Using TLS Fingerprints for OS Identification in Encrypted Traffic

Martin Laštovička*†, Stanislav Špaček*†, Petr Velan* and Pavel Čeleda*
*Masaryk University, Institute of Computer Science, Brno, Czech Republic
†Masaryk University, Faculty of Informatics, Brno, Czech Republic
Email: {lastovicka|spaceks|velan|celeda}@ics.muni.cz

***Abstract*—Asset identification plays a vital role in situational awareness building. However, the current trends in communication encryption and the emerging new protocols turn the well-known methods into a decline as they lose the necessary data to work correctly. In this paper, we examine the traffic patterns of the TLS protocol and its changes introduced in version 1.3. We train a machine learning model on TLS handshake parameters to identify the operating system of the client device and compare its results to well-known identification methods. We test the proposed method in a large wireless network. Our results show that precise operating system identification can be achieved in encrypted traffic of mobile devices and notebooks connected to the wireless network.**

I. Introduction

The first step in understanding the situation in a network is obtaining the knowledge of what devices connect to it. However, this is not easily fulfilled, especially in unmanaged networks where devices can connect without any network access control mechanism. These devices are typically identified through dynamically assigned IP addresses. Administrators and researchers address this issue with device identification (fingerprinting) from network traffic. Most methods leverage the HTTP User-Agent strings, which directly describe the device and are inherently present in device communication. However, the current trends in communication evolution go in the direction of user privacy. They tend to move as much of the traffic as possible into encrypted payloads to counter the identification and tracking.

A typical solution to the encryption challenge is to scan the devices in the network actively or to inspect the encryption handshake parameters from packets using deep packet inspection. The contribution of our work lies in the analysis of TLS (Transport Layer Security) handshake parameters suitable for OS (Operating System) identification in IP flows. This shift towards passive flow identification allows security administrators to asses the OS of each device even in large networks. Moreover, enabling identification from encrypted traffic will ensure the identification usability in the ever-evolving network communication.

First, we describe in detail the parameters of TLS handshake and the nuances of distinct versions of the protocol and how TLS 1.3 [1] redefined the semantics of some of the data fields. The information from the handshake is incorporated into network flows, exported using IPFIX [2], and further processed. We employ machine learning classifier to build a model mapping the TLS parameters feature vector to the OS of the client device. We further explore the OS identification from encrypted traffic by extending our previous work on TCP/IP parameters fingerprinting [3] where we replaced manual statistical analysis with a decision tree algorithm.

We evaluate the methods on a large dataset collected from campus wireless networks. Students and employees can bring and connect any device which guarantees high diversity of the devices. We pair records from infrastructure servers logs to the collected traffic to establish ground truth and calculate the accuracy metrics of our methods. Furthermore, we measured OS identification results for two established methods (i.e., based on HTTP User-Agent and connections to specific domains) to provide a comparison with methods based on plaintext traffic.

II. Related Work

We surveyed the state of the art in the field of device fingerprinting in our previous work on this topic [3]. Hence, we focus mainly on the new contributions in the area. Shen et al. [4] published an article concerning the fingerprinting of devices in the industrial control system environment. Sanchez et al. [5] proposed hardware features, specifically the internal clock signals, to discern different devices. However, none of these publications deals with the challenge to identify the operating system of a device from network traffic, posed by the currently common encryption.

The recent approach to network communication is favoring user privacy and promotes encrypting as much of the transferred data as possible. The purpose is to hide the content and other relevant information about the transfer that might be captured and analyzed by an external observer. However, even encrypted connections might disclose some information about the participants and the purpose of the transfer [6]. During the handshake, before the encrypted connection is established, all the exchanged data can be seen unencrypted.

The TLS handshakes have already been analyzed in the past. Anderson et al. [7] provide a comprehensive study of malware's use of TLS by observing the unencrypted TLS handshake messages. They also identify handshake features, that can be used to cast some light on the data transferred by TLS, e.g., Server Name Indication (SNI), and server certificate. The usage of SNI for flow analysis was further explored by Shbair et al. [8]. However, these articles focus more on the server side of the handshake, while the identification of connected devices is based on the client-side parameters.

Identification of client applications using TLS handshake monitoring was proposed by Korczyński et al. [9]. The authors showed the identification from encrypted data to be possible. A client device identification algorithm based on the specific parameters of the TLS handshake was proposed by Husák et al. [10]. They utilized the simultaneous monitoring of the HTTP and HTTPS connections to create a dictionary, pairing the User-Agent from HTTP with the TLS handshake from HTTPS connections generated by one device. The dictionary was then used to assign User-Agents to HTTPS connections captured in the network.

III. Network Data Acquisition

To be able to identify the operating system of a particular device from its network traffic, we need to measure and analyze relevant features from the traffic of the device. For this purpose, we utilize IPFIX data provided by a flow monitoring system [2]. The flow monitoring system uses multiple Flowmon probes [11] and IPFIXcol flow collector [12].

The accuracy of any identification method depends on the number and quality of features used as an input. We use several different features in this work: TCP/IP parameters of observed connections, values from plaintext HTTP headers, and parameters of TLS connections. The rest of this section describes the features we use for OS identification.

A. TCP/IP Features

Each flow record contains the basic fields, i.e., flow start time, flow end time, source IP address, source port, destination IP address, and destination port. Existing analyses show that the size of TCP SYN packet, initial TCP window size, and TTL (Time to Live) value of packets differ between operating systems. Therefore, these fields are recorded as well for the flows describing TCP connections.

B. HTTP Headers

Although most of the HTTP traffic is secured by TLS protocol nowadays, there are still services using plaintext HTTP. When a device communicates using this protocol, the flow probe analyses the HTTP request header and extracts the visited domain name from the URL and the User-Agent field. Since there are domains tied to operating system specific update services, they can be used as an indication of the used OS. The User-Agent field usually contains not only the name of the application but also the platform (i.e., operating system) on which it is running. Moreover, some applications, such as antivirus software, are often platform-specific, and its presence can reveal the used OS as well. Therefore, the HTTP User-Agent is often used for OS identification.

C. TLS Handshake

To identify the operating system of devices that communicate through encrypted protocols, we have to rely on the information remaining in cleartext. Since the unencrypted initialization phase precedes every encrypted communication, as shown in Figure 1, we can inspect the negotiation (handshake) between the client and the server.

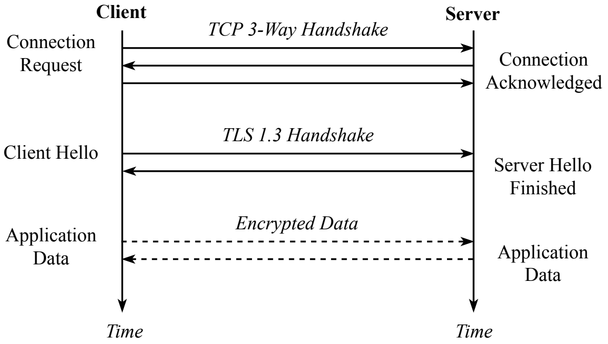

Fig. 1. TLS 1.3 handshake – negotiation of the encrypted connection.

The first parameter we explore is the *Client Version* of the TLS protocol sent by the client. The version influences further parameters and extensions the client uses, which plays an essential role in the OS identification. However, stating the client version from *Client Hello* message is not as straightforward as one would expect. Every version of TLS is identified differently due to backward compatibility, and each version sends 0x0301 bytes (TLS 1.0) in the record header field *Version* of the TLS handshake which specifies the version of TLS used. Then the client constructs the *Client Hello* header, in which it fills in another Version field identifying the TLS version of the client application. This identification was valid until TLS 1.3 was introduced. TLS 1.3 clients send their version as 0x0303 (TLS 1.2) [1], and their correct version is located in extensions of the Client Hello message. Specifically, in the extension *Supported Versions* it sends a list of all versions it can use, and one of the versions is the TLS 1.3. This leads to the current situation when TLS 1.3 clients use identifiers of three different versions of the protocol. Flowmon monitoring probes rely on the Version field from Client Hello header and export this value into the flow. Hence, probes correctly identify TLS versions 1.0 to 1.2, and TLS 1.3 clients are exported as TLS 1.2 flows. To distinguish those two versions, we look at extension types list and look for extension number 43 Supported Versions, which is exclusively and mandatorily used by TLS 1.3 clients. However, this version mapping was not implemented in the version (v10.02.05) of the Flowmon probe used for our experiments.

The *Cipher Suites* is a field from Client Hello header which specifies a list of supported encryption algorithms together with the key length and hash algorithm to be used. This list is ordered descendingly according to the client preference, and we assume this preference can help identify the underlying operating system. The TLS 1.3 specification defines only five cipher suites to be used with TLS 1.3, but the clients usually append more suites at the end of the list to ensure compatibility with older servers. The list has variable length depending on the client and the flow exporter stores only the first 16 bytes of the field to the flow. As a result, we have IDs of the first eight cipher suites most preferred by the client.

Similarly to cipher suites, the client can offer named groups for the key exchange and cryptography based on elliptic curves. This upgrade was introduced in TLS 1.2 as *elliptic_curves* extension and defined 25 named curves to chose from [13]. TLS 1.3 specification then reduced this number to only five supported curves, but added the option to use finite field groups and defined five groups with audited parameters resistant to known attacks [14]. This change also led to the renaming of the extension to *supported_groups* and introduced ordering with the most preferred group first. Flowmon exporter store the IDs of the first eight groups in the flow records.

Other parameters parsed from the TLS handshake are the extension types and lengths. The extensions can specify additional options for the handshake and greatly varies between TLS versions and implementations. IANA maintains a list of known extensions [15]; however, in the real traffic, we can see unassigned extension types in use. Most of those are so-called GREASE (Generate Random Extensions And Sustain Extensibility) values which are nowadays only an Internet-Draft [16] but already deployed in many TLS implementations. Flowmon exporter parses the extensions and stores IDs of the first 23 extensions used together with a list of their lengths.

The final parameter extracted from the TLS handshake is the value of Server Name Indication extension [17]. It is present in almost every HTTPS communication to differentiate between multiple virtual servers so that the server know which TLS certificate to send to the client. We treat the extracted server name the same way as the domain name (host) extracted from HTTP headers. Table I shows the features we measure from the traffic using flow monitoring and use for passive OS identification.

TABLE I
FEATURES EXTRACTED FROM NETWORK DATA.

TCP, HTTP Flow Features	TLS Flow Features
TCP SYN packet size	TLS server name indication
TCP window size	TLS client version
TTL of TCP SYN packet	TLS cipher suites
HTTP User-Agent	TLS extension types
HTTP hostname	TLS extension length
	TLS supported groups
	TLS elliptic curves point formats

IV. Dataset

To test the OS identification methods on real-world data, We measured the flow data from the university uplink to the Internet. The dataset consists of data from three different sources; flow records collected from the university backbone network, log entries from the two university DHCP (Dynamic Host Configuration Protocol) servers and a single RADIUS (Remote Authentication Dial In User Service) accounting server. The data was collected from 2019-07-12 00:00 to 2019-07-16 23:59 with a few hours overhead on both sides of the interval for the log entries to cover long connection sessions overlapping to and from the time frame. We made the anonymized dataset publicly available on the Zenodo platform [18]. In the dataset, we kept only flows with source IP addresses from university wireless networks (Eduroam). This step significantly reduced the amount of data and left only the relevant flows to identify the OS of devices in our network, which we can enrich with information from DHCP and RADIUS servers.

The DHCP log data was chosen as the ground truth for our experiment. The log typically archives the information that connects a unique MAC (Media Access Control) address of a specific device with the IP address it got assigned within a specific time frame. However, it is not possible to estimate the length of the session initiated by a specific device just from the DHCP log. A device might end the connection long before its IP lease time expires and the DHCP server does not log this action. The RADIUS accounting logs supplement the DHCP logs in this regard, as they contain the *session start* and *session end* parameter for a specific device identified by its IP and MAC address.

The DHCP log data was collected from the two central PPPoE concentrators of the university network. The original DHCP logs contained a large amount of data not relevant or redundant to OS identification, so we applied several filters to keep them as concise as possible. The resulting DHCP log includes following parameters: *timestamp*, *IP address*, *IP range*, *MAC address*, and *hostname* of the client device. The *timestamp* denotes the precise time of the request and with the requested *IP address* and the device's *MAC address* allows pairing an address to a specific device within the given time frame. The information that identifies the device's operating system is provided by the *hostname* parameter. We discovered that some of the DHCP REQUESTS in the dataset demanded IP addresses that do not belong to any of the known Eduroam address pools. We removed these addresses and to confirm that the device requests a valid Eduroam IP address, we specify for each one the corresponding Eduroam address pool.

After that, all data sources needed to be combined to connect the captured sessions with the ground truth. At first, we have correlated both logs sources. We created a log pair if both logs entries contained the identical IP and MAC addresses and the DHCP REQUEST timestamp lied within the interval of RADIUS start and end timestamps with a tolerance of one minute to deal with possible time dyssynchronization of logging servers. This correlation resulted in tuples containing *ID*, *IP*, *MAC*, *device name*, *start time*, *end time*, and *ground truth OS* derived from the hostname. Finally, we have enriched every flow record with the session ID and ground truth. The key was the flow source IP address and flow start timestamp, which had to fall exactly into the session time interval. We experimented with different time tolerances for this mapping, but even tolerances of tens of minutes added only a negligible number of unpaired flows (i.e., ten minutes tolerance added 0.38 % of flows). Altogether, our dataset consists of or is a result of the activity of:

- 18 708 983 enriched flows,
- 10 734 unique users,
- 45 602 unique Wi-Fi sessions,
- 11 962 unique MAC addresses,
- 8 071 unique IPv4 addresses assigned.

V. OS Identification Methodology

We utilize four different methods to identify the OS of each flow record. In this section, we describe the processing of collected data, computing ground truth, and the settings of the identification methods with a focus on the machine learning algorithms. A brief overview of the identification level of detail for each approach is presented in Table II.

TABLE II
OS identification methods level of detail

Method	Vendor	Name	Major Version	Minor Version
TCP/IP parameters	✓	✓	(✓)	(✓)
TLS handshake	✓	✓	(✓)	(✓)
User-Agent	✓	✓	✓	✓
Specific domains	✓	✓	✗	✗
Ground truth	✓	(✓)	✗	✗

A. Preprocessing

Machine learning algorithms were used for two methods, TCP/IP, and TLS, and they require data transformations before the learning or classification phase. We had already experimented with machine learning for TCP/IP parameters in our previous work [19] and the methodology stayed the same. The three selected features (i.e., IP TTL, TCP Window Size, and the size of initial TCP SYN packet) are all numerical values, and during preprocessing we only round up the TTL value to the nearest higher power of two according to Lippmann et al. [20] to remove the influence of monitoring probe location.

In the case of TLS, we treat each feature as categorical. The TLS version is an identifier which numerical value has no real meaning, and the semantic of order relation on numbers does not hold. The *cipher suites* and *supported groups* are both ordered lists of IDs where the ordering is relevant for the identification as it represents the preference of the client. Similarly, we take the lists of extensions IDs and their lengths as ordered to keep their semantics and position in the original packet. During the preprocessing, we encode each feature into a binary vector using one-hot encoding. The encoding ensures all features retain their information value and that the encoding does not introduce any new relations as if the values would be treated as numbers. The encoder is persistently stored and used on both learning and testing datasets.

For the User-Agent method, we did not use any preprocessing. For the specific domains method, the information from SNI is exported as a binary vector and HTTP host (domain name) as a string. Therefore, the SNI values were converted to string as well during the preprocessing.

B. OS Identification

The TCP/IP and TLS methods use a Decision tree to classify the flows with labels corresponding to a specific minor version of the OS. To train the classifier, we use the methodology proposed by Husák et al. [10] and further extended by Matoušek et al. [21] for flow monitoring. We pair the TCP/IP parameters and User-Agents directly as they are present in the same flow. For TLS, we pair HTTP and HTTPS requests from the same device. Finally, we split the dataset into a training one consisting of the annotated flows from the first day of collected traffic. The rest of the flows were used as testing dataset. Specific domains and User-Agent method stayed the same as presented in previous paper [3] and could serve as a comparison of traffic evolution in time.

C. Ground Truth

The ground truth of our experiment is based on the data obtained from combining the DHCP and RADIUS auditing logs, specifically, from the *hostname* parameter of DHCP REQUEST events. Using this parameter to establish the devices' OS works well for Apple and Google operating systems. Those use identifiable device name set by default, and the user is usually unable to change it freely. On the other hand, the prevailing desktop operating systems, Windows and Linux, are not as easily discerned. The Windows hostname is readily editable, and by default, it is derived from the user name during OS installation. The Linux hostname represents a similar case; it is easily editable, and its default value is set during installation, but also may vary for different Linux distributions.

Despite the aforementioned shortcomings, the ground truth gained from DHCP log's *hostname* should prove sufficient for our experiment. The university network that we collect the dataset from, Eduroam, is a wireless network environment, so mobile devices, whose operating systems' are rather easily determined, prevail over the desktop ones by a large margin.

VI. Results

In this section, we present how the implemented OS identification methods performed on our dataset. We also include basic statistics of the traffic based on the usage of different operating systems and versions of TLS and SSL protocols.

A. OS Identification Coverage

The first important measure of identification method capabilities is its ability to identify the operating system from available data regardless of the result accuracy. We have evaluated the coverage from two points of view. The first one represents the ratio of flow records containing all features needed for the identification method. The second is an aggregation of flows into connection sessions where a session represents all flows since the device connected to the network until it disconnected. If the device sends at least one flow with all required features during the session, the device OS for the whole session can be identified.

The coverage results are summarized in Figure 2. A generally working method proved to be the TCP/IP parameters as it depends on network and transport layer information and over 71 % of the captured traffic used TCP, and only a marginally low number of sessions did not establish any TCP connection. Similarly, almost every device (97.1 %) sent at least one TLS

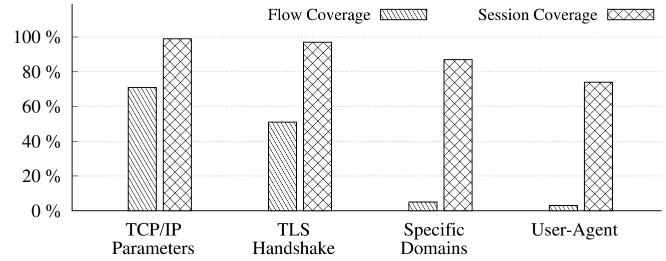

Fig. 2. Coverage of operating system identification methods.

handshake message, and the TLS traffic was responsible for more than half (50.86 %) of the flows. The amount of traffic for the other methods is significantly lower. As of Specific domains and User-Agent parsing, the number of flows is 5 %, resp. 3 %. Even so, they were able to identify the OS for 87 %, resp. 74 %, of the connections of the devices. This ratio indicates their usability in large networks as they can filter out a large amount of traffic before the identification and still keep high coverage.

B. OS Identification Accuracy

We measure the identification accuracy using standard performance metrics of accuracy, precision, recall, and F-score. To deal with the multi-class classification of our methods, we calculated the confusion matrix with true positive (TP), false positive (FP), true negative (TN), and false negative (FN) values for each of the l classes (OS names), treating each class as separate binary classifier. We assigned classification prediction as TP if the ground OS name of the flow matched the OS name of prediction regardless of the version as ground truth does not cover this level of detail. From those values, we calculated average accuracy and micro averaging of precision, recall and F-score ($\beta = 1$) according to Sokolova et al. [22].

Detailed results of our experiments are listed in Table III, and for visual comparison also on Figure 3. At first, we discuss the results of repeated measurements of unchanged methods (i.e., User-Agent and Specific domains) followed by a description of the new or upgraded ones. The User-Agent method produced the best and most consistent results. It is based directly on the OS filled-in by the device itself and the number of devices (intentionally) sending wrong information in this field is very low, which is consistent with our previous results. Also, Specific domains method provides consistent results but with much lower F-score.

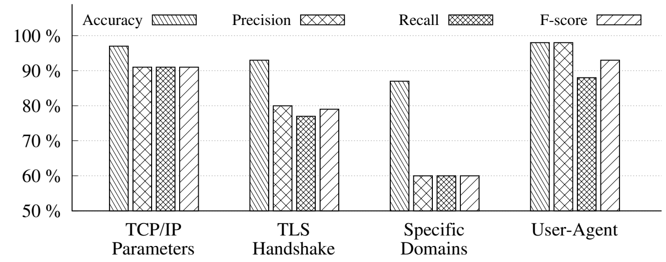

Fig. 3. Micro averaging of accuracy metrics.

TABLE III
MICRO AVERAGING OF ACCURACY METRICS

Method	Accuracy	Precision	Recall	F-score
TCP/IP parameters	0.9711	0.9137	0.9130	0.9133
TLS handshake	0.9312	0.8048	0.7749	0.7896
Specific domains	0.8659	0.5978	0.5974	0.5976
User-Agent	0.9764	0.9797	0.8763	0.9251

Our method based on TCP/IP parameters change was twofold. The error in flow exporter which was present in our previous work was fixed and it significantly increased the coverage and provided correct data for prediction. The second change was the shift from statistical analysis to machine learning which improved the accuracy metrics notably. The precision and recall over 90 % and almost complete coverage makes TCP/IP method widely usable. Its main drawback is in distinguishing versions of the same OS core used. Examining the predictions, we found out that the primary source of false positives and false negatives are the OS names *iOS* and *MAC OS* which the classifier has troubles to distinguish as Apple uses too similar parameters.

The new method based on TLS handshake parameters generally proved very good results with the accuracy metrics around 80 %. Surprisingly, this method was not able to identify a single flow from *Windows Phone* and classified every one of them as different versions of desktop Windows.

C. Traffic Statistics

Operating systems identified in the dataset are depicted on Figure 4. Google Android takes the first place with 39.93 % followed by Apple iOS and MAC OS with 30.03 % and Microsoft Windows with 28.49 %. The remaining operating systems (1.55 %) represent several Linux distributions and minor mobile devices vendors (e.g., BlackBerry).

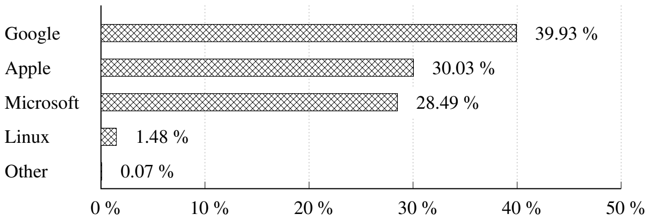

Fig. 4. Operating system usage share grouped by vendor.

Figure 5 shows the use of TLS versions among the clients. Recent TLS versions 1.2 and 1.3 dominate the network traffic. SSL was used only in 1887 flows with 292 of them in version 2.0; the rest was SSL 3.0.

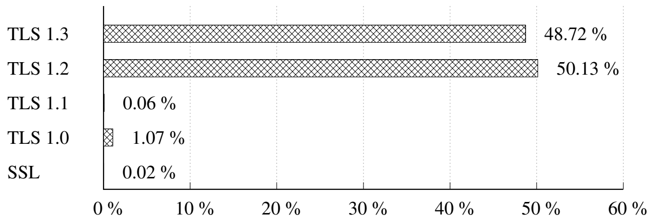

Fig. 5. TLS and SSL protocol versions used by clients.

VII. Conclusion

In this paper, we proposed a method of passive identification of the operating system based on flow monitoring data that leverages information from TLS handshake. To support the idea of OS identification in encrypted traffic, we enhanced OS identification from TCP/IP parameters by exploiting machine learning algorithms. Finally, we repeated our experiment with OS identification using Specific domains and HTTP User-Agent for comparison of the new methods to established ones.

Our results prove that the OS identification from encrypted traffic is possible, and the used methods exhibited high accuracy metrics. The method based on TCP/IP parameters is comparable to unencrypted User-Agent identification with F-score 91.33 % (compared to 92.51 % of User-Agent method). The method based on TLS handshake parameters performed a bit worse with accuracy metrics around 80 %, however, excelled in the coverage. It was able to identify more than 97 % of the devices connected to the network, which is significantly better portion than the Specific domains or User-Agent methods could achieve.

Concerning lessons learned from the experiments, we would argue that methods for OS identification are mature enough and work in dynamic networks with the majority of traffic encrypted. However, data acquisition is becoming more complex. The source flow data need to be enhanced with information from applications protocols which are continuously evolving and changing the specifications of the data fields from previous versions. This evolution requires the flow exported to be continuously updated as new protocols and protocol versions are created. Also, the correlation of data from multiple data sources required a lot of manual work and the use of heuristics to correctly match log records to corresponding flows. In our future work, we plan to focus on the automation of data retrieval from the infrastructure elements and their normalization for (near) real-time flow annotation. We will also keep our close cooperation with Flowmon Networks to apply results of this research in their monitoring solution.

Acknowledgement

This research was partly supported by the CONCORDIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927 and partly by the ERDF project "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822). Martin Laštovička is Brno Ph.D. Talent Scholarship Holder – Funded by the Brno City Municipality.

References

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Internet Engineering Task Force, Aug. 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446

[2] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, Fourthquarter 2014.

[3] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive OS Fingerprinting Methods in the Jungle of Wireless Networks," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[4] C. Shen, C. Liu, H. Tan, Z. Wang, D. Xu, and X. Su, "Hybrid-augmented device fingerprinting for intrusion detection in industrial control system networks," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 26–31, 2018.

[5] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock Around the Clock: Time-Based Device Fingerprinting," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, 2018, pp. 1502–1514.

[6] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A Survey of Methods for Encrypted Traffic Classification and Analysis," *Netw.*, vol. 25, no. 5, pp. 355–374, Sep. 2015.

[7] B. Anderson, S. Paul, and D. McGrew, "Deciphering malwares use of TLS (without decryption)," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 195–211, 2018.

[8] W. M. Shbair, T. Cholez, A. Goichot, and I. Chrisment, "Efficiently bypassing SNI-based HTTPS filtering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 990–995.

[9] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 781–789.

[10] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 6, Feb 2016.

[11] Flowmon Networks. Flowmon Probe. [Online]. Available: https://www.flowmon.com/en/products/flowmon/probe

[12] P. Velan and R. Krejčí, "Flow Information Storage Assessment Using IPFIXcol," in *Dependable Networks and Services*, ser. Lecture Notes in Computer Science, vol. 7279. Springer, 2012, pp. 155–158.

[13] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)," RFC 4492, Internet Engineering Task Force, May 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4492.txt

[14] D. K. Gillmor, "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)," RFC 7919, Internet Engineering Task Force, Aug. 2016. [Online]. Available: https://tools.ietf.org/html/rfc7919

[15] Internet Assigned Numbers Authority. (2019, Sep.) Transport Layer Security (TLS) Extensions. [Online]. Available: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml

[16] D. Benjamin, "Applying GREASE to TLS Extensibility," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-grease-04, Aug. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-grease-04

[17] D. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions," RFC 6066, Internet Engineering Task Force, Jan. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6066

[18] M. Laštovička, S. Špaček, P. Velan, and P. Čeleda, "Dataset Using TLS Fingerprints for OS Identification in Encrypted Traffic," Sep 2019. [Online]. Available: http://doi.org/10.5281/zenodo.3461771

[19] M. Laštovička, A. Dufka, and J. Komárková, "Machine learning fingerprinting methods in cyber security domain: Which one to use?" in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2018, pp. 542–547.

[20] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, "Passive operating system identification from TCP/IP packet headers," in *Workshop on Data Mining for Computer Security*, 2003, p. 40.

[21] P. Matoušek, O. Ryšavý, M. Grégr, and M. Vymlátil, "Towards Identification of Operating Systems from the Internet Traffic: IPFIX Monitoring with Fingerprinting and Clustering," in *2014 5th International Conference on Data Communication Networking (DCNET)*, Aug 2014, pp. 1–7.

[22] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.