BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MOBILE APPLICATION MONITORING USING TLS FINGERPRINTS

MONITOROVÁNÍ MOBILNÍCH APLIKACÍ POMOCÍ OTISKŮ TLS

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAN KOČÍ**
AUTOR PRÁCE

SUPERVISOR **Ing. PETR MATOUŠEK, Ph.D. M.A.**
VEDOUCÍ PRÁCE

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Kočí Jan, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Kybernetická bezpečnost
Název: **Monitorování mobilních aplikací pomocí otisků TLS**
Mobile Application Monitoring Using TLS Fingerprints
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se s metodou JA3/JA3S pro identifikaci mobilních aplikací pomocí otisků TLS. Vytvořte databázi otisků TLS běžných mobilních aplikací.
2. Navrhněte způsob získávání TLS otisků z komunikace pomocí rozšířeného monitorování toků IPFIX (využijte sondy nprobe, flowmon probe apod.).
3. Navrhněte a implementujte systém detekce mobilních aplikací ze síťových dat.
4. Ověřte přesnost detekce aplikací a dále možnost profilování uživatelů, detekci nebezpečných aplikací, sledování vytíženosti sítě mobilními aplikacemi, apod.
5. Demonstrujte využití monitorování mobilních aplikací v reálném provozu.
6. Zhodnoťte svou práci. Diskutujte výhody i omezení daného přístupu pro správu sítě.

Literatura:

- MATOUŠEK Petr, BURGETOVÁ Ivana, RYŠAVÝ Ondřej a VICTOR Malombe. On Reliability of JA3 Hashes for Fingerprinting Mobile Applications. In Proceedings of ICDF2C 2020, s. 20.
- van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D.J., Lindorfer, M., Cho ness, D., van Steen, M., Peter, A.: FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Trac. In: NDSS. The Internet Society, 2020.
- Anderson, B., McGrew, D.: TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In: Proceedings of the Internet Measurement Conference. pp. 379-392, 2019.
- Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J.: Coming of age: A longitudinal study of TLS deployment. In: Proceedings of the Internet Measurement Conference 2018. pp. 415-428, 2018.
- Dokumentace k sondě nprobe dostupná online na URL https://www.ntop.org/guides/nprobe/ [září 2021].

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Petr, Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 26. října 2021

Abstract

The main purpose of this thesis is to study the possibility of using TLS fingerprints for mobile application monitoring and apply these methods to monitor network flows created by the Flowmon probe. To create the fingerprints the JA3 and JA3S methods are used. Apart from the TLS fingerprints, the implemented classifier uses SNI values to classify input flows. First, a dataset containing fingerprints of selected applications is created. This dataset is used together with the implemented classifier to classify input flows. Following is a description of the proposed classification methods and the implemented classifier. The classifier is evaluated using the Accuracy, Precision and Recall evaluation metrics. Finally, the classifier is used in several experiments that demonstrate its possible applications.

Abstrakt

Tématem této diplomové práce je monitorování mobilních aplikací pomocí otisků TLS a použití těchto metod pro rozšířené monitorování toků pomocí sondy Flowmon. K vytvoření otisků byly využity metody JA3 a JA3S. Kromě samotných TLS otisků jsou ke klasifikaci použity také hodnoty SNI. Na začátku práce je vytvořen dataset obsahující otisky vybraných aplikací. Tento dataset slouží ke správné klasifikaci aplikací v datových tocích pořízených sondou Flowmon. V práci se dále nachází popis implementovaného klasifikátoru a jeho metod. Na závěr je implementovaný klasifikátor vyhodnocen pomocí metrik Accuracy, Precision a Recall. Činnost klasifikátoru je demonstrována v několika experimentech, které slouží jako možné příklady jeho využití v praxi.

Keywords

TLS fingerprinting, JA3 fingerprints, JA3S fingerprints, mobile app monitoring, traffic monitoring, IPFIX, Flowmon probe.

Klíčová slova

Otisky TLS, JA3 otisky, JA3S otisky, monitorování mobilních aplikací, IPFIX, Flowmon sonda.

Reference

KOČÍ, Jan. *Mobile Application Monitoring Using TLS Fingerprints.* Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr Matoušek, Ph.D. M.A.

Rozšířený abstrakt

Monitorování síťového provozu je proces zkoumání příchozí a odchozí komunikace v daném místě sítě pomocí specializovaného hardwaru či softwaru. Jedná se o základní prvek umožňující operátorům sítě provádět operace jako profilování uživatelů, monitorování kapacity sítě či detekce nebezpečných aplikací. Jeden z důležitých aspektů monitorování sítě je schopnost klasifikovat procházející komunikaci. Klasifikace komunikace znamená přiřazení paketů či datových toků do jedné z několika známých tříd.

Tématem této práce je monitorování mobilních aplikací pomocí otisků TLS. Cílem práce je vytvořit systém, který je schopen, za pomoci TLS otisků, klasifikovat datové toky získané ze sondy Flowmon a úspěšně v nich detekovat vybrané mobilní aplikace. Metody použité k vytvoření TLS otisků se nazývají JA3 a JA3S. Kromě těchto metod používá vytvořený klasifikátor také hodnoty SNI, indikující název serveru, se kterým klient komunikuje.

Úvodní část práce nabízí stručné vysvětlení metod pro vytváření TLS otisků. Prezentovány jsou metody JA3 a JA3S. Následující část se podrobně věnuje vytváření datasetu s otisky vybraných aplikací. V této kapitole je vysvětlen proces odchytávání komunikace, extrakce potřebných hodnot ze zpráv TLS handshaku a vytvoření JA3 a JA3S otisků. Následně jsou v této kapitole uvedeny výsledky analýzy vytvořeného datasetu. Hlavním tématem této analýzy je kvalita vytvořených JA3 a JA3S otisků, jejich počet a schopnost jednoznačně identifikovat danou aplikaci.

Po úspěšném vytvoření datasetu a jeho následné analýze, bylo potřeba vybrat vhodnou sondu, která bude v této práci použita k vytvoření vstupních datových toků pro navržený klasifikátor. Následující kapitola se tedy věnuje srovnání dvou exportéru IPFIX flow dat. Jedná se o sondu Flowmon a sondu nProbe. Obě sondy jsou vyhodnoceny a porovnány s důrazem na jejich schopnost zpracovat dané hodnoty TLS protokolu, nutné pro vytvoření TLS otisků.

Poté co je úspěšně vybrána vhodná sonda, jsou v práci prezentovány metody použité ve vytvořeném klasifikátoru. První metoda používá ke klasifikaci vstupního datového toku jeho JA3 otisk. Tato metoda porovnává vytvořený otisk vstupního toku s otisky aplikací, uloženými v datasetu. Pokud vstupní otisk odpovídá některé z aplikací, je tato aplikace vyhodnocena jako výsledek klasifikace. Další metoda implementovaná v klasifikátoru, používá řetězec hodnot extrahovaných z TLS handshaku. Klasifikátor následně hledá aplikaci, jejíž řetězec je nejvíce podobný vstupnímu řetězci. Poslední dvě metody provádí klasifikaci pomocí hodnoty SNI. Klasifikátor hodnotu SNI zpracovává pomocí dvou různých přístupů - Jaccardův index a metoda TF-IDF.

Po představení jednotlivých metod klasifikátoru, jsou stručně uvedeny důležité implementační detaily práce. Následně je implementovaný klasifikátor vyhodnocen pomocí tří evaluačních metrik - Accuracy, Precision a Recall. Jednotlivé metody jsou vyhodnoceny samostatně a porovnány. Dále je klasifikátor prezentován v několika experimentech, které ukazují potenciální oblasti jeho použití. Na závěr je zhodnocena vhodnost použití JA3 otisků pro klasifikaci datových toků sondy Flowmon.

Mobile Application Monitoring Using TLS Fingerprints

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Petr Matoušek. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jan Kočí
May 16, 2022

Acknowledgements

I hereby thank my supervisor Mr. Petr Matoušek for his professional advice and guidance throughout the process of writing this thesis. I am also very thankful to my lovely girlfriend Miss Žanda Nováková, for her amazing support and encouragement.

Contents

Chapter 1

Introduction

In the year 2020, the use of malware increased by 358% compared to the previous year as stated in [16]. In fact, according to [20], cybercrime as a whole has increased by 600% since the beginning of the global pandemic. The annual Cost of a Data Breach Report by IBM [17] found out that remote work has increased the average cost of a data breach by $137,000 as employees do not have the same level of security at home as they would working in the office.

The importance of cybersecurity is a well-known topic discussed on a daily basis all around the world. And as the world continues to lean toward the use of technology and automation it is expected to be more and more relevant. One of the methods that can make computer networks safer is traffic monitoring. Traffic monitoring is the process of using specialized hardware or software to investigate the incoming and outgoing traffic on a computer network. And an essential part of traffic monitoring is traffic classification. Traffic classification is defined in [33] as assigning a label from a pre-defined finite set of labels to a particular network traffic object, e.g., a traffic flow. It is a significant feature for network operators as it enables them to perform tasks such as capacity management or performance monitoring. But most importantly it allows them to detect malicious software in the network.

Numerous methods have been developed for the traffic classification task. Some of the methods have even proven that it is possible to classify activities on the internet from encrypted network traffic, e.g., [21] and [3]. One of those methods uses JA3 and JA3S fingerprints.

The purpose of this thesis is to demonstrate the task of traffic classification and create a system able to monitor mobile applications in a network based on their JA3 fingerprints. The thesis starts by discussing similar work in the field of traffic classification in Chapter 2. After that, TLS fingerprinting and the JA3/JA3S methods are described in Chapter 3. Chapters 5 and 6 provide an introduction of two network probes - the nProbe and the Flowmon probe - and analyse the outputs they produce. After that, Chapter 7 provides a description of the methods used in the proposed classifier. Chapter 8 presents the implementation details of the most important classes and methods implemented in this thesis. In Chapter 9 the classifier is evaluated using the Accuracy, Precision and Recall evaluation metrics. Finally, Chapter 10 conducts several experiments to demonstrate possible applications in which the proposed classifier can be used.

Chapter 2

Related work

The purpose of this chapter is to show similar work related to this thesis. The problem of traffic classification has been visited many times in the past. It is an important task used in many areas of network monitoring systems. It supplies internet providers with valuable information that can be used to provide different levels of Quality-of-Service, detecting anomalies and frauds etc. In addition, some enterprise services may need a higher priority than other traffic.

As mentioned in study [33], some of the early methods of traffic classification used a port-based approach. In these systems, the source and destination port numbers of a packet were used to identify the application. This approach alone, however, does not deliver a reasonable Accuracy and is therefore usually used in combination with other techniques. A study by Bakhshi et al. [5] used machine learning techniques together with port numbers in a two-phase system.

Another method known as *deep packet inspection* (DPI) utilises the patterns and signatures available in the application layer of a packet. This technique is used for instance by an open-source tool for traffic classification called nDPI [11].

Nowadays there is a rising interest in systems that use statistical, machine learning and deep learning methods. These systems extract statistical features from a given flow, traffic or even a single packet. Paper [31] describes a method using the cumulative sum of packet lengths.

The authors of paper [31] created a method using the cumulative sum of lengths of the first 100 packets in the flow and then using a k-Nearest Neighbor classifier to fingerprint web pages. They achieved an Accuracy of 91,6%.

Another interesting approach was presented in paper [39], where they used the first 784 bytes of each flow representing it as a 28x28 matrix and feeding it into a convolutional neural network. They achieved an Accuracy of 99.41% in the task of malware traffic classification.

All the above-mentioned methods presented different approaches to traffic classification. In this thesis, the approach of TLS fingerprinting was used. The development of this technique is connected to the research of Ivan Ristic. In his research in 2008 [1], he developed an Apache module to fingerprint connected clients based on their cipher suites. Blake Anderson et al. investigated encrypted TLS flows in paper [4]. They discovered a set of observable data features from TLS *Client* and *Server Hello* messages. These were, e.g., the TLS versions, TLS cipher suites and TLS extensions.

When it comes to using TLS fingerprinting for mobile applications several interesting approaches could be mentioned. In a study [28] from 2017, the authors analyzed the TLS usage in Android applications creating their own app Lumen that was intercepting TLS

communication and gathering various statistics. Stöber et al. created a classifier for mobile applications described in [35]. The classifier used side-channel information such as timing and data volume. Side channels were also used in paper [36] by Taylor et al. They used machine learning techniques to identify smartphone apps from this side-channel data. In addition to merely fingerprinting and identifying smartphone apps, they also investigated how app fingerprints change over time, across devices and across different versions of apps.

The approach used in this thesis is different from the techniques mentioned above. It builds on top of the JA3/JA3S TLS fingerprinting technique and applies it to classify selected mobile applications in network flows. Moreover, it also adds values from the SNI field to improve the classification. This approach was chosen as it allows the classification of encrypted traffic. The following section explains the JA3 and JA3S methods used to create the fingerprints.

Chapter 3

TLS fingerprinting

This chapter is going to discuss the methods of TLS fingerprinting used in this thesis. First, it briefly describes the TLS protocol and how it can be used to identify an application. It then proceeds by explaining TLS fingerprinting focusing on the JA3 and JA3S hash methods.

3.1 TLS protocol

The TLS Protocol Version 1.3 is described in RFC [30]. TLS stands for Transport Layer Security. As the name suggests it is a protocol designed to facilitate confidentiality and data security for communications over a computer network. The security is achieved by encrypting the communication between web applications and servers. TLS is widely used in many applications such as email or instant messaging and is also used in the HTTPS protocol. HTTPS is an implementation of TLS encryption on top of HTTP which is the foundation of any data exchange on the World Wide Web. The description of HTTPS can be found in RFC [29]. As Figure 3.1 shows the TLS protocol is between the Application and Transport layer in the TCP/IP model.

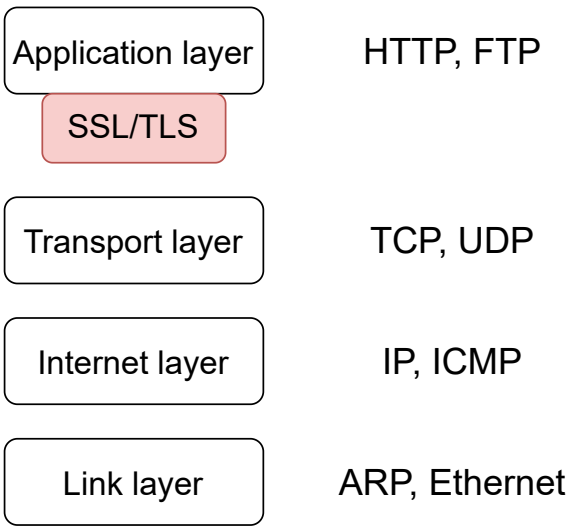

Figure 3.1: TCP/IP model and the TLS protocol.

Before any two applications are able to start the encrypted communication using TLS, they need to initialize the connection. During the initialization, the two parties exchange certain messages to acknowledge and verify each other, select the encryption algorithms they will use and derive session keys. This process is known as the TLS handshake. Its description can be found in RFC [30].

The main goal of the TLS handshake is to agree on a shared symmetric encryption key that will be used to encrypt the communication. To do so, asymmetric cryptography is used. Following are the TLS handshake messages in the order they occur as described in [37][9][30]:

1. Client Hello: The first message of the handshake is sent by the client and is called the *Client Hello Message*. It contains some important information such as the TLS versions supported by the client, the cipher suites that are available and some random number called *Client Random* that will be further used to derive the symmetric key.[9][30]

2. Server Hello: The server responds with its *Server Hello Message*. It contains the chosen TLS version and cipher suite. It further includes the digital certificate of the server with a corresponding public key and another random number *Server Random*. The server then proceeds by sending its digital signature. The signature is created as a summary of the previous messages hashed and encrypted using its private key. The server then ends its move by sending the *Server Hello Done* message.[9][30]

3. Premaster Secret: Client verifies the validity of the provided certificate and its public key by decrypting the digital signature. The client usually does not have to authenticate itself. Therefore it continues by sending the *Client Key Exchange* message in which it sends another random value called *Premaster Secret* encrypted with the public key presented by the server.[9][30]

4. Deriving session keys: The server uses its private key to decrypt the *Premaster Secret*. At this point, both client and server know the *Client Random*, the *Server Random* and the *Premaster Secret*. They use these values to calculate the session keys. Since the *Premaster Secret* is a secret value that can be decrypted only by the private key of the server, it is guaranteed that no one else will be able to derive the session key.[9][30]

5. Client Finished: The client concludes by sending the *Change Cipher Spec* message indicating that from now on all communication will be encrypted. At last, it sends the *Client Finished* message which combines all the previously sent messages encrypted with the newly derived symmetric key.[9][30]

6. Server Finished: Server verifies that the handshake was successful by decrypting the *Client Finished* message. It concludes by sending its *Change Cipher Spec* and *Server Finished* messages.[9][30]

Figure 3.2 demonstrates the messages of the handshake as they appear and displays them in a sequence diagram. First of all the TCP 3-way handshake takes place. After that, the client starts the TLS handshake with its *Client Hello Message*. After the handshake is completed both sides can start transferring their encrypted data.

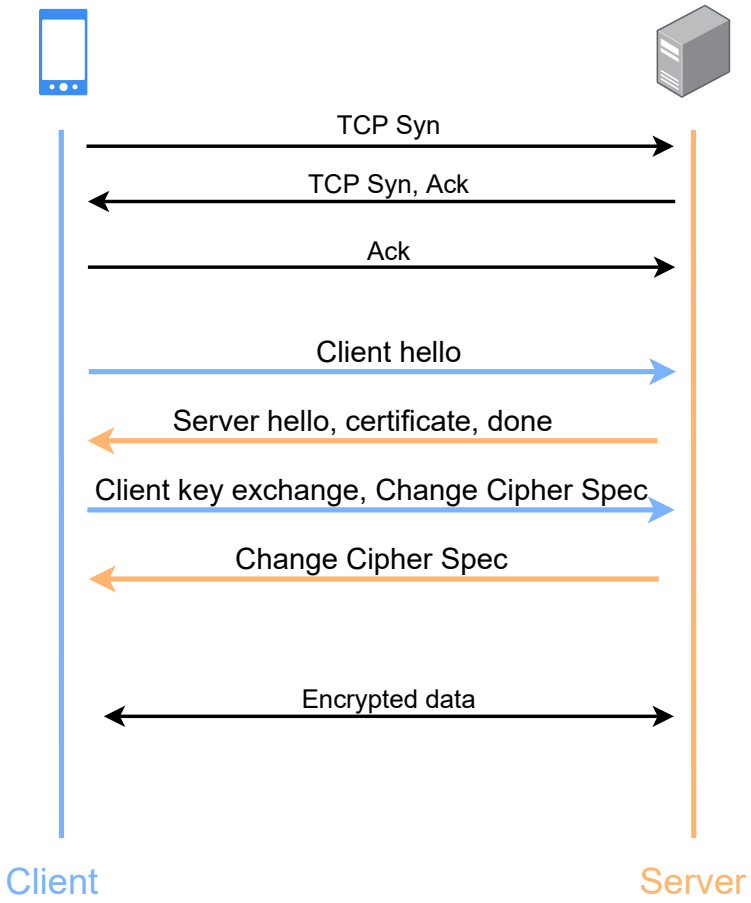

Figure 3.2: TLS handshake.

The TLS handshake is without a doubt an essential part of the protocol. It establishes the cryptography needed to create a secure channel between the two parts involved. In the next section we are going to discuss how the TLS handshake can be used to create fingerprints of applications.

3.2 TLS fingerprinting

Fingerprinting is a technique of creating some sort of a mark with the intention of using this collected mark as a way of identification. The main idea behind TLS fingerprinting is the fact that certain values exchanged during the TLS handshake could be used to identify the client and server applications. These values are actually depending on the modules and libraries that the application is using and therefore provide a stable and valuable method of identification.

The methods used to create TLS fingerprints in this thesis are called JA3 and JA3S. They were presented by John B. Althouse, Jeff Atkinson and Josh Atkins in 2015.[1] The primary concept of TLS fingerprinting came from Lee Brotherston in his research in 2015.[2]

[1] See https://github.com/salesforce/ja3 [Accessed 2021-12-30]
[2] See https://blog.squarelemon.com/tls-fingerprinting [Accessed 2021-12-30]

3.3 JA3 method

The JA3 method is described in this [3] article by John Althouse. The method is used to create a fingerprint for the client application. It operates by extracting certain values from the *Client Hello Message* of the TLS handshake and converting the bytes into a decimal format. The following fields are extracted:

- **TLS version**: The TLS version supported by the client.
- **Supported cipher suites**: A list of the supported cipher suites. A cipher suite is a set of security algorithms. It specifies one algorithm for each of the following tasks: key exchange, bulk encryption, message authentication.[38]
- **List of extensions**: Specifies which extensions are supported by the client. These extensions present a special mechanism for adding functionality to the protocol without the need of modifying it. Some of the commonly used extensions are, e.g., the Server Name Indication (SNI) extension and the Elliptic curves extension.[7]
- **Elliptic curves**: Also called Supported groups. An extension specifying the elliptic curves supported by the client. Elliptic-curve cryptography is an approach to public-key cryptography based on the algebraic structure of elliptic curves.[25]
- **Elliptic curves formats**: An extension specifying the formats of elliptic curves supported by the client.[25]

This method then concatenates these values using a comma to delimit each field and a dash to delimit each value within the field while also preserving their exact order. The concatenated result is then hashed using the MD5 hash function creating the final 32-character-long JA3 TLS client fingerprint. Figure 3.3 illustrates this process in a simple way.

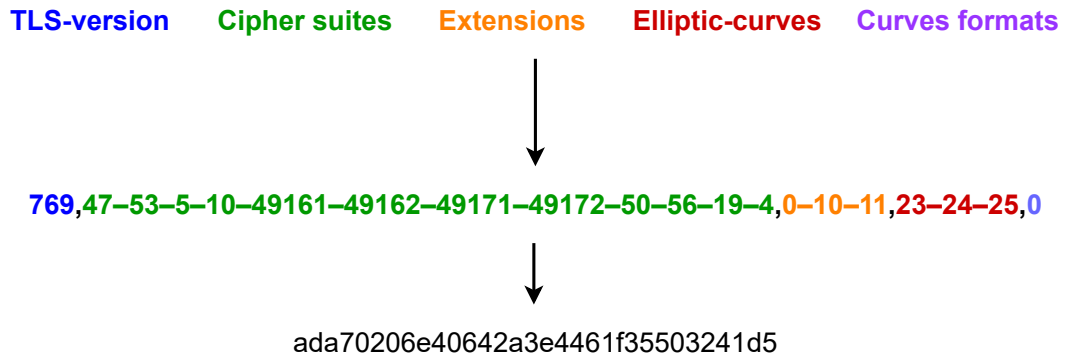

Figure 3.3: Construction of the JA3 hash.[3]

The fields extracted from the *Client Hello Message* are illustrated in Figure 3.4.

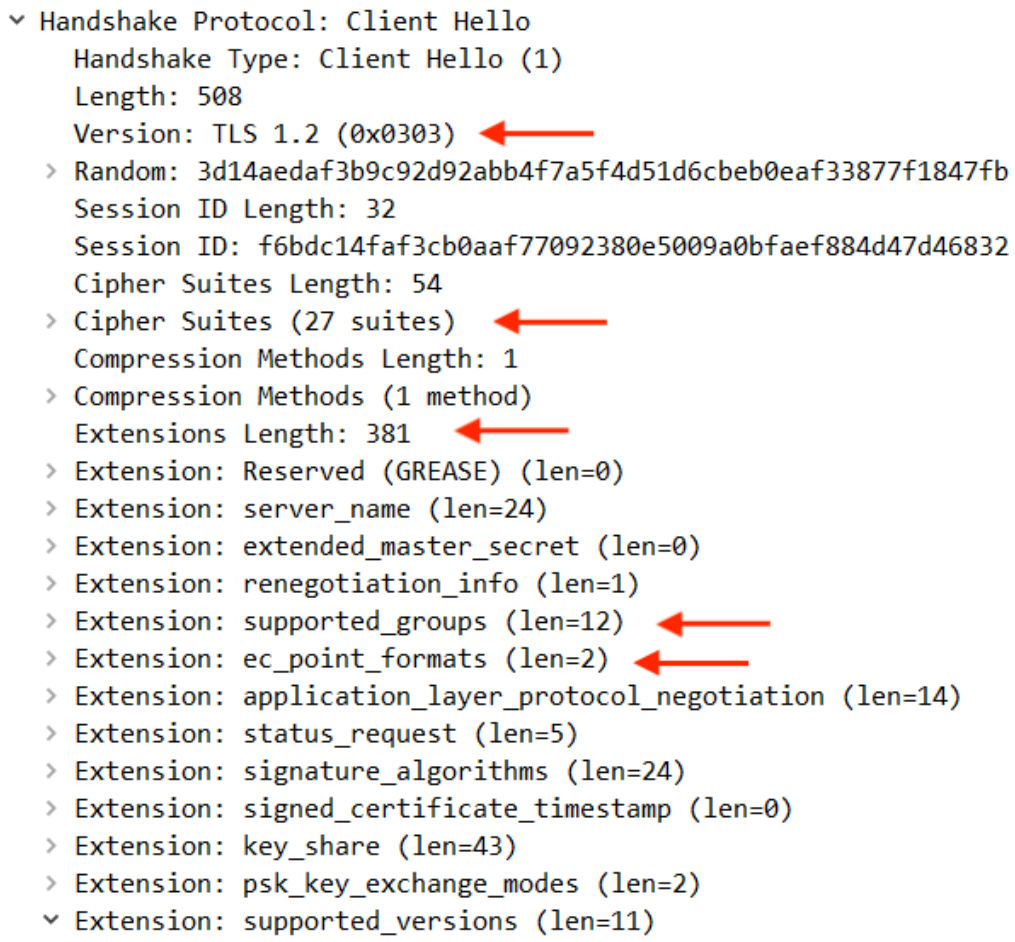

Figure 3.4: Values extracted from the *Client Hello Message*[3].

3.4 JA3S method

The JA3S method is used for the server side of the communication and represents how the server responds to a certain client. Description of this method was also obtained from the [3] article. The following fields are extracted from the *Server Hello Message* of the TLS handshake:

- **TLS version**: The TLS version used by the server.[30]
- **Accepted cipher**: Specifies the cipher suite that will be used from the list of cipher suites supported by the client.[30]
- **List of extensions**: Defines the extensions that will be used within this session.[30]

The JA3S method extracts these values from the *Server Hello Message* converting them into a decimal format and concatenating and separating them in the exact same way as the JA3 method does. Finally these values are once again hashed using the MD5 hash function creating the final fingerprint.[3]

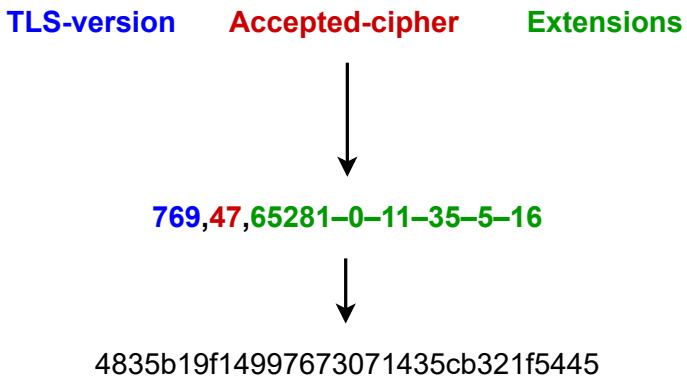

Figure 3.5: Construction of the JA3S hash.[3]

The same server formulates its *Server Hello Messages* differently depending on the contents of the *Client Hello Message* it receives. This means that the JA3S hash cannot be used to fingerprint a server just based on its *Server Hello Message*. However, as mentioned in [3], even though the same server responds differently to different clients, it always responds the same to the same client. The JA3S hash can therefore be used to increase the ability to identify the client application.

This chapter provided an introduction to the problem of TLS fingerprinting. The two methods used to obtain the fingerprints in this thesis were explained. These are the JA3 and JA3S methods. The next section discusses the application dataset that contains fingerprints of selected applications created using the JA3 and JA3S methods. This dataset will then be used by the classifier, described in section 7, to classify input flows.

Chapter 4

Creating the application dataset

This chapter describes the process of creating the application dataset with JA3 and JA3S fingerprints. It explains the network topology used to capture the traffic, provides an overview of applications used in the dataset, discusses the necessary preprocessing of the captured packets and finally conducts an analysis of the created fingerprints.

The main goal of this thesis is to be able to detect mobile applications in network traffic using TLS fingerprints. In order to be able to identify the applications, it is necessary to compare the obtained fingerprints with fingerprints of applications that have already been seen. Therefore, the first step of this thesis is to create a dataset that contains JA3 and JA3S hashes of known applications. The network topology used to create the dataset is shown in Figure 4.1.

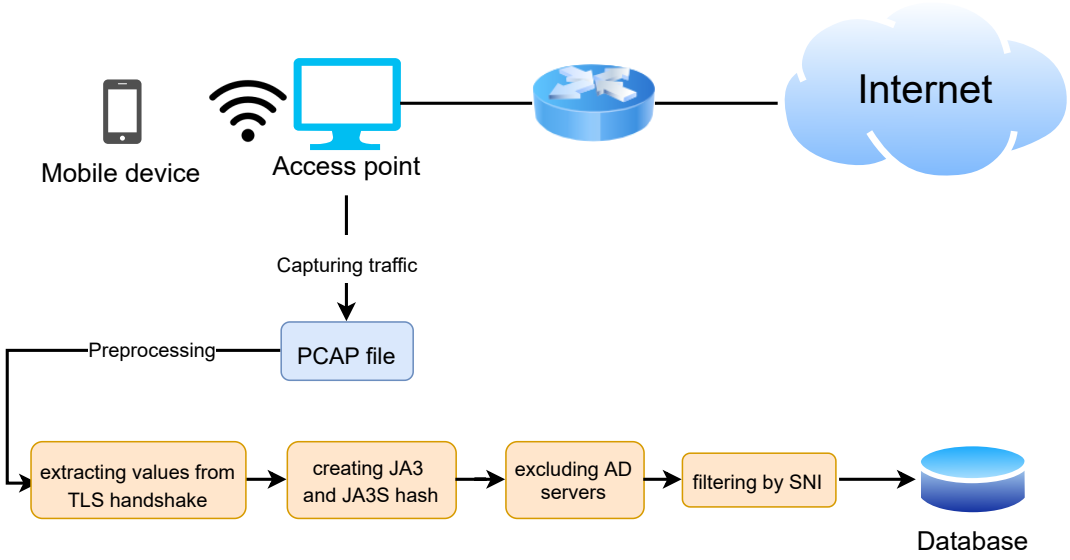

Figure 4.1: Topology of the environment used to acquire the dataset.

To acquire the data from the communication between the applications and their servers, a wireless access point was used. The access point was created on a computer connected to a wired network through an Ethernet cable. The computer uses its own Wi-Fi to broadcast the internet connection allowing other devices to wirelessly connect to it. The computer then acts as a router for the connected devices. All their traffic is going through the

computer. Each application was run on a device connected to this access point and was used for several minutes to generate as much traffic as possible. In the meantime, on the access point, a program called Wireshark[1] was intercepting all the traffic going from the device. Finally, the intercepted traffic was exported into a *pcap* file for further processing.

The processing includes tasks like extracting the desired TLS values from the handshakes, creating the fingerprints from these values, excluding advertising servers from the traffic and filtering the traffic by the SNI field to only include the traffic between the applications and their servers. The filtering was performed for each application in the dataset and is done by comparing the SNI field of each application to a list of keywords belonging to the application. The keywords of each application usually include the name of the application and several other special identifiers. This whole preprocessing part is performed by the TLSParser class described in Chapter 8.

4.1 Tested applications

To create the dataset, the applications that the classifier is going to work with need to be selected. These applications are selected across several different areas of the industry from social networks to games, finance, etc. The applications used in this thesis are described in Table 4.1. Apart from the application name, the dataset also needs to contain the version of each application. This is needed because different versions of the same application can use different libraries and hence generate different JA3 hashes. Another important information is the operating system of the device on which these applications were used. In this case, it was an Apple smartphone with the iOS 15.0.2 operating system.

Application	Version	Category	Description
Airbnb	21.43	Traveling	Online marketplace for lodging.
Booking.com	31.3	Traveling	Online travel agency.
Couchsurfing	5.38.1	Traveling	Social networking and hospitality exchange service.
Crypto.com	3.122	Finance	Cryptocurrency exchange app.
Duolingo	6.139.0	Education	Language-learning app.
Equabank	22.0.0	Finance	Mobile banking app.
Facebook	346.0	Social	Social media and networking service.
George	21.30.11	Finance	Mobile banking app.
Livesport	4.12.1	Sports	Instant sports data and results.
Pinterest	9.42	Lifestyle	Image sharing and social media service.
Quora	8.3.4	News & Magazines	Social question-and-answer app.
Shazam	15.1.0	Music & Audio	Music recognition app.
MS Teams	3.17.1	Business	Business communication platform.
Tiktok	21.7.0	Social	Video-focused social networking service.
Uber	3.481.10002	Maps & Navigation	Transportation mobility service.
Whatsapp	2.21.210	Communication	Instant messaging app.
Youtube	16.43.2	Video Players & Editors	Video sharing and social media platform.
Zalando	5.15.1	Shopping	Fashion and lifestyle e-commerce app.

Table 4.1: Application used in the dataset.

[1] See https://www.wireshark.org/ [accessed 2021-12-31]

4.2 Data preprocessing

The captured traffic contains a lot of information that needs to be filtered. It includes the entire communication of the device with the internet. Even though the capturing was performed only when the application was being used, there is still a lot of traffic happening in the background. This additional traffic could be generated by the system or even by applications running in the background.

The captured *pcap* file is passed to the TLSParser class. This class uses the *Scapy*[2] Python library to extract the desired values from packets inside the *pcap* file and outputs a pandas DataFrame object. In particular, the parser extracts the IP source and destination addresses and all the fields necessary for creating the JA3 and JA3S fingerprints. In order to be able to assign a JA3S hash of a server to the correct JA3 hash of a client, the parser needs to match the packets according to their IP addresses and their Sequence and Acknowledgment Numbers.

Another important thing that the parser does is removing the GREASE values from the TLS fields. The GREASE stands for Generate Random Extensions And Sustain Extensibility. It was designed by David Benjamin, a software engineer at Google, as a new mechanism to spot potential compatibility issues in the TLS protocol. The description of GREASE can be found in this [6] document. The main objective of this mechanism is to detect interoperability issues early. It does so by using some reserved and currently unused TLS values and advertising them randomly within the fields in the TLS handshake. Servers and software that implement TLS correctly should be able to receive these GREASE values from a client without the connection failing or terminating. If one of the values provided by the client is not recognized by the server, it should be able to ignore it and successfully establish a connection using the best available options. For example, if the client supports some new cutting edge cipher but the server does not, the server is just going to ignore that cipher and it will pick one that is supported by both. More about the mechanism of GREASE can be found in this [23] article.

After parsing the values and creating the fingerprints the parser also filters the traffic using the Server Name Indication (SNI) field. The SNI was initially created with the intention of enabling the existence of virtual servers and their reachability through HTTPS. It indicates the hostname to which the client is attempting to connect. The hostname needs to be known during the TLS handshake. This allows the server to respond with one of the multiple possible certificates each corresponding to different services running on the same IP address. Therefore, the SNI field was added to the TLS protocol.

In this thesis, the SNI helps to identify the servers corresponding to the applications. For each application, there is a list of keywords. Usually, an application has just one keyword - the name of the application. The list of keywords was constructed manually by inspecting the traffic. First, all packets with an empty SNI value are dropped. Then only those *Client* and *Server Hello* messages with the SNI value containing at least one of the keywords of the corresponding application are selected. The rest of the packets are excluded from the dataset. By this, messages that were not generated by the application but rather by some other programs running in the background and creating noise in the capturing process are omitted. Finally, a list of advertising and tracking servers is used to filter out the traffic generated by these servers and exclude it from the final dataset.

[2] https://scapy.net/

Application	Keywords	SNIs
Duolingo	[duolingo]	goals-api.duolingo.com, friends-prod.duolingo.com, ...
Teams	[teams, microsoft]	teams.microsoft.com, ic3.events.data.microsoft.com, ...
George	[george, csas]	george.csas.cz, storego.csas.cz, ...

Table 4.2: Example of application keywords and their corresponding SNIs

The final dataset includes not only the JA3 and JA3S hashes but also the actual values from TLS handshakes that the hashes are created from. These values are stored in the decimal format described in Section 3.3 and are used in the classifier to compute similarities with input flows. This topic is described in Chapter 7. Table 4.3 shows the fields included in the final dataset and the values they contain.

app_name	duolingo	youtube
os_version	ios_15.0.2	ios_15.0.2
app_version	6.139.0	16.43.2
sni	brb.duolingo.com	youtubei.googleapis.com
ja3_hash	03b2be863a...	89688f68c3...
ja3s_hash	35b476bf06...	eb1d94daa7...
ja3_decimal	771,4865-4866...	771,4865-4866...
ja3s_decimal	771,49199,11-16	771,4865,51-43

Table 4.3: Example of the final datset.

As already mentioned before, this dataset is used in the classification of input flows by comparing the values extracted from these flows with those already collected in the dataset. The next section provides a brief analysis of the created application dataset.

4.3 Dataset Analysis

Analysing the dataset is an important task that allows to better understand the collected data. The applications that were used were chosen from a variety of different areas and categories. This should make the results more unique and distinguishable.

The dataset consists of 18 different applications. It contains a total of 120 records (record = one row in the dataset) with an average of 7 records per application. The median being 3 and mode 2. It is good to mention, however, that the number of records for each application is quite variable. The maximum is 31 records for TikTok and the minimum 1 record for Uber and Livesport. Following is a table showing the number of SNIs, JA3 and JA3S hashes for each application.

Application	SNIs	JA3 hashes	JA3S hashes
airbnb	2	1	1
booking.com	6	1	2
couchsurfing	2	1	2
crypto.com	3	1	3
duolingo	12	1	4
equabank	2	1	3
facebook	6	4	2
george	4	2	2
livesport	1	1	1
pinterest	2	1	2
quora	19	1	2
shazam	2	1	2
teams	16	2	5
tiktok	26	5	10
uber	1	1	1
whatsapp	2	2	1
youtube	2	1	1
zalando	3	1	2

Table 4.4: The number of SNIs, JA3 and JA3S hashes for each application.

Table 4.4 shows the number of SNIs, JA3 and JA3S hashes collected for each application. These values are unique within the application meaning that e.g. Facebook has four unique JA3 hashes but these hashes can be shared with other applications. They are not unique throughout the whole dataset.

As we can see the number of SNIs is also quite variable with the biggest value being 26 SNIs for TikTok and the lowest 1 for Uber. This value indicates with how many servers an application communicates. Most applications have only one JA3 hash. This could have been beneficial if all of these hashes would be unique and therefore would identify each application with certainty.

The important thing that needs to be discussed and learned from this analysis is whether the created JA3 and JA3S hashes can be used to identify each application. In other words we need to find out if each application has a hash that is unique and only occurs for this application. If an application does not have a unique JA3 hash it can still be successfully identified if it has a unique JA3S hash. Table 4.5 shows the results of this examination for each application.

Application	Unique JA3 hash	Unique JA3S hash	Result
airbnb	YES	NO	YES
booking.com	NO	NO	NO
couchsurfing	NO	YES	YES
crypto.com	NO	YES	YES
duolingo	NO	YES	YES
equabank	NO	YES	YES
facebook	YES	NO	YES
george	YES	YES	YES
livesport	NO	NO	NO
pinterest	NO	NO	NO
quora	NO	YES	YES
shazam	NO	YES	YES
teams	YES	YES	YES
tiktok	YES	YES	YES
uber	NO	YES	YES
whatsapp	YES	NO	YES
youtube	NO	NO	NO
zalando	NO	NO	NO

Table 4.5: Unique hashes for applications and the result of the ability to identify them.

The result column displays whether an application can be identified using JA3 and JA3S hashes. It is computed as a logical disjunction of the results in the previous two columns. That means an application can be successfully identified if it has a unique JA3 hash or a unique JA3S hash. The uniqueness of the JA3S hash does not need to be for the whole dataset. It only needs to be unique within the applications that reached their server with the same JA3 hash.

As shown in Table 4.5, 5 applications cannot be identified. In particular, they are Booking.com, Livesport, Pinterest, Youtube and Zalando. Consequently, it is clear that JA3 and JA3S hashes are not enough to solve our identification problem. That is why additional techniques were used to enhance the classification abilities, for example SNI values.

Additional inspections of the created JA3 hashes were performed. For the 18 applications and 126 collected records, there are 13 unique JA3 hashes. Some correspond to only one application some are shared between many. The fact that two and more applications generate the same hash is a common problem. It is caused by the fact that these applications use the same libraries or frameworks etc. Table 4.6 presents the results of this analysis in greater detail.

Only two hashes are shared between multiple applications. The first one corresponds to eight applications while the other one to even nine different applications. There are some applications in these two groups that do not have any unique JA3 hash. They could only be distinguished from the other applications from their group by the JA3S server hash. Table 4.7 and 4.8 show the JA3S hashes and their applications corresponding to the two common JA3 hashes. Applications that do not have their unique JA3S hash are not able to be identified. These applications are marked in following tables using bold font and include Livesport from the first group and Pinterest, Zalando, Youtube and Booking.com from the second.

JA3 hash	SNIs	Apps	App names
03b2be863a6f11cb9269bcc646797545	40	8	duolingo, equabank, facebook, uber, quora, tiktok, livesport, whatsapp
13aa429c5dbaaf587d8fd5d52149ab36	3	1	facebook
1f3c530fc35e41300422550c3c980e85	10	1	tiktok
2438872a1892fbb853f8ded46fd72da2	1	1	whatsapp
7376b944d8a406b95e9408b3c52b8b5a	1	1	facebook
7a39a2b5aef9100fb75dd4169f560b95	6	1	tiktok
89688f68c3d80a111d642ed73232e1e6	38	9	booking.com, teams, george, crypto.com, pinterest, shazam, couchsurfing, youtube, zalando
8c7f19a25372add41febf9879dd33d0a	1	1	facebook
a839cfeed30d55439b09de5f1b47fa3a	13	1	tiktok
d38a313f1a8b2dad3cbc52c69bf3e31c	1	1	george
d889531a0389787425d5638caf6d84b3	1	1	tiktok
e4d448cdfe06dc1243c1eb026c74ac9a	3	1	teams
f469e7c27055847567d4c5d8e45bb564	2	1	airbnb

Table 4.6: This table shows unique SNIs and applications per JA3 hash.

03b2be863a6f11cb9269bcc646797545	
JA3S hash	**Applications**
040e57cd679445d961adbb68d4f9873c	equabank
15af977ce25de452b96affa2addb1036	**livesport**, tiktok
35b476bf06ae1c4b036bf7246dcf499c	duolingo
63ddcc036b96a2854f2db888363204d8	equabank
70745099b394fe3f42264227c098cc98	quora
896415616b22361262d7a961b6325cfd	duolingo
af830039016ecfd26e620b69f9a3bc14	equabank
eb1d94daa7e0344597e756a1fb6e7054	uber
f4febc55ea12b31ae17cfb7e614afda8	duolingo, facebook, quora, whatsapp

Table 4.7: This table shows all the JA3S hashes corresponding to JA3 hash 03b2be863a6f11cb9269bcc646797545 and their applications.

89688f68c3d80a111d642ed73232e1e6	
JA3S hash	**Applications**
040e57cd679445d961adbb68d4f9873c	shazam
0f5a6116c5ad5ff8cbe0a8ccfb2fbd95	george
15af977ce25de452b96affa2addb1036	**pinterest**, teams, **zalando**
1ea9f5ee48f642a7f190cfe16bc3de12	crypto.com
37193c4da5f334b9c542d38a26296d08	teams
545ffb1bd88f345709bd65c96be77da1	teams
63ddcc036b96a2854f2db888363204d8	george
70745099b394fe3f42264227c098cc98	couchsurfing
78c23e5cac676891c5f192ebf55ee966	teams
eb1d94daa7e0344597e756a1fb6e7054	crypto.com, **pinterest**, shazam, **youtube**, **zalando**
eff7860f129871c984e2dc46e87717ae	teams
f4febc55ea12b31ae17cfb7e614afda8	**booking.com**, couchsurfing

Table 4.8: This table shows all the JA3S hashes corresponding to JA3 hash 89688f68c3d80a111d642ed73232e1e6 and their applications.

This chapter discussed the created application dataset. The dataset consists of 18 applications and is stored as a *csv* file. Each row in the dataset contains the following fields: name and version of the application, version of the operating system, SNI, JA3 hash, JA3S hash, a string of decimal JA3 values and a string of decimal JA3S values. The analysis of the dataset revealed that using only JA3 and JA3S hashes would not enable the classifier to certainly classify each application. Therefore, the SNI values are included in the dataset and are also used to correctly classify input flows. The next section explains the IPFIX protocol and describes the essentials of flow exporters.

Chapter 5

Choosing the Flow Exporter

The purpose of this chapter is to explain the process of collecting flow monitoring data from the network. The first section describes the IPFIX protocol used for exporting the flows. The following two sections introduce two network probes the Flowmon probe and the nProbe. These two probes are evaluated for their ability to harvest the fields necessary for the creation of JA3 and JA3S hashes.

5.1 The IPFIX protocol

This chapter is going to discuss the IPFIX protocol and explain the meaning of a network flow. The source of information for this chapter were two RFC documents, [18] and [2], describing the IPFIX protocol.

A network flow is defined as a set of IP packets passing through an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties. In the standards defining network flows these properties are usually the following:

- **Source and destination IP address**: Identify the location of both the sender and the receiver of the traffic.
- **Source and destination port**: Identify the specific process or service running on the sender and receiver sides.
- **The protocol field**: Identifies the protocol of the transferred packets.

A packet is defined to belong to a flow if it completely satisfies all the defined properties of the flow. Meaning all packets with the same source and destination IP address, port numbers and the same protocol belong to the same flow. An *observation point* is a location in the network where packets can be observed (e.g. a router)

IPFIX is an IETF protocol for exporting flow information from routers, probes and other devices that are used by mediation systems, accounting systems and network management systems. This standard defines how the collected IP flow information should be formatted and transferred from an exporter to a collector. Previously, before the IPFIX protocol, many network systems were relying on the Netflow version 9 protocol developed by Cisco Systems. IPFIX is heavily based on the implementation of Netflow version 9. It is often identified as the Netflow version 10.

The protocol works as follows. A pool of metering processes collects data packets at one or more observation points. An exporter then gathers all records from the observation points together and sends them to a collector using the IPFIX protocol. Similarly as mentioned above, IPFIX considers a flow to be a sequence of packets sharing the same properties such as the source and destination IP addresses, port numbers and protocol.

The IPFIX is an important protocol used for network monitoring. This thesis uses it to collect flow traffic information and detect mobile applications in the traffic. The next section describes two probes used for exporting flow data and compares their abilities.

5.2 nProbe

The first flow traffic exporter software that is going to be discussed is the nProbe[1]. It is a software probe able to collect, analyze and export network traffic reports using the standard Netflow v5/v9 and IPFIX format. It is available for most platforms and operating systems. When installed on a PC, nProbe turns the PC into a network-aware monitoring appliance.

The first thing that needs to be discussed is the fields that are supported by nProbe to be exported using the IPFIX protocol. Table 5.1 shows the availability of the TLS fields required for the JA3 fingerprints to be created obtained from the official guide[2].

	TLS field	**nProbe field**	**Availability**
Client	TLS version	TLS_VERSION	YES
	Cipher suites	-	NO
	Extensions	-	NO
	Elliptic curves	-	NO
	Curve formats	-	NO
	SNI	TLS_SERVER_NAME	YES
Server	TLS version	TLS_VERSION	YES
	Cipher suite	TLS_CIPHER	YES
	Extensions	-	NO

Table 5.1: TLS fields supported by nProbe.

As the above table suggests, the support of the fields that nProbe provides is not sufficient. The only fields provided are the client and server TLS versions, the SNI and the server cipher suite. The abundance of the other fields results in the consequence that nProbe could not be used for this task efficiently. It does, however, allow users to develop custom plugins that could add the desired functionality.

The nProbe, in fact, offers its own implementation of the JA3 fingerprinting method. Both client and server fingerprints can be obtained from the provided flow using the *JA3C_HASH* and *JA3S_HASH* fields. However the method that creates the fingerprints does not exclude the GREASE values from the TLS fields. This makes the fingerprints unusable for applications that implement adding the GREASE values into their messages.

The conclusion is that nProbe is not suitable for the task of this thesis. It would be necessary to extend its functionality with a custom plugin parsing the desired TLS fields from the flow traffic. This would, however, exceed the scope of this thesis.

[1] https://www.ntop.org/guides/nprobe/

[2] https://www.ntop.org/guides/nprobe/cli_options.html

5.3 Flowmon Probe

The next probe to be discussed is the Flowmon Probe developed by the Flowmon Networks company based in Brno, Czech Republic[3]. Its main focus is on developing products for network performance monitoring and anomaly detection.

The Flowmon Probe is a powerful flow data exporter. It uses a non-intrusive approach to access the network traffic. As explained at the official website [14], the probe connects passively through a SPAN port or a network TAP. The SPAN (Switch Port Analyzer) is a designated port on a network appliance (switch) that is programmed to send a copy of all packets seen on one port or an entire VLAN to another port, where the packets can be analyzed.

On the other hand, a network TAP (Test Access Point) is a purpose-built hardware device that sits in a network segment, between two appliances (router, switch or firewall), and allows to access and monitor the network traffic that flows through it. A comparison of a SPAN and a TAP can be found at this [15] webpage.

Other characteristics of the Flowmon Probe include the following:[14]

- **Visibility into L2, L3/4, L7**: Collects information from L2, L3/4 and L7 layers.
- **Network Performance Monitoring**: Provides a wide collection of metrics to analyze the performance of a network.
- **Encrypted traffic analysis**: Collects network traffic metadata in the IPFIX format using probes and enriches it with TLS protocol information.
- **Application recognition and usage data**: Reports on the use of applications by analyzing L7 packets.

An important feature that the Flowmon probe provides is the encrypted traffic analysis. The probe enriches the collected traffic with TLS metadata that is necessary to make TLS fingerprinting possible. Table 5.2 shows all the TLS fields supported by the Flowmon probe[4].

	TLS field	Flowmon field	Availability
Client	TLS version	FLOWMON_TLS_CLIENT_VERSION	YES
	Cipher suites	FLOWMON_TLS_CIPHER_SUITES	YES
	Extensions	FLOWMON_TLS_EXTENSION_TYPES	YES
	Elliptic curves	FLOWMON_TLS_ELLIPTIC_CURVES	YES
	Curve formats	FLOWMON_TLS_EC_POINT_FORMATS	YES
	SNI	FLOWMON_TLS_SNI	YES
Server	TLS version	FLOWMON_TLS_SERVER_VERSION	YES
	Cipher suite	FLOWMON_TLS_CIPHER_SUITE	YES
	Extensions	-	NO

Table 5.2: TLS fields supported by Flowmon.

As Table 5.2 shows, the probe provides all fields necessary for the creation of JA3 hashes. The probe is in fact able to generate the JA3 hash by itself. This build-in feature is available

[3] https://www.flowmon.com/cs

[4] https://support.kemptechnologies.com/hc/en-us/articles/4405949707789-Flow-standards-specifications

under the *FLOWMON_TLS_JA3_FINGERPRINT* field. Unfortunately, the probe does not provide any support for the server extensions field. The JA3S hash therefore cannot be created as described in section 3.4.

This chapter introduced two potential flow exporters that could be used to obtain input flows for the classifier. After analysing the capabilities of both probes it was decided that Flowmon probe will be the flow exporter used in this thesis as it supports most of the TLS fields that are necessary to create the fingerprints. The next section describes the flows produced by the probe and provides an analysis of their contents.

Chapter 6

Analyzing Flowmon flows

This chapter offers an in-depth analysis of the flows obtained from the Flowmon probe. These flows are exported from the traffic using the IPFIX protocol and are stored inside *csv* files.

To perform the analysis it is first needed to capture the communication between an application and its server. The traffic was captured using Wireshark and saved as a *pcap* file. This file served as a reference point to which the created *csv* file was compared to, as it holds the actual packets that were transported.

There are two ways to obtain the *csv* file with the captured flows. The first one is to route the traffic through the virtual machine where the Flowmon probe is installed. The probe listens at a specific port analyzing all the traffic coming through it. The exported flows can be then downloaded from the web interface of the Flowmon collector.

The second option is to use a command line tool available at the Flowmon virtual machine called *flowmonexp5*. This tool can be supplied with a *pcap* file that is then processed and exported to a *csv* file while parsing all the desired fields. To parse a *csv* file with the *flowmonexp5* tool, the following command is used:

```
flowmonexp5 -I pcap-replay:file=input.pcap -E csv: -P \
            tls:fields=MAIN#CLIENT#JA3 > output.csv
```

where -I specifies the capture and process options to use a *pcap* file as input, -E specifies the export options to export the flows as a *csv* file and -P specifies the plugins to be used to process the input. Using this command the exported *csv* file will contain all the necessary TLS fields described in Table 5.2. One of the fields is the JA3 fingerprint generated by the Flowmon exporter. The exported flows, however, need to be further processed before passing them to the classifier. The processing is performed by a class called FlowParser described in Chapter 8. Figure 6.1 provides a graphical illustration of how the input flows are created. The capturing process is the same as described in section 4. The captured *pcap* file is passed to the *flowmonexp5* tool that creates a *csv* file with the exported flows. The *csv* file is then passed to the FlowParser class that performs the necessary preprocessing of the flows. After that, the flows can be passed to the classifier.

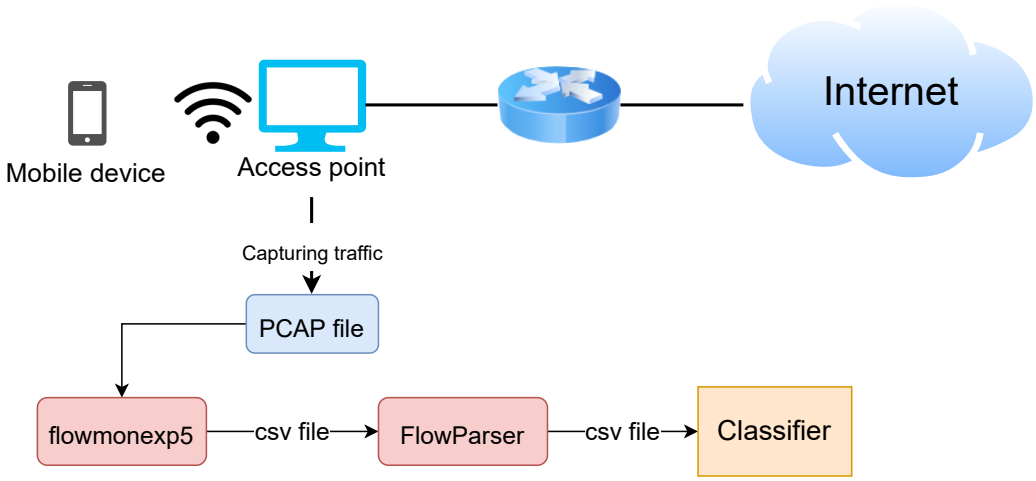

Figure 6.1: Processing of the input flows.

The following part of this section describes the preprocessing necessary to prepare the flows for the classifier. First, several fields in the *csv* files created by *flowmonexp5* have to be converted from a hexadecimal to a decimal format. These fields include the cipher suites, elliptic curves, elliptic curve formats, and other TLS extensions. The fields have to be converted to a decimal format as it is used by the JA3 method to create the fingerprints. Figure 6.2 describes the process of converting the cipher suites from hexadecimal to decimal format.

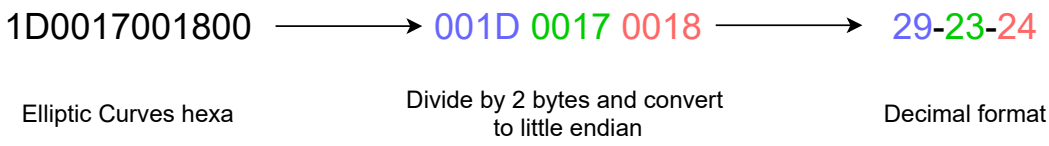

Figure 6.2: Converting elliptic curves to a decimal format.

The hexadecimal string in the elliptic curves TLS extension consists of several segments, each describing one key exchange algorithm that is supported by the client. Each of these segments is exactly two bytes long. The virtual machine where the Flowmon probe is running uses a big-endian order of bytes. Therefore the two bytes within each segment have to be switched before converting the segment to its decimal value. For example, as RFC [26] describes, the 0x0018 hexadecimal value represents the 384-bit prime field Weierstrass curve, also known as *secp384r1*. Finally, all the decimal values are concatenated together using a dash symbol to delimit each value.

Another field that needs to be converted is the TLS extensions field. This field, just like the elliptic curves, consists of two-byte segments and so the exact same process repeats. Moreover, one more operation that needs to be performed during these conversions is excluding the GREASE values. Figure 6.3 shows the GREASE values that are excluded from TLS extensions.

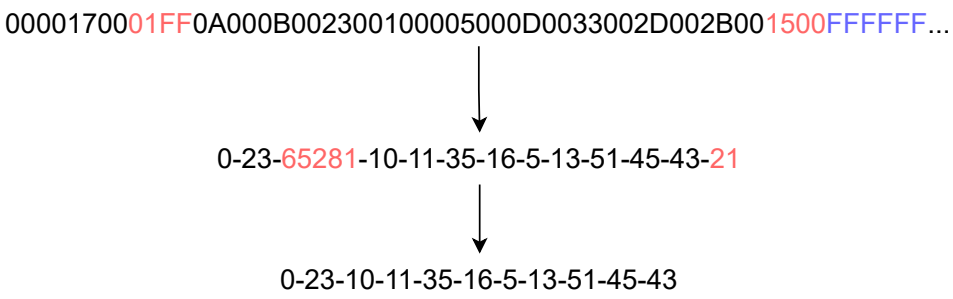

Figure 6.3: Excluding the GREASE and padding values from TLS extensions.

The extraction is performed using a list containing all the values reserved for GREASE. Whenever one of these values occurs in the string it is excluded. From the above example in Figure 6.3 the GREASE value is 65281. Number 21 indicates that everything that follows until the end of the string are padding values. The padding is also removed from the extensions.

The following part of this section describes the problems that emerged from using the Flowmon probe to create JA3 fingerprints.

Flowmon probe JA3 fingerprints

The Flowmon probe allows to create JA3 fingerprints using the *flowmonexp5* tool. As the official Flowmon website states the JA3 method *"combines these five parameters of TLS communication: version, ciphers, extensions, elliptic curves and its formats and produces an MD5 hash."*[13]

According to this explanation the generated fingerprints should be identical to those in the application dataset described in Section 4. After an attempt to recreate the fingerprints, however, the resulting hashes were different from those created by the Flowmon probe. This was caused by the fact that the Flowmon probe does not exclude GREASE values when creating the fingerprints. The GREASE values, however, are not used in the application dataset. Therefore, the JA3 fingerprints created by the Flowmon probe cannot be used.

Flowmon probe cipher suites

After some examination, it was discovered that Flowmon only takes the first eight cipher suites from the TLS *Client Hello Message*. As the specification [1] of the Flowmon collector 11.1 states, the *TLS_CIPHER_SUITES* field supports the input length of only sixteen bytes. This proves the above mentioned discovery as one TLS cipher suite takes two bytes resulting in only eight cipher suites parsed by the Flowmon collector. Figure 6.4 shows the comparison of the cipher suites retrieved by Flowmon and those found in a *pcap* file.

[1] See https://support.kemptechnologies.com/hc/en-us/articles/4405949707789-Flow-standards-specifications

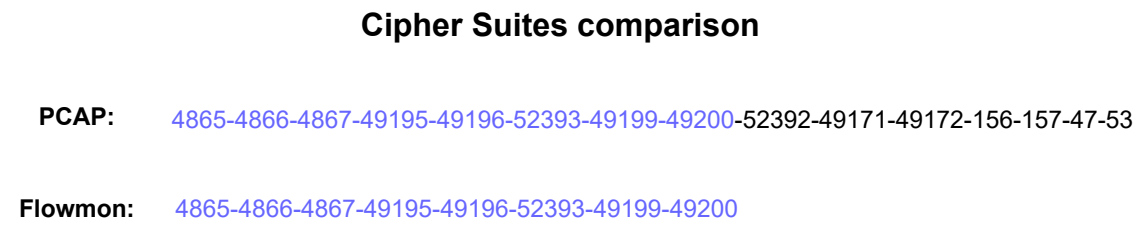

Figure 6.4: Comparison of cipher suites parsed by the Flowmon probe and obtained from a *pcap* file.

The fact that Flowmon does not parse all the cipher suites needs to be mirrored in the creation of the application dataset so that it is possible to receive identical hashes. Section 6.1 describes how this change affected the application dataset.

Flowmon probe client TLS version

Another problem that was discovered in the Flowmon probe was the client TLS version in the *csv* files. The JA3 method uses the client TLS version to create the fingerprints. The client TLS version is obtained from the *Client Hello Message*. It was discovered that to obtain the TLS version, Flowmon uses the *supported_versions* TLS extension. This extension contains all the TLS versions that are supported by the client. The problem is that the Flowmon probe takes only the first supported version from the list without excluding the GREASE values. The TLS client version in the *csv* files, therefore, often contains GREASE values instead of the actual TLS version. The TLS version extracted by the probe is illustrated in Figure 6.5. The next section describes how the encountered problems affected the application dataset.

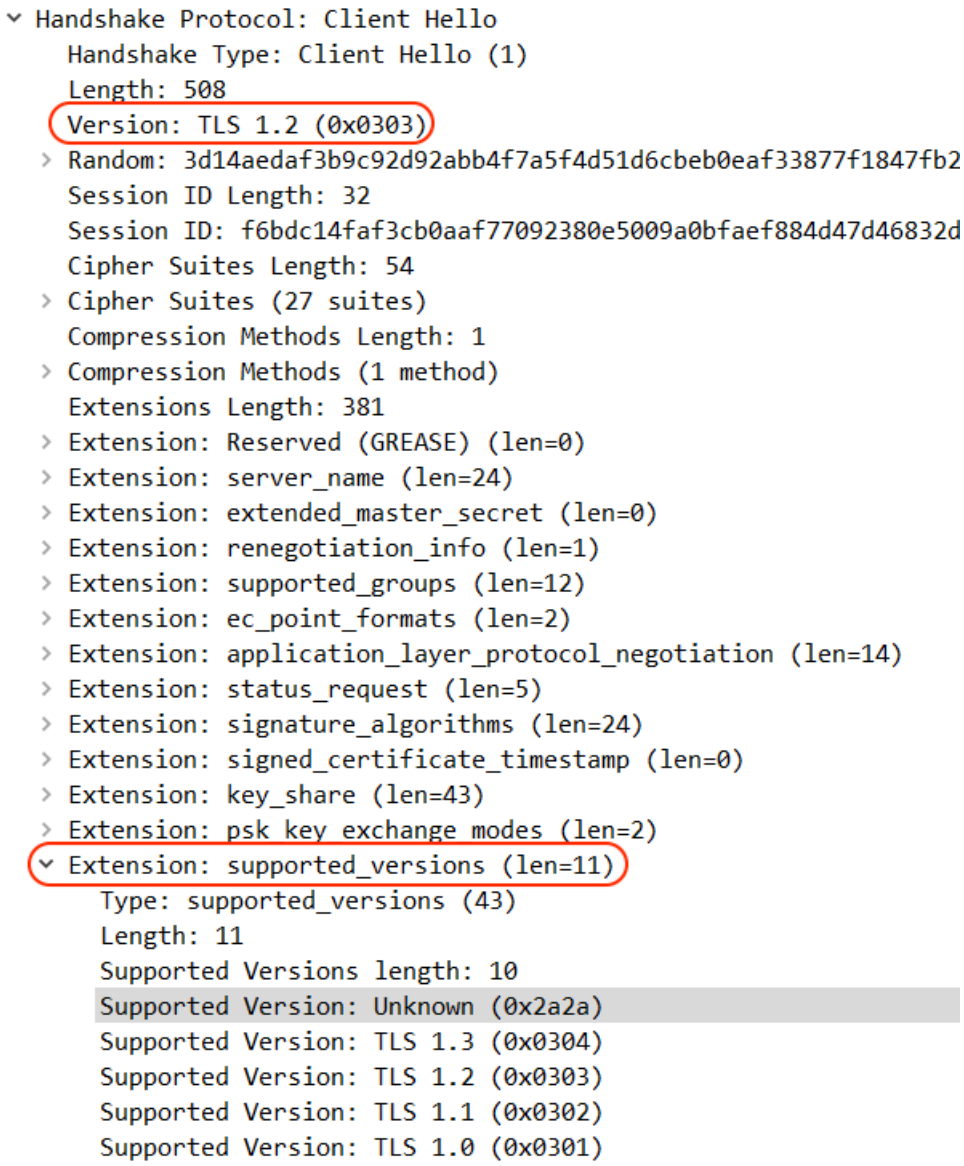

Figure 6.5: TLS version extracted by the Flowmon probe.

6.1 Updating the application dataset

After analysing the Flowmon records it is necessary to update the application dataset in order to make it possible to work with the probe. The changes that need to be applied relate to the missing TLS versions and cipher suites in the *csv* files.

The JA3 fingerprints in the application dataset are created using the method described in section 3.3. The *csv* files created by the Flowmon probe are, however, missing the correct TLS version and also contain only the first eight cipher suites. In order to be able to compare the fingerprints in the application dataset with those created from the *csv* files, both fingerprints have to be constructed using the same values.

The application dataset was therefore updated to use only the following fields to create the JA3 fingerprints:

- First eight supported cipher suites
- List of extensions
- Elliptic curves
- Elliptic curves formats

Figure 6.6 shows a comparison of the values used originally and after updating the dataset.

Figure 6.6: The TLS fields used to create the JA3 fingerprints before and after update.

Losing the information about the TLS version and some of the cipher suites may potentionaly result in decreasing the variability of the created fingerprints. It is possible that it would, therefore, decrease the number of unique JA3 fingerprints in the dataset. Table 4.5 shows the results of comparing the number of unique fingerprints in the original and the updated dataset for each application with the intention to show how this changes affect the dataset.

Application	**Unique JA3 hash before update**	**Unique JA3 hash after update**
airbnb	YES	NO
booking.com	NO	NO
couchsurfing	NO	NO
crypto.com	NO	NO
duolingo	NO	NO
equabank	NO	NO
facebook	YES	YES
george	YES	YES
livesport	NO	NO
pinterest	NO	NO
quora	NO	NO
shazam	NO	NO
teams	YES	YES
tiktok	YES	YES
uber	NO	NO
whatsapp	YES	YES
youtube	NO	NO
zalando	NO	NO

Table 6.1: Unique hashes for applications before and after updating the dataset.

As described in Table 6.1, even before this change only six applications out of the eighteen in the dataset had a unique JA3 fingerprint that allowed them to be distinguished from

other applications. After applying the changes, this number has dropped to five which does not introduce a big downgrade. However, as it is not possible to create JA3S fingerprints from the Flowmon records the ability to distinguish our applications is limited to the JA3 hash. This decreases the final number of distinguishable applications from thirteen to just five. It is therefore necessary to boost the classification using other information, e.g., the SNI values.

This chapter provided an analysis of the *csv* files with flows created by the Flowmon probe. The preprocessing necessary to prepare the flows for the classifier was also discussed. As it was already mentioned, the abilities of the Flowmon probe do not enable the creation and usage of JA3S fingerprints. Therefore, only the JA3 fingerprints are used in this thesis. The fingerprints have to be constructed from the fields inside the exported *csv* files. After encountering several problems the original JA3 method described in section 3.3 had to be updated. The new method only uses the first 8 cipher suites and does not use the client TLS version to create the fingerprints.

Chapter 7

Proposed method

This chapter presents the methods and algorithms used to perform the classification. In total, four different classification methods are discussed. Figure 7.1 provides a graphical illustration of how the classifier works.

The input of the classifier is a *csv* file with network flows created by the Flowmon exporter. The classifier uses the application dataset to compare values extracted from the input flow with values in the dataset to perform classification. Finally, the output of the classifier is an ordered list of tuples containing the names of applications and their scores, as illustrated in Figure 8.3.

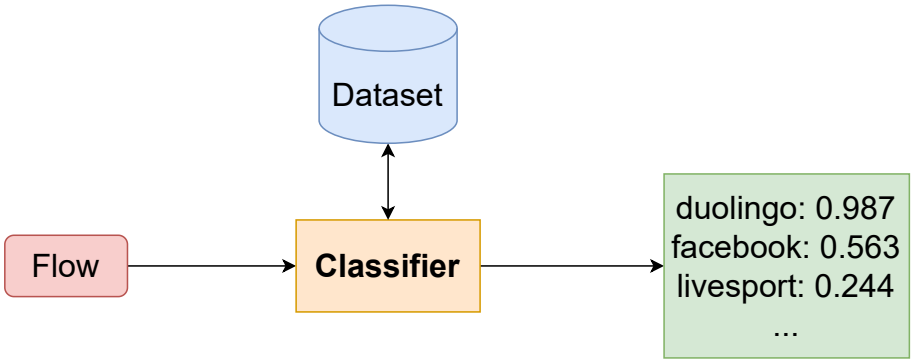

Figure 7.1: Input and output of the classifier.

The following sections provide a detailed explanation of the implemented classification methods. The first section describes the method using JA3 fingerprints. The following section presents another method that uses a string of values extracted from TLS handshakes to perform the classification. The last two methods use the Server Name Indication (SNI) TLS extension field to classify the input flows. All these methods can be used individually or can be combined to ensure the best results for the classifier. This option is also discussed at the end of this chapter. More information about the implementation of the classifier can be found in Chapter 8. The evaluation of each method is further discussed in Chapter 9.

7.1 Classification using JA3 fingerprints

The first method that was implemented in the classifier uses the already discussed TLS fingerprinting technique. The main principle of this technique is relying on the fact that certain values exchanged during the TLS handshake could be used to identify the client and server applications. These values are actually depending on the modules and libraries that the application is using and therefore provide a rather stable and valuable method of identification. The principle of this method was discussed in Chapter 3.3. To use the JA3 method together with the Flowmon probe, the process of creating the fingerprints had to be modified. The reasons for this modification were discussed in Chapter 6.

First, the classifier computes the JA3 fingerprint of an input flow. After that, the classifier compares the obtained fingerprint with all unique fingerprints inside the application dataset searching for an exact match. If the exact match is found the classifier looks at the number of applications that exactly matched the fingerprint. If only one application matches the fingerprint, the score for the application is 1 and all the other applications have a score equal to 0. If the fingerprint matches more applications, the score for each of these applications is computed as 1 divided by the number of matches. The score of an application A and input flow f is computed as:

$$score(A, f) = \begin{cases} h_f \notin H_A, & 0 \\ h_f \in H_A, & \frac{1}{n} \end{cases} \tag{7.1}$$

where h_f is a JA3 hash created from the input flow, H_A is a set of all hashes in the application dataset related to application A and n is the number of applications in the dataset that contain an exact match with flow f.

Figure 7.2 provides a graphical illustration of the JA3 classification method.

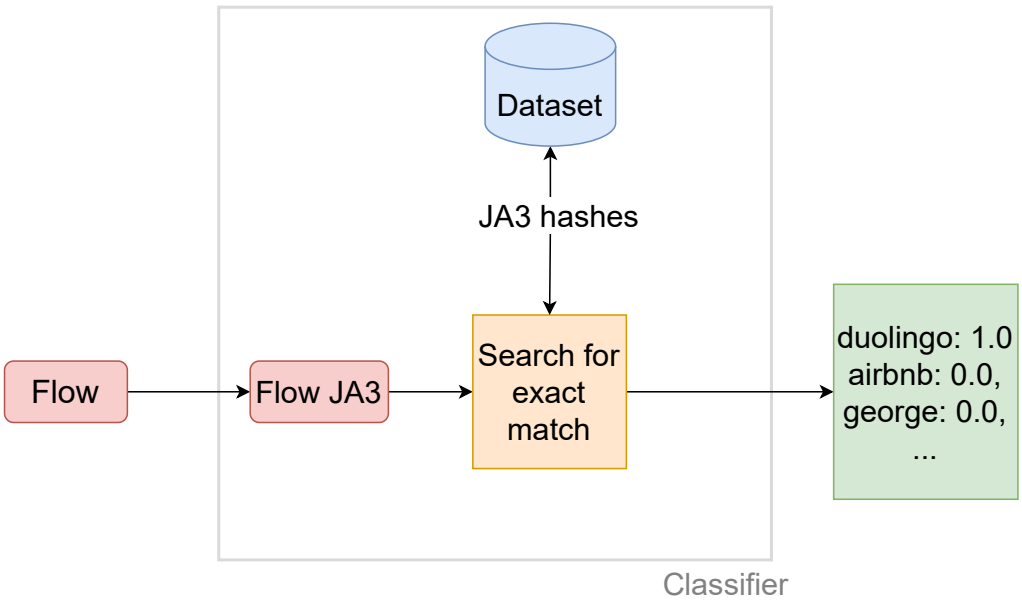

Figure 7.2: Diagram of the JA3 classification method.

7.2 Classification using TLS values

Another method of classification implemented in this thesis is the TLS string classification. This method is very similar to the previous one using JA3 fingerprints in that it uses the same values from which the fingerprints are created. Those are the values extracted from a TLS handshake *Client Hello Message*. The difference is that this method does not use the MD5 hash function to create the fingerprints.

This method also does not search for an exact match. Instead, it takes the constructed string and compares it to all unique strings in the application dataset searching for the most similar. The similarity is computed using the longest matching substring as shown in Figure 7.3.

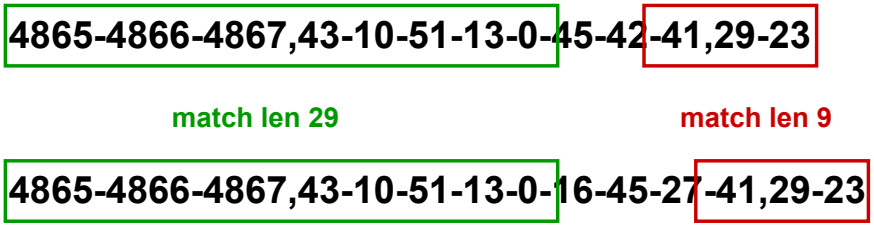

Figure 7.3: Searching for the longest match between two strings.

Once the classifier finishes comparing all the strings it returns the result of the classification which is an ordered list of tuples containing the names of the applications with the computed score.

The score is computed as the length of the longest match between a TLS string from the application dataset and the input flow divided by the length of the whole string. Therefore, if an application has the exact same string as the input flow, the computed score is 1. If the match is equal to only a half of the string the score is 0.5, etc. The score for an application A and an input flow f is computed as follows:

$$score(A, f) = \frac{\max_{t \in T_A} len(longest\_match(t, t_f))}{len(t_f)} \tag{7.2}$$

where T_A is a set of all TLS strings in the application dataset corresponding to application A, *len* is a function returning the length of the string that is passed to it, *longest_match* is a function returning the longest common substring found between the two compared strings and t_f is the TLS string constructed from the input flow.

As already mentioned, this method is very similar to the method using JA3 fingerprints. The TLS string used in the comparison is the same that is used for the fingerprints. This, of course, means that the number of unique strings is the same as the number of unique fingerprints. This method, therefore, does not introduce any additional value or information that the classifier could benefit from.

The main reason to implement this method, however, is the fact that it searches for similarities. As the method searches for the longest match and not for an exact match it should be able to identify an application even in case of a change in the cipher suites used by the application. This could happen for example after an update of the application. Figure 7.4 shows a diagram of the TLS values method.

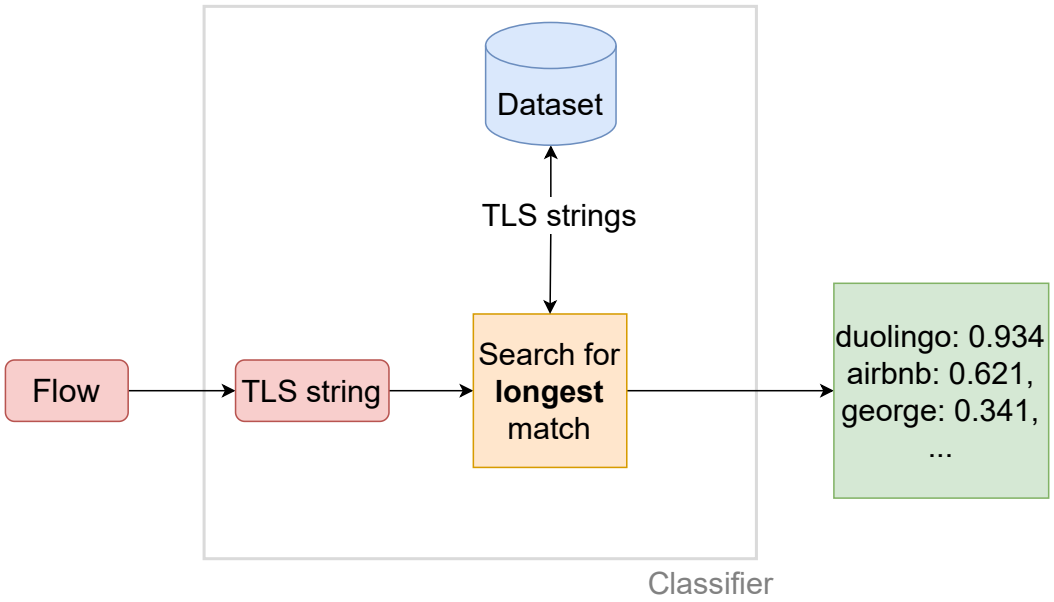

Figure 7.4: Classification using TLS values.

7.3 Classification using SNIs

The last classification method implemented in this thesis is the SNI classification. It uses the Server Name Indication (SNI) field extracted from the *Client Hello Message*. The source of information for this section is the Transport Layer Security (TLS) Extensions RFC [7] and the Transport Layer Security (TLS) Extensions: Extension Definitions RFC [12].

The SNI field is an extension to the TLS protocol. It is used by the client to indicate which server it is attempting to connect to during the handshaking process.

With HTTP protocol this value would be specified inside the HTTP headers. When using encryption and the HTTPS protocol, however, the server needs to be able to know this value without the need to decrypt the packet. It needs to be accessible unencrypted in order to allow multiple certificates to be presented by the server at the same IP address.

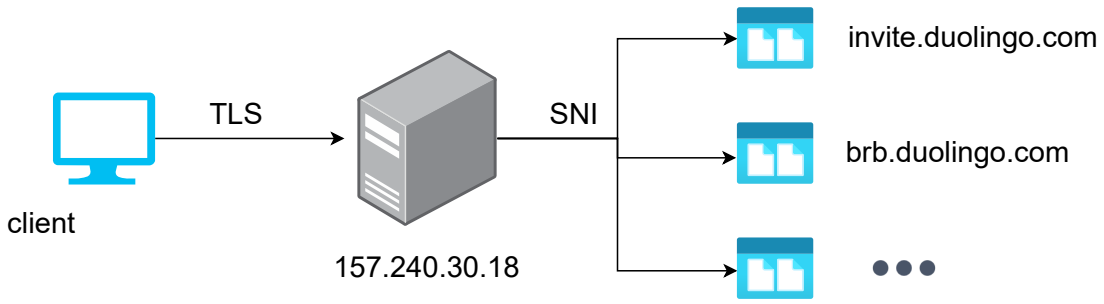

Figure 7.5: SNI allows multiple HTTPS websites to be served at the same IP address

Comparison of an SNI of an input flow with SNIs stored in the application dataset should result in a successful classification even in cases when the SNI of the flow is not in the dataset as it usually contains words specific for that application (name of the application, names of servers, etc.).

Application	SNIs
Airbnb	api.airbnb.com, www.airbnb.com, ...
Booking.com	account.booking.com, experiences.booking.com, ...
Duolingo	brb.duolingo.com, goals-api.duolingo.com, ...

Table 7.1: Example of SNIs and their applications.

Table 7.1 shows examples of the SNIs corresponding to certain applications in the application dataset. As the table shows, most of the SNIs contain some keywords that are typical for each application. For example, the name of the application typically occurs in the SNI field.

Figure 7.6 shows a graphical interpretation of the SNI classifier. Two different approaches that use SNIs were implemented - the Jaccard similarity index and the TF-IDF method.

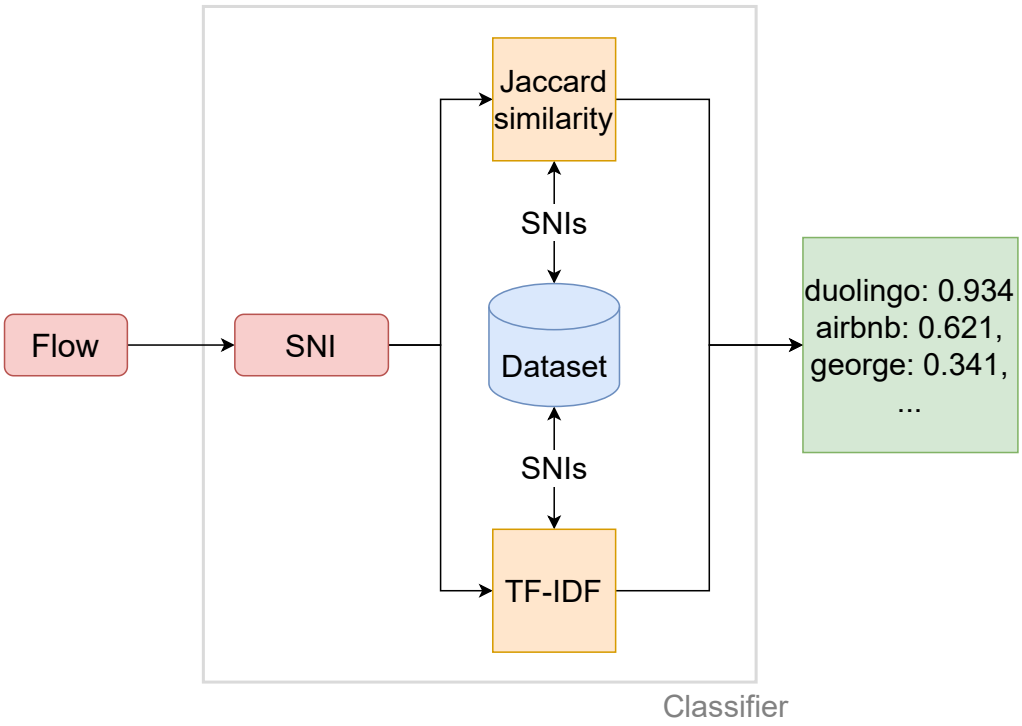

Figure 7.6: Classification using the SNI values.

7.3.1 Classification using Jaccard similarity

The first method that is used to implement classification of SNIs is the Jaccard similarity index developed by Paul Jaccard[19]. The Jaccard index, also known as the Jaccard similarity coefficient, is a statistical method that measures the similarity and diversity of two finite sample sets. It defines the similarity as the size of the intersection divided by the size of the union of the two sample sets. This method is described by Equation 7.3:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{7.3}$$

where A, B are finite sets of values. Note that the value of the Jaccard similarity coefficient can be from interval $\langle 0; 1 \rangle$.

The more values two different sets have in common, the more similar they are. The classifier applies this method on SNIs. To compute the similarity of two SNIs, the SNIs first need to be transformed into sets. To do so each SNI is split by delimiters.

The delimiters are defined in the implementation of the classifier and include special characters that are usually used to delimit or separate the words within SNIs (a dot, comma, dash etc.).

The words that remain after the split are then used to create the set representing the SNI. After that, two different sets of words are compared using the Jaccard similarity as described in Equation 7.3. The transformation of an SNI into a set is also illustrated in Figure 7.7.

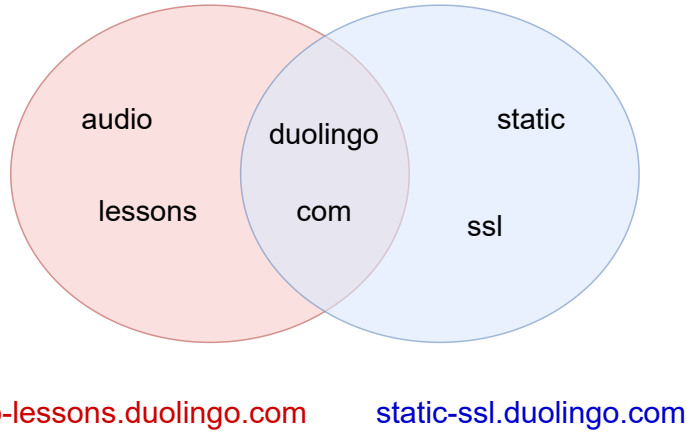

Figure 7.7: Transforming SNIs into sets for the Jaccard similarity index.

In this particular example depicted in Figure 7.7, the SNIs are split by dots and dashes resulting in two sets containing four words. The Jaccard similarity between the two sets is computed as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{6} = \frac{1}{3} \tag{7.4}$$

To use this method for the classification of an input flow it is used as follows. First, the SNI is extracted from the input flow and transformed into a set. After that, the set is compared with all SNI sets in the application dataset.

The final score for each application is computed as the maximum of all the Jaccard similarities between the set of the flow and all SNI sets of the application. The score for application A and input flow f is computed as follows:

$$score(A, f) = \max_{s \in S_A} J(s, s_f) \tag{7.5}$$

where S_A is a set of SNIs in the application dataset related to application A, and s_f is the SNI extracted from the input flow. The output of the classifier is an ordered list of tuples containing applications and the computed score.

7.3.2 Classification using TF-IDF

The last approach that was implemented for the classification using SNIs is the term frequency-inverse document frequency (TF-IDF) method. TF-IDF is a numerical statistics whose main intention is to reflect the importance of a word to a document given a collection of documents. As stated in [27], the TF-IDF method and its variations are often used in various information retrieval tasks such as text mining, text summarization and classification, user modelling, etc. This [8] survey conducted in 2015 also showed that TF-IDF is used by 83% of text-based recommender systems in digital libraries.

In order to be able to fully understand the method, it is necessary to understand the terminology that it uses.

- Term = word
- Document = collection of words
- Corpus = collection of documents

To compute how important a word t is with respect to a document d, the TF-IDF method uses two separate statistics: Term frequency and Inverse document frequency.

Term frequency

The first form of term frequency was developed by Hans Peter Luhn (1957)[22]. Term frequency determines the relative frequency of term t in a document d. It is proportional to the number of times the term occurs in the document. That means the more times term t occurs in a document d the bigger the resulting term frequency of t is. The term frequency of word t in document d is defined as[24]:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{7.6}$$

where $f_{t,d}$ is the number of times term t occurred in document d, and the sum is essentially the number of all terms in document d.

The term frequency alone could be directly used to determine the importance of a term to a document. This method, however, has a small weakness that reduces its capabilities. It gives bigger values to words commonly used in the language. In English, for example, to words like: “the”, “a”, “be”, “to”, “of”, etc. These words do not have any importance in defining documents, because they do not describe their content. Therefore the inverse document frequency (IDF) statistics is used together with the term frequency.

Inverse document frequency

The cornerstone of inverse document frequency was laid by Karen Spärck Jones (1972)[34]. The inverse document frequency is used to define how much information a word provides. That means whether it is a commonly used word or it is rare across the documents. The inverse document frequency of term t in a corpus D is computed as a logarithmically scaled fraction of the total number of documents divided by the number of documents that contain term t. The inverse document frequency for term t and corpus D is defined as[24]:

$$idf(t,D) = \log \frac{N}{|d \in D : t \in d|} \tag{7.7}$$

where D is the corpus, N is the number of documents in the corpus ($N = |D|$) and the denominator represents the number of documents from corpus D where term t appears (meaning that $tf(t, d) \neq 0$). If the term t is not present in any document across the corpus, the above equation results in a division by zero. It is therefore a common practice to adjust the denominator to $1 + |d \in D : t \in d|$.

If a word appears in every document, its IDF value is 0. That is because such a word does not have any special value for a single document and therefore does not represent the uniqueness of a document. On the other hand, the fewer documents term t occurs in, the bigger significance it has.

Term frequency-inverse document frequency

The term frequency-inverse document frequency metric is defined as the product of the two statistics: term frequency and inverse document frequency. The TF-IDF of term t for document d given a corpus of documents D is therefore defined as[24]:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{7.8}$$

The TF-IDF metric gives higher weight to a term that has higher frequency in a given document and is not present in other documents. Such term is expected to be of greater importance for the given document. The value of TF-IDF is always greater than or equal to 0. The higher the value the more significant the given term is.

To apply the TF-IDF method for SNI classification it is necessary to identify the connections and similarities between these two approaches and redefine the terminology of the TF-IDF method to fit it to this specific use case.

The input of the classifier is an SNI string. The string is split by delimiters into separate words as described in Section 7.3.1 and illustrated in Figure 7.8.

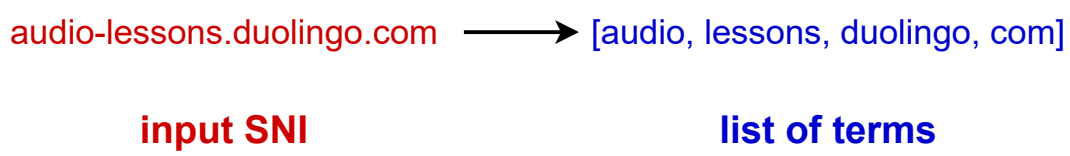

Figure 7.8: Converting an SNI into a list of terms.

After that, the SNI is represented as a list of words. Each word in this list can be perceived as a term in the TF-IDF method. Finding the correct application for the input SNI can now be addressed as finding a document that is, given the SNI as a query, most relevant to that query.

In this interpretation, each document represents one application and consists of all words from SNIs of that application that are stored in the application dataset. The entities of the TF-IDF method are therefore redefined as:

- Term = a word from the SNI string
- Document = a list of words from SNIs belonging to a given application
- Corpus = a set of all application documents

Figure 7.9 illustrates how the SNIs of an application are transformed into a document that represents the application.

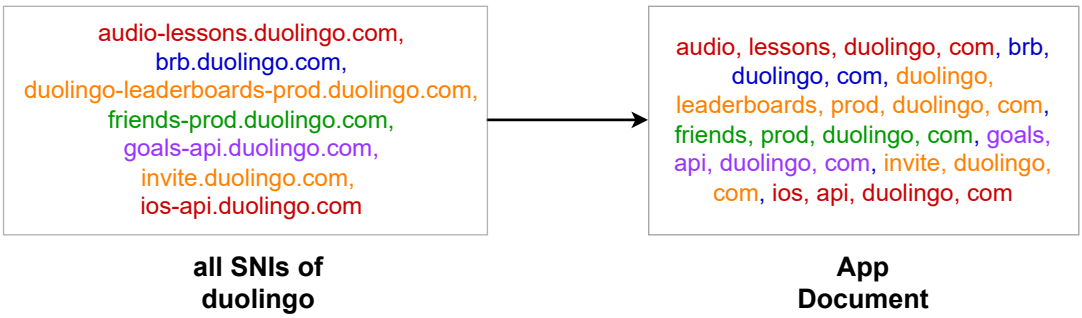

Figure 7.9: Converting SNIs of Duolingo application into a document.

All SNIs are split and concatenated together to create a big list of terms. To create the corpus, all applications in the application dataset are transformed in this way. It is then possible to compute a TF-IDF value of each term in an SNI against a document. Table 7.2 illustrates the TF-IDF values computed for the Duolingo application.

Term	TF-IDF value
duolingo	2.2512
leaderboards	1.2663
audio	1.1959
brb	1.1959
friends	1.1959
goals	1.1959
invite	1.1959
ios	1.1959
lessons	1.1959
prod	1.0225
api	0.5928
com	0.2767

Table 7.2: The TF-IDF value of certain words for the Duolingo application.

Table 7.2 shows normalized TF-IDF values of some words for the Duolingo application ordered from the biggest (most significant word) to the smallest (least significant). It is obvious that the term "duolingo" is the most significant to the application as it appears many times in the SNIs and does not appear in SNIs of other applications. On the other hand, terms like "api" and "com" do not have a big significance even though they appear many times in the SNIs. That is because they commonly appear in the SNIs of other applications across the corpus.

The TF-IDF method can be used to compute the significance of a given word to a document of SNIs belonging to a certain application. This approach was applied to compute the score of an input flow. The score of input flow f and an application A is computed as a sum of the TF-IDF values for each word obtained from the SNI of the input flow. The computation of the score is described by the following equation:

$$score(A, f) = \sum_{t \in s_f} tfidf(t, d_A, D) \tag{7.9}$$

where s_f is the SNI of the input flow f, t is a term, d_A is the document created for application A, and D is the corpus of all application documents. Figure 7.10 provides a graphical illustration of the computation. The output of the classifier is an ordered list of tuples containing applications and the computed scores.

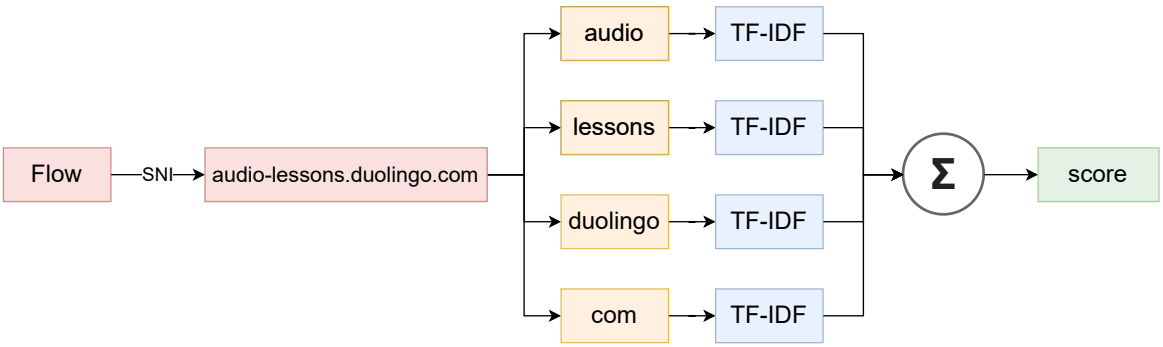

Figure 7.10: Computing the score using the TF-IDF method.

7.4 Combining the classifiers

The last classification method implemented in the classifier combines all the above-mentioned approaches and computes the score of an input flow as a sum of the scores computed by the individual methods. The score of input flow f for an application A is computed as:

$$score(A, f) = \sum_{c \in C} c(A, f) \tag{7.10}$$

where C is a set of all the implemented classifiers. The output of the final classifier is an ordered list of tuples containing applications and the computed scores.

This chapter provided an overview of the classification methods used to classify input flows. These methods use different approaches to perform the classification, in particular: exact matching of JA3 fingerprints, searching for the longest match of a string of TLS values, applying Jaccard similarity for SNIs, applying the TF-IDF method for SNIs and a combination of the approaches. The next chapter presents the most important implementation details including several parsers, the classifier, the application dataset, etc.

Chapter 8

Implementation

This chapter gives a comprehensive description of the work completed in the course of this thesis, including the implementation of classification methods, parsers, evaluation tools, etc. All classification models are constructed using the object-oriented programming paradigm and implemented in Python version 3.9. Several Python libraries used for data science were also used throughout this work, such as numpy[1], scipy[2], pandas[3], etc. Figure 8.1 provides a class diagram of the most important classes.

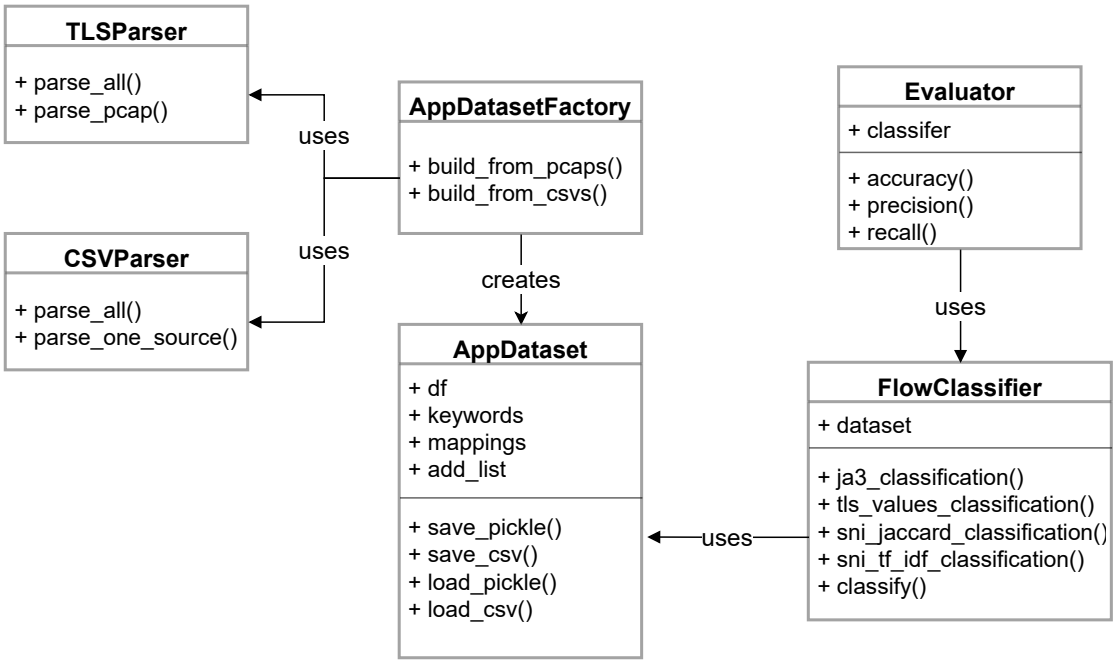

Figure 8.1: Class diagram of the implemented classes.

Section 8.1 starts by describing the parsers created to construct a dataset of applications from various input sources. Section 8.2 describes the classes used to represent the application dataset itself. After that, a class used for parsing the *csv* files containing the input flows created by the Flowmon probe is covered in Section 8.3. Section 8.4 provides

[1] https://numpy.org/
[2] https://scipy.org/
[3] https://pandas.pydata.org/

a thorough view into the implementation details of all the classification methods that were explained in Chapter 7. Finally, Section 8.5 describes the Evaluator class that implements several metrics used to evaluate the classifier.

8.1 Dataset Parsers

The first step in the implementation phase of this thesis was to create the application dataset containing the selected applications and their information. For this purpose, two parser classes were created.

CSVParser

The first class is the CSVParser that is implemented inside the csv_parser.py file. This parser is intended to be used together with a Perl script that was created as part of the project Integrated platform for analysis of digital data from security incidents (Tarzan).[4] This script receives a *pcap* file as input and creates JA3 and JA3S fingerprints by extracting the underlying TLS communication. The output of this script is a *csv* file containing the created fingerprints. The list of advertising servers used to exclude the servers from traffic was also obtained from the Tarzan project.

The CSVParser takes this file and performs the preprocessing necessary to transform the data into a form that suits the needs of the classifier. This includes excluding empty fields, filtering by SNI, dropping traffic created by advertising servers, adding new columns with the string of TLS values, version of the application, etc. The DataFrame data class from the pandas library is used to represent the dataset. The preprocessing is illustrated in Figure 8.2.

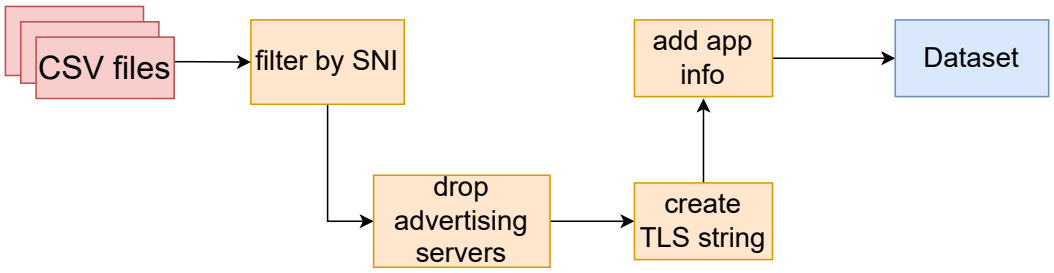

Figure 8.2: The preprocessing performed by the CSVParser.

Columns of the resulting dataset are shown in Table 4.3. The CSVParser provides the following public methods:

- `parse_all(folderpath, keywords, sep)`: This method is used to parse all *csv* files located in the folder. The first parameter indicates a path to the folder, the second parameter is a dictionary containing special keywords for each application and the last parameter is the separator used inside the *csv* files to separate each value.
- `parse_one_source(filepath, config_file, sep)`: This method parses the application data from one *csv* file that contains data of all applications. The first parameter

[4] https://github.com/matousp/ja3s-fingerprinting

is a path to the file, the second parameter indicates the path to a configuration file that contains metadata about each application, and the last parameter is the separator.

The `parse_all` method parses all files in the folder. Each file in the folder needs to have a name in the following format:

`app-name_app-version_os-version_.csv`

meaning that a *csv* file containing the fingerprints of the Duolingo application version 6.139.0 running on a device with the ios-15.0.2 operating system would be called:

`duolingo_6.139.0_ios-15.0.2_.csv`

The keywords parameter is a dictionary containing special keywords for each application. These keywords are used to filter only communication that corresponds to the correct application. The keys in this dictionary are the application names and the value for each key is a list of the keywords. Usually, the list of keywords for an application contains the name of the application and some other keywords that were discovered by manual inspection of the traffic.

The `parse_one_source` method, on the other hand, uses one *csv* file containing the communication of all applications. The name of this file does not have to follow any special format. The method, however, needs to receive a configuration file in JSON format containing metadata of the applications. The file specifies for each application the name of the application, version of the application, version of the operating system and the keywords.

TLSParser

The second parser class is the TLSParser. The TLSParser replaces the Perl script and creates the fingerprints directly from *pcap* files. It uses the *Scapy*[5] Python library to extract the necessary TLS fields from packets. *Scapy* is a powerful tool for interactive packet manipulation running on top of the *libpcap* library. The TLSParser provides the following public methods:

- `parse_all(folderpath, keywords)`: This method is used to parse all *pcap* files inside the folder indicated by the folderpath parameter. The second parameter is a dictionary with a list of keywords for each application.
- `parse_pcap(path, keywords)`: This method can be used to parse one *pcap* file. The first parameter is a path to the PCAP file and the second parameter is a dictionary with keywords.

For both methods the names of the files to be parsed need to follow the format defined in the CSVParser section. Both parsers are intended to be used only by the AppDataset-Factory class to create an AppDataset object representing the application dataset.

8.2 Dataset

This section introduces the classes used to work with the application dataset. There are two classes that serve this purpose. The first one is a factory class called AppDatasetFactory that creates the dataset. The second class called AppDataset represents the dataset itself.

[5] https://scapy.net/

AppDatasetFactory

The AppDatasetFactory class follows the factory design pattern. The purpose of this class is therefore to create an instance of the AppDataset class. The dataset is intended to be constructed only in this way. The AppDatasetFactory class provides the following methods:

- `build_from_pcaps(folder_path, keywords_path, addlist_path)`: A static method that is used to create the dataset from *pcap* files. It uses the TLSParser presented in the previous section. The first parameter is a path to the folder containing *pcap* files that the dataset is created from. The second parameter is a path to the file with the keywords in JSON format. The last parameter is a path to the file containing names of advertising servers that are excluded from the dataset.
- `build_from_csvs(folder_path, keywords_path, sep)`: A static method that creates the dataset from *csv* files created by the Perl script. It uses the CSVParser to parse the files. The first parameter is a path to the folder with *csv* files, the second parameter is a path to the file with keywords in JSON format and the last parameter is a separator used inside the *csv* files.

The return value of both methods is an instance of the AppDataset class representing the application dataset.

AppDataset

The AppDataset class represents the application dataset of selected applications. The dataset is represented as a two-dimensional array stored inside a pandas DataFrame object. Each record in this dataset consists of the following columns: name of the application, version of the application, version of the operating system, SNI, JA3 hash, JA3S hash, the string of TLS values for the JA3 hash and the string of TLS values for the JA3S hash. The AppDataset class contains the following properties:

- `df`: This property holds the DataFrame object containing the dataset.
- `keywords`: A dictionary with keywords for applications inside the dataset.
- `mappings`: A dictionary that maps each application to an integer id. It is used by other classes to get the name of an application from its integer id.
- `add_list`: A list containing the advertising servers.

Besides the above properties, the AppDataset class provides the following methods:

- `save_pickle(filename)`: Used to save an AppDataset instance into the file as a pickle.
- `save_csv(filename)`: Used to save an AppDataset instance into the *csv* file.
- `load_pickle(filename)`: A class method used to load an AppDataset instance from the pickle.
- `load_csv(filename)`: A class method used to load an AppDataset instance from the *csv* file.

The AppDataset class is used by the classifier to classify input flows. The classifier compares the values extracted from the flows with those stored inside an AppDataset object. The dataset can be saved into a pickle or a *csv* format and restored later on.

8.3 Flow Parser

The FlowParser class is used to parse *csv* files containing the flows that represent input for the classifier. It converts the values in the flows from hexadecimal to decimal format, excludes GREASE values and padding values from TLS fields, drops flows containing empty fields, etc. Besides that, it also creates the JA3 fingerprints and attaches them as a new column to the resulting pandas DataFrame. The FlowParser provides the following methods:

- `parse_nprobe_csv(df)`: This method parses a *csv* file containing flows exported by the nProbe. The only parameter is a pandas DataFrame that contains the unprocessed flows.
- `parse_fm_csv(df)`: This method parses a *csv* file containing flows exported by the Flowmon probe. The only parameter is a pandas DataFrame that contains the unprocessed flows.

The FlowParser class is used to parse flows from both nProbe and Flowmon probes. Both methods return a pandas DataFrame object that contains processed flows. These flows are then passed to the classifier.

8.4 Classifier

The classifier is implemented inside the FlowClassifier class. It works together with an AppDataset object that must be provided in the constructor. The classifier offers several different methods of classification and a method that combines all results of these methods together. The only property of the classifier is the application dataset:

- `dataset`: An instance of the AppDataset class that contains the application dataset.

The classification methods provided by the FlowClassifier class are the following:

- `ja3_classification(flow)`: This classification method uses JA3 fingerprints to find an exact match between the input flow and the application dataset.
- `tls_values_classification(flow)`: This method performs the classification using the string of TLS values as it searches for the longest substring match between the input flow and the application dataset.
- `sni_jaccard_classification(flow)`: This method implements the Jaccard similarity coefficient to classify the input flow by its SNI value.
- `sni_tf_idf_classification(flow)`: This method implements the TF-IDF algorithm to classify the input flow by its SNI value.
- `classify(flow)`: This method combines all the approaches and computes the final score as the sum of their results.

The only parameter for all classification methods is the input flow represented as a pandas DataFrame object. The return value is an ordered list of tuples containing names of applications and their resulting scores. The first tuple in this list, therefore, indicates the application with the biggest score. Figure 8.3 shows an example of the return value.

```
[
    ('duolingo', 0.85),
    ('airbnb', 0.43),
    ('livesport', 0.21),
    (...)
]
```

Figure 8.3: Example return value of the classifier.

8.5 Evaluator

The Evaluator class is used to evaluate the classification methods of the FlowClassifier. To evaluate the classification it uses three evaluation metrics implemented as separate methods. The only constructor parameter of the Evaluator is an instance of the FlowClassifier class. The following methods are provided to perform the evaluation:

- `accuracy(test_flows, method)`: This method computes the Accuracy of the classifier. The first parameter is a pandas DataFrame containing test flows used to evaluate the classifier. The second parameter specifies what classification method of the FlowClassifier will be used.
- `precision(test_flows, method)`: This method computes the average Precision of the classifier. The first parameter contains test flows and the second parameter specifies the classification method.
- `recall(test_flows, method)`: This method computes the average Recall value of the classifier. The first parameter contains test flows and the second parameter specifies the classification method.

All evaluation methods require the same parameters. The `test_flows` parameter contains a labeled dataset of flows. Each flow is passed to the classifier and evaluated by comparing the result to the correct label of the flow. The `method` parameter is used to choose the classification method to evaluate. The evaluation metrics implemented in these methods are explained in section 9.1. Table 8.1 shows an example of a labeled test flow.

Column	Value
label	airbnb
L3_IPV4_SRC	192.168.137.222
L3_IPV4_DST	92.123.238.128
L4_PORT_SRC	64262
L4_PORT_DST	443
BYTES	192204
PACKETS	260
TLS_SERVER_VERSION	772
TLS_CIPHER_SUITE	4866
TLS_SNI	api.airbnb.com
TLS_CLIENT_VERSION	56026
TLS_CIPHER_SUITES	4865-4866-4867-49196-49195-52393-49200
TLS_EXTENSION_TYPES	0-23-10-11-16-5-13-18-51-45-43-27
TLS_ELLIPTIC_CURVES	29-23-24-25
TLS_EC_POINT_FORMAT	0
ja3_hash	207f234e15864581680474c785657ae8

Table 8.1: Example of a labeled test flow.

This chapter described the most important classes and methods implemented in this thesis. The next chapter explains the evaluation metrics used by the Evaluator class - Accuracy, Precision, Recall - and presents the results on two test datasets.

Chapter 9

Evaluation

The implemented classification methods need to be evaluated in order to determine their Accuracy. This chapter is going to discuss the evaluation process. Section 9.1 provides an introduction to the evaluation metrics and explains how these metrics work. Finally, Section 9.2 presents the evaluation results of the implemented classifier and provides a comparison of its classification methods.

9.1 Evaluation metrics

All evaluation metrics are implemented inside the Evaluator class discussed in Chapter 8. The source of information for this chapter was the Advanced Data Mining Techniques book [10]. This chapter also takes inspiration from this [32] article by Boaz Shmueli (2019).

The metrics used in this thesis are well-known methods frequently used for evaluating pattern recognition, information retrieval and classification systems. They are used for binary classification, multi-label classification as well as multi-class classification.

To explain the metrics it is important to understand the concept of a confusion matrix. The confusion matrix represents a summary of prediction results on a classification problem and is used for summarising the performance of a classification method. It is a square two-dimensional matrix where the number of rows and columns is equal to the number of classes in the dataset.

The confusion matrix is best explained on a binary classification problem. In binary classification, the model chooses from two classes when making a prediction. These classes are often called positive and negative. For example, given a model that predicts whether a picture contains a dog, the positive class would be "yes" the picture contains a dog, while the negative class would be "no" there is no dog in the picture. For a binary classifier, the confusion matrix contains two rows and two columns and shows how many positive samples were predicted as positive or negative and how many negative samples were predicted as positive or negative. Figure 9.1 shows an example of a confusion matrix.

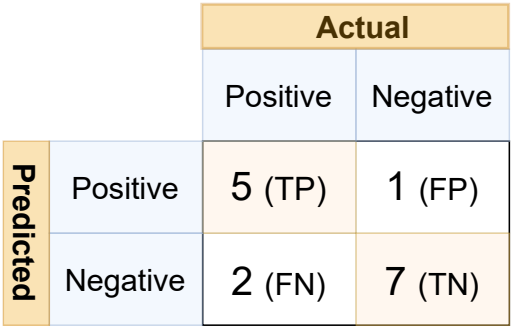

Figure 9.1: Example of a binary confusion matrix [32].

This confusion matrix shows, that five positive samples were correctly predicted as positive, two positive samples were incorrectly predicted as negative etc. The cells in a confusion matrix are often referred to using the following terms[10]:

- True Positive (TP): Positive samples correctly classified as positive.
- False Positive (FP): Negative samples falsely classified as positive.
- True Negative (TN): Negative samples correctly classified as negative.
- False Negative (FN): Positive samples falsely classified as negative.

this terminology is used to define the evaluation metrics used in this thesis. The first metric computes the Accuracy of the classifier. The Accuracy is simply defined as the number of correct predictions divided by the total number of predictions made[10]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.1}$$

Another evaluation metric that is used in this thesis is Precision. Precision is used to express what proportion of the predicted positive classes is truly positive. It can be described with the following formula[10]:

$$Precision = \frac{TP}{TP + FP} \tag{9.2}$$

The last metric that is frequently used together with the two previous metrics is Recall. Recall describes what proportion of the actual positive samples were correctly classified[10]:

$$Recall = \frac{TP}{TP + FN} \tag{9.3}$$

When dealing with a multi-class classification problem, the confusion matrix contains all the possible classes in its rows and columns. The Precision and Recall metrics need to be computed for each class separately. The average value of the Precision and Recall computed for each class is then used to evaluate the whole model. Figure 9.2 shows an example of a confusion matrix with three classes.

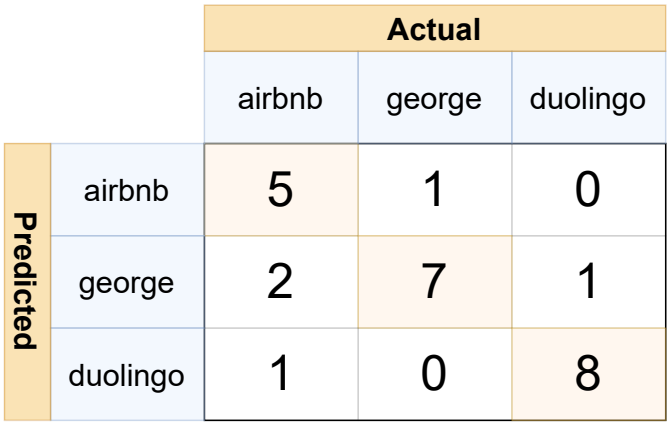

Figure 9.2: Example of a confusion matrix.

The rows in the matrix correspond to the classes predicted by the classifier. The columns represent the actual/true labels of the classified samples. In this particular example, five samples of the Airbnb application are classified correctly as the Airbnb application, two are incorrectly classified as the George application and one sample is predicted to be the Duolingo application.

To compute the Precision of class Airbnb the number of samples correctly classified as Airbnb is divided by the total number of samples classified as Airbnb.

$$Precision(airbnb) = \frac{5}{5+1+0}$$

which would result in a Precision of 83% for the Airbnb class. The Recall of class Airbnb is computed as the number of samples correctly classified as Airbnb divided by the total number of Airbnb samples in the test dataset.

$$Recall(airbnb) = \frac{5}{5+2+1}$$

this results in a Recall of 62.5% for the Airbnb application. The principle of computing the Precision and Recall values using the confusion matrix is illustrated in Figure 9.3.

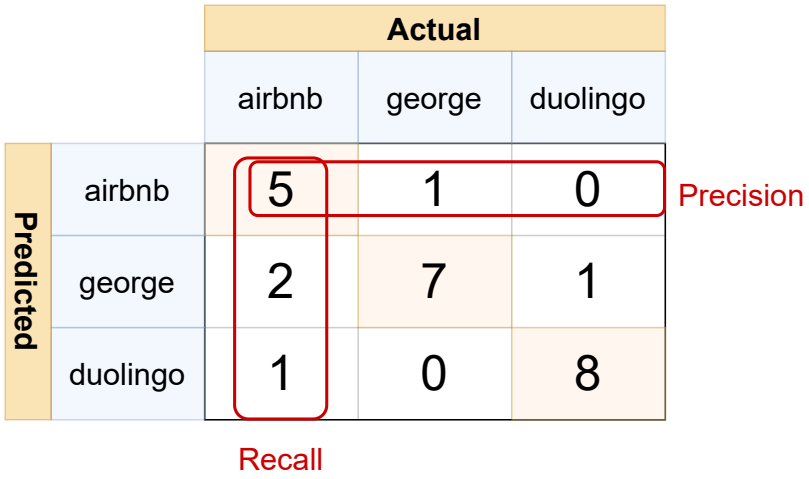

Figure 9.3: Computing Precision and Recall from the confusion matrix.

All classification methods implemented in this thesis are evaluated using the Accuracy, Precision and Recall metrics. The next section presents the evaluation results.

9.2 Evaluation results

The classification methods were evaluated using two test datasets. These datasets were created using the Flowmon probe as described in section 6 and consist of the same applications as the application dataset. The first dataset is referred to as "Test Dataset 1". It consists of 384 flows and contains applications with the same versions as the application dataset. The second dataset, called "Test Dataset 2", consists of 268 flows and contains applications in newer versions. The second dataset, therefore, should give an insight into how the implemented methods are able to classify newer versions of applications.

First of all, the method using JA3 fingerprints is evaluated. The inabilities of the Flowmon probe did not make it possible to create the fingerprints using the original approach described in section 3.3. Additionally, most of the fingerprints in the application dataset are shared between many applications. Table 9.1 shows the Accuracy, Precision and Recall of the method using JA3 fingerprints.

JA3 method	**Test Dataset 1**	**Test Dataset 2**
Accuracy	0.0130	0.0149
Precision	0.1764	0.1428
Recall	0.0050	0.0053

Table 9.1: Evaluation of the JA3 fingerprints method.

As the results show, the JA3 fingerprints method produces a very low Accuracy and Recall. The Precision is slightly better with over 17% on the first test dataset. The reason for such bad results is the fact that when an unknown fingerprint is passed into the classifier, it is not able to make a prediction. Also, when the classifier receives a fingerprint, that is shared by several applications it is not able to decide which application is the right one. Therefore, the classifier is only able to make a prediction if the input flow has a fingerprint that corresponds to just one application. Unfortunately, only around 1-2% of the flows meet this condition.

The next evaluated method uses the string of values extracted from a TLS handshake. This method, in contrast with the JA3 fingerprints, searches not for an exact match, but for the longest substring shared between the input flow and a value in the application dataset. Table 9.2 shows evaluation results of this method.

TLS string method	**Test Dataset 1**	**Test Dataset 2**
Accuracy	0.1719	0.0932
Precision	0.1765	0.2142
Recall	0.0265	0.0183

Table 9.2: Evaluation of the TLS string method.

The TLS string method produced an Accuracy of 17.19% for the first test dataset and 9.32% for the dataset containing the applications in newer versions. The Precision of this method reached to 17.65% on the first dataset and 21.42% on the second dataset. The

Recall values were only 2.65% and 1.83%. It is thus clear, that this method is also not good enough for the classification task. It uses the same values that are used to create the JA3 fingerprints and therefore faces the same problems as the JA3 method.

The better of these two methods, however, turned out to be the method using TLS strings. None of these methods, however, is capable enough to be declared usable neither on its own nor combined with the other one. Combining these methods resulted in the same Accuracy as the TLS string method alone. It is therefore necessary to boost the classification with methods using the Server Name Indication (SNI) field.

Two different SNI methods were implemented. The first one uses Jaccard similarity index while the other one the TF-IDF method to classify input flows. Table 9.3 shows the comparison of these two methods.

	Test Dataset 1		**Test Dataset 2**	
	Jaccard	**TF-IDF**	**Jaccard**	**TF-IDF**
Accuracy	0.9972	0.9609	0.9776	0.9776
Precision	1.0	0.9510	1.0	1.0
Recall	0.9782	0.9338	0.9024	0.9345

Table 9.3: Comparison of the Jaccard index and the TF-IDF method.

As the results suggest, the method using Jaccard similarity performs better on the first test dataset. The TF-IDF method, on the other hand, performs better for the second dataset with the updated versions of applications. Regardless of this fact, the Jaccard similarity outperforms the TF-IDF method and was therefore selected to be used for the classification task.

The last method that is evaluated computes the predictions as a sum of results of the following three methods: JA3 fingerprint method, TLS string method and SNI Jaccard similarity method. The evaluation results of this approach are illustrated in Table 9.4.

	Test Dataset 1	**Test Dataset 2**
Accuracy	0.9972	0.9776
Precision	1.0	1.0
Recall	0.9782	0.9024

Table 9.4: Evaluation results of the final classifier.

As the evaluation shows, combining the three methods results in the same performance as the SNI Jaccard similarity method. The JA3 fingerprints and TLS string methods did not introduce any significant improvement at least on the test datasets used for evaluation.

To conclude this chapter, it is safe to say that none of the methods using values extracted from TLS handshakes is capable enough for the classification task. The only methods that performed well are the two methods using the SNI field. The best method turned out to be the method using the Jaccard similarity index. The approach combining all implemented methods together did not outperform the Jaccard method either. The next chapter presents possible applications of the implementer classifier and conducts several experiments.

Chapter 10

Experiments

This chapter is going to briefly discuss the potential usage of the implemented classifier. One of the requirements in the assignment of this thesis was to conduct several experiments with the implemented classifier and show how it could be used for tasks such as user profiling, malware detection etc.

The usage of JA3 fingerprints for malware detection is arguably one of the main applications of this method. The classifier implemented in this thesis, however, did not manage to use the full potential of the JA3 fingerprints. The fingerprints are created from input flows manually using a slightly modified approach than the original method described in section 3.3. In order to be able to detect malware, a new dataset of known malware fingerprints would have to be created using this modified method. It was decided that this would exceed the scope of this thesis.

10.1 User profiling

An interesting application of the implemented classifier is using it for user profiling. Users can be detected in input flows using the source IP address. It is then possible to detect what applications each user uses. To conduct this experiment a new class called UserProfiler was implemented. The input for this class is a *csv* file containing all the flows that were captured on a network. The profiler then groups the flows by their source IP address and looks at all the SNIs in the groups. For each SNI it uses the Jaccard similarity method of the classifier to predict what application the SNI corresponds to. If the score of the prediction is greater than a given threshold, the predicted application is added to a set of applications for the given user.

To evaluate the UserProfiler class a new dataset was created. This dataset is stored inside the *users.csv* file. To construct this dataset, four different users connected to the same access point. Each user had a set of applications that he interacted with within the session. The captured traffic was then exported using the *flowmonexp5* tool and parsed by the FlowParser class resulting in a *csv* file with 389 rows - where each row represents one flow. The set of applications each user interacted with within the captured session is shown in Table 10.1.

User	IP address	Applications
user1	192.168.137.20	Airbnb, Booking.com, Couchsurfing
user2	192.168.137.35	Crypto.com, Equabank, George, Quora
user3	192.168.137.48	Facebook, Pinterest, Whatsapp
user4	192.168.137.90	Zalando

Table 10.1: Users and their applications in the *users.csv* dataset.

For the input *csv* file, the profiler returns a dictionary containing source IP addresses as keys and the detected applications as values. The predicted applications rely heavily on the threshold value. In the profiler, the threshold value determines the minimal difference between the score of the most probable and the second most probable predicted application. If the difference is greater than the threshold, it means that the classifier is certain enough with the prediction.

For example, if the classifier predicts for an input flow that the most probable application is Airbnb with a score of 1.0 and the second most probable application is Booking.com with a score of 0.5, the difference between the two scores is 0.5. If this difference is greater than the given threshold, the profiler trusts this prediction.

Following are the results of the profiler with different threshold values.

Threshold = 0.4	
Key	**Value**
192.168.137.20	Airbnb, Booking.com, Couchsurfing
192.168.137.35	Crypto.com, Equabank, **Facebook**, George, Quora
192.168.137.48	Facebook, Pinterest, Whatsapp
192.168.137.90	Zalando

Table 10.2: Result of the profiler for a threshold value of 0.4.

When the threshold is set to 0.4, all the desired applications are detected. For user2 the classifier also detected the Facebook application. This application was not actively used by the user and was probably running in the background.

Threshold = 0.3	
Key	**Value**
192.168.137.20	Airbnb, Booking.com, Couchsurfing, **Facebook**, **Youtube**
192.168.137.35	Crypto.com, Equabank, **Facebook**, George, Quora, **Youtube**
192.168.137.48	Facebook, Pinterest, Whatsapp
192.168.137.90	**Facebook**, **Youtube**, Zalando

Table 10.3: Result of the profiler for a threshold value of 0.3.

When the threshold value is lowered to 0.3, more applications start to appear. For users 1,2 and 4 the classifier detected Facebook and Youtube. The detection of Youtube was caused by some Google services running in the background.

Threshold = 0.6	
Key	**Value**
192.168.137.20	Booking.com, Couchsurfing
192.168.137.35	Equabank, George, Quora
192.168.137.48	Facebook, Pinterest, Whatsapp
192.168.137.90	Zalando

Table 10.4: Result of the profiler for a threshold value of 0.6.

On the other hand, when the threshold is set to 0.6, all the applications running in the background disappear. For user1, however, the classifier is no longer able to detect the Airbnb application.

10.2 Traffic monitoring

Another interesting application of the classifier is using its abilities for network traffic monitoring. In particular, the classifier can be used to determine the volume of traffic that was generated by mobile applications.

To demonstrate this usage of the classifier, the *users.csv* dataset was used. For each flow in the dataset, the classifier computes the prediction using the Jaccard SNI method. If the computed score is high enough - given a threshold value - all packets of the flow are assigned to the predicted application. Each flow contains information about the number of packets and bytes that were transferred within the flow. In this fashion, the classifier is able to compute the statistics of how much traffic was transferred by each application as shown in Table 10.5.

Application	**Packets**
Airbnb	914
Booking.com	504
Couchsurfing	256
Crypto.com	16442
Duolingo	0
Equabank	2227
Facebook	94
George	1145
Livesport	0
Pinterest	1092
Quora	4137
Shazam	0
Teams	0
Tiktok	0
Uber	0
Whatsapp	22
Youtube	0
Zalando	127

Table 10.5: Number of packets transferred by each application detected in the *users.csv* dataset.

Table 10.5 shows the result of computing the number of packets transferred by each application in the *users.csv* dataset using the implemented classifier with a threshold value of 0.4. With this kind of data available, it is then possible to compute the percentage of traffic that was generated by mobile applications. It is computed as the sum of traffic generated by each detected application divided by the total number of traffic. Table 10.6 presents the results of such statistics for the *users.csv* dataset.

	% of Applications
Bytes	58.24%
Packets	52.72%
Flows	25.19%

Table 10.6: Percentage of traffic generated by mobile applications in the *users.csv* dataset.

Over 52% of packets were generated by mobile applications. This represents over 58% of bytes and over 25% of flows in the captured communication.

This chapter demonstrated possible applications of the implemented classifier. The first section showed how the classifier could be used for user profiling. Each IP address in the communication was assigned with a set of applications that were detected. This usage could be used for tasks like user identification, recommendations of new applications etc.

The next section revealed how to use the classifier to compute multiple statistics of the captured flows. These statistics included, e.g., the number of packets, bytes and flows that were generated by each application.

Overall, it can be stated that the classifier offers several interesting applications that utilize its abilities and apply them in different areas of computing.

Chapter 11

Conclusion

The main goal of this thesis was to study the possibilities of using TLS fingerprinting techniques for monitoring mobile applications and create a system that would be able to detect mobile applications and work with a state-of-the-art flow exporter software implementing the IPFIX protocol.

This paper started by discussing the related methods that have been used for the problem of traffic monitoring in the past. After giving a few examples of such methods the course shifted towards the method that was intended to be used in this thesis - TLS fingerprinting. The discussion of this method initiated with a brief overview of the concepts essential to its realization - the TLS protocol itself and the JA3/JA3S methods.

After explaining the essentials it was necessary to create the application dataset. The main purpose of this dataset is to be used as a reference when classifying the input flows to predict their corresponding classes (applications). With the application dataset ready it was time to choose the right flow exporter software. Two different probes were being considered - the Flowmon probe and the nProbe. After considering the abilities and weaknesses of both the Flowmon probe was selected. This, however, introduced several obstacles to the TLS fingerprinting method. The probe was only able to parse the first 16 bytes of the TLS cipher suites and also used a TLS extension field that contained GREASE values to extract the TLS version used by the client. These problems resulted in decreasing the variability of the TLS values stored in the application dataset. Following was a comprehensive description of the methods implemented inside the classifier. Three different approaches were combined to construct the final classifier - the JA3 fingerprints, a string of TLS values, and the Server Name Indication (SNI) field.

Despite all the effort, the proposed architectures using JA3 fingerprints and the TLS values did not perform as expected. They were not found to be suitable for working with the Flowmon probe. The evaluation of each method revealed that the classification relied entirely on the performance of the SNI method. Even though the two methods were not successful, this thesis produced other variable results. The created application dataset together with the entire codebase was made available to the public and can therefore help with further development of this idea. Another valuable outcome is the created TLSParser class, that was used to create the application dataset. This parser can be used to extract TLS values and create the JA3 and JA3S fingerprints directly from PCAP files.

The field of cyber security is still rapidly moving forward. It is, therefore, possible, that the Flowmon probe will be more suitable for this task in a newer version. And as more and more traffic is being encrypted new classification methods may occur and present a new challenge to this field.

Bibliography

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

[2] Aitken, P., Claise, B. and Trammell, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information.* IETF RFC 7011, september 2013.

[3] Althouse, J. B. *TLS Fingerprinting with JA3 and JA3S.* [Online; accessed 2021-12-30]. Available at: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.

[4] Anderson, B., Paul, S. and McGrew, D. A. Deciphering Malware's use of TLS (without Decryption). *CoRR.* 2016, abs/1607.01639. Available at: http://arxiv.org/abs/1607.01639.

[5] Bakhshi, T. and Ghita, B. On Internet Traffic Classification: A Two-Phased Machine Learning Approach. *Journal of Computer Networks and Communications.* january 2016, Volume 2016 (2016), p. 21 pages. DOI: 10.1155/2016/2048302.

[6] Benjamin, D. *Applying GREASE to TLS Extensibility.* 2018. [Online; accessed 2022-01-19]. Available at: https://tools.ietf.org/id/draft-ietf-tls-grease-01.html.

[7] Blake Wilson, S., Mikkelsen, J., Nystrom, M., Hopwood, D. and Wright, T. *Transport Layer Security (TLS) Extensions.* IETF RFC 3546, june 2003.

[8] Breitinger, J. B. C., Langer, S. and Gipp, B. Research-paper recommender systems: a literature survey. In:. 2016. International Journal on Digital Libraries ; 17 (2016), 4. - pp. 305-338. - ISSN 1432-5012. - eISSN 1432-1300.

[9] Cloudflare. *What happens in a TLS handshake? | SSL handshake.* [Online; accessed 2021-12-30]. Available at: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/.

[10] David L. Olson, D. D. *Advanced Data Mining Techniques.* Springer Berlin, Heidelberg, 2008. ISBN 978-3-540-76916-3. Page 138.

[11] Deri, L., Martinelli, M., Bujlow, T. and Cardigliano, A. NDPI: Open-source high-speed deep packet inspection. *2014 International Wireless Communications and Mobile Computing Conference (IWCMC).* 2014, p. 617–622.

[12] Eastlake, D. E. *Transport Layer Security (TLS) Extensions: Extension Definitions.* IETF RFC 6066, january 2011.

[13] Flowmon Networks. *Encrypted Traffic Analysis*. [Online; accessed 2022-01-20]. Available at: https://www.flowmon.com/en/solutions/security-operations/encrypted-traffic-analysis.

[14] Flowmon Networks. *Flowmon Probe, NetFlow and IPFIX Exporter*. [Online; accessed 2022-01-02]. Available at: https://www.flowmon.com/en/products/appliances/probe.

[15] Garland Technology. *TAP vs SPAN*. [Online; accessed 2022-01-02]. Available at: https://www.garlandtechnology.com/tap-vs-span.

[16] Help Net Security. *Malware increased by 358% in 2020*. February 2021. [Online; accessed 2022-01-26]. Available at: https://www.helpnetsecurity.com/2021/02/17/malware-2020/.

[17] IBM. *How much does a data breach cost?* [Online; accessed 2022-01-26]. Available at: https://www.ibm.com/security/data-breach.

[18] J. Quittek, T. Zseby, B. Claise, S. Zander. *Requirements for IP Flow Information Export (IPFIX)*. IETF RFC 3917, 2004.

[19] Jaccard, P. The Distribution of the Flora in the Alpine Zone.1. *New Phytologist*. 1912, vol. 11, no. 2, p. 37–50. DOI: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. Available at: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x.

[20] Jason Firch. *10 Cyber Security Trends You Can't Ignore In 2021*. [Online; accessed 2022-01-26]. Available at: https://purplesec.us/cyber-security-trends-2021/.

[21] Lotfollahi, M., Zade, R. S. H., Siavoshani, M. J. and Saberian, M. Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning. *CoRR*. 2017, abs/1709.02656. Available at: http://arxiv.org/abs/1709.02656.

[22] Luhn, H. P. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*. 1957, vol. 1, no. 4, p. 309–317. DOI: 10.1147/rd.14.0309.

[23] Lynch, V. *Google Wants to GREASE Up Chrome*. 2016. [Online; accessed 2022-01-19]. Available at: https://www.thesslstore.com/blog/google-wants-grease-chrome/.

[24] Manning, C. D., Raghavan, P. and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 9780521865715.

[25] Moeller, B., Bolyard, N., Gupta, V., Blake Wilson, S. and Hawk, C. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*. IETF RFC 4492, may 2006.

[26] Nir, Y., Josefsson, S. and Pégourié Gonnard, M. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*. IETF RFC 8422, august 2018.

[27] Rajaraman, A., Ullman, J. and Leskovec, J. Mining of Massive Datasets. In:. 2011, chap. Data Mining, p. 1–17. DOI: 10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.

[28] Razaghpanah, A., Niaki, A. A., Vallina Rodriguez, N., Sundaresan, S., Amann, J. et al. Studying TLS Usage in Android Apps. In: *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2017, p. 350–362. CoNEXT '17. DOI: 10.1145/3143361.3143400. ISBN 9781450354226. Available at: https://doi.org/10.1145/3143361.3143400.

[29] Rescorla, E. *HTTP Over TLS*. IETF RFC 2818, may 2000.

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

[31] Shen, M., Liu, Y., Chen, S., Zhu, L. and Zhang, Y. Webpage Fingerprinting using Only Packet Length Information. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. May 2019, p. 1–6. DOI: 10.1109/ICC.2019.8761167. ISSN 1938-1883.

[32] Shmueli, B. Multi-Class Metrics Made Simple, Part I: Precision and Recall. *Towards Data Science*. 2019. Available at: https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2.

[33] Siavoshani, M. J., Khajepour, A., Ziaei, A., Gatmiri, A. A. and Taheri, A. Machine Learning Interpretability Meets TLS Fingerprinting. *CoRR*. 2020, abs/2011.06304. Available at: https://arxiv.org/abs/2011.06304.

[34] Sparck Jones, K. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. In: *Document Retrieval Systems*. GBR: Taylor Graham Publishing, 1988, p. 132–142. ISBN 0947568212.

[35] Stöber, T., Frank, M., Schmitt, J. and Martinovic, I. Who Do You Sync You Are? Smartphone Fingerprinting via Application Behaviour. In: *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. New York, NY, USA: Association for Computing Machinery, 2013, p. 7–12. WiSec '13. DOI: 10.1145/2462096.2462099. ISBN 9781450319980. Available at: https://doi.org/10.1145/2462096.2462099.

[36] Taylor, V. F., Spolaor, R., Conti, M. and Martinovic, I. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Transactions on Information Forensics and Security*. 2018, vol. 13, no. 1, p. 63–78. DOI: 10.1109/TIFS.2017.2737970.

[37] TechNet, Microsoft Docs. *SSL/TLS in Detail*. 2009. [Online; accessed 2021-12-30]. Available at: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc785811(v=ws.10).

[38] TechNet, Microsoft Docs. *Cipher Suites in TLS/SSL (Schannel SSP)*. 2021. [Online; accessed 2021-12-30]. Available at: https://docs.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel.

[39] Wang, W., Zhu, M., Zeng, X., Ye, X. and Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In: *2017 International Conference on Information Networking (ICOIN)*. Jan 2017, p. 712–717. DOI: 10.1109/ICOIN.2017.7899588.

Appendix A

Installing the Flowmon probe

The Flowmon collector and probe are both commercial software. In order to be able to use them properly a license is needed. After reaching out to the Flowmon support they provided a link to a download page[1] with virtual appliances as well as a demo license for them. From this page the Flowmon collector virtual appliance for VMWare version 11.1.7 was downloaded and installed into the VMWare virtualization software. This virtual appliance also includes the Flowmon probe. The probe provides two monitoring ports that generate the IPFIX data and send them locally to the embedded collector. The collector can be accessed using a web interface running inside the virtual appliance. This interface allows its users to set up the environment, inspect and analyse the captured flows with various visualisation techniques and also export them to a *csv* file for further examination. The virtual appliance also provides the *flowmonexp5* tool that can be used for exporting the flows from a captured *pcap* file.

[1] https://support.kemptechnologies.com/hc/en-us/articles/4404209325965-Download-Flowmon-11-1

Appendix B

Contents of the included SD card

The SD card included as part of this thesis consists of the following files:

```
/
├── data
│   ├── csv_flows ...  Directory containing csv files with test flows
│   ├── csv_flows_new ...  Directory containing csv files with test flows in
│   │                      newer versions
│   ├── experiments ...  Directory with csv files used for experiments
│   ├── pcap_files ...  Directory with pcap files used to create the
│   │                   application dataset
│   ├── application_dataset.csv ...  Contains the application dataset
│   ├── test_flows1.csv ...  Contains test flows in original versions
│   └── test_flows2.csv ...  Contains test flows in newer versions
│
├── doc   ...  Directory with the PDF documentation of this
│   │          thesis and the performed experiments
│   ├── experiments.html
│   └── thesis.pdf
│
├── doc_src ...  Directory containing all source files of the
│                latex documentation
│
├── src   ...  Contains all implemented classes and other source
│              files
│
├── README.md
└── requirements.txt ...  File containing all required libraries
```