Towards Identification of Network Applications in Encrypted Traffic

1st Ivana Burgetová
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
burgetova@fit.vutbr.cz

2nd Ondřej Ryšavý
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
rysavy@fit.vutbr.cz

3rd Petr Matoušek
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
matousp@fit.vutbr.cz

***Abstract*—Network traffic monitoring for security threat detection and network performance management is challenging because most communications are protected by encryption. This paper addresses the problem of identifying applications associated with Transport Layer Security (TLS) network connections. We evaluate three primary approaches to classifying TLS traffic: fingerprinting methods, SNI-based identification, and machine learning-based classifiers. Each method has strengths and limitations: fingerprinting relies on a regularly updated database of known hashes, SNI is vulnerable to obfuscation or missing information, and an AI technique such as machine learning requires sufficient labelled training data. The comparison of these methods that we present highlights the challenges of identifying individual applications, as TLS properties are significantly shared across applications. The simpler task of identifying a collection of candidate applications still provides valuable insights for network monitoring and can be achieved with high accuracy by all methods considered. Finally, we suggest practical use cases and identify future research directions to further improve application identification methods.**

***Index Terms*—TLS fingerprinting, JA4, encrypted traffic, application identification, machine learning**

I. Introduction

Identifying and classifying Transport Layer Security (TLS) communications has become an increasingly difficult problem in modern network environments. With the widespread adoption of encryption protocols such as TLS, a significant portion of network traffic is now encrypted, rendering traditional monitoring tools less effective. As a result, network administrators struggle to gain visibility into network activity, making it difficult to detect security threats, enforce policies, and optimize network performance.

The complexity of identifying applications in TLS communications is due to several factors. First, encryption does not allow for the content inspection previously used for application identification. Second, the new version of TLS uses encryption to protect parameters previously used for application identification, such as Server Name Indication (SNI) and certificates. To address this issue, several methods for classifying TLS traffic have been proposed:

- *TLS fingerprinting*. It relies on unique patterns in the encrypted data to match connections with known application behavior. However, the TLS fingerprinting often struggles with new or updated applications, where the patterns can be significantly different.
- *Server Name Identification*. SNI-based identification uses the server name information available in the TLS handshake to identify the communicating application. However, the new proposal considers the use of Encrypted Server Name Indication (ESNI) to increase user privacy. ESNI keeps the SNI secret by encrypting the SNI part of the client hello message.
- *Machine Learning Classification*. ML-based classifiers use statistical models trained on various features of TLS traffic, such as negotiated security parameters, to predict the associated application. ML-based methods can adapt to new and changing traffic patterns, but require significant training data and computational resources.

The ultimate goal of these techniques is to accurately identify the application associated with each TLS connection. Due to the inherent challenges, it is not always possible to identify a single application. In such cases, an acceptable solution may be to provide a (ranked) list of possible applications associated with the connection, allowing network administrators to make informed decisions based on likely candidates.

A. Contribution

The main contributions are (i) the introduction of a novel annotated dataset of TLS communications from common desktop and mobile applications, enabling further research on traffic classification; (ii) a comprehensive comparison of three approaches to identify encrypted traffic, i.e., TLS fingerprinting, ML-based methods, and SNI matching, and evaluation of their coverage, accuracy, and feasibility of deployment; (iii) identification of challenges in TLS application detection due to common TLS properties among applications; and (iv) discussion of possible application use cases and future work to improve the accuracy of these methods.

B. Structure of the paper

The structure of the paper is as follows: Section II presents related work, providing an overview of previous studies that address the same problem as this paper. Section III describes the principles of TLS encryption, explaining the key features of the TLS handshake and how they are used for fingerprinting.

Section IV describes the experimental setup, including the process of creating new datasets and the methodology used to evaluate the different identification methods. Section V discusses the experiments and results, providing a comprehensive analysis and addressing practical considerations. Finally, Section VII summarizes the paper with concluding remarks and suggestions for future research.

II. Related Work

Classification of encrypted traffic and identification of network applications have been researched since the widespread adoption of encrypted communication protocols. This section gives an overview of the major work on identification methods, primarily TLS fingerprinting, machine learning, and neural networks, which have received significant attention recently.

The use of TLS fingerprints for malware detection in encrypted traffic was addressed by Anderson et al. in [1]. The authors extract the cipher suites, the TLS extensions, and the length of the client's public key from the TLS client/server hello records. In addition to the TLS attributes, they observe flow statistics, such as packet lengths, inter-arrival times, byte distribution. By combining these attributes on a large dataset of malicious and legitimate TLS flows, they are able to achieve 99.6% accuracy in classifying malware. When using TLS attributes alone, accuracy ranges from 63% to 100%. Unfortunately, the proposed method is applied to private datasets, which limits comparison with other approaches.

Machine learning detection of encrypted malware communications was also investigated by De Lucia and Cotton [2]. The authors applied a support vector machine (SVM) and a convolutional neural network (CNN) to streams extracted from captured TLS connections. As features they used record size, type, and direction. Their results show high accuracy, but they do not discuss important issues such as similarity of TLS features, overlap of malware families, etc. In contrast to their approach, we focus on identifying network applications in encrypted traffic rather than malware families.

The application of machine learning and deep learning to the identification of encrypted traffic was also explored by Barut et al. [3]. The authors combined flow metadata (port numbers, payload size, bytes transferred) with TLS features to classify encrypted application traffic. They used random forest and k-NN classifiers to select the features. However, the best RF classifier selected the source port number as the most important feature, which is a value randomly generated by the operating system. Therefore, its use for classification is questionable. For TLS features, both statistical data (the number of cipher suites, extensions, key lengths, etc.) and pre-processed lists of cipher suites with extensions were used. To use the CNN classifier, the authors transformed the input data to overcome the bias due to the imbalance of the data set. Their conclusion showed the importance of TLS cipher suites for application identification, which was a part of TLS fingerprints.

In [4], Anderson and McGrew examined the evolution of TLS usage in applications over time. They tracked the use of different TLS versions, cipher suites, and extensions, and collected session data such as associated processes, destination IP addresses, and ports. They clustered similar fingerprints using Levenshtein distance. While their work focused on general TLS trends, we aim to address application identification.

Fingerprint overlap was discussed by Anderson and McGrew in [5]. They extended TLS fingerprinting to include the destination address, port, and SNI so that their fingerprints were more accurate using the destination context. This was similar to server fingerprinting: JA3S and JA4S hashes. We also use the server attributes along with the SNI values. The authors measured the similarity between fingerprints using Levenshtein distance and added weights to the attributes based on the information gain ratio. Rather than identifying families of applications as in [5], our work attempts to identify the individual applications where possible. In the case of shared fingerprints, we compute the most likely one or present a set of matching applications.

Our paper extends the previous work of Matousek et al. [6], which investigated the reliability of TLS fingerprints for mobile application identification. We compare JA4 fingerprints with ML techniques and SNI matching to highlight their advantages and limitations. To the best of our knowledge, we have not found any published work comparing machine learning approaches with TLS fingerprinting.

Another popular direction for identifying encrypted traffic are AI techniques such as deep learning, which using CNN [7], [8] or RNN classifiers [9]. Unlike TLS fingerprinting or machine learning, the deep learning approaches presented here work with the full payload, i.e., they encode incoming packet payloads into vectors or images that are then used for training and testing. While the authors claim to achieve a high levels of accuracy, they ignore many important issues related to the nature of the encrypted data. First, they treat the input data as uninterpreted, without distinguishing between a packet header and a payload. This has a significant impact on the stability of the model, as encrypted traffic typically has a different payload distribution depending on the negotiated algorithm, e.g., HTTPS transmissions would have different characteristics between the same hosts if a different encryption algorithm is chosen. In addition, different applications transmit different amounts of data, e.g., an encrypted file transfer has much more data than an encrypted email. This omission results in an unbalanced training dataset that needs to be artificially normalized. Another problem with modelling full-packet payloads is the huge amount of processing data. For example, the popular ISCX dataset contains 21 GB of data (including payload), but only 2,436 TLS connections. Therefore, storing and processing the full payload is not feasible for real-world networks, so the TLS-based methods are more preferable for practical use.

The previously cited papers mostly work with the VPN-nonVPN dataset (ISCXVPN2016)[1] in experiments and evaluation. However, this dataset was created in 2016, before the new encryption algorithms and TLS standards were adopted (e.g.,

[1]See https://www.unb.ca/cic/datasets/vpn.html [Sept 2024].

TLS 1.3). Therefore, the classification models trained on this dataset do not correspond to current encrypted transmissions. For this reason, we provide our own annotated dataset of recent encrypted communications, see Section IV-D. For comparison, we also include results from the original ISCX dataset.

III. TLS Encryption

Transport Layer Security (TLS) [10], [11] is a protocol defined on top of the transport layer that provides encryption, data integrity, and authentication for application protocols. Typically, TLS is implemented on top of TCP. After a TCP connection is established, the TLS client and server perform a TLS handshake where they negotiate security parameters to establish a secure TLS channel. The TLS handshake is an essential part of the TLS fingerprint, specifically the client hello and server hello packets, which contain a list of cipher suites, extensions, and other parameters supported by the client and server. TLS defines a large number of possible parameter values. Their combinations represent a distinctive feature of the client or server that is used for application fingerprinting.

The TLS handshake data is sent unencrypted. Once the TLS handshake is complete, subsequent packets between the client and server are encrypted, see Figure 1.

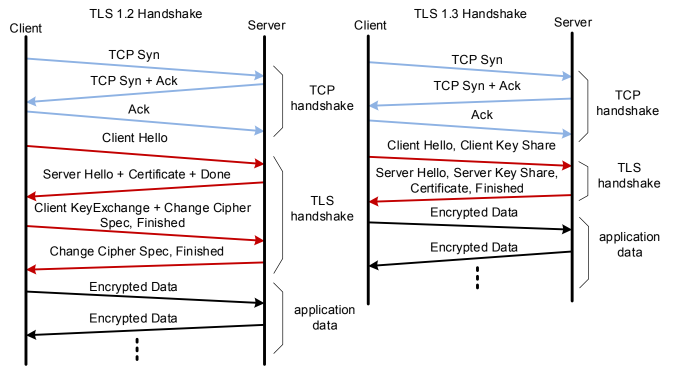


Fig. 1: TLS handshake version 1.2 and 1.3

A. TLS Features

To identify encrypted applications, there are three sources of features for classification models: (a) TLS attributes extracted from the TLS handshake, (b) metadata about the TLS flow (e.g., number of bytes and packets transmitted, duration), and (c) the full packet payload[2]. The TLS fingerprinting and machine learning presented in this paper uses the first approach, so we briefly introduce common TLS attributes that form JA3/JA4 fingerprints[3] used in our research.

- *Version*. The version of the TLS handshake protocol.
- *A list of cipher suites*. It includes possible combinations of a key exchange method, algorithms for authentication, encryption, and data integrity. Valid combinations are standardized by IANA[4]. The current IANA cipher suite list contains 351 different combinations.
  The cipher suite list may contain random GREASE values [12] for client or server compatibility testing. These values introduce instability into TLS fingerprinting, and are therefore excluded from the fingerprint calculation.
- *A list of extensions*. TLS extensions define additional TLS features. There are approximately 63 different extensions.
- *Supported Groups (SG)*. This TLS extension specifies the named groups that the client supports for key exchange. They are ordered from most to least preferred.
- *Elliptic Curve Point Format (EC Format)*. It specifies the encoding supported by the client for transmitting EC values.
- *Server Name Indication (SNI)*. The SNI specifies a domain name of the server that the client is contacting [13]. The SNI is not a part of the TLS fingerprint, but plays an important role in annotating the requested service. However, it is only useful for application identification when the client is contacting a fixed service, such as a weather forecast server. In the case of web browsers, the SNI value changes with each new web server requested.
- *Application Layer Protocol Negotiation (ALPN)*. If multiple application protocols are supported by a single server, the client and server must negotiate an application protocol to be used for each connection [14].
- *Supported Versions*. This extension contains a list of TLS versions supported by client, ordered by preference [11].
- *Signature Algorithms*. A list of supported hash algorithms used for signatures.

Table I presents different sets of TLS attributes for the the client (JA3, JA4) and server (JA3S, JA4S) TLS fingerprints. It

TLS Attribute	**JA3**	**JA3S**	**JA4**	**JA4S**
TLS/QUIC protocol			x	x
Handshake Version	x	x	x	x
Cipher Suites	x	x	x	x
Extensions	x	x	x	x
Supported Groups	x			
EC Format	x			
SNI				
ALPN			x	x
Supported Versions			x	x
Signature Algorithms			x	

TABLE I: TLS attributes used in JA3 and JA4 fingerprints

shows which TLS attributes are shared by different types of TLS fingerprints. The TLS cipher suites are ordered for JA4/S, while JA3/S keeps the original order. This plays an important role when for application identification.

B. TLS Attributes Characteristics

We analyzed the uniqueness and importance of TLS attributes. To evaluate the importance of TLS attributes, we use entropy, which represents the degree of uncertainty of the value in the full range of possible values. This means that attributes

[2]The use of the encrypted packet payload is limited, because encrypted data does not provide any information for analysis.

[3]See https://blog.foxio.io/ja4+-network-fingerprinting [Sept 2024].

[4]See IANA TLS Parameters.

with higher entropy contribute more to the uniqueness of the fingerprint and help to better identify the application. Low entropy means that many TLS connections share the same attribute value, in which case the attribute does not contribute much to distinguishing applications.

Table II contains the entropy of TLS attributes in the MDA and ISCX datasets (see Section IV-D). The MDA dataset contains 21,301 TLS connections. After filtering out incomplete connections and connections to analytics and advertising servers, we obtained 16,427 connections from 77 different applications. The table shows the number of unique values for the attribute, the percentage of connections containing an empty value for the attribute, and the entropy. For comparison, we have also included values from the ISCX dataset. This dataset contains 1,494 TLS connections from 16 applications.

TLS Attribute	**Uniqueness**		**Emptiness (%)**		**Entropy**	
	MDA	ISCX	MDA	ISCX	MDA	ISCX
TLS Version	1	3	0	0	0	0.09
Client Cipher Suites (unsrt)	35	22	0	0	0.48	0.81
Client Cipher Suites (srt)	31	24	0	0	0.41	0.81
Client Extensions (unsrt)	8202	28	0	0	0.73	0.79
Client Extensions (srt)	59	23	0	0	0.57	0.79
EC Format	2	3	1.78	6.09	0.22	0.46
SNI	731	116	0.03	22.56	0.86	0.70
ALPN	10	7	7.94	59.1	0.3	0.70
Client Supported Versions	48	1	10.08	100	0.83	0
Signature Algorithms	17	8	0	1.7	0.45	0.68
Server Cipher Suites	11	22	0	0	0.62	0.71
Server Extensions (unsrt)	53	40	0	0	0.52	0.58
Server Supported Versions	3	1	22.64	100	0.59	0

TABLE II: Entropy of TLS features in the MDA/ISCX datasets

As expected, the most important TLS attributes for fingerprinting are SNI, client extensions and server cipher suites for the MDA dataset, and client cipher suites, client extensions, and server cipher suites for the ISXC dataset.

We can see a difference between the MDA dataset created in 2024 and the ISCX dataset created in 2016. The numbers differ due to the removal of obsolete cipher suites and the addition of new extensions. In addition, no server- or client-supported versions were found at all in the ISCX TLS connections.

The client-supported versions show a high entropy, but there are about 10% of TLS connections with the empty value for the MDA. The TLS version also contains only one value for the MDA, and no values in the ISCX datasets, making it useless for application detection.

IV. Test Environment

Our test environment included an Android emulator for mobile applications and a Windows-based virtual sandbox for desktop applications. Both tools captured network communications in PCAP files and annotated connections based on application processes. We focused on Android and Windows applications due to their market dominance, although applications from other environments can be similarly analyzed.

A. Emulation of Mobile Applications

To generate the TLS fingerprints of mobile applications, we have developed a tool[5] that emulates the behaviour of mobile apps using the Android Virtual Device (AVD). Our tool downloads an APK file containing the mobile app and installs it on the virtual device. Its behaviour is then emulated using the ADB shell and the `monkey` command. Typically, the app opens a number of connections to the application server, which are captured using `tshark`. The extracted TLS connections are then parsed for TLS client and server hello packets to generate JA3/S, JA4/S, and JA4X fingerprints, which are stored in the fingerprint database. The output forms a part of the MDA dataset as described in Section IV-D.

B. Sandboxing Windows Applications

TLS connections from desktop applications are captured by running them in a sandboxed Windows environment. We capture all network communications from the host and filter out only application-related connections by monitoring open sockets. This approach allows us to automatically tag each communication with the corresponding process name. We created a dataset of 42 popular Windows applications. The annotated data has been processed and is part of the MDA dataset as described in Section IV-D.

We also found that many desktop applications available in the Microsoft Store, such as Instagram, TikTok, Pinterest, and Facebook, are deployed as Progressive Web Apps (PWAs). PWAs are web applications that provide an app-like experience and can be installed on a device to run in a dedicated window without the traditional browser interface. Because PWAs run inside the web browser process, it is impossible to identify them by their process name. As a result, we focus exclusively on native Windows applications. Analysis of PWA-based applications has been reserved for the future.

C. ISCX2016 Dataset

For an objective comparison, we utilized publicly available dataset named ISCXVPN2016[6] created by the Canadian Institute for Cybersecurity. The dataset contains annotated samples of network application communications such as web browser, email, chat, streaming, file transfer, etc, see [15]. The full dataset contains 21 GB of captured communications, covering 2,436 TLS connections from 16 different applications.

D. MDA Dataset

The Mobile Desktop Applications (MDA) dataset is an annotated dataset created by our research team for experiments with encrypted traffic[7]. The dataset contains complete communications in the form of PCAP files from 35 mobile and 42 desktop applications.

Table III shows the number of TLS connections of MDA mobile and desktop applications compared to the ISCX2016. For both datasets, we preprocessed and cleaned the data to prepare it for use in the experiments. First, we filter out incomplete connections and connections to advertising and tracking servers. Then we divide the whole dataset into a training and a test part based on the separate runs of our

[5]See https://hashapp.netology.sk [Sept 2024].

[6]See https://www.unb.ca/cic/datasets/vpn.html [Sep 2024].

[7]Available at https://github.com/matousp/tls-fingerprinting [2024].

	Mobile MDA	Desktop MDA	ISCX2016
Total TLS connections	6227	15074	2436
Complete connections	6133	15047	2422
Filtered connections	4142	12285	1494
Train part	3095	8144	1063
Test part	1047	4141	431
Number of apps	35	42	16

TABLE III: Statistics of the MDA and ISCX2016 datasets

applications, using 2/3 of the runs for training and 1/3 for testing. Since each run consists of a different number of TLS connections, the numbers are not exact.

We also computed basic statistics for the MDA dataset. Table IV provides selected properties of individual fingerprints, their combinations, and the SNI value for mobile and desktop application connections. These properties evaluate different types of fingerprints, their combinations, and show which combination seems to be more promising for correctly identifying applications. We provide the total number of distinct values, the percentage of unique fingerprints vs. our application set, the percentage of applications covered by unique fingerprints, and the efficiency of the fingerprint.

Fingerprint type	Total	Uniqueness	Covered apps	Efficiency
JA3	8208	99.5%	67.5%	1.02
JA4	111	54.1%	41.6%	3.36
JA3S	77	44.2%	20.8%	4.53
JA4S	97	48.5%	27.3%	4.06
JA3+JA3S	8330	99.2%	80.5%	1.02
JA4+JA4S	264	66.7%	70.1%	2.16
JA3+JA4+JA3S+JA4S	8349	99.2%	84.4%	1.02
SNI	728	88.0%	89.6%	1.27

TABLE IV: Efficiency of different TLS fingerprints in mobile and Windows application filtered dataset.

The efficiency E expresses the average number of applications sharing the same fingerprint. It is calculated as follows: let $i = 1 \dots n$ be the number of unique fingerprints and the function $f(i)$ maps each fingerprint to an application. For example, if a fingerprint is assigned to two different applications, $f(i) = 2$. Then the efficiency E of the fingerprint type is calculated as follows:

$$E = \frac{\sum_{i=1}^{n} f(i)}{n} \tag{1}$$

Table IV shows that we found 8,208 different JA3 fingerprints, of which 99.5% were unique, i.e., used by only one application. The rest of the fingerprints were shared by several applications. However, the JA3 fingerprints only cover 67.5% of the applications, the rest of the applications do not have unique fingerprints. The efficiency of 1.02 means that, on average, one fingerprint is used for 1.02 applications. Thus, the optimal fingerprint has a coverage of 100% and an efficiency of 1. From this point of view, the combinations of fingerprints or SNI value are more promising for application identification, so we focused on them in our experiments.

E. Evaluation methodology

Due to the number of shared fingerprints, as shown in Table IV, it is not easy to assign a specific application to each observed TLS connection. On the other hand, it is helpful for network traffic monitoring to assign a small set of possible applications. To address the issue of shared fingerprints, we use two different evaluation methods to compare the accuracy of different classification approaches:

- *Probabilistic method* where a single, most likely application is associated with the tested TLS connection.
- *Set-based method* where each TLS connection is associated with a set of applications that match the connection based on the chosen classification approach. The classification result is considered correct if the real application is included in the set of possible candidates.

The *accuracy* provides a comparison between different fingerprinting methods. We measure the accuracy in terms of the percentage of correctly classified (*OK*), misclassified (*Error*) and unrecognized connections (*Unknown*).

The accuracy for *set-based classification* is calculated by evaluating all classifiers on the test data. For each TLS connection, there are three possible results:

- *OK:* At least one classifier correctly identifies the application.
- *Unknown:* None of the classifiers recognizes the application.
- *Error:* One or more classifiers match the application, but none is correct.

Calculating the accuracy of a *probabilistic classification* means that each application classifier generates a probability score for each row of the test data, indicating how likely it is that the row belongs to a particular application. The classifier with the highest probability score is selected as the predicted classification for that row of data. The result is calculated as *OK* if the selected classifier correctly identifies the application. The *Unknown* and *Error* results are calculated as described for set-based classification. The overall accuracy is then calculated by aggregating the number of *OK*, *Unknown* and *Error* results over each dataset.

V. Experiments

We have performed experiments using different identification methods. All experiments follow the same processing steps as shown in Figure 2. The input is the labelled dataset with TLS communication, which is filtered in the first step to remove known connections to public ad servers. The filtered dataset is divided into a training and a test part. The training part is used to train ML-based detectors and to map fingerprints and SNIs to applications. Once the models are created, they are applied to the test data. During the evaluation, two types of classification (probabilistic and set-based) are considered, see Section IV-E.

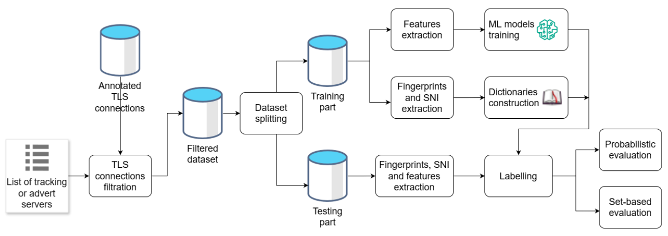

Annotated TLS connections
List of tracking or advert servers
TLS connections filtration
Filtered dataset
Dataset splitting
Training part
Testing part
Features extraction
ML models training
Fingerprints and SNI extraction
Dictionaries construction
Fingerprints, SNI and features extraction
Labelling
Probabilistic evaluation
Set-based evaluation

Fig. 2: The processing pipeline

	Mobile apps			Windows apps			All			ISCX2016		
Classification type	**OK**	**Unknown**	**Error**	**OK**	**Unknown**	**Error**	**OK**	**Unknown**	**Error**	**OK**	**Unknown**	**Error**
Probabilistic classification												
JA3+JA3S	47.6%	13.7%	38.8%	30.4%	61.9%	7.7%	32.3%	52.2%	15.6%	79.4%	6.5%	14.4%
JA4+JA4S	54.4%	1.8%	43.7%	56.8%	0.4%	42.7%	54.0%	0.7%	43.4%	80.0%	6.3%	13.7%
JA3+JA3S+JA4+JA4S	49.2%	13.7%	37.2%	30.4%	61.9%	7.7%	32.6%	52.5%	15.2%	79.4%	6.3%	14.4%
ML-based	46.0%	28.6%	25.4%	47.0%	0.8%	52.2%	44.7%	4.3%	51.1%	77.4%	1.1%	21.4%
SNI	86.3%	5.9%	7.8%	75.0%	1.5%	23.5%	76.6%	1.9%	21.5%	85.9%	5.7%	8.4%
Set-based classification												
JA3+JA3S	79.9%	13.7%	6.5%	37.4%	61.9%	0.7%	46.0%	52.2%	1.9%	86.5%	6.3%	7.2%
JA4+JA4S	90.9%	1.8%	7.3%	98.7%	0.4%	0.9%	97.1%	0.7%	2.2%	86.5%	6.3%	7.2%
JA3+JA3S+JA4+JA4S	78.8%	13.7%	6.6%	37.4%	61.9%	0.7%	45.9%	52.2%	1.9%	86.5%	6.3%	7.2%
ML-based	67.3%	13.6%	19.1%	97.5%	0.2%	2.3%	91.9%	1.4%	6.7%	87.7%	2.5%	9.8%
SNI	88.3%	5.9%	5.8%	98.2%	1.5%	0.3%	96.2%	1.9%	1.9%	91.3%	5.7%	3.0%

TABLE V: Accuracy of TLS connection classification with different classification approaches.

A. *Fingerprints classifiers*

Our experiments evaluate the performance of different fingerprint combinations for application identification. We compare the original version of the fingerprints (JA3+JA3S) with the newer version (JA4+JA4S). We also test the combination of all four fingerprints (JA3+JA3S+JA4+JA4S), see Table V.

We use a dictionary-based exact match method for classification. We created a dictionary of fingerprints from the training set and applied it to find the most likely application or a set of applications for each TLS connection in the test set. The fingerprints not seen in the training set were marked *Unknown*.

The experiments yielded some unexpected results. Although the basic properties described in Table IV show very high uniqueness and percentage of covered applications for the JA3+JA3S combination, its accuracy for connection classification is the lowest, see Table V (both probabilistic and set-based methods). This is more obvious for the Windows applications, where JA3 fingerprints tend to be more unique due to the different order of the TLS client extensions. This high degree of uniqueness leads to a high percentage of unseen *Unknown* fingerprints in the test set. In this situation, the JA4 fingerprints work better because they use the client extensions differently.

In general, the JA4+JA4S fingerprints perform better, but they tend to form a larger cluster of associated applications, on average three more applications than in the case of JA3, see Table VI. The performance of the combination of all four fingerprints is comparable to the JA3+JA3S fingerprints as the included JA3 fingerprints significantly increase the percentage of unseen fingerprints.

Fingerprint type	**mobile**	**desktop**	**all**	**ISCX2016**
JA3+JA3S	5.46	2.27	3.85	2.09
JA4+JA4S	5.77	5.03	6.71	2.09
JA3+JA4+JA3S+JA4S	5.29	2.26	3.78	2.09
ML-based	1.79	3.22	3.55	1.57
SNI	1.17	2.12	2.00	1.77

TABLE VI: Average number of predicted applications for one fingerprint (set-based method).

B. *SNI classifier*

For SNI classification, we use the same dictionary-based and exact match method as for fingerprinting. Unsurprisingly, this classifier achieves the best accuracy for the majority of our datasets. It is slightly outperformed by the set-based classifier, but at the cost of a higher average number of applications assigned to each connection.

C. *ML-based classifiers*

To facilitate comparison with fingerprinting methods, we also trained ML-based binary classifiers for application identification. A separate classifier was trained for each application, with the primary goal of evaluating whether ML algorithms could achieve higher accuracy in identifying applications. We deliberately did not perform advanced feature engineering or use flow metadata, relying only on the information available in the TLS handshakes. This allowed us to directly compare the accuracy of ML-based detectors with JA3/JA4 fingerprinting.

The dataset was divided into training and test parts, following a similar methodology used for fingerprinting. The training data represents 2/3 of the captured TLS handshakes. Importantly, all applications were present in the training data. We applied this approach to all datasets.

a) Features: The input features represented categorical data[8], which required appropriate encoding. We used one-hot encoding, which, while potentially generating a large number of columns, avoids introducing unintended relationships between values. The input features are listed in the table VII. String values were encoded directly using one-hot encoding. Lists of strings were converted into a single string by concatenating the individual values in their original order. The combined string was then one-hot encoded. In the case of extension types these were first converted to an ordered list, and then concatenated into a single string using the same encoding process as for lists. We chose this approach after analysing the dataset and observing that ordering the information keeps the number of different values manageable without losing much information. We also remove grease values from the list before encoding.

Encoding categorical data results in many boolean columns. Table VIII shows the size of the boolean vector after encoding categorical columns for each dataset. The total number

[8]Although most of the input fields are numeric representing constant values, e.g., TLS versions, cipher suites, extension types, etc., we treat them as strings because they have categorical rather than ordinal meaning.

No	Feature	Type	Description
F1	**TlsVersion**	String	The version of the TLS protocol used during the connection.
F2	**TlsClientCipherSuites**	String[]	An array of the cipher suites supported by the client.
F3	**TlsClientExtensionsSet**	String[]	An ordered array of TLS extensions supported by the client.
F4	**TlsClientSupportedGroups**	String[]	An array of supported elliptic curve groups by the client.
F5	**TlsClientAlpns**	String[]	An array of Application-Layer Protocol Negotiation (ALPN) protocols supported by the client.
F6	**TlsClientSupportedVersions**	String[]	An array of TLS protocol versions supported by the client.
F7	**TlsClientSignatureAlgorithms**	String[]	An array of signature algorithms supported by the client for verifying the integrity of the TLS handshake.
F8	**TlsServerExtensions**	String[]	An array of TLS extensions used by the server, responding to the client's supported extensions.
F9	**TlsServerCipherSuite**	String	The cipher suite selected by the server for encryption during the TLS handshake.

TABLE VII: Features used for ML-based binary classifiers

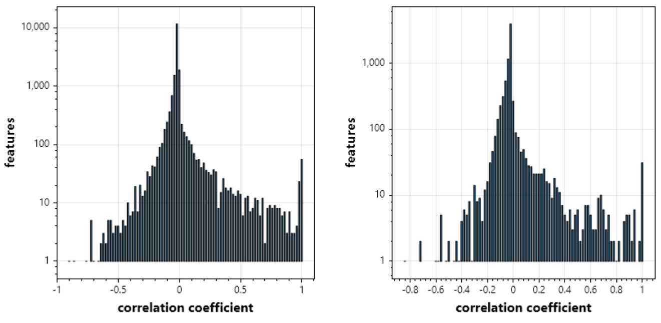
represents the size of the input vector for classifier training, which is obtained by concatenating the individual one-hot encoded source features. The table also shows the size for each source feature after applying one-hot encoding. Due to space limitations, the header refers to the feature index (defined in Table VII) instead of the full feature name. As can be seen, the largest vectors in all datasets are the client and server extension features, which contain the most diverse values within the source datasets. The total size of the feature vector varies slightly between the datasets, reflecting the difference between the Windows and mobile datasets and the older ICSX dataset. For example, in the mobile dataset, we observed only a single TlsVersion, but a richer variation of extension values.

Dataset	Total	F1	F2	F3	F4	F5	F6	F7	F8	F9
Windows	136	3	17	33	10	3	3	17	42	8
Mobile	142	1	23	46	10	8	5	10	30	9
ICSX	123	3	20	25	4	6	1	7	37	20

TABLE VIII: Dataset feature dimensions

We also computed the correlation between the encoded features. The graph in Figure 3 shows the distribution of Pearson correlation coefficients between pairs of features in the datasets. The histogram shows a distribution of correlation coefficients between -1 and 1, with the majority of values centered around 0, indicating that most pairs of features are uncorrelated. The low positive correlation is likely due to the nature of one-hot coding, as some categories may rarely co-occur, resulting in very low positive correlations between pairs of features that are activated together in some cases. The histogram also shows a tail towards correlation coefficients close to 1, meaning that some pairs of features are highly correlated. We further examined these correlations and found that some specific client cipher suites are always used together with certain extension sets (TlsClientExtensionsSet), signature algorithms (TlsClientSignatureAlgorithms), and supported groups (TlsClientSupportedGroups).

b) Dimension reduction: We investigated two methods of dimensionality reduction to improve classifier training. Specifically, we compared the performance of binary classifiers trained on the original one-hot-encoded feature vectors to those trained after applying dimensionality reduction techniques. The first method tested was Principal Component Analysis (PCA), which is the orthogonal projection of the data onto a lower dimensional linear space such that the variance



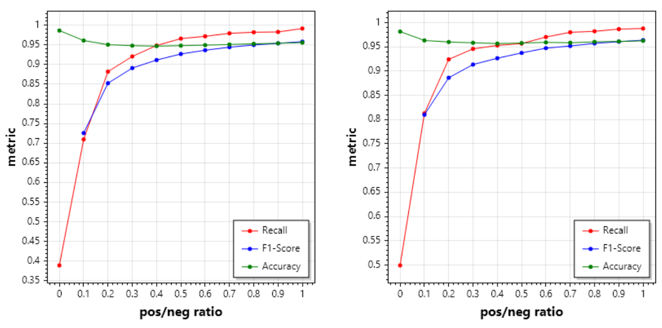
(a) MDA Dataset (b) ICSX Dataset

Fig. 3: Correlation among features

of the projected data is maximized. The second method was autoencoder-based dimensionality reduction, where a neural network (autoencoder) is trained to compress the data into a lower dimensional space. The results showed that neither of these dimensionality reduction methods led to significant improvements in classification performance metrics. In addition, they did not reduce the computation time. In contrast, the autoencoder-based approach required a significant amount of time to train the encoder, which increased the overall computational cost.

c) Performance metrics: We evaluated the classifiers using several performance metrics, including recall, F1 score, AUC-ROC, and AUC-PRC [16]. These metrics were computed independently for each application classifier. We used several algorithms, i.e., Random Forest, FastTree, LightGBM, LBFGS Logistic Regression, and Linear Support Vector Machine to develop the application classifiers. For each classifier, we selected the algorithm with the highest recall (for the set-based identification task) and F1 score (for the probabilistic identification task). As we build the binary classifier for each application in the dataset, the input data is necessarily unbalanced. For each application, there are significantly fewer positive class elements (connections of the application) because the connections of all other applications form a negative class. Therefore, a random oversampling technique was applied to mitigate this problem. This method duplicates instances of the minority class in the categorical dataset to increase its representation[9]. The graphs in Figure 4 illustrate

[9]While more sophisticated methods such as SMOTENC could be considered, even this simple approach sufficiently improved the false negative rate.



(a) MDA Dataset (b) ICSX Dataset

Fig. 4: The metrics of unbalanced datasets

the impact of dataset imbalance on the key performance metrics of the trained classifiers. As shown, both recall and F1-score improve when the dataset is balanced. This is because a balanced dataset allows the classifier to learn more effectively and identify the minority class that is underrepresented in unbalanced datasets. As expected, the accuracy decreases slightly when the dataset is balanced. In an unbalanced dataset, accuracy is often inflated because the classifier can simply correctly predict the majority class most of the time, even though it may fail to identify instances from the minority class. In contrast, a balanced dataset forces the classifier to better consider both classes, resulting in a more realistic measure of its true performance.

The resulting best performing model was designated as the application detector, which was then further evaluated and compared using the methodology described in Section IV-E. As a result, the application detectors can use different algorithms depending on their performance. Among the algorithms tested, the binary decision tree classifier was chosen in most cases (177x), with a few classifiers using either field-aware factorisation to binary classification model (4x) or a linear logistic regression model trained with L-BFGS method (5x).

VI. Discussion

A. Results Evaluation

The aim of our experiments was to compare different approaches to identifying the application based on TLS features. Table V summarises the achieved accuracy for two modes: (a) a single application identification (probabilistic classification), which suggests the most likely application identified for the TLS connection, and (b) a set of application identifications (set-based classification), which provides a list of candidate applications. Obviously, the first mode is more difficult and the results have lower accuracy. We evaluated five different methods, three based on TLS fingerprinting, one using an ML-trained model, and the last one using a simple dictionary of known SNIs.

Although the experiments were limited to the MDA and ISCX datasets, there are some interesting observations:

- The SNI approach provides consistent performance across the different datasets, supporting the observation that applications tend to use only a limited set of SNIs to communicate with their application servers. In some cases, the SNI is not available, or the applications contact the share services, making such connections ambiguous.
- In fingerprinting, the different combinations of JA3 and JA4 fingerprints give different levels of accuracy. The best solution is the combination of JA4 and JA4S, which proves the correct design of the hash calculation and supports the claim of the JA4+ authors about the usefulness of this fingerprinting method for identifying encrypted communications. The performance of the widely used JA3 fingerprint is poor, suggesting that it is becoming obsolete. This is because the TLS libraries use GREASE values and random order in the TLS parameters.
- Finally, we tried to build ML-based classifiers to identify applications. However, using only TLS attributes resulted in classifiers with inferior performance. When we analysed the results, we found that the source data contained many overlapping samples (see Section III-B), which negatively affected the metrics of the trained classifiers.

Despite the limited amount of data available for the experiments, it is apparent from the results that the traditional fingerprinting approach can achieve reasonable performance in the task of identifying the candidate application for the TLS connection. This is obvious because the method is based on exact matching of known fingerprints. Due to the characteristics of the input data, the ML-based approach does not help in the case of previously unseen samples, and due to the overlapping samples, it is even worse overall than the JA4+JA4S fingerprinting.

B. Deployment Issues

While achieving high accuracy in identifying individual applications through encrypted traffic analysis is challenging, identifying a collection of candidate applications is easier, but beneficial for network monitoring. In real-world scenarios, additional heuristics, such as IP address-based Internet service identification, can complement and refine the detection process, improving overall accuracy. This approach allows network administrators to narrow down potential applications and services, which is particularly useful in large networks.

In practice, all of the proposed methods require regular updates of the fingerprint database or retraining of the models. This ongoing maintenance task is non-trivial and requires significant resources, especially as applications are frequently updated and evolve. Ensuring that the fingerprinting method remains accurate and up to date is critical to maintaining its effectiveness in identifying encrypted traffic.

All of the methods evaluated rely on the TLS connection information, which can be achieved in flow-based monitoring environments through approaches such as IPFIX [17], where network probes analyze and extract selected information from TLS handshakes. Once this information is captured, any of the presented identification methods can be effectively applied. While the machine learning approach is more computationally intensive during training, the performance of all the methods discussed is efficient enough to allow real-time deployment in operational environments. This makes them suitable for use in live network monitoring systems where timely detection of application traffic is critical for security monitoring.

VII. Conclusion and Future Work

Identification of network applications is one of the desired features of network monitoring tools. We have tested methods based on fingerprinting techniques such as JA3/JA4, SNI-to-application mapping, and ML-based classification. Our experiments showed that accurate identification of individual applications is challenging. However, for administrators, identifying a set of possible applications provides valuable insight.

Among the fingerprinting methods, the combination of client-side and server-side parameters proved effective in distinguishing between a range of applications, with the JA4 and JA4S methods achieving accuracy rates above 90%. In contrast, the older JA3/JA3S fingerprints showed lower performance, mainly due to limitations in the way their hashes are computed, rendering them inapplicable. The accuracy of ML-based detectors varies with the input data set.

Further feature engineering and the use of larger datasets are required to improve performance. Nevertheless, ML-based classifiers offer the advantage of reducing the number of unknown results, which is an inherent advantage of their underlying principles compared to traditional fingerprinting techniques. For comparison, we also evaluated the SNI-based method, which takes advantage of the fact that applications often use unique SNI values to identify their servers. As expected, the SNI method gave the best results for single application identification tasks.

Our evaluation was limited to two available datasets, which limits our ability to assess the stability and robustness of the methods under different conditions. Future research should test the methods on a wider range of datasets, including those with different network environments and application types, to provide a more comprehensive evaluation. In addition, the automated generation of annotated datasets of mobile and Windows applications can only partially capture the behaviour of these applications.

Future work may include the development of more sophisticated methods for annotating connections in real-world scenarios that capture a wider range of application activities and interactions. This would result in richer data sets that more accurately reflect actual patterns. The ML-based classification approach used a simple methodology that leaves significant room for improvement, particularly through the use of advanced feature engineering techniques that better capture the nuances in the TLS characteristics of applications. In addition, alternative techniques to improve classification accuracy can be explored to address the issue of class imbalance.

Finally, while the methods have shown promise in controlled environments, their performance in real-world settings has yet to be thoroughly evaluated, for example by deploying these methods in live environments to test their effectiveness under real-world conditions.

Acknowledgment

This work is supported by the project “Flow-based Encrypted Traffic Analysis (FETA)”, 2022-2025, no. VJ02010024, funded by the Ministry of the Interior of the Czech Republic, and the internal BUT project ”Smart Information Technology for a Resilient Society”, 2023-2025, no. FIT-S-23-8209. The authors would like to thank to Kristián Kičinka, who participated in the creation of fingerprints.

References

[1] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware’s use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, 2018.

[2] Michael J. de Lucia and Chase Cotton. Detection of encrypted malicious network traffic using machine learning. In *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2019.

[3] Onur Barut, Rebecca Zhu, Yan Luo, and Tong Zhang. Tls encrypted application classification using machine learning with flow feature engineering. In *10th In. Conf. on Communication and Network Security*, ICCNS ’20, page 32–41, New York, NY, USA, 2021. ACM.

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

[5] Blake Anderson and David A. McGrew. Accurate TLS fingerprinting using destination context and knowledge bases. *CoRR*, abs/2009.01939, 2020.

[6] Petr Matoušek, Ivana Burgetová, Ondřej Ryšavý, and Malombe Victor. On Reliability of JA3 Hashes for Fingerprinting Mobile Applications. In *Digital Forensics and Cyber Crime, ICDF2C 2020*, volume 351 of *LNICST*, pages 1–22. Springer, 2021.

[7] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning, 2018.

[8] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48, 2017.

[9] Haipeng Yao, Chong Liu, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Shui Yu. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data*, 8(1):241–252, 2022.

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

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

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

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

[14] S. Friedl, A. Popov, A. Langley, and E. Stephan. *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension*. IETF RFC 7301, July 2014.

[15] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of Encrypted and VPN Traffic using Time-related Features. In *2nd Int. Conf. on Information Systems Security and Privacy*, pages 407–414. SciTePress, 2016.

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

[17] B. Claise, B. Trammel, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, IETF, September 2013.