,53%	0,14%	0,60%	0,07%
ARP	2,11%	1,01%	1,62%	3,17%	0,06%
Total	*44,92%*	*5,10%*	*9,99%*	*8,56%*	*0,77%*

Table 4.1: Encrypted and Unencrypted Mobile Communication in 2018 and 2019

activity on analysing behavior of encrypted communication in order to obtain meta data about encrypted protocols and services. One research direction is focused on statistical analysis of encrypted transmissions [37, 12], the other effort oriented on extracting features from TLS handshake and computation of so called *TLS fingerprint* [22, 7, 3, 32]. One of the popular TLS fingerprinting implementations called *JA3 fingerprinting* was proposed by John B. Althouse, Jeff Atkinson and Josh Atkins in 2015[1]. Their method was also incorporated into network monitoring and intrusion detection systems like Flowmon, Bro, or Suricata, where it is employed for malware detection [4], identification of network applications [23], or blacklisting[2].

In our research, we focus on mobile devices, especially on detection of mobile devices based on a set of installed applications. Mobile apps regularly communicate over the Internet without explicit user interaction in order to update software, synchronize local data, or retrieve remote status [32]. This makes possible to identify a mobile device based on a characteristic set of applications and their versions that are installed on the device [24]. Mobile application can be identified from captured TLS traffic using JA3 hashes (retrieved from client communication) or JA3S hashes (retrieved from server communication).

However, there are important questions related to the digital forensics: *Are these fingerprints reliable enough to identify a specific application? How*

[1]See https://github.com/salesforce/ja3 [April 2020]
[2]See SSLBL project at https://sslbl.abuse.ch/ [April 2020]

stable are they? How can we create a unique fingerprint database of mobile apps? The goal of this paper is to study reliability of JA3 fingerprints on selected mobile apps, present a way how unique fingerprints can be generated and discuss the application of JA3 fingerprinting to digital forensics.

4.1.1 Preliminaries

Transport Layer Security (TLS) [13, 34] is a transmission protocol that works on top of TCP where it provides privacy and data integrity for communicating applications. The protocol is composed of two parts: TLS Handshake Protocol and TLS Record Protocol. TLS Handshake Protocol negotiates security parameters, e.g., protocol version, methods for key exchange, encryption, authentication, and data integrity, secure channel options, etc. TLS handshake communication is not encrypted. The TLS Record Protocol encapsulates high-level protocol data and transmits encrypted packets. Example of TLS handshake is in Figure 4.1.

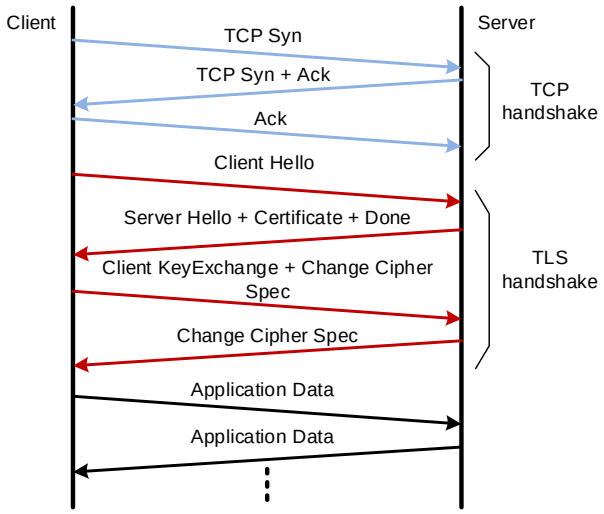

Figure 4.1: Establishing TLS connection.

After opening TCP connection by three-way handshake, the TLS negotiates security parameters using TLS Client and Server Hello packets. The client application offers a set of supported encryption and authentication methods using the TLS Client Hello. The TLS server processes these options and sends back options that are supported on the server side. The server can also include a server certificate to authenticate itself. After all security parameters are agreed, application data encapsulated by TLS Record Protocol are exchanged.

Most of TLS fingerprinting methods use the first packet sent by the client: *Client Hello*. The Client Hello contains an imprint of TLS configuration of

the client application that depends on the used TLS library and operating system. In this paper we study *JA3 fingerprint* that is computed as MD5 hash from five TLS handshake fields: TLS Handshake version, Cipher suites, Extensions, Supported Groups (former Elliptic Curve), and Elliptic Curve point format, see Figure 4.2. Some TLS fingerprinting implementations use different TLS fields, e.g., Kotzias et al. [23] omit TLS version.

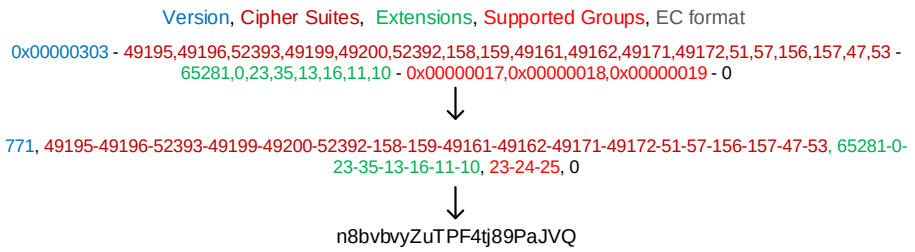

Figure 4.2: Computing JA3 hash

Computation of JA3 fingerprint includes (i) extraction of selected fields from TLS Hello packet, (ii) concatenation of extracted data in decimal format into one string, and (iii) application of MD5 hash algorithm on the string. The result is a 32-bit string in hexadecimal format. There are open implementations of JA3 fingerprinting[3]. Unlike `nmap` or web browser fingerprinting methods which actively request the source device or application, JA3 fingerprinting uses passive approach. Application of TLS fingerprints for identification of network applications requires TLS fingerprint values to be unique, accurate and stable. There are several aspects that limits reliability of TLS fingerprints:

TLS library.

TLS fingerprint of an application depends on the TLS library that was used during implementation. There are plenty of TLS libraries available to developers, e.g., GnuTLS, Oracle JSSE, BSD LibreSSL, OpenSSL, or Mozilla NSS. When two applications are implemented using the same TLS library, it may happen that their TLS fingerprints are the same. TLS fingerprints can change with a new version of the application, version of the TLS library, or the operating system. Table 4.2 shows JA3 hashes for common web browsers: Mozilla Firefox v.73, Chrome v.80, and Opera v.66 under four operating systems: Linux Ubuntu, Windows 10, Kali Linux and Mac OS.

We can see that Firefox has four unique JA3 fingerprints. Two of them are present in all tested operating systems. In case of Chrome and Opera, one JA3 fingerprint values corresponds to both browsers under all operating systems. These browser were possibly compiled with the same TLS library.

[3]See https://github.com/salesforce/ja3 or https://ja3er.com/ [April 2020].

	Firefox				Chrome				Opera			
JA3 hash	Ubuntu	Win	Kali	MacOS	Ubuntu	Win	Kali	MacOS	Ubuntu	Win	Kali	MacOS
0e6f3c8f2b18f3011f1d6cbbdcfcbd65						x				x		
1344ed2e9d7d8e3e84e6ab655047ba32	x	x	x	x								
1f3c530fc35e41300422550c3c980e85							x	x	x	x	x	x
4863015f73b8332cf91cfa3a14a4893d		x										
5a291b49748c50adf1da70f8142d4cc4					x				x			
756094f51da8214018fbfba93211d59f	x	x	x	x								
a839cfeed30d55439b09de5f1b47fa3a					x	x	x	x	x	x	x	x
d889531a0389787425d5638caf6d84b3					x	x	x	x	x	x	x	
d90d517f72e9b8af9a8c1e2fe1fb2da8	x			x								

Table 4.2: JA3 hashes of common Web browsers

This experiment proves that TLS fingerprints change with the version and operating system. More observations related to JA3 fingerprinting of web browsers are written in Section 4.3. Similar experiment over larger dataset is also mentioned in [32].

Random values in TLS.

In 2016, Google started to Generate Random Extensions And Sustain Extensibility (GREASE) values to TLS. This technique was adopted by IETF in January 2020 as RFC 8701 [5]. GREASE values are randomly generated numbers of cipher suites, extensions and supported groups present in TLS Hello packets. They prevent extensibility failures in TLS ecosystem. During TLS handshake, the responding side must ignore unknown values. Peers that do not ignore unknown values fail to inter-operate which means a bug in implementation. Therefore, RFC 8701 adds GREASE values as a part of the list of cipher suites, extensions and supported groups to detect invalid implementations.

When experimenting with Opera browser under Win 10 we noticed that the browser generates 155 unique JA3 fingerprints out of 207 TLS handshakes. By excluding GREASE values, the number of unique JA3 fingerprints decreased to four. The high number of JA3 fingerprints was caused by random GREASE values in TLS handshakes. Table 4.3 shows six JA3 fingerprints of Opera browser under Ubuntu with all extracted TLS values (the upper six lines). The last six lines presents TLS values without GREASE values. The brown values in the upper table represent GREASE values as

List of Cipher Suites	List of Extensions	Supported Group	JA3 hash
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-65281-16-18	29-23-24-25	839868ad711dc55bde0d37a87f14740d
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-65281-16-18	29-23-24-25	839868ad711dc55bde0d37a87f14740d
56026-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	60138-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-19018-21	35466-29-23-24	ee972d7d47ec01a9cb9b04efb7346e32
60138-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	39578-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-56026-21	23130-29-23-24	cb4415a180704432d2e3f70f8dca5783
31354-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	47802-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-51914	43690-29-23-24	74a57a5f55ce2c9fa637b1f4567308b4
14906-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	31354-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-43690	56026-29-23-24	a10f93ffdc89d383db0f4437a0530569
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-16-18	29-23-24-25	5a291b49748c50adf1da70f8142d4cc4
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-16-18	29-23-24-25	5a291b49748c50adf1da70f8142d4cc4
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a

Table 4.3: JA3 hashes with and without GREASE values

defined in RFC 8701. When ignoring these values, last four lines in the upper table would have the same JA3 hash.

In addition to GREASE values, it is also good to omit extension value 65281 from TLS fingerprinting. This value represents renegotiation option in TLS handshake [35], see red numbers in the list of extensions. The last option that can be ignored is the TLS Client Hello Padding Extension defined by RFC 7685 [31]. The padding extension (value 21, depicted by green value in the table) is added by a client to make sure that the packet is of a desired size.

In order to keep JA3 fingerprints stable, it is necessary to eliminate above mentioned values. Most of JA3 implementations usually exclude GREASE value from TLS fingerprinting.

Ads, tracking services and web analytics.

By observing TLS handshakes of mobile apps, we noticed that an app does not sent data only to its application server, but it also opens many connections to various sides without explicit user activity. These connections include ad servers, tracking services, or web analytic servers. Destinations of these services are dynamic which means that each time the application is launched, it connects to different sites using different TLS fingerprints. This causes problem for finding ground-truth communication for learning TLS fingerprints.

Dynamic behavior of ad connections is caused by mobile advertising auctions that redirect the application from the ad server to the content provider based on the results of an auction [26]. Since different applications include same ad, tracking or analytic plugins, thus captured communication of these applications may contain the same TLS fingerprints. We call this extra traffic *a noise*, other researches call it *ambiguous traffic* [39]. Table 4.4 shows TLS fingerprints obtained from communication of Gmail app.

SrcIP	DstIP	Server Name Indication	JA3 Fingerprint
10.0.2.15	172.217.23.193	ci5.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737
10.0.2.15	172.217.23.225	ci3.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737
10.0.2.15	172.217.23.229	mail.google.com	81d2604dcc31ff39cdddb6079692b0b0
10.0.2.15	216.58.201.106	www.googleapis.com	193c522402283ed9e84b8bb38137829f
10.0.2.15	216.58.201.106	www.googleapis.com	3d9a16cdc1b2a98f6046af1c833054b8
10.0.2.15	216.58.201.74	android.googleapis.com	ca75d9d90e40897206fa2a08d9100df0
10.0.2.15	216.58.201.97	ci4.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737

Table 4.4: JA3 hashes of Gmail App

There are five different JA3 fingerprints computed of TLS communication of Gmail app. Using SNI extension in the TLS Client Hello, we are able to exclude communication to Google API (www.googleapis.com) and user

content (googleusercontent.com) which is not directly related to the app. The remaining fingerprint with SNI mail.google.com characterizes Gmail app.

Thus, it is necessary to exclude ad, tracking and analytic communication from TLS fingerprinting. One solution how to recognize this noise traffic is using black lists of ad and tracking servers[4]. By comparing server names in SNI field with names in ad server lists, we can partially clean up the captured TLS communication from the noise. Table 4.5 shows a percentage of ad traffic in communication of selected mobile apps based on ad and tracking server lists. Especially free apps include ad plugins in order to receive funding from ads providers.

App	All TLS handshakes	AD servers	Percentage
Accuweather	520	232	45%
BoomPlay Music	361	53	15%
Gmail	60	6	10%
Tor Browser	9	0	0%
Reddit	1892	840	44%
Muj vlak	451	195	43%
Viber	12	6	50%
Discord	24	0	0%
TitTok	203	20	10%
WhatsApp	243	30	12%
NextBike	444	39	9%
Facebook	178	13	7%
EquaBank	387	14	4%

Table 4.5: The number of TLS connections to Ad servers for selected Apps

Time stability.

Very important issue is stability of TLS fingerprints over time. We demonstrated, that a TLS fingerprint depends on TLS library and operating system. Update of TLS library, adding new cipher suites or excluding weaker ciphers can change the fingerprint. The longitudinal study of TLS fingerprints of Kotzias et al. [23] shows that the maximum duration of TLS fingerprints is 3 years and 4 months, the median is 1 day and the mean 158,8 days. If application is not updated, it keeps its original TLS fingerprint.

The time instability means that for successful identification of mobile applications based on TLS fingerprints, we need to update the fingerprint database by a new app version. However, our experiments show that variability of TLS fingerprints is not so big. On the other hand, a unique fingerprint of a particular version of the mobile app can identify communication of the app in network traffic. When a new version is released, a new fingerprint

[4] E.g., https://hosts-file.net/ad_servers.txt, https://pgl.yoyo.org/adservers/, or https://gitlab.com/ookangzheng/dbl-oisd-nl [April 2020]

should be added to the fingerprint database. This is especially important for digital forensics.

4.1.2 Datasets

This section introduces our datasets used in experiments. First we observed available datasets with mobile traffic. ReCon datasets[5] [33] was created to observe leakage of personal identifiers through mobile communication. Recon dataset contains HTTP(s) logs of 512 mobile applications. Logs do not contain TLS headers that are important for TLS fingerprinting. However, for a given mobile app, we can extract a list of sites the application usually connects. For example, for `accuweather` app, we get fonts.googleapis.com or ssl.google-analytics.com (noise servers), and vortex.accuweather.com or accuwxturbo.accu-weather.com directly related to the app. These domain names can be traced in SNI extension during TLS analysis which is important for creating unambiguous fingerprints.

Interesting mobile apps dataset is Panoptispy[6] [30] that was created to study media permissions and leaks from Android apps. The dataset consists of network traffic that have instances of media in an HTTP requests body. Besides dumps of HTTP requests, it contains a list of apps with package name, version, app name and app md5. However, this is not sufficient for TLS fingerprinting.

Andrubis and Cross Platform datasets mentioned in [41] were not located by the authors of this paper. Nevertheless, we explored Mirage dataset[7] [2] which contains mobile app traffic for ground-truth evaluation. The captured traffic is stored in JSON format and contains bi-flows with source and destination ports, number of bytes, inter-arrival times, TCP window size, L4 raw data, and various statistics. Since TLS header is hidden in byte-wise raw L4 payload, it is not easy to extract TLS values that are interesting for our research. However, we plan to use this data for ground-truth evaluation of our method presented in the paper.

For our experiments we created own dataset with communication of selected mobile applications, see Table 4.6.

Dataset	Apps	Device	OS	PCAP (MB)	Packets	TLS Handshakes	List of Apps
Web browsers (WB)	3	PC	Linux, MacOS, Win 10	193	224.717	2.621	Chrome, Firefox, Opera
Mobile Apps I (MA1)	5	Sony, Huawei	Android 9, 7.0	2	5.700	79	Discord, Messenger, Slack, Telegram, WhatsApp
Mobile Apps II (MA2)	4	Android Studio	Android 6	35	50.820	595	Accuweather, Gmail, Tor, Viber
Mobile Apps III (MA3)	4	Android Studio	Android 7.1, 8,1, 9	827	642.919	3.180.345	Cestovne Poriadky, Mujvlak, Reddit, Seznam
Mobile Apps IV (MA4)	14	Tecno	Android 5.1	446	578.812	5.308	Boomplay Music, Chrome, EquaBank, Facebook, ...
Total	*30*			*1504*	*1.502.968*	*3.188.948*	

Table 4.6: Mobile apps communication dataset

We agree that our dataset is not representative, however, it is sufficient

[5] See `https://recon.meddle.mobi/appversions/` [April 2020].
[6] See `https://recon.meddle/mobi/panoptispy` [April 2020].
[7] See https://ieee-dataport.org/open-access/mirage-mobile-app-traffic [April 2020].

for studying typical features of TLS mobile apps fingerprints. The dataset is available in PCAP format and as CSV traces of network and TLS parameters[8]. It contains five parts:

Web Browsers (WB).

The first dataset consists of TLS communication of web browsers Chrome v80, Firefox 68.2, Firefox 73.0, Firefox 70.0, Opera 66.0 and Opera 67.0. These browsers were running under four different operating systems: Kali Linux, Mac OS, Windows 10 and Linux Ubuntu. During experiments we requested 10 different URLs. We created TLS fingerprints for all browsers based on TLS handshakes related to requested URLs. The dataset contains 2.621 TLS handshakes used for testing stability of TLS fingerprints with respect to application versions and underlying operating system. More detailed description of this dataset is in Table 4.7.

Mobile Apps I (MA1).

The second dataset includes five mobile applications: Discord v16.3, Messenger v253.0, Slack v20.03, Telegram v6.0 and WhatsApp v2.20. The applications were installed on two mobile devices: Sony Xperia X71 Compact with Android 9 (API level 28) and Huawei P9 with Android 7 and EMUI 5.0.1 (API level 24). Devices were connected to a PC and TLS data captured using `tshark`. To make sure that packets include initial handshake, the tested application were restarted using ADB commands. The dataset contains 79 TLS handshakes.

Mobile Apps II (MA2).

The third dataset includes communication of four mobile applications: Accuweather, Gmail, Tor and Viber. For tests, we used Android Emulator which is a part of Android Studio. In the Android Emulator, we created two virtual devices: Google Pixel C with Android 8.1 and Google Nexus 10 with Android 6.0. Using ADB interface we installed the above mentioned mobile applications on the virtual device and simulated user behavior using command-line tool *Monkey*. The Monkey emulates user behavior on a given app, so the captured communication is initiated by that app. An example of emulating Viber app on the virtual device is below. The dataset contains 595 TLS handshakes.

```
$adb shell monkey -p viber -v 500
```

Mobile Apps III (MA3).

This dataset includes communication of following mobile applications: Cestovne Poriadky (Time Table), Muj vlak (My train), Reddit and Seznam. TLS fingerprints of these apps were obtained using Virtual Box where a virtual Android device with these apps was installed. The apps were tested on Android version 7.1, 8.1 and

[8] See https://github.com/matousp/tls-fingerprinting [April 2020]

9. On each Android system, the app was repeatedly launched and communication captures. We also observed if the application cache has influence on communication, so each App was running twenty times without cache and twenty times with cache on each system. Together, we obtained 3.180.245 TLS handshakes.

Mobile Apps IV (MA4).

The last dataset was focused on variety of mobile apps installed on a real device Tecno J8 with Android 6.1. Dataset includes following apps: BoomPlay Music, Chrome Browser, Equa Bank app, Facebook app, Gmail app, Google calender, KB klic, Messenger, Mobilni Banka app, NextBike, Telegram, TikTok, WhatsApp and Youtube app. Each app was running five times on the restarted device so that captured communication corresponds to a typical usage. We extracted 5.308 TLS handshakes from the captured traffic.

The above mentioned datasets were used for experiments with JA3 and JA3S fingerprints, see Section 4.4 and 4.

4.2 Related Work

TLS fingerprinting is not a new technique and its development is connected with security research of Ivan Ristić who developed in 2008 an Apache module that passively fingerprinted connected clients based on cipher suites. Using this technique he created a signature base that identified many browsers and operating systems [1]. This technique was later applied on identification of HTTP clients [22] and implemented in IDS systems Bro and Surikata for passive detection.

Blake Anderson et al. in [4] studied millions of TLS encrypted flows and introduced a set of observable data features from TLS client and server hello messages like TLS version, TLS ciphers suites and TLS extensions that they used for malware detection. They also observed server's certificate and the client's public key length, sequence of record lengths, times and types of TLS sessions. They identified cipher suites and extensions that were present in malware traffic and missing in normal traffic. The authors defined TLS client configurations for the 18 malicious families. Similarly, they identified TLS server configurations most visited by 18 malicious families. They applied TLS features together with other features (flow data, inter-arrival times, byte distribution) to malware classification where achieved accuracy from 96.7% to 98.2%. As demonstrated by their study, omitting TLS features lead to significantly worse performance.

Platon Kotzias et al. in [23] passively monitored TLS and SSL connections from 2012 to 2015 and observed changes in TLS cipher suites and extensions offered by clients and accepted by servers. They also used client TLS fingerprinting with features similar to JA3 fingerprinting. From handshakes they omitted GREASE values. Using captured data, they observed

7.3% fingerprint collisions in TLS fingerprints. They also mapped fingerprints to a program or library and the version. One of their results was observation of TLS fingerprints stability. They noticed that maximum duration when a fingerprint was seen in their databases was 1.235 days (3 years, 4 months). However, the median of duration a fingerprint was seen was 1 day, the mean 158.8 days. They noticed some fingerprints that were seen very briefly and did not reappear later. There found out that 1,203 fingerprints of the 69,874 fingerprints were responsible for 21.75% connections. Further, they analysed vulnerability of TLS against various attacks which is different direction unlike our research. Their results related to stability and collisions of TLS fingerprints was also observed during our experiments.

Another interesting approach published by Anderson and McGrew [3] combines end host data with network data in order to understand application behavior. This approach, however, requires access to both end hosts and connected network. Their fingerprint database represent the real traffic generated by 24,000 hosts and having 471 million benign and millions of malware connections [9]. Using end point data, the authors were able to associate destination information with end point data like timestamp, endpoint ID, operating system and process name. During fingerprint analysis they observed that while GREASE values are generated randomly, their position is deterministic. Thus, instead of removing GREASE values, they set them to 0a0a. They also studied similarity of TLS fingerprints which was defined using Levenshtein distance. Two TLS fingerprints were similar if their distance was less than or equal to 10% of the number of cipher suites, extension types, and extension values. The authors stated that the Levenshtein distance was an intuitive method for identifying close fingerprints. Especially TLS libraries often make minor adjustments to default cipher suites or extensions between minor version releases and more drastic changes between major version releases. They also noticed that some TLS libraries change their default parameters to better suit the platform on which they are running. Another interesting point is prevalence of application categories in the dataset where 37.1% connections belong to browsers, 19.3% to email applications, 17,2% to communication tools, 9% to the system, etc. Longevity of fingerprints like system libraries, tools osquery and DropBox, and browsers was 6 months or greater.

The above mentioned approaches worked mostly with common network traffic and network application. Another work closer to ours deals with TLS usage in Android Apps [32]. The authors analyzed behavior of TLS in mobile platforms. They developed Android app Lumen that was installed on a mobile device where intercepted TLS connection and gathered statistics about the traffic. Using Lumen app, the authors observed how 7.258 apps

[9] Tools for capturing data are available at https://github.com/cisco/mercury [April 2020]

use TLS. They analyzed handshakes with respect to TLS API and library that the app used. Their work was focused on apps security and TLS vulnerabilities. They showed that TLS libraries and OS API modified supported cipher suites across versions which caused changes in TLS fingerprints. They also showed that each TLS library and OS version had a unique cipher suite lists. They built a database of fingerprints paired with corresponding OSes and libraries where they observed influence of major and minor revisions of OS or TLS libraries on the fingerprint. Unfortunately, Lumen app was not able to captured TLS handshakes which would be useful for our research. Thus, we used additional approach how to obtain reliable TLS fingerprints of mobile apps.

The mobile application fingerprinting using characteristic traffic was considered by Stöber et. al [36]. They created a classifier that identified communicating applications based on the analysis of side-channel information such as timing and data volume. Mobile application fingerprinting has been tackled by machine learning techniques using timing and size of packets [39], which improved previous work presented in [38] that observed the traffic that was common among more than one apps. The method is applicable to encrypted traffic, which is used by most smartphone applications and relies only on information available from the side channel. The fingerprinting system was trained and tested on 110 most popular Android applications. The training was done automatically using the implemented application AppScanner. The significant feature of the method was that it analyzed the traffic represented as bursts. A burst was defined as a group of packets within TCP flow representing an interaction for a typical smartphone application that communicated using HTTPS protocol. Statistical features were then extracted for bursts and used for training random forests classifier. The method did not rely on any other source of information, e.g., DNS, TLS, IP addresses, etc. The achieved accuracy as presented by the authors was between 73 to 96 percent for the selected set of applications. Recently, the work was extended by [41] that used a semi-supervised method for both app recognition and detection of previously unseen apps.

Another line of research that considered mobile device identification is represented by Govindaraj, Verma and Gupta [16]. They proposed a methodology for extracting and analyzing ads on mobile devices to retrieve user-specific information, reconstruct a user profile, and predict user identity. As the published results showed it was possible to identify a user in various settings even if he/she used multiple devices or different networks. Their work stemmed from the study by Castelluccia, Kafar and Tran [9] who demonstrated the possibility to infer the interests of users from targeted ads.

Our work uses previously published results and focuses on passive identification of mobile apps using JA3 fingerprints. It also observes traffic that is common to multiple apps and that should be excluded from fingerprinting. Based on the app, we employ JA3, JA3S, and SNI features to accurately

identify the unknown traffic that was sent by a mobile apps. We are able to detect only apps that were previously learnt and stored in the fingerprinting database. Unlike some of the above mentioned approaches, our technique for mobile app detection is simple, fast and reliable. Its accuracy depends on the quality of learnt fingerprints.

4.3 JA3 Fingerprinting for Web Browsers

This part includes our preliminary experiments with JA3 fingerprinting of web browser that were mentioned in Section 4.1.1. In this study we analyzed JA3 fingerprint of common web browser and observed stability, uniqueness and reliability of these values for web browser fingerprinting.

Several experiments with annotated web traffic from various web browsers including Google Chrome, Opera and Firefox were conducted to generate unique fingerprints and create a database for comparison with unknown datasets. The study shows possible utilisation of JA3 fingerprinting in browser identification.

Encrypted communications make it difficult to conduct device or user fingerprinting that require visibility of protocol headers such as HTTP, IMAP and others. In order to avoid this limitation, new methods such as TLS fingerprinting are now becoming popular. Our work has made it possible to positively identify specific web browsers based on captured network dumps generated from an unknown environment.

4.3.1 Background

As stated before, during an SSL handshake, most client user agents initiate an TLS handshake request in a unique way. This includes web browsers in different operating systems such as Linux, Mac OS and windows. The fingerprint relies on data from ClientHello messages in the SSL handshake. We focused on JA3 technique which is a standard for creating SSL client fingerprints. As mentioned above, JA3 gathers the decimal values of the bytes for the following fields in the Client Hello packet; SSL Version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats. It then concatenates those values together in order, using a comma to delimit each field and a dash to delimit each value in each field. These strings are then MD5 hashed to produce an easily consumable and shareable 32 character fingerprint. This is called JA3 SSL Client Fingerprint.

This study extends JA3 functionalities by making it possible to easily identify the type of web browser based on network communication. Three additional SSL handshake fields are introduced to make the data more informative. This includes time since reference or first frame, source IP address, and destination IP address.

4.3.2 Testing Environment

The data used in this experiment consisted of 13 PCAP files from different browsers and various operating systems. While capturing the traffic, specific domains and URLs were accessed in all browsers to ensure that communication can be reliably filtered by analysing DNS records. These include:

- superuser.com/questions/247127/what-is-and-in-linux/247131
- linuxsig.org/files/bash_scripting.html
- strathmore.edu
- vutbr.cz/en
- facebook.com
- adobe.com
- amazon.com
- bitbucket.org/dashboard/overview
- forums.kali.org
- offensive-security.com

The packets relating to these URLs were identified by examining DNS records. Corresponding IP addresses were gathered and used to filter tls.hello packets. Table 4.7 describes the captured data in details.

For experiments with JA3 hashes we implemented a tool that consists of a shell script that processes PCAP files and computes JA3 fingerprints of known web browsers. Tshark commands are used to extract the relevant fields from the Client Hello packets. Unix string manipulation commands parse the fields to prepare for fingerprint generation and hashing. Computed fingerprints are saved into CSV files so that unknown PCAP files can be compared.

The script implementation includes various steps intended to extend the JA3 functionality by being able to identify the web browsers used in a particular communication. Various command based tools are used to read and analyse PCAP files in order to reveal the browser identity. Figure 4.3 illustrates the architecture of our web browser fingerprinting tool.

The first step uses `tshark` to extract following packet fields from a PCAP file: `frame.time, ip.src, tcp.srcport` and `tcp.dstport`. In order to fingerprint only packets to known destinations (creating a database of web browser profiles), packets representing noise from other applications should not be examined. A full packet capture includes traffic to many destinations, including Operating System, background applications and other running apps communicating with remote services. Since this experiment is

			All IP Addresses			Filtered IP Addresses				
Browser	Operating System	Browse Version	Packets #	tls.hello Packets	Unique Fingerprints	Packets #	tls.hello Packets	Unique FIngerprints	File Size (Bytes)	Timestamp
Google Chrome	Kali Linux	80.0.3987.106	12013	179	174	4730	33	31	18379776	Feb 17 21:19
Google Chrome	Mac OS	80.0.3987.106	26914	155	149	4431	16	16	20612464	Feb 18 13:15
Google Chrome	Ubuntu	80.0.3987.132	5835	76	73	260	6	6	3368680	Mar 11 14:57
Google Chrome	Windows	80.0.3987.106	19686	180	17	10739	41	11	19693784	Feb 17 15:50
Mozilla Firefox	Kali Linux	68.2.0esr	9864	164	3	3307	36	3	12880012	Feb 18 14:46
Mozilla Firefox	Mac OS	73	20217	194	5	4634	29	3	13399464	Feb 18 13:15
Mozilla Firefox	Ubuntu	73.0.1	20474	183	4	6584	20	3	14374332	Mar 11 15:14
Mozilla Firefox	Windows	70.0.2	18014	220	4	8516	42	4	16870532	Feb 17 15:50
Opera	Kali Linux	66.0.3515.72	12558	191	186	2968	30	28	17763452	Feb 17 20:43
Opera	Mac OS	66.0.3515.72	28300	198	192	8016	27	27	20853084	Feb 18 13:15
Opera	Ubuntu	67.0.3575.53	22053	204	199	6184	18	18	16335228	Mar 11 15:01
Opera	Windows	66.0.3515.95	8445	117	8	5315	22	4	7969388	Feb 17 15:50
Opera	Windows	67.0.3575.53	20344	207	200	5991	25	23	17217192	Mar 5 14:54

Table 4.7: Overview of web browser dataset

focused on web browser TLS fingerprinting only, traffic from other applications should be eliminated. However, browser traffic includes communication by browser plugins, advertisements, and other remote services not explicitly initiated by the user. This should also be eliminated so that it remains only communication to destinations initiated by the user. This helps to ensure that the fingerprints are clean and able to identify browsers across different operating systems or versions. This also has a secondary benefit of minimising the size of the dataset to be analysed, hence increasing the tool efficiency.

This is achieved by filtering traffic based on known DNS destinations. Extracted records are analysed and IP addresses matched with known domain names. TLS Client Hello packets to the identified destinations are extracted and fingerprints generated. The steps below were followed to achieve this:

1. Identify a set of URLs to use (see above), and run them in a browser. Capture and save traffic using Wireshark. Use Bulk URL Opener browser plugin to load multiple URLs at once.

2. Extract DNS A records and DNS response names from the PCAP files and save the results in CSV files. Combine these in a single CSV file.

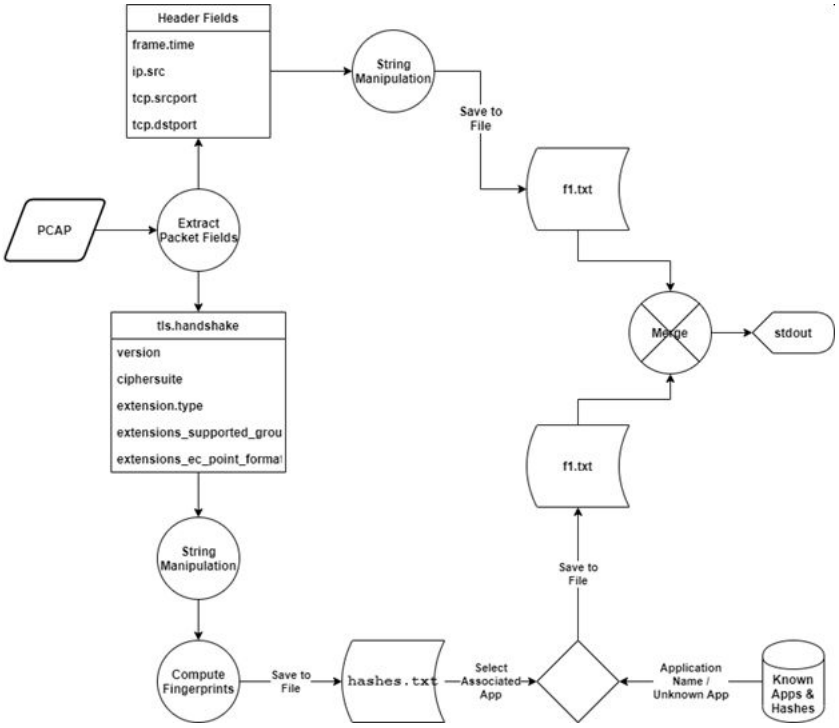

Figure 4.3: Architecture of a web browser fingerprinting tool

3. Search for the DNS response names for the domains identified, and match with corresponding IP addresses.

4. Calculate JA3 fingerprints, hashed from the above mentioned TLS handshake fields.

4.3.3 Results

Three web browser under different OSes were used, see Table 4.8.

Firefox

Browser identification was done for Firefox communication across four operating systems i.e. Kali Linux, Mac OS, Ubuntu and Windows. JA3 fingerprints were generated from known PCAP files for Firefox. To ensure that noise is not fingerprinted, DNS analysis was conducted whereby packets in the PCAP file were filtered based on the destination IP addresses of the domains entered during the traffic capture. DNS A records were matched with corresponding DNS response names in order to identify the destination IP addresses of selected domains. These were identified and all client hello packets with such destination IPs were fingerprinted. Unique fingerprints across the four operating systems were identified, see Figure 4.4.

Browser	Operating System	Browser version
Google Chrome	Kali Linux	80_0_3987_106
Google Chrome	Mac OS	80_0_3987_106
Google Chrome	Windows	80_0_3987_106
Google Chrome	Ubuntu	80_0_3987_132
Firefox	Kali Linux	68_2_0esr
Firefox	Mac OS	73_0
Firefox	Windows	70_0_2
Firefox	Ubuntu	73_0_1
Opera	Kali Linux	66_0_3515_72
Opera	Mac OS	66_0_3515_72
Opera	Windows	66_0_3515_95
Opera	Ubuntu	67_0_3575_53

Table 4.8: Tested Web Browsers

```
MariaDB [fingerprints]> SELECT * FROM ja3;
+--------+---------+------------+-----------+----------------------------------+
| ja3_id | app     | version    | os        | fingerprint                      |
+--------+---------+------------+-----------+----------------------------------+
|      1 | Firefox | Kali Linux | 68_2_0esr | 334da95730484a993c6063e36bc90a47 |
|      2 | Firefox | Kali Linux | 68_2_0esr | b20b44b18b853ef29ab773e921b03422 |
|      3 | Firefox | Mac OS     | 73_0      | 334da95730484a993c6063e36bc90a47 |
|      4 | Firefox | Mac OS     | 73_0      | b20b44b18b853ef29ab773e921b03422 |
|      5 | Firefox | Windows    | 70_0_2    | 334da95730484a993c6063e36bc90a47 |
|      6 | Firefox | Windows    | 70_0_2    | b20b44b18b853ef29ab773e921b03422 |
|      7 | Firefox | Ubuntu     | 73_0_1    | 334da95730484a993c6063e36bc90a47 |
|      8 | Firefox | Ubuntu     | 73_0_;    | b20b44b18b853ef29ab773e921b03422 |
+--------+---------+------------+-----------+----------------------------------+
8 rows in set (0.001 sec)
```

Figure 4.4: Firefox Fingerprint Entries

Google Chrome and Opera browsers did not have any matching fingerprints using this JA3 method.

Google Chrome and Opera

Browser identification was done for Google Chrome and Opera communication across three operating systems i.e. Kali Linux, Mac OS and Windows. The browser version was different across Kali Linux, Ubuntu, Mac OS and Windows operating systems. Unique fingerprints could not be identified across these browsers using the JA3 method.

This led to deep examination of Client Hello fields used in JA3 fingerprinting and identifying the differences in comparison to Firefox. Google Chrome PCAP files were analysed using Wireshark. It was noted that the `tls.handshake.ciphersuite` field was different. It contained Cipher Suite: Reserved (GREASE) (0x9a9a) which is not in Firefox. GREASE value was added to Chrome in version 55. GREASE values were also seen in `tls.handshake.extension.type` and `tls.handshake.extensions_sup-`

`ported_group`.

It has been proved that the hexadecimal numbers in GREASE are random, and change every time a page is refreshed. This explains the instability of fingerprints that we observed across different browser sessions and operating systems. The results were similar for Opera browser because it is built on the Chromium and Blink engine just like Chrome.

Because GREASE has been found to introduce random values, its occurrences in the Client Hello messages will be eliminated in the respective fields and fingerprints generated without it. Tshark was used to extract the Client Hello fields. These were processed, manipulated and saved as comma separated values. GREASE related fields were removed and finally MD5 hash values calculated for each record. A significant decrease in unique records indicate that GREASE values are quite random, and are different for communications with the same host. Elimination of these values gives a more consistent flow, which increases the chances of effective fingerprinting.

This process was done for Google Chrome PCAP files from Windows, Mac OS, Kali Linux and Ubuntu operating Systems. Initial tests show that one fingerprint (9ff0023372e249c161e03a71055216ca) is unique for Google Chrome across all the operating systems under review, Table 4.9.

chrome-klinux-80_0_3987_106	chrome-mac-80_0_3987_106	chrome-windows-80_0_3987_106	chrome-ubuntu-80_0_3987_132
7ea9c5678db69497cac5ea5efedbbcf3	7ea9c5678db69497cac5ea5efedbbcf3	29928ea197b2dd37dbdc3144040d3bb9	9ff0023372e249c161e03a71055216ca
8551ff8dc5d6c8379e28cf8ecfdbf7e9	9ff0023372e249c161e03a71055216ca	664c79a566a55e42851b76c6e245915b	
9ff0023372e249c161e03a71055216ca		9ff0023372e249c161e03a71055216ca	

Table 4.9: Google Chrome Fingerprints

PCAP files from Opera browser belonging to Windows, Mac OS, Kali Linux and Ubuntu operating Systems were analysed and indicated a common unique fingerprint (9ff0023372e249c161e03a71055216ca) across the four operating systems as seen in Table 4.10.

The similarity between Google Chrome and Opera Browser is because they share the same engine, Chromium and Blink engine.

To validate the tool for Chrome and Opera fingerprinting, PCAP files for Firefox were used and different fingerprints generated as indicated in Table 4.11.

Since Chrome and Opera fingerprints are indistinguishable, we classify these browsers as one app, see the following Figure.

opera-klinux-66_0_3515_72	opera-mac-66_0_3515_72	opera-ubuntu-67_0_3575_53	opera-windows-66_0_3515_95
3534a4d8fcabf5fee54ed010c34b45a0	7ea9c5678db69497cac5ea5efedbbcf3	29928ea197b2dd37dbdc3144040d3bb9	664c79a566a55e42851b76c6e245915b
7ea9c5678db69497cac5ea5efedbbcf3	9ff0023372e249c161e03a71055216ca	6a7f738f44c5ad879841dbe0f688fb66	9ff0023372e249c161e03a71055216ca
9ff0023372e249c161e03a71055216ca	eeb35a05bfa15e7b7dc92f84e3cc3fd7	7ea9c5678db69497cac5ea5efedbbcf3	
		9ff0023372e249c161e03a71055216ca	

Table 4.10: Opera Browser Fingerprints

firefox-klinux-68_2_0esr	firefox-ubuntu-73_0_1	firefox-mac-73_0	firefox-windows-70_0_2
21890d87fa7da98f2b6cda22df895bcb	21890d87fa7da98f2b6cda22df895bcb	21890d87fa7da98f2b6cda22df895bcb	4411f0b337f1c2708cb8d98f58b9a447
74f477829a69ba89ff7942171e4f6f54	74f477829a69ba89ff7942171e4f6f54	74f477829a69ba89ff7942171e4f6f54	4b186ca6dfb44d2a496773fdc2b6944a
942292e4839c47998b8c32b97e45694c	942292e4839c47998b8c32b97e45694c	942292e4839c47998b8c32b97e45694c	74f477829a69ba89ff7942171e4f6f54
			942292e4839c47998b8c32b97e45694c

Table 4.11: Firefox Fingerprints

4.3.4 Discussion

Preliminary results have indicated that Firefox web browsers can be accurately identified across different operating systems. This is because of its unique fingerprint in the Client Hello TLS message. The application was modified in order to fingerprint Google Chrome and Opera browsers across the four operating systems.

Finally, the browser version does not matter with regard to fingerprint generation. The tests were done using different versions of browsers across the four operating systems, and similar fingerprints were identified.

4.4 JA3 Fingerprinting for Mobile Apps

4.4.1 Learning Phase

As mentioned above, the crucial task for mobile apps identification using TLS fingerprints is to create a reliable fingerprint database with unambiguous fingerprints. Even if an app is running in controlled environment like Android virtual studio, the captured traffic contains mixture of app traffic

```
Feb17,202020:15:35.144460835CET 192.168.10.163  45452   443     Chrome/Opera
Feb17,202020:15:35.144635635CET 192.168.10.163  45450   443     Chrome/Opera
Feb17,202020:15:35.250526167CET 192.168.10.163  33082   443     Chrome/Opera
Feb17,202020:15:35.250688300CET 192.168.10.163  33084   443     Chrome/Opera
Feb17,202020:15:35.284125370CET 192.168.10.163  38592   443     Chrome/Opera
Feb17,202020:15:35.284289878CET 192.168.10.163  38594   443     Chrome/Opera
Feb17,202020:15:35.364255338CET 192.168.10.163  57310   443     Chrome/Opera
Feb17,202020:15:35.364418825CET 192.168.10.163  57308   443     Chrome/Opera
Feb17,202020:15:35.388697844CET 192.168.10.163  59454   443     Chrome/Opera
Feb17,202020:15:35.394075447CET 192.168.10.163  59452   443     Chrome/Opera
Feb17,202020:15:35.401418329CET 192.168.10.163  52988   443     Chrome/Opera
Feb17,202020:15:35.404517725CET 192.168.10.163  56238   443     Chrome/Opera
Feb17,202020:15:35.405553918CET 192.168.10.163  56236   443     Chrome/Opera
Feb17,202020:15:35.405907450CET 192.168.10.163  52986   443     Chrome/Opera
Feb17,202020:15:35.425155704CET 192.168.10.163  59168   443     Chrome/Opera
Feb17,202020:15:35.425315285CET 192.168.10.163  59170   443     Chrome/Opera
Feb17,202020:15:35.427707637CET 192.168.10.163  40326   443     Chrome/Opera
Feb17,202020:15:35.444331974CET 192.168.10.163  40328   443     Chrome/Opera
Feb17,202020:15:35.445924856CET 192.168.10.163  39710   443     Chrome/Opera
Feb17,202020:15:35.462204150CET 192.168.10.163  39712   443     Chrome/Opera
Feb17,202020:15:35.530689350CET 192.168.10.163  40326   443     Chrome/Opera
Feb17,202020:15:35.562482720CET 192.168.10.163  40328   443     UnknownApp
Feb17,202020:15:35.602059221CET 192.168.10.163  44874   443     Chrome/Opera
Feb17,202020:15:35.602252839CET 192.168.10.163  44876   443     Chrome/Opera
Feb17,202020:15:35.666002421CET 192.168.10.163  58386   443     Chrome/Opera
Feb17,202020:15:35.673807362CET 192.168.10.163  56132   443     Chrome/Opera
Feb17,202020:15:35.709443697CET 192.168.10.163  38820   443     Chrome/Opera
Feb17,202020:15:35.748621705CET 192.168.10.163  37614   443     Chrome/Opera
Feb17,202020:15:35.748771156CET 192.168.10.163  53872   443     UnknownApp
Feb17,202020:15:35.817811227CET 192.168.10.163  60226   443     UnknownApp
```

Figure 4.5: Chrome and Opera Identification

and communication of OS, pre-installed apps and plugins that are common to multiple apps. Here, we introduced a technique, how to clean up the captured traffic in order to receive only TLS handshake related to the given app, see Figure 4.6.

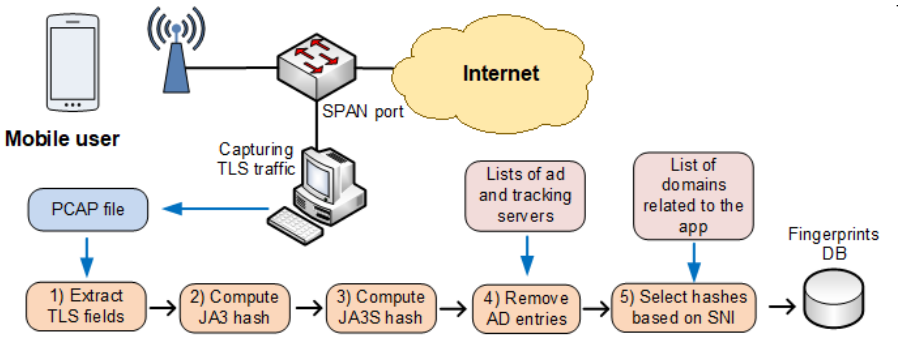

Figure 4.6: Creating TLS fingerprints

First, we need to launch an app communication on the mobile device. We made experiments both with virtual devices running on Android virtual studio (datasets MA2 and MA3) and on real devices (datasets MA1 and MA4). When using virtual environment, we can capture network traffic on the interface connected to the virtual environment. However, there can be also communication of virtual OS and other applications installed on the system. When using real smart phones, we can create a WiFi connection

only for this device and capture traffic on the WiFi interface. Fingerprints creation include the following steps:

1. Extract TLS Client Hello packets using `tls.handshake.type==1` Wireshark filter and obtain the following data: source and destination IP address, source and destination port, TLS handshake type (client or server hello), SNI, a list of TLS cipher suites, extensions, supported groups, and EC point format. Before computing JA3 hash, we preprocess TLS parameters and exclude GREASE values, padding and renegotiation options, see Section 4.1.1.

2. Compute JA3 hash using TLS values in TLS Client Hello packet as explained in Section 4.1.1. Apply MD5 hash function on TLS version, a list of cipher suites, list of extensions, supported groups and EC point format. JA3 hash uses MD5 function with 32-bit output in hexadecimal format.

3. Compute JA3S hash using TLS values in TLS Server Hello packet. JA3S hash is linked with JA3 hash using IP addresses and ports in Hello packet. JA3S hash is a MD5 digest of TLS version, cipher suite and extensions only.

4. Based on the list of ad servers and tracking servers, we remove TLS fingerprints where SNI matches any domain name from these lists. This excludes TLS fingerprints related to the noise traffic, see Table 4.5.

5. Now we need to select from candidate TLS fingerprints only those that are really related to the app. Selection is based on matching SNI field of TLS entries with keywords related to the app. There are several ways how to obtain app keywords. Applications like Lumen and AppVersion reveal information about the app which also includes domain names related to the app, see also Chapter 3. An example of keywords that match app SNI names is listed in Table 4.12. For many apps, keywords respond to app names. Formally, the keyword is the maximum common substring of SNI names related to the app.

The above written procedure describes how to select TLS fingerprints so that we are sure that the fingerprint belongs exactly to the given mobile app. Unfortunately, this procedure does not guarantee uniqueness of the obtained fingerprints which is essential for successful detection. When analysing JA3 fingerprints learnt from our datasets, we noticed that there were 30 distinct fingerprints, however, many of them belonged to multiple applications. Only 21 JA3 hashes were assigned unambiguously reaching uniqueness of 70%. Thus, we added JA3S fingerprint to a feature set and obtained 122 distinct combinations with 114 unique fingerprints related to only one app. Remember that one app can have a set of unique fingerprints.

Application	keyword	SNI
BoomPlay Music	boomplaymusic	source.boomplaymusic.com, android.boomplaymusic.com
Accuweather	accuweather, accu-weather	api.accuweather.com, cms.accuweather.com, vortex.accuweather.com
Viber	viber	content.cdn.viber.com
Discord	discord	best.discord.media, discordapp.com, dl.discordapp.net, gateway.discord.gg, ...
Mobilni Banka	mojebanka.cz, mobilnibanka.cz	www.mojebanka.cz, wa.mojebanka
KB klic	kb.cz	login.kb.cz, www.kb.cz
Nextbike	nextbike.net	api.nextbike.net, maps.nextbike.net, my.nextbike.net, static.nextbike.net, ...
EquaBank CZ	equa.cz, equamobile.cz	acs.equa.cz, ma.equamobile.cz
TitTok	tiktok	abtest-va-tiktok.byteoversea.com, mon.tiktokv.com
Duolingo	duolingo	android-api.duolingo.com, duolingo-leaderboard-prod.duolingo.com, ...
Youtube	youtube	www.youtube.com, youtubei.googleapis.com
Google Calendar	calendarsync	calendarsync-pa.googleapis.com
WhatsApp	whatsapp	media-prgl-l.cdn.whatsapp.net, pps.whatsapp.net, static.whatsapp.net
Gmail	mail.google.com, inbox.google.com	mail.google.com, inbox.google.com
Muj vlak	timetable.cz	ipws2.timetable.cz

Table 4.12: Example of app keywords

Combination of JA3+JA3S increases uniqueness to 93,44% but there were still several JA3+JA3S combinations that belonged to more than one app. After adding SNI to a feature set we received 154 distinct combinations with 153 combinations related to only one app. The results are given in Tab. 4.13.

Feature	Distinct items	Unique items	Uniqueness
JA3	30	21	70,00%
JA3+JA3S	122	114	93,44%
JA3+JA3S+SNI	154	153	99,35%

Table 4.13: Uniqueness of features in the TLS fingerprint

The number of unique fingerprints does not express, how many applications from our MA datasets we can cover with these fingerprints. We further analysed the sets of unique fingerprints to reveal this information. When using JA3 hash only, we can uniquely identify only 33,33% of apps from our dataset. Remaining apps cannot be identified with JA3 hashes only (66,67%), because there are no unique JA3 hashes for these apps in our datasets. When adding JA3S hash, we are able to cover 79,17% of apps from our dataset. Adding SNI to a feature set increase the coverage of apps to 91,67%. Using all three features, we were not able to cover 2 of 24 apps - Messenger and Telegram.

It seems that combination of JA3, JA3S and SNI provides unique and reliable TLS fingerprints of mobile apps. This statements is not always true. It depends on the function of the mobile app. Most mobile apps communicate only with limited number of servers related to the app. Some apps, for instance web browsers, communication with an open set of destinations based on user activity. For such apps we cannot use JA3S and SNI because these features depend on the destination. Also for applications that connect to servers with random or anonymized domain names, only JA3 hash can be employed for app identification. Interestingly, the JA3 hash of Tor app was

unambiguous, thus it could be used for mobile app identification.

4.4.2 Detection Phase

The above written procedure describes generation of TLS fingerprints from captured TLS traffic. The process includes TLS data pre-processing and refinement that produces a unique TLS fingerprint composed of JA3 hash only, combination of JA3+JA3S or JA3+JA3S+SNI. Having such fingerprints, we can monitor unknown network traffic, retrieve selected values from TLS Client and Server Hello packets, and compute JA3 and JA3S hashes. When these hashes match the fingerprint, we can deduce that the communication was initiated by the mobile app that is related to the fingerprint.

In real networks, detection engine retrieves TLS data from extended Netflow/IPFIX records or IDS logs. Of course, we are able to detect only those apps whose fingerprints are in the fingerprint database.

4.4.3 Stability and Reliability

As mentioned in Section 4.1.1, stability of TLS fingerprints of s a mobile app depend on an app version, TLS library, and operating system. When using JA3S hashes, it also depends on server version and its TLS library. This means that we have to check a fingerprint of a new app version. The fingerprint can be generated using procedure described in Section 4.4.1 which is not complicated. If TLS fingerprint(s) of a new app version are not in the fingerprint database, it should be added. In some cases, a new version may keep the same fingerprint as the previous one. Fingerprint stability is demonstrated on experiments with dataset MA3 where we observed TLS fingerprints of four apps on Android 7.1, 8.1 and 9. The results are presented in Table 4.14.

Mobile App	Android 7	Android 8.1		Android 9	
Feature	present	added	missing	added	missing
CP JA3	1	1	1	0	0
CP JA3S	2	2	2	0	0
CP SNI	2	0	0	0	0
Mujvlak JA3	1	1	1	0	0
Mujvlak JA3S	1	0	0	0	0
Mujvlak SNI	1	0	0	0	0
Reddit JA3	1	2	1	0	0
Reddit JA3S	1	2	1	0	0
Reddit SNI	9	3	2	4	0
Seznam CZ JA3	3	3	3	2	1
Seznam CZ JA3S	11	0	0	2	0
Seznam CZ SNI	12	0	1	1	0

Table 4.14: Stability of TLS fingerprints over OS version

The first column represents the number of unique feature values in TLS

fingerprint of a mobile app under Android 7. Columns Android 8.1 and 9 show the number of features that were added or missing in comparison to the previous version. We can see that SNI for CP app and Mujvlak are stable across versions. JA3S hash of CP app was changed when migrating from version 7 to 8 but it stayed unchanged to version 9. Adding new values does not negate stability of the fingerprint because the original fingerprint can still identify the app. If there are more additions, it may happen that the fingerprint of the older version would not match newly added features (false negative). However, when updating the fingerprint database by a new fingerprint, the accuracy of identification is preserved.

4.5 Evaluation

We evaluated TLS fingerprinting method on dataset MA4. Dataset contains communication of 14 apps captured in five distinguished time windows. Thus, we used the first four sets for training and creating fingerprints and the last set for detection. For training, we used procedure described in Section 4.4.1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	X
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	14
D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	12
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	13
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X	5	5	0	0	1	6	1	0	4	9	0	1	1	0	0	127

Table 4.15: Detection of mobile apps based on JA3 hash

Table 4.15 shows confusion matrix of app classification based on JA3 hash only. Letters A to O represent mobile apps, letter X describes unknown traffic. Rows contain predicted values, columns represent real values.

We can see limits of JA3 fingerprinting that works well with apps C (Facebook), D (Gmail), M (WhatsApp) and N (Youtube) but other apps have JA3 hashes same as unknown traffic (X class). By adding JA3S hash to TLS fingerprint, the number of correctly classified apps increases, see Table 4.16. However, there is still high number of false positives (column X).

Table 4.17 presents classification results for three features JA3+JAS3+SNI. We can see the classification is more accurate when using all these features.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	X
A	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	13
D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	18
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	4
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	17
M	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	10
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
X	0	2	0	0	1	4	1	0	3	3	0	0	2	0	0	101

Table 4.16: Detection of mobile apps based JA3+JA3S

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	X
A	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	8
D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X	0	1	0	0	0	6	0	0	1	2	0	1	1	1	0	158

Table 4.17: Detection of mobile apps based on JA3+JA3S+SNI

Table 4.18 shows accuracy, precision and recall of classification.

	Total items	Accuracy	Precision	Recall
JA3	211	65,88%	23,53%	26,67%
JA3+JA3S	211	61,61%	30,85%	64,44%
JA3+JA3S+SNI	211	90,05%	80,00%	71,11%

Table 4.18: Evaluation of combination of TLS features

JA3 hash is reliable only for specific apps and produces many false negatives (row X). JA3+JA3S classification has worst accuracy but better recall. This means that it produces a lot of false positives. The best result shows combination of JA3+JA3S+SNI. It also places some samples into X (unknown app) category, however, this can be improved by extending a list of keyword and inserting additional SNIs into fingerprint database.

4.6 Use Cases for Digital Forensics

Identification of smartphone apps can be applied to digital forensics as a complementary method to obtain forensically valuable information. First, the background traffic of installed applications can be analyzed to identify a communicating device [36]. This can distinguish a smartphone model from different vendors based on a pre-installed set of apps [15] and the background traffic. Next, by identification of communicating applications, user-specific information, habits and interests can be observed [11].

Most digital investigations that include mobile device analysis use logical extraction method to access the existing files such as call history, text messages, web browsing history, pictures and other files available on the smartphone. However, logical extraction requires the possession of the device and also the way to bypass passcodes protecting the access to the device. With smartphones better protected against the unauthorized access the passive monitoring of their activities stands for the complementary source of data for forensic analysis. The possibility to identify the smartphone application, and therefore the device or even the user of a device is applicable at least to the following scenarios:

- Forensic analysis. Law enforcement agencies may send a preservation order to the ISP to collect the communication for a specific device. The captured information can be used for learning about the activities of a suspect at different points in time. Creating a profile of the suspect and correlating identified activities with the information obtained from the other sources can bring an important insight to the case being investigated. Even if the most traffic of smartphone apps is encrypted, the presented method can detect installed applications. The presence and usage of a specific application at a given point of time may reveal the intention of the suspect. Later, after the device is physically available to an investigator the information obtained from the monitoring phase can be corroborated with the findings of the logical acquisition outcomes to support the reliability of evidence.

  As an example we consider the common situation today when a criminal communicates using instant messaging (IM) apps available on smartphones instead of voice calls, which makes this communication difficult to observe and decode. Traditional call record analysis upon which law enforcement agencies rely is thus not possible as we do not have access to the communication channels [40]. Before the specific method for reconstructing call records based on the observation of traffic patterns is applied, it is required to identify the IM app in use.

  Criminals aware of secrecy provided by IM apps can use them for communication to protect against traditional call record analysis [40]. However, if we can identify the activity of mobile apps, the traffic generated

by the communicating IM hosts can be used to record communication between suspects. Also, posts published under the anonymous social network account can be revealed by comparing the time of the public posts with the time of the actions as inferred by the application usage aiding to hate crime investigations.

Although the social network providers are willing or required to cooperate With LEA in these cases, it may be sometimes difficult to obtain enough information for the identification of an author of hate speech. The knowledge of installed applications and their usage patterns can be one of many sources for establishing the cyber profile of a suspect. As several marketing surveys pointed out mobile app usage varies by generation. The use of this information for creation of a profile, however, requires a reliable source of up-to-date data.

- Intelligence operations. The agencies may be able to trace certain individuals on basis of tracking the communication characteristic of the applications installed on their smartphones. To be feasible, the amount of information that needs to be collected and processed has to be limited. For instance, NetFlow-based monitoring is considered as suitable technique for this purpose [42].The advantage of TLS fingerprinting is that it can be applied at massive scale. Accommodating TLS fingerprinting to existing NetFlow monitoring system requires to include TLS fingerprints to NetFlow records, which many existing monitoring solutions already provide for the purpose of detection of security threats that use encrypted communication.

4.7 Summary

Mobile application fingerprinting can be considered a practical method with potential applications in digital forensics. In this paper, we have presented a study on the reliability of JA3-based methods for mobile application identification. The advantage of the method is that it only depends on the TLS handshake information that can be obtained from the initialization phase of the secure channel establishment.

We have shown that using JA3 only is not sufficient for accurate identification of apps. More reliable results were obtained by a combination of JA3 hash, JA3S hash, and Server Name Identification (SNI). All these features can be easily computed from TLS handshake messages. We have also considered the issues of TLS fingerprint volatility. Based on our experiments the variability of TLS fingerprints is not so large. Also, when a new major version of the application is released, it is not difficult to obtain a new fingerprint and update the fingerprint database.

The presented results are valid for existing TLS versions that provide

access to the source information necessary for computing the fingerprints. However, ongoing work on TLS protocol suggests an increase of user privacy by hiding more currently available fields, e.g., SNI[10], or even encryption of TLS ClientHello message. Addressing these emerging challenges may be a topic for future work.

[10]See Internet Draft at https://tools.ietf.org/html/draft-ietf-tls-esni-06 [March 2020].

Bibliography

[1] Robert Abel. SSL/TLS fingerprint tampering jumps from thousands to billions. *SC Magazine*, 2019.

[2] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé. MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation. In *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, pages 1–8, 2019.

[3] Blake Anderson and David McGrew. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In *Proceedings of the Internet Measurement Conference*, pages 379–392, 2019.

[4] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, pages 195–211, 2018.

[5] D. Benjamin. *Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility.* IETF RFC 8701, January 2020.

[6] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile Device Identification via Sensor Fingerprinting. *CoRR*, abs/1408.1416, 2014.

[7] Konstantin Böttinger, Dieter Schuster, and Claudia Eckert. Detecting Fingerprinted Data in TLS Traffic. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS Â'15, pages 633–638, New York, NY, USA, 2015. Association for Computing Machinery.

[8] Ralph Broenink. Using Browser Properties for Fingerprinting Purposes. In *16th Twente Student Conference on IT*, January 2012.

[9] Claude Castelluccia, Mohamed-Ali Kaafar, and Minh-Dung Tran. Betrayed by your ads! In Simone Fischer-Hübner and Matthew

Wright, editors, *Privacy Enhancing Technologies*, pages 1–17, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] G. Chittaranjan, J. Blom, and D. Gatica-Perez. Who's who with big-five: Analyzing and classifying personality traits with smartphones. In *2011 15th Annual International Symposium on Wearable Computers*, pages 29–36, June 2011.

[11] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, page 297–304, New York, NY, USA, 2015. Association for Computing Machinery.

[12] George Danezis. Traffic Analysis of the HTTP Protocol over TLS. 2009.

[13] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF RFC 5246, August 2008.

[14] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] Julien Gamba, Mohammed Rashed, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. An analysis of pre-installed android software. In *41st IEEE Symposium on Security and Privacy*. IEEE, 2020.

[16] Jayaprakash Govindaraj, Robin Verma, and Gaurav Gupta. Analyzing Mobile Device Ads to Identify Users. In Gilbert Peterson and Sujeet Shenoi, editors, *12th IFIP International Conference on Digital Forensics (DF)*, Advances in Digital Forensics XII, pages 107–126, New Delhi, India, January 2016. Springer International Publishing.

[17] Michael C. Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe Exposure Analysis of Mobile In-app Advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[18] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[19] P. Hoffman and P. McManus. *DNS Queries over HTTPS (DoH)*. IETF RFC 8484, October 2018.

[20] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. *Specification for DNS over Transport Layer Security (TLS)*. IETF RFC 7858, May 2016.

[21] Thomas Hupperich, Davide Maiorca, Marc Kührer, Thorsten Holz, and Giorgio Giacinto. On the robustness of mobile device fingerprinting: Can mobile users escape modern web-tracking mechanisms? In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 191–200, New York, NY, USA, 2015. ACM.

[22] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. Https traffic analysis and client identification using passive ssl/tls fingerprinting. *EURASIP Journal on Information Security*, 2016.

[23] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of TLS deployment. In *Proceedings of the Internet Measurement Conference 2018*, pages 415–428, 2018.

[24] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. Fingerprinting mobile devices using personalized configurations. *Proceedings on Privacy Enhancing Technologies*, pages 4–19, 2016.

[25] Andreas Kurtz, Andreas Weinlein, Christoph Settgast, and Felix Freiling. DiOS: Dynamic Privacy Analysis of iOS Applications. Technical Report CS-2014-03, Friedrich-Alexander-Universität Erlangen-Nürnberg, June 2014.

[26] Gabe Kwakyi. How Do Mobile Advertising Auction Dynamics Work? *Incipia Blog*, 2018.

[27] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[28] T. Matsunaka, A. Yamada, and A. Kubota. Passive os fingerprinting by dns traffic analysis. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 243–250, March 2013.

[29] Aruba Networks. *ArubaOS DHCP Fingerprinting*. Technical report, Aruba Networks, 2011.

[30] Elleen Pan, Jingjing Ren, Martina Lindorfer, Christo Wilson, and David Choffnes. Panoptispy: Characterizing Audio and Video

Exfiltration from Android Applications. *Proceedings on Privacy Enhancing Technologies*, pages 33–50, 10 2018.

[31] Jon Postel. *User Datagram Protocol*. IETF RFC 768, August 1980.

[32] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. Studying TLS Usage in Android Apps. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, page 350–362, New York, NY, USA, 2017. ACM.

[33] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 361–374, New York, NY, USA, 2016. ACM.

[34] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. IETF RFC 8446, August 2018.

[35] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. *Transport Layer Security (TLS) Renegotiation Indication Extension*. IETF RFC 5746, February 2010.

[36] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. 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*, WiSec '13, page 7–12, New York, NY, USA, 2013. ACM.

[37] G. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li. An Novel Hybrid Method for Effectively Classifying Encrypted Traffic. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5, 2010.

[38] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE, 2016.

[39] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust Smartphone App Identification Via Encrypted Network Traffic Analysis. *CoRR*, 2017.

[40] F. Tsai, E. Chang, and D. Kao. Whatsapp network forensics: Discovering the communication payloads behind cybercriminals. In

2018 20th International Conference on Advanced Communication Technology (ICACT), pages 1–1, 2018.

[41] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffness, Maarten van Steen, and Andreas Peter. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *NDSS*. The Internet Society, 2020.

[42] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 218–227, 2014.

[43] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM.