Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 220 (2023) 94–101

Procedia Computer Science

www.elsevier.com/locate/procedia

The 14th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 15-17, 2023, Leuven, Belgium

Categorizing TLS traffic based on JA3 pre-hash values

Jenny Heino[a,b,*], Antti Hakkala[a], Seppo Virtanen[a]

[a] *Department of Computing, University of Turku, 20014 Turku, Finland*
[b] *Forcepoint LLC, Itälahdenkatu 22 A, 00210 Helsinki, Finland*

Abstract

The JA3 algorithm for fingerprinting TLS client traffic has become a popular additional tool in the tool set of network security professionals. The pre-hash value of the JA3 fingerprint lists parameter values from the TLS handshake supported by the TLS client. In this paper we present two different machine learning methods for identifying the endpoint application from TLS traffic based on the JA3 pre-hash string. Both methods were able to identify applications from Mozilla in our sample set, but had more variation with other applications. The methods can be used for improving network security accuracy.

Keywords: Traffic categorization; Network security; TLS; Endpoint aware inspection

1. Introduction

Gaining a better overall visibility into the traffic in your monitored network is a goal that anyone responsible for network security can stand behind. Understanding what is normal in your network helps identify and filter the abnormal traffic. By being able to categorize traffic, for example by identifying the endpoint applications initiating connections in the network, it is possible to gain valuable information about the network. In a network security solution, it can help with providing more accurate deep packet inspection based security.

One method of gaining information about the endpoint application from TLS traffic is the JA3 fingerprinting algorithm. The JA3 algorithm is a network protocol hash fingerprinting algorithm for TLS clients, and a fairly recent addition in the tool set of network security specialists. The algorithm collects a set of distinctive and significant values from the TLS Client Hello message, concatenates these values into a so-called pre-hash string, and finally calculates an MD5 hash from it. It is a quick method for observing which TLS clients have previously produced the same fingerprint, and in some cases even identify the endpoint application which has initiated the TLS connection. However, as the JA3 fingerprint itself is an MD5 hash calculated from the pre-hash string, even a minimal change

* Corresponding author Jenny Heino.
E-mail address: jeahei@utu.fi

1877-0509
10.1016/j.procs.2023.03.015

in the original feature set, such as one additional extension, will produce an entirely different fingerprint. Because of this characteristic of the fingerprint resulting from the MD5 hash, we have in previous work argued that, instead, it is useful to focus on the pre-hash string of the JA3 fingerprinting algorithm [1]. The information available in the JA3 pre-hash value is a fruitful ground for metadata about the network connection. As this value is already calculated and stored by many network analysis tools, including Wireshark [2], a generic visualization and categorization tool based on the JA3 pre-hash fingerprint could be utilised by many network professionals.

In this paper, we present two machine learning methods for categorizing TLS traffic based on JA3 pre-hash strings that have been collected from a monitored network. We use Latent Dirichlet Allocation (LDA) and K-Means clustering to categorize the data based on the endpoint application. To validate our methods, we have crafted pre-trained example models as well as the tools for creating new models from new data. The tools are freely available in Github [3]. In our initial tests, both example models were able to correctly categorize the endpoint applications developed by Mozilla, while having more variation with other endpoint applications. This is most likely due to the wide use of the Chromium Embedded Framework.

Our incentive to develop the methods arises from the need to improve the accuracy of deep packet inspection based security measures in network security solutions. The occurrence of false positive and false negative assessments reduces the usability of any security solution every time, and thus new methods for reducing their occurrence are always needed. We believe that gaining a better awareness of the communicating endpoints can be a valuable tool for reducing the occurrence of false positive and false negative security assessments. Many false positives and false negative assessments could be avoided if the security measures were better aimed specifically at the endpoint applications that are at risk of being compromised by the traffic.

The structure of the rest of the paper is as follows. We first take a brief look into related work regarding categorization of TLS traffic in Section 2. We then introduce the concept of hash fingerprinting in Section 3, with a more detailed look into the JA3 fingerprint. In Section 4 we introduce the methods and tools we have selected for categorizing the traffic based on the JA3 pre-hash string. In Section 5 we demonstrate the results from the pre-trained methods and sample data set included with the tools, and provide a few notes regarding our findings and potential future uses. Finally, in Section 6 we conclude our paper.

2. Related work

As our interest has especially centered around the JA3 fingerprinting algorithm for TLS client traffic, we focused our scan for related work on TLS traffic. We were interested in seeing how different aspects of the TLS traffic, especially the TLS client handshake, have been used for categorizing traffic. We also wanted to understand how accurately the features selected for the JA3 pre-hash string can be used to identify the original endpoint application.

In 2016, Husák *et al.* [4] experimented how well the list of supported cipher suites in the TLS Client Hello message can be used for identifying the original HTTPS client. The researchers first created a dictionary mapping the list of supported cipher suites to the corresponding User-Agent. After this, they compared their dictionary to the traffic from the campus network at Masaryk University. They observed that the dictionary they had collected could be used to map the TLS connections in the network to a User-Agent string for 99,6% of the connections.

In 2017, Muehlstein *et al.* [5] analysed how a large amount of encrypted TLS traffic can be mapped to an operating system, browser and application purely based on the encrypted data. The researchers used a more detailed data set than what was used by [4]. Their data set did include information from the TLS Client Hello message, such as the cipher suites and the extensions, but they also used information from the TCP connection, such as packet sizes and traffic flow bursts. The accuracy of their classification method is 96,06%. However, due to the use of extended features, their result is not entirely relevant for us.

We were also interested in using Levenshtein distance for calculating how close two JA3 pre-hash strings are to each other. For this, we considered existing papers evaluating usage of Levenshtein distance for different features of a TLS handshake. We found two especially interesting papers approaching this topic. In 2019, Frolov and Wustrow [7] used Levenshtein distance for clustering TLS traffic. The features they collect from the TLS Client Hello are to some extent similar as the features included in the JA3 pre-hash string, such as the supported cipher suites, supported groups and elliptic curve point formats. They, however, include some additional information in their analysis as well, such as the contents from the ALPN extension (Application Layer Protocol Negotiation), and the GREASE extensions (Generate

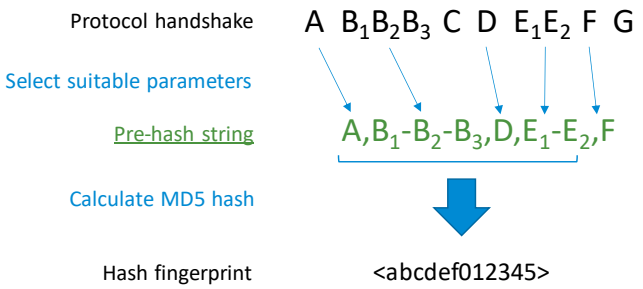

Fig. 1. Procedure for generating a hash fingerprint.

Random Extensions And Sustain Extensibility). By using Levenshtein distance for clustering the traffic, they were for example able to distinguish one cluster comprising of various versions of Microsoft Exchange, and another cluster comprising of fingerprints from Google Chrome version 65. Interestingly, the cluster for Google Chrome contained two loosely connected sub-clusters, which the researchers were able to map to connections following the early draft of TLS 1.3 and to connections following the current standard.

Later in 2019, Anderson and McGrew [6] also performed an analysis on fingerprinting TLS traffic, and used Levenshtein distance for calculating the distance. The information included in their TLS fingerprint is very close to the information contained in the JA3 pre-hash string: it includes the list of cipher suites, the list of extensions, as well as information from suitable extensions, such as supported groups. There are some differences to the JA3 pre-hash string, however, such as that the researchers also utilise the GREASE extensions to some extent. By using Levenshtein distance for considering if a previously unknown fingerprint is ”close” to an existing fingerprint, the researchers were able to cover nearly all observed TLS connections after a year of data collection. However, the researchers also performed leave-one-out cross-validation on the top 100 fingerprints for which they had endpoint information. When comparing to the approximate fingerprint's process list, the researchers noted that the actual target processes were included on the list only approximately 73.5% of the time. The researchers still note that using Levenshtein distance for this purpose can be an appropriate method of last resort.

3. Hash fingerprinting

The concept of network protocol hash fingerprinting originated with the first hash fingerprinting algorithm, JA3 [8]. This algorithm paved way for other similar algorithms for other network protocols. In this section, we first give a high level description of the concept of hash fingerprinting in Subsection 3.1. In Subsection 3.2 we take a more detailed look at the JA3 fingerprint, and then in subsection 3.3 we take a brief look into the other hash fingerprinting algorithms.

3.1. Hash fingerprinting Concept

On the application layer of the network protocol stack, when two endpoints begin a network connection, they first perform a handshake. For many network protocols, the handshake process makes it possible for the endpoints to find common features that both support. In these cases commonly the client endpoint will first list all features it supports, and out of these features, the server endpoint will then select the ones it supports. The hash fingerprinting algorithms focus on such network protocols.

In the network protocol hash fingerprinting algorithms, a protocol specific set of distinctive and significant parameters are selected from the handshake message. When a fingerprint is being calculated, the values for these parameters are concatenated into a string. Depending on the protocol, some parameters can have multiple values during the handshake, when others might only have one value. When a parameter has multiple values, these values are separated by a dash. Values for different parameters are separated by a comma. We call this string the *pre-hash string*. To calculate the hash fingerprint itself, an MD5 hash is taken from this pre-hash string, which comprises the final fingerprint. This process is visualized in Figure 1.

3.2. JA3 and JA3 pre-hash string

The JA3 fingerprint is a hash fingerprinting algorithm for fingerprinting TLS clients. It is by far the most widely utilised hash fingerprinting algorithm. Support for the JA3 fingerprinting algorithm has been included in several projects, many of which are listed in [9]. It was introduced by the Salesforce employees J. B. Althouse, J. Atkinson and J. Atkins in 2017, who came up with the concept based on a presentation by L. Brotherston in 2015 [10].

For generating the JA3 fingerprint, the following values are collected from the Client Hello message: version, list of supported cipher suites, list of extensions, supported groups, and supported elliptic curve point formats. These values are then concatenated into the pre-hash string. Finally, an MD5 hash is calculated from the pre-hash string, which makes up the JA3 fingerprint.

3.3. Other hash fingerprinting algorithms

In this work we have focused on TLS traffic and the JA3 fingerprint, specifically the pre-hash string. The information contained in this string provides interesting prospects for visualization and categorization of TLS traffic in a monitored network. As shown in Section 2, similar information has been shown to produce reliable categorization of TLS traffic. However, our methods can easily be modified to be applicable for any other hash fingerprinting algorithm as well. We will shortly review other hash fingerprinting algorithms that have been proposed since the introduction of the JA3 fingerprint, but we will not go into deeper detail with them.

In 2018, the inventors of the JA3 fingerprint introduced the JA3S fingerprint [11], which is the server side fingerprinting algorithm for TLS. It is highly dependent on the initial Client Hello message, and thus not as practical as the JA3 fingerprint. The HASSH fingerprint [12] is an algorithm for fingerprinting SSH clients and servers, and was also introduced in 2018 by Salesforce employee B. Reardon. An algorithm was proposed for the initial protocol version of Google QUIC by Salesforce employee C. Yu in 2019 [13], but it is not applicable for the version of QUIC protocol standardized in RFC 9000 [14]. The standardized QUIC protocol uses TLS for the handshake phase, and thus the JA3 fingerprinting algorithm is applicable for it. Finally, several algorithms have been introduced by independent researchers, including RDFP for fingerprinting RDP clients [15] and SMBFP for fingerprinting SMB clients [16].

4. Methods and tools

We chose two different approaches for categorizing the JA3 pre-hash strings. The first approach is to consider each value, such as a cipher suite or an extension, as an individual word, and one JA3 pre-hash string as a document. We first separate the values into a bag-of-words model, and then perform topic modeling on it using *Latent Dirichlet Allocation* (*LDA*). In this approach, the order of the values is ignored, and only the presence of a word (such as a cipher suite or an extension) matters in the categorization. LDA was chosen due to its popularity and wide utilisation in topic modeling.

$$JA3_1 = A_1,B_1\text{-}B_2\text{-}B_3,C_1,D_1,E_1\text{-}E_2$$
$$JA3_2 = A_1,B_1\text{-}B_2,C_1,D_2,E_3$$

$$\begin{aligned} lev(JA3_1, JA3_2) = {} & lev(A_1, A_1) \\ & + lev(B_1\text{-}B_2\text{-}B_3, B_1\text{-}B_2) \\ & + lev(C_1, C_1) \\ & + lev(D_1, D_2) \\ & + lev(E_1\text{-}E_2, E_3) \\ = {} & 0 + 1 + 0 + 1 + 2 = \underline{4} \end{aligned}$$

Fig. 2. Calculating Levenshtein distance between two JA3 pre-hash strings.

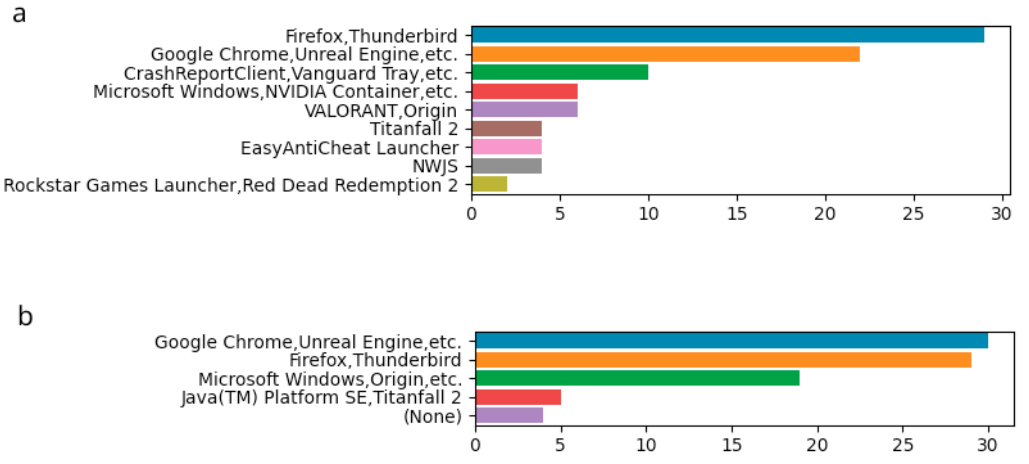

Fig. 3. (a) Example data set categorization based on K-Means; (b) Example data set categorization based on LDA.

The second approach focuses more on the order of the values, in addition to their presence. We implemented the *Levenshtein distance* algorithm for the JA3 pre-hash strings, and use it to perform *K-Means clustering* on the values. To calculate Levenshtein distance for a JA3 pre-hash string, we first calculated the Levenshtein distance of each section of the pre-hash string (version, cipher suites, extensions, supported groups and supported point formats) and considered the Levenshtein distance to be the sum of the distances of each section. This is demonstrated in Figure 2. Again, K-Means was chosen due to its popularity and wide utilisation as a clustering algorithm.

The tools for implementing these methods were written in the Python programming language. For implementing LDA, we used the Gensim library [17]. For K-Means we created a custom implementation of the algorithm. The tools are freely available in our GitHub repository at [3]. In the repository, we have also included an example data set of JA3 pre-hash values with mappings to the endpoint applications that have produced them in our test environment. We also provide example models created based on the example data set. The scripts use the example data sets by default, but can be modified to point at a custom data set instead to create new models based on other data sets. It is possible to use the pre-defined example models to see which category a JA3 pre-hash string provided by the user would fall into.

We have collected data for the example data set using a Forcepoint Next Generation Firewall in a local network segment with one external internet connection. The period for data collection was a three month period from 1st June until 31st August 2022. Five endpoints in the network were chosen as the source, as they had the Forcepoint Endpoint Context Agent (ECA) installed, which enables mapping endpoint information to the network connection in the Forcepoint NGFW. Each endpoint had Windows 10 operating system, and was in leisure use during the time period. The example data was collected from a set of endpoint applications that were considered trustworthy and were permitted to make external connections. During this time period, 87 unique JA3 pre-hash values and 178 unique JA3 pre-hash value to endpoint application mappings were observed. As long as the format of the data set remains the same, any other network monitoring tool capable of calculating JA3 pre-hash strings and mapping network connections to an endpoint application can be used for collecting data.

5. Results

Figure 3 shows the categorization results from our example models when run on the sample data set. The categorization result from our example K-Means model is presented in Figure 3 (a) and the result from our example LDA model is presented in Figure 3 (b). We can see that in both cases, there are 29 values categorized under the ”Firefox, Thunderbird” category. This includes, in fact, all the values produced by these two endpoint applications in the sample set, and nothing else. Firefox and Thunderbird are both developed by Mozilla Corporation [18]. Thus, our

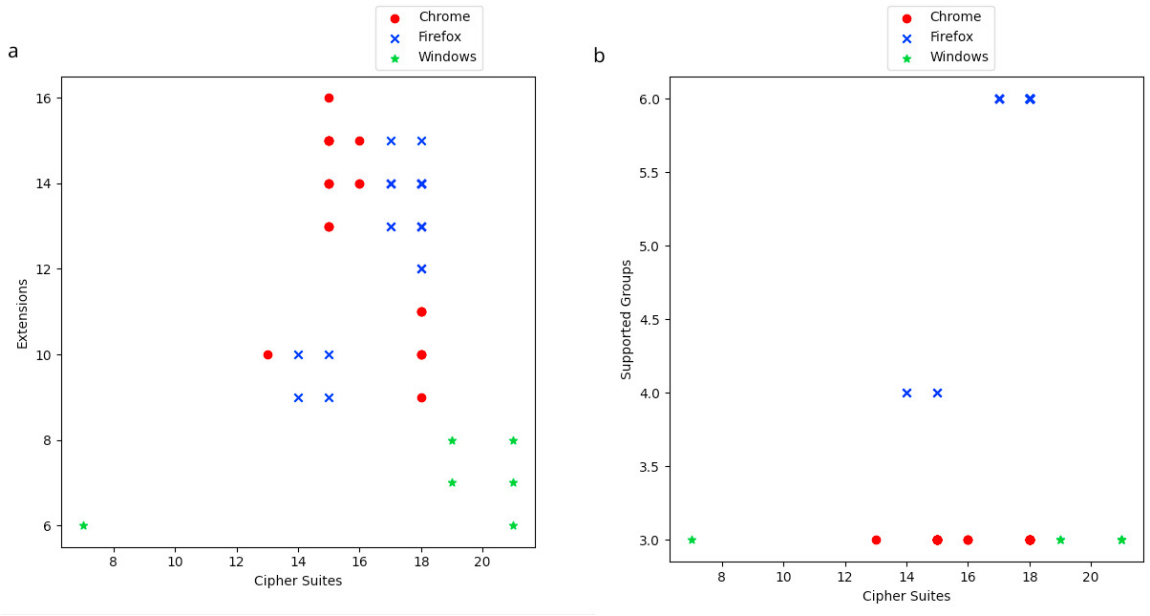

Fig. 4. Visualizing the (a) extensions and cipher suites; and (b) supported groups and cipher suites of Google Chrome, Mozilla Firefox and Microsoft Windows as Cartesian coordinates.

example models were both able to correctly and without false positives categorize all values produced by the Mozilla products from the sample data set.

There is more variation in categorizing the rest of the values. With LDA, the largest category includes Google Chrome, whereas with K-Means this is only the second largest category with the Mozilla category taking the top place. The rest of the categories do not seem to align. One big reason for this variation is that Firefox and Thunderbird use the Network Security Services (NSS) TLS libraries from Mozilla [19]. Many other endpoint applications utilise the open source Chromium Embedded Framework (CEF) [20], which means that the underlying TLS library is the same for such applications, producing common and similar JA3 fingerprints and making separation more difficult or impossible.

We can also see that with our example models, K-Means has more separate categories identified than LDA. One reason for this is that when training the models, we searched for the best accuracy by varying the amount of groups for the model in question. We could not find a number of groups which would have produced a sufficient result on both models.

In addition to categorizing the traffic, we used parts of the JA3 pre-hash string (cipher suites, extensions, supported groups and elliptic curve point formats) to visualize different aspects of the traffic in general. To better visualize the Levenshtein distances, we projected the JA3 pre-hash strings into two-dimensional Cartesian coordinates by calculating their distance from the origin. In Figure 4 (a) we can see the projection of the extensions and cipher suites for Firefox, Google Chrome and Microsoft Windows, as categorized by the example K-Means model. We observed that with this projection the values for Microsoft Windows form one cluster with one value as an outlier, while Firefox and Google Chrome were more difficult to separate. However, when projecting into the supported groups and cipher suites, Firefox forms two separate clusters while Google Chrome and Microsoft Windows are more uniform. This is visualized in Figure 4 (b). We can thus see that different two-dimensional projections reveal different clusters.

In Figure 5 we have also presented bar graphs visualizing the frequencies of the extensions used by Firefox (a) and all extensions observed in the sample set (b). From this, we can observe that the Record Size Limit extension, used in all Firefox samples, is not as popular in the overall sample set by comparison. We can also see that the unassigned TLS extension number 65037 is present nine times in the Firefox samples as well as in the whole sample set. In fact, this extension is only used by Firefox in the sample set. These findings show that in this environment it would be possible to identify the usage of Firefox with high confidence purely based on the presence of these two extensions.

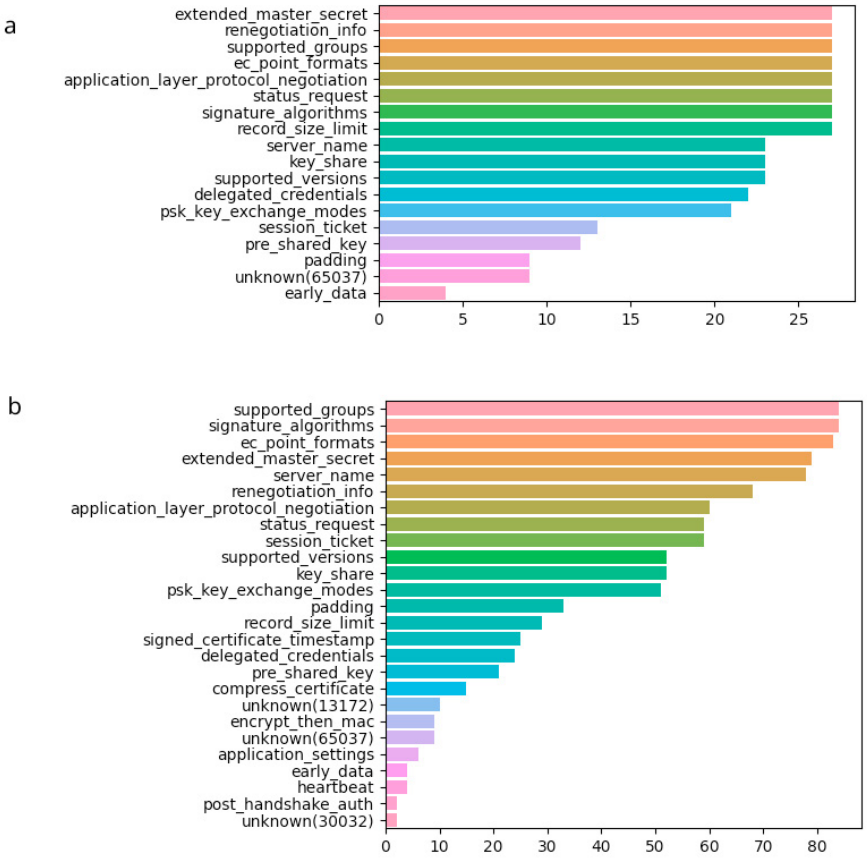

Fig. 5. (a) Frequencies of TLS Client Hello extensions from Firefox; (b) Frequencies of all observed TLS Client Hello extensions in the example data set.

Due to the small size of the sample set, this finding cannot be generalized, but it demonstrates the usefulness of this approach.

It should be noted that the environment from which the training data set is collected heavily affects the results. The sample data set has been collected from Windows machines, so it can give invalid results for values from other Operating Systems. In addition, some amount of manual work may be needed to make sure the training set is optimal. As an example, we found that Firefox uses a scheduled Windows task for telemetry purposes, titled ”default-browser-agent.exe”, but with the product name ”Firefox” in the executable metadata [21]. As the TLS connections made by this process are made by the Microsoft Windows Operating System, we renamed it as such in our training set to avoid invalid categorization.

The methods used for categorization in this paper provide a useful tool for improving deep packet inspection based security measures in network security solutions. We have seen that some endpoint applications have very distinctive JA3 pre-hash values. In addition, we have seen that it is possible to identify usage of common application frameworks, such as CEF, based on the JA3 pre-hash values. This makes it possible to better target security measures at appropriate endpoint applications if vulnerabilities in such frameworks are discovered.

6. Conclusion

In this paper we have presented two machine learning methods for categorizing TLS traffic. The methods categorize traffic based on endpoint application, and use the JA3 pre-hash strings as source material. We utilised LDA and

K-Means for categorizing the traffic, using Levenshtein distance for implementing a distance metric for K-Means clustering. To validate our methods, we produced pre-trained example models based on traffic samples collected from a small local network segment. Both models were able to correctly categorize all values from Mozilla based products in our sample set, but the results for other applications had more variation. To better visualize the Levenshtein distances, we projected the JA3 pre-hash into two dimensions and presented them as Cartesian coordinates. We observed that each projection reveals different clustering. The results are promising and show that such methods can be used for categorizing network traffic. One potential application domain for such methods is improving the security capabilities of network security solutions capable of deep packet inspection. Based on our results, there is justification for further research into the categorization methods of network traffic utilized in this paper.

Acknowledgements

The authors wish to thank Dr. Jukka Heikkonen and Dr. Antti Airola of the University of Turku, Finland, for their informative input regarding potential data analysis methods.

References

[1] Heino, J., Gupta, A., Hakkala, A., Virtanen, S., 2022. On Usability of Hash Fingerprinting for Endpoint Application Identification. *2022 IEEE International Conference on Cyber Security and Resilience (CSR)* 38–43. IEEE, Virtual Conference.
[2] Heilmeier, U., 2021. Commit f18ee30a3dc8495a3913043aa64c506d3e92fe13: 'TLS: Adding JA3 and JA3S fingerprints'. https://github.com/wireshark/wireshark/commit/f18ee30a3dc8495a3913043aa64c506d3e92fe13. (Accessed: August 8, 2022).
[3] Heino, J., Hakkala, A., Virtanen, S., 2022. JA3-CAT - Categorize TLS traffic based on JA3 pre-hash values https://github.com/ExtremeEmpress/ja3_cat. (Accessed: October 28, 2022).
[4] Husák, M., Čermák, M., Jirsík, T., Čeleda, P., 2016. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security* (1) 1–14.
[5] Muehlstein, J., Zion, Y., Bahumi, M., Kirshenboim, I., Dubin, R., Dvir, A., Pele, C., 2017. Analyzing HTTPS encrypted traffic to identify user's operating system, browser and application. *14th IEEE Annual Consumer Communications & Networking Conference* 1–6. IEEE, Las Vegas, NV, USA.
[6] Anderson, B., McGrew, D., 2019. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. *Proceedings of the Internet Measurement Conference* 379-–392. Association for Computing Machinery, New York, NY, USA.
[7] Frolov, S., Wustrow, E., 2019. The use of TLS in Censorship Circumvention. *Network and Distributed Systems Security (NDSS) Symposium*. San Diego, CA, USA .
[8] Althouse, J., Atkinson, J., Atkins, J., 2017. Open Sourcing JA3. https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41/. (Accessed: August 8, 2022).
[9] Althouse, J., Atkinson, J., Atkins, J., 2017. JA3 - A method for profiling SSL/TLS Clients. https://github.com/salesforce/ja3. (Accessed: August 8, 2022).
[10] Brotherston, L., 2015. Stealthier attacks and smarter defending with TLS fingerprinting. *DerbyCon V*. Louisville, KY, USA.
[11] Althouse, J., Atkinson, J., Atkins, J., 2018. TLS Fingerprinting with JA3 and JA3S. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967. (Accessed: August 19, 2022).
[12] Reardon, B., 2018. Open Sourcing HASSH. https://engineering.salesforce.com/open-sourcing-hassh-abed3ae5044c. (Accessed: August 19, 2022).
[13] Yu, C., 2019. GQUIC Protocol Analysis and Fingerprinting in Zeek. https://engineering.salesforce.com/gquic-protocol-analysis-and-fingerprinting-in-zeek-a4178855d75f. (Accessed: August 19, 2022).
[14] Iyengar, J., Thomson, M., 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. *RFC 9000, Internet Engineering Task Force (IETF)*
[15] Karimishiraz, A., 2019. RDP Fingerprinting. https://medium.com/@0x4d31/rdp-client-fingerprinting-9e7ac219f7f4. (Accessed: August 19, 2022).
[16] Torres, M. R., 2020. SMBFP SMB Fingerprinting Zeek package. https://github.com/micrictor/smbfp. (Accessed: August 19, 2022).
[17] Řehůřek, R., 2022. Gensim: Topic modeling for humans. https://radimrehurek.com/gensim/. (Accessed: August 26, 2022).
[18] Mozilla Corporation, 2022. Internet for people, not profit – Mozilla. https://www.mozilla.org/. (Accessed: October 7, 2022).
[19] Mozilla Corporation, 2022. Network Security Services (NSS). https://firefox-source-docs.mozilla.org/security/nss/index.html. (Accessed: September 23, 2022).
[20] Greenblatt, M. A., 2022. Chromium Embedded Framework. https://bitbucket.org/chromiumembedded/cef/wiki/Home. (Accessed: September 23, 2022).
[21] Mozilla Corporation, 2022. Default Browser Agent. https://firefox-source-docs.mozilla.org/toolkit/mozapps/defaultagent/default-browser-agent/index.html. (Accessed: September 23, 2022).