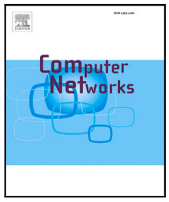
Computer Networks 250 (2024) 110606

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



JAPPI: An unsupervised endpoint application identification methodology for improved Zero Trust models, risk score calculations and threat detection

Jenny Heino [a,b,*], Christian Jalio [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

ARTICLE INFO

Keywords:
Network security
Intrusion detection
Machine learning
Zero Trust
Risk score

ABSTRACT

The surge in global digitalization triggered by COVID-19 has led to a significant increase in internet traffic and has precipitated a rapid transformation of the network security landscape. Despite being increasingly difficult, accurate traffic inspection is vital for ensuring productivity while reliably protecting internal assets. Endpoint application identification enables high accuracy inspection and detection by providing network security solutions with specific context on individual connections. However, achieving it in real-time with standard fingerprinting methods based only on client-side traffic has proven to be a challenging problem with no comprehensive solution thus far. In this article, we present a new methodology for identifying endpoint applications from network traffic, utilizing machine learning. Our methodology leverages similarities in the pre-hash string of the JA3 algorithm for fingerprinting application specific TLS Client Hello messages. By utilizing well-known clustering algorithms it is possible to identify the underlying TLS libraries and the application from the traffic remarkably better than with simple string-based matching. Our model can categorize 99,5% of the traffic in a controlled network, and 93,8% in an uncontrolled network, compared to 0,1% and 0,2% using simple string matching. Our methodology is especially effective for enhancing Zero Trust models, calculating a risk score for network events, and improving threat detection accuracy in network security solutions.

1. Introduction

The network security landscape has been going through a big transition for the past several years, as an increased number of companies are moving their previously local resources to the cloud. In addition, the COVID-19 pandemic transformed the concept of the workplace in such a way that it is now common for employees to either be fully remote or to have a hybrid arrangement by distributing their work time between remote and non-remote work. This rapid change has forced many network security vendors to reconsider their strategy and produce innovative approaches to stay current.

The problem of providing secure connectivity between offices, data centers, cloud resources and the mobile workforce has resulted in many innovative solutions. The *Secure Access Service Edge* technology, or *SASE*, has raised the most interest in many global market research and advisory companies such as Forrester [1] and Gartner [2]. The SASE architecture builds on top of the Zero Trust access model, enforcing security and access policies based on the identity of a device or entity. A SASE solution often consists of multiple intertwined services, such as Software-Defined Wide Area Network (SD-WAN), Secure Web Gateway (SWG), Next Generation Firewall (NGFW), Data Leak Prevention (DLP) and Cloud Access Security Broker (CASB).

A Zero Trust access model, or Zero Trust model, is a security model built upon the concept that every attempt to access a secured resource is verified. A Zero Trust model typically relies on identifying the user or the device and verifying that it has the correct permission to access the resource. This level of identification does not, however, give a comprehensive understanding about what exactly is attempting to access the resource. One device, or one user account on the device, typically has several different software components, or *endpoint applications*, actively running. When the device initiates a network connection to a resource, it is not necessarily visible which endpoint application initiated the connection. In some instances, this can have a significant impact. An employee responsible for human resources should be able to access a database containing highly personal information using a specific endpoint application designed for the task. But if another, unknown endpoint application running on the employee's device suddenly starts fetching information from the database, it should be considered highly suspicious and potentially malicious. Without awareness of the endpoint applications initiating the network connections, a network

* Corresponding author at: Department of Computing, University of Turku, 20014 Turku, Finland.
E-mail address: jeahei@utu.fi (J. Heino).

https://doi.org/10.1016/j.comnet.2024.110606
Received 18 March 2024; Received in revised form 24 May 2024; Accepted 15 June 2024
Available online 19 June 2024

security solution does not have visibility into which application is receiving the data, only that it is the particular employee's device and account.

This kind of endpoint awareness in a network security solution is also a useful tool when considering the risk score of a network event. The concept of risk can be useful when designing security policies. When a network security solution has perception of the endpoint applications initiating network connections, it can either raise or lower the risk score of the network event depending on if it is considered normal or not for the particular endpoint application to make such a network connection. The network security solution then has an improved ability to analyse whether the traffic should be terminated or permitted to continue.

Another feature of a network security solution which benefits from awareness of the endpoint applications is threat detection. False positives are a common issue among different security solutions, and they can disrupt regular network usage. It is not always possible to develop threat detection in a way that would only catch truly malicious traffic patterns. It is instead common that a subset of the threat detection signatures may also occasionally trigger from benign traffic. One source for this problem among network security solutions is the lack of context about the protected endpoint application. If the traffic triggers an uncertain threat detection signature that is intended to protect a vulnerable version of the Chrome browser, but the recipient of the traffic is a Firefox browser, it is not necessary to terminate the traffic. However, without endpoint awareness, the network security solution, or the administrator of the solution, needs to make the decision whether to terminate the connection and risk blocking benign traffic, or let the traffic through and risk a potential exploit.

For a network security solution to be able to properly utilize the endpoint application information for enhancing security, the information needs to be available when the connection is still ongoing, preferably when it is still at the handshake stage, or even earlier. This gives the network security solution the ability to apply security measures on the connection. As comparison, some methods use features that are collected from the entire connection, meaning that proper identification can only be done after the connection has already been closed. These kinds of methods can be useful for a network security solution for other purposes, such as reporting on past incidents. In the context of this article, however, such methods cannot be utilized.

There are various methods for mapping the endpoint application to the network connection. Some network security vendors [3,4] have a specific endpoint component which can be used for reporting endpoint metadata for each network connection. This gives the network security solution certainty about the endpoint application, and the information is typically available already when the connection is opened. But this level of awareness is usually not available, and the network security solution needs to use the network traffic itself to gain information about the endpoint application. For some protocols, the task is easier due to a plain-text header, such as the User-Agent header for HTTP, broadcasting the endpoint application. But for many other protocols, network security solutions need to rely on more sophisticated methods. Achieving accurate and reliable endpoint application identification in real-time based only on client-side traffic has proven to be a challenging problem with no comprehensive solution found in research thus far.

In this article we introduce a new methodology utilizing machine learning for identifying endpoint applications from network traffic. Our methodology is called *JAPPI*, which comes from *JA3-based Application Identification*. We will focus on TLS traffic, but the method can be utilized for many other network protocols as well, where the handshake procedure presents a similar set of distinctive parameter values. JAPPI leverages similarities in the pre-hash string of the JA3 algorithm for fingerprinting application specific TLS Client Hello messages. Unlike other approaches, JAPPI is run in real-time and its inspection result is based only on client-side traffic. By providing awareness and specific context on individual connections between communicating endpoint applications, JAPPI brings value to a network security solution in multiple ways. Firstly, JAPPI can be used for enhancing Zero Trust models by ensuring that unwanted endpoint applications are not permitted access to a sensitive remote resource. Secondly, JAPPI can bring additional context for calculating the risk score of a network event, as the risk score can be lowered or increased depending on whether the JAPPI identification indicates a trusted application or something else. Thirdly, by providing awareness of the protected endpoint application, JAPPI can help in providing more targeted threat detection, and decrease false positive and false negative threat identifications.

To validate the coverage of JAPPI, we apply it to a large-scale experiment with live data from two different network environments. We also compare the results from JAPPI to results using a methodology from our previous research [5] which leveraged basic string matching. Our previous methodology used the MD5 hash based JA3 fingerprint analysis, whereas JAPPI is built on our recent findings on the benefits of using JA3 pre-hash strings instead of JA3 hash fingerprints [6] and on the application of machine learning methods to categorize pre-hash strings to the corresponding endpoint applications [7]. Our hypothesis is that JAPPI will be able to categorize the traffic considerably better than our previous methodology.

The rest of the article is structured as follows. In Section 2 we look at existing research and implementations related to identifying the source endpoint application of a network connection. We take a deeper look into the concept of hash-fingerprinting algorithms, especially the JA3 algorithm. In this section we also give a brief introduction to our previous methodology. In Section 3 we present a specification for JAPPI, a novel machine learning based methodology for identifying endpoint applications from TLS Client Hello messages. In Section 4 we validate JAPPI by performing a large-scale experiment on two different real-life network environments. We also compare the results with our previous methodology. In Section 5 we compare JAPPI to other methods that have been proposed for similar purposes in literature. In Section 6 we discuss the results as well as potential use-cases of JAPPI. Finally, we conclude our article in Section 7.

2. Related work

In this section we will cover related research on identifying the endpoint application based on network traffic and existing implementations. In Section 2.1 we will first focus on recent research on identifying endpoint applications based on network traffic. In Section 2.2 we cover some existing implementations in network security solutions. We will then introduce the concept and history of hash fingerprinting algorithms in Section 2.3. Finally, we will provide a high-level description of our previous methodology in Section 2.4.

2.1. Identifying endpoint applications based on network traffic

Several methods for identifying the endpoint applications using various protocol parameters from network traffic have previously been proposed. For some protocols, such as HTTP, the task is easier due to the existence of a specific protocol header intended for identifying the endpoint application [8]. Such direct indicators do not, however, exist in most network protocols that are currently in active use, which brings about the need for more complex identification methods.

Using the list of supported cipher suites presented in a TLS Client Hello message for identifying the client endpoint application has been experimented with by Husák et al. [9]. The researchers showed that the approach can be effective. The main issue with this approach is that there is no variation allowed for the list. The list is effective for a limited time, but as new versions of the applications are released, with added or modified support for different algorithms, the identification decays.

Another article by Muehlstein et al. [10] explores using an extensive feature set collected from a TLS connection for identifying the client

endpoint application. The features include information collected from the TLS messages, such as the set of supported cipher suites and the number of extensions, but also information about the network connection, such as the number and size of packets transferred on both sides. Machine learning (support vector machine with radial basis function) is applied on the feature set to identify the client endpoint application. This method uses information collected both from the client and server side, which makes it depend on the endpoint application on both sides of the connection and thus less adequate for our purpose. In addition, this method uses information from the entire connection, which again does not suit our purpose to gain awareness of the endpoint application at the beginning of the connection.

Two recent papers, by Frolov and Wustrow [11] and Anderson and McGrew [12], have solved this issue by using the Levenshtein distance algorithm on a feature set collected from the TLS Client Hello. Both research teams calculate the distances between two parameter lists and observe if the lists are close to each other, instead of relying on finding an exact match. The teams observed that the method identified a larger part of the network traffic than a simple matching would produce. Anderson and McGrew [12] also mention using clustering on the collected feature set, but do not mention the clustering algorithm used.

Many articles have also been published about distinguishing malicious network traffic from non-malicious. Mostly these methods use a binary classification of traffic, which cannot be used for our purposes. One recent paper by Gomez et al. [13] does approach the issue using unsupervised clustering for identifying different malware samples from TLS traffic. Using their method, the researchers reach a high accuracy in identifying different malware families. Like Muehlstein et al. [10], however, this method uses a feature set collected both from the client and the server side during the entire TLS connection, which makes it not applicable for our purposes.

2.2. Existing implementations for identifying endpoint applications in network security solutions

There are a few different existing implementations for gaining awareness on the endpoint applications in network security solutions. In this section we will cover a few implementations that are relevant to this article. We will leave out approaches like the *Endpoint Detection and Response* method which moves the network security process to the client side, and active scanning methods, such as NMAP [14] and Nessus [15], as they do not provide endpoint application information for new network connections initiated by the protected endpoints.

Receiving endpoint metadata from the protected endpoint for each network connection is one approach to gaining awareness of the endpoint applications in a network security solution. With this approach, a separate endpoint application is installed on the protected endpoint, which sends metadata about each new network connection to the network security solution. This method gives the network security solution an infallible understanding of each endpoint application. The downside to this method is the need to install a separate endpoint application on the protected endpoint. There are two existing implementations for such endpoint metadata: the *McAfee Endpoint Intelligence Agent* [3], and the *Forcepoint One Endpoint* [4].

A proprietary method of identifying endpoint applications from TLS traffic, called *Encrypted Visibility Engine* or *EVE*, has been implemented in the Cisco Secure Firewall [16]. The method collects information from the TLS Client Hello and utilizes machine learning on the information to gain understanding of the endpoint application. Based on public information this method sounds similar to the machine learning methods we experimented with previously [7]. The detail about their method has, however, not been made public, so we cannot tell which fields of the TLS Client Hello their method utilizes. There is a reference to enhancing their feature to identify malware connections, but there does not seem to be any plans to utilize the information for improving the accuracy of existing intrusion detection signatures.

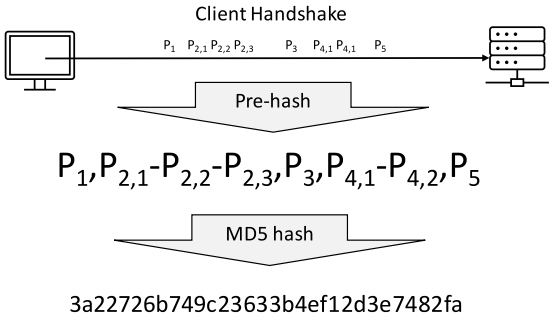


Fig. 1. A high level presentation of a hash fingerprinting algorithm.

2.3. Hash fingerprinting

One method for fingerprinting the source and destination endpoint application of a network connection is *hash fingerprinting*. The first such algorithm published was the JA3 algorithm by Althouse et al. [17]. Since then, there have been several hash fingerprinting algorithms proposed for fingerprinting various network protocols. We will present the JA3 algorithm in more detail in Section 3.1 and give a more general description of the hash fingerprinting algorithms here.

The underlying idea behind all hash fingerprinting algorithms is the same. When a network connection is opened between two endpoints, more specifically two endpoint applications, the applications will need to perform a handshake to let the other endpoint know what versions and features they support and prefer. The handshake procedure may also include transferring other relevant information, such as indicating the host name that the client is trying to access. The hash fingerprinting algorithms pick up this information from a handshake message sent by the client or server, depending on which side the algorithm is fingerprinting, and concatenates the values into a string format. The string is structured so that different parameters are separated by a comma and multiple values for one parameter are separated by a dash. We refer to this string as the *pre-hash string*. The final fingerprint, which is the output of the hash fingerprinting algorithm, is the MD5 hash taken of the pre-hash string.

The high-level function of a hash fingerprinting algorithm is clarified by Fig. 1. At the top, we can see a client endpoint initiating a connection with a server endpoint. The client sends a Client Handshake message to the server, which lists a set of parameters related to the supported and preferred capabilities that the client has. These parameters are collected into a pre-hash string, which in this example consists of five sections. The first, third and fifth section contain the values given for parameters P_1, P_3 and P_5, while the second and fourth section list all values that were provided for parameters P_2 and P_4. The pre-hash string for this example is "$P_1, P_{2,1} - P_{2,2} - P_{2,3}, P_3, P_{4,1} - P_{4,2}, P_5$". Finally, an MD5 hash is taken of this string, which comprises the final hash fingerprint.

After the publication of the JA3 algorithm for fingerprinting the TLS client, researchers have proposed hash fingerprinting algorithms for various other network protocols. First, Althouse et al. [18] proposed JA3S for fingerprinting TLS servers. The researchers have since acknowledged that the JA3S is highly dependent on the initial client request, and thus not as useful as the JA3 fingerprint. Since then, algorithms for fingerprinting SSH clients and servers (*HASSH* by Reardon [19]), gQUIC clients (*CYU* by Yu [20]), RDP clients (*RDFP* by Karimishiraz [21]) and SMB clients (*SMBFP* by Torres [22]) have been introduced. Out of these, the JA3 algorithm gained the highest popularity, due to most internet traffic happening over TLS.

2.4. Our previous methodology

We have previously introduced a method for endpoint aware inspection in a Network Security Solution [5]. In this section we summarize the key features of our previous methodology. We will remain on a higher level regarding the method description and leave a deeper definition for our original article.

Our previous methodology is based on the idea that a subset of the protected network can be used as a source for information about the endpoint's applications used in the network, and this information can then be leveraged for the entire protected network. The subset of the protected network which provides this information is called the *Source* group. The Source group should provide a good overall sample of the entire protected network, which is referred to as the *Target* group.

The endpoints in the Source group send metadata to the network security solution about each endpoint application initiating a new network connection. This endpoint application information is correlated by the network security solution to a hash fingerprint which has been taken from the associated network connection, and the pair of hash fingerprint and endpoint application metadata is stored in a database.

After the database has been populated for a suitable amount of time, the information in it will be utilized for the entire Target group. A hash fingerprint will be taken of each suitable network connection produced by any endpoint in the Target group, from which the network security solution does not receive endpoint metadata. The hash fingerprint will be compared to the information in the database, which will provide a list of endpoint applications that have produced the same hash fingerprint in the Source group. This will give the network security solution an educated assumption about the potential endpoint application initiating the network connection. This information can be utilized for producing more accurate deep packet inspection-based decisions, such as whether to terminate a potentially malicious connection or not.

3. JAPPI

In this section we introduce a novel machine learning based methodology for identifying endpoint applications from TLS Client Hello messages. Our methodology uses a distance algorithm and clustering for categorizing the JA3 pre-hash values instead of a simple string comparison of the hash fingerprints, which was used in our original method. We have named the methodology *JAPPI* for clarity. The acronym comes from *JA3-based Application Identification*.

After proposing our previous methodology, we recognized that it had a key weakness in its logic. The weakness is that the original method solely relies on the hash fingerprints themselves. Due to the nature of how hash algorithms, such as the MD5 algorithm, work, the hash fingerprint itself will vary even if two pre-hash strings only have a small difference [6]. This may happen for example if the latest version of an endpoint application has added support for one new feature, but the rest of the values presented during the handshake remain the same.

This issue was heavily highlighted by a recent change in the TLS library used by the Chromium Framework [23]. A feature was implemented into the Chromium TLS library which presents the extensions of a TLS Client Hello message in a randomized order. This does not alter how the TLS handshake is processed by servers, but it will produce a distinct hash fingerprint each time, making the hash fingerprint database concept useless for Chromium based endpoint applications. This is highly relevant, as the Chromium framework is used by a large variety of different endpoint applications, such as Spotify [24], Adobe [25] and Microsoft Business Intelligence [26].

JAPPI uses the pre-hash values of the JA3 algorithm instead of the final hash fingerprints. As the pre-hash value lists all supported features and other parameters in the order in which they were presented by the application, it can be used to compare how *close* two different values are. Due to this, it is possible to use clustering algorithms on the values.

We will first introduce the concepts that JAPPI uses. In Section 3.1 we introduce the JA3 fingerprinting algorithm. In Section 3.2 we introduce the Levenshtein distance algorithm in the form that we use it with the pre-hash strings, and in Section 3.3 we give a high-level introduction to clustering and a few relevant clustering algorithms. After this, we will present a definition for JAPPI in Section 3.4. Finally, we go through how JAPPI can improve the use cases of Zero Trust models, risk score calculation and threat detection in Section 3.5.

3.1. JA3 algorithm

The JA3 hash fingerprinting algorithm is a fingerprinting algorithm intended for identifying the client endpoint application based on the TLS Client Hello message. It was invented and published in 2017 by Salesforce employees John Althouse, Jeff Atkinson and Josh Atkins [17]. Since then, it has become a key component for security analysts. The JA3 fingerprint of a malware or an exploit kit is often provided with Indicators of Compromise (IOC) information, and the algorithm has been integrated into various applications and platforms, such as Wireshark [27] and Cloudfront [28].

The JA3 algorithm collects the following information from the TLS Client Hello message: version, list of supported cipher suites, list of extensions, list of supported groups, and list of supported elliptic curve point formats. As depicted in Section 2.3, this information is then concatenated into the pre-hash string where the values for different sections are separated by commas, and multiple values for one section are separated by a dash. An example JA3 pre-hash string produced by Firefox version 125.0 on a Linux based operating system, as demonstrated by Fig. 2, is 771,49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-156-157-47-53,0-23-65281-10-11-35-16-5-13- 28,29-23-24-25,0. The JA3 fingerprint is the MD5 hash taken from this string, which in this case is fb0aa01abe9d8e4037eb3473ca6e2dca. As stated previously, we will be focusing on the pre-hash string of the algorithm.

3.2. Levenshtein distance

To perform clustering on the JA3 pre-hash values, we need the ability to compare how close two values are to each other. For this purpose, we use the Levenshtein distance algorithm. We calculate the distances of each individual section of the pre-hash string and take the sum of these distances. The definition of the Levenshtein distance, as we use it in this context, is as follows.

Definition 3.1 (*Levenshtein Distance Algorithm for Pre-hash Strings*)**.** Let $A = (A_1, \dots, A_h)$ and $B = (B_1, \dots, B_h)$ be pre-hash strings of a hash fingerprinting algorithm, where $h \geq 2$. Let each section A_n, B_n consist of 0 or more values $A_{n,m}$, $B_{n,m}$ such that $A_n = (A_{n,1}, \dots, A_{n,l})$ and $B_n = (B_{n,1}, \dots, B_{n,k})$. The Levenshtein distance between A and B is defined as follows:

$$lev_{hashfp}(A, B) = \sum_{n=1}^{h} lev_{section}(A_n, B_n) \tag{1}$$

where for sections A_n, B_n,

$$lev_{section}(A_n, B_n) = \begin{cases} |A_n| & \text{if } |B_n| = 0 \\ |B_n| & \text{if } |A_n| = 0 \\ lev_{section}((A_{n,2}, \dots, A_{n,l}), (B_{n,2}, \dots, B_{n,k})) & \text{if } A_{n,1} = B_{n,1} \\ 1 + min \begin{cases} lev_{section}((A_{n,2}, \dots, A_{n,l}), B_n) \\ lev_{section}(A_n, (B_{n,2}, \dots, B_{n,k})) \\ lev_{section}((A_{n,2}, \dots, A_{n,l}), \\ \qquad (B_{n,2}, \dots, B_{n,k})) \end{cases} & \text{otherwise.} \end{cases} \tag{2}$$

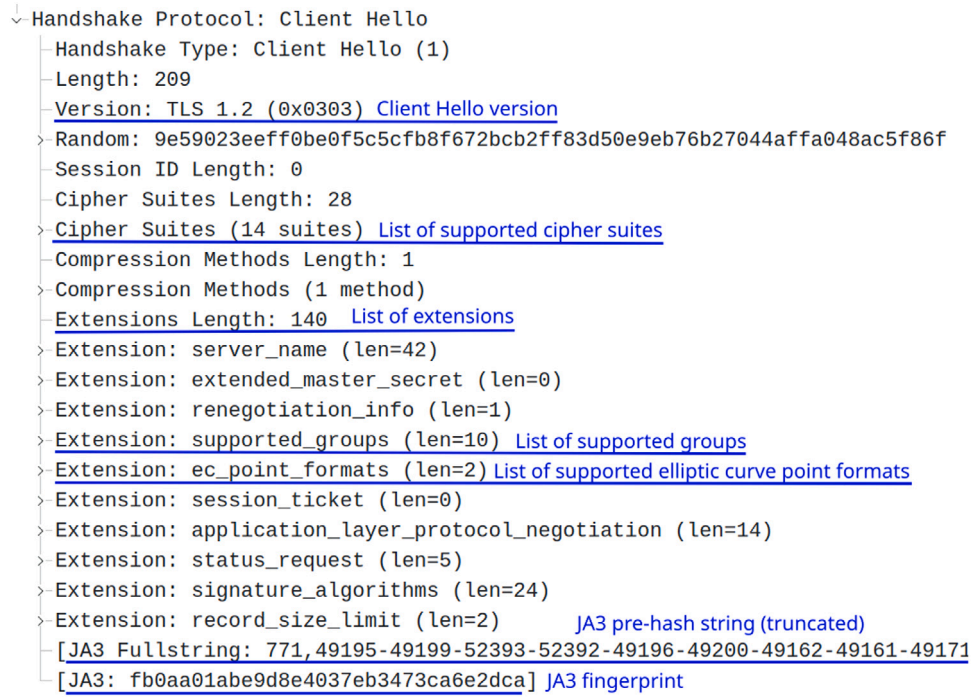


Fig. 2. A screen capture from Wireshark of a captured TLS Client Hello message sent by Firefox 125.0, showing the generation of the JA3 fingerprint.

Each hash fingerprinting algorithm has its own number h of sections. The JA3 hash fingerprinting algorithm has 5 sections, so the distance is calculated using value $h = 5$.

For a JA3 pre-hash string A, the sections are as follows: A_1 = version, A_2 = list of supported cipher suites, A_3 = list of extensions, A_4 = list of supported groups, and A_5 = list of supported elliptic curve point formats. Due to the randomization of the list of extensions by the Chromium Framework mentioned in the introduction of Section 3, we have also made one additional adjustment when calculating the distance between two JA3 pre-hash strings: before calculating the distance $lev_{section}(A_3, B_3)$ between two lists of extensions, we sort the lists.

3.3. Clustering

We use clustering to identify distinctive groups within a collection of JA3 pre-hash strings. With the Levenshtein distance algorithm defined in the previous section, we find the values that are close to each other, and thus form a cluster. There are various clustering algorithms that can be used for finding the clusters among a set of JA3 pre-hash strings using the Levenshtein distance, and JAPPI is not restricted to any one algorithm. In Section 4 we selected two algorithms, which we will shortly introduce here: *K-Means* and *DBSCAN*. Both algorithms are popular and widely used clustering algorithms.

3.3.1. K-means

The *K-Means* algorithm is the most popular and widely used clustering algorithm. A big reason for its popularity is that it is easy to understand and implement. One drawback with K-Means is that the algorithm requires that the number of clusters be known prior to running the algorithm. If the right number of clusters is not known in advance, it may be necessary to run the algorithm with different numbers to find the best one. Another drawback is that it is sensitive to the selection of the initial cluster centroids, which again may raise the need for repeating the algorithm several times before selecting the best outcome.

Due to its popularity, there have been many variations proposed for the algorithm. We will next describe on a high-level how we use K-Means in Section 4.

1. Select a number c which represents the number of clusters. Select c elements among the list of unique JA3 pre-hash strings in the data set as the initial *centroids* for the final clusters.
2. Repeat the following until the clusters are considered stabilized.
   (a) For each JA3 pre-hash string in the data set, find the closest centroid. A cluster is formed by the JA3 pre-hash strings that have the same closest centroid.
   (b) For each cluster, find the JA3 pre-hash string which has the smallest average distance to the other JA3 pre-hash strings in the same cluster. Assign this JA3 pre-hash string as the new centroid for the cluster.

As mentioned above, K-means is sensitive to the initial selection of centroids. There are various methods of selecting the initial c centroids. The three initialization methods we used in Section 4 are described next.

- **Random initialization.** A random set of c elements is selected among the data set.
- **K-Means++.** The first initial centroid is selected randomly. The following centroids are selected by taking the element furthest away from the already selected centroids, until c centroids have been selected.
- **Set of known elements.** A pre-determined set of elements is given as the initial centroids. If less than c elements are given, the rest are selected randomly.

3.3.2. DBSCAN

Density-based spatial clustering of applications with noise, or *DBSCAN* is also one of the most popular clustering algorithms. Unlike K-Means, DBSCAN does not require the desired number of clusters to be provided in advance. As DBSCAN is a density-based clustering algorithm, it identifies the elements that are packed close to each other and flags the other elements as outliers. In Section 4 we experiment with two approaches to the outliers: in one approach we consider them unique clusters, and in the other we discard them. Next, we give a high-level description of how we use DBSCAN in Section 4.

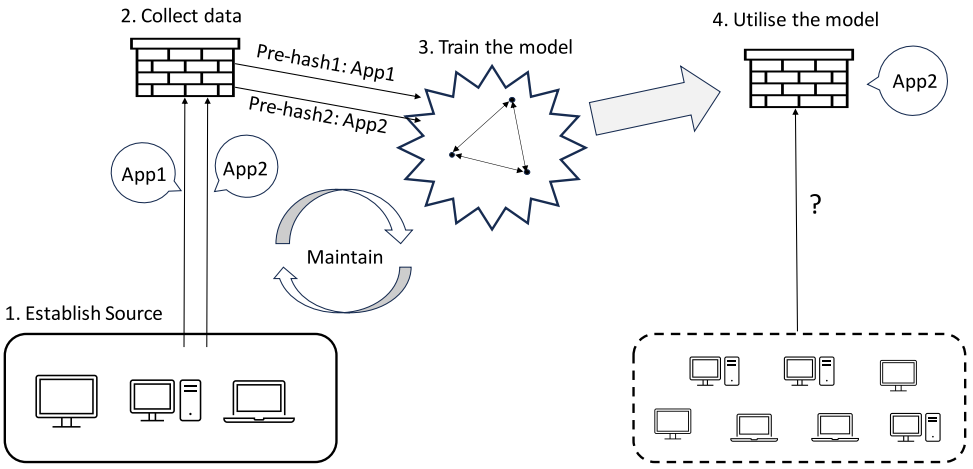


Fig. 3. High level description of JAPPI process. Steps 1 and 2 depict establishing the Source for the data among the protected devices, collecting the endpoint application metadata from them along with the calculated JA3 pre-hash string, and storing the data. Step 3 depicts running a clustering model on the data to find the JAPPI clusters. Step 4 depicts the JAPPI model being utilized by the network security solution on the rest of the protected network. The JAPPI model is constantly maintained by collecting new data from the Source.

1. Select a distance *eps* which is used for deciding if two JA3 pre-hash strings are considered to be neighbours: the distance between neighbours is less than *eps*.
2. Select a number *minpts* for determining if a JA3 pre-hash string is a part of a cluster or an outlier: if the JA3 pre-hash string has less than *minpts* neighbours, it is considered an outlier.
3. Consider all JA3 pre-hash strings to be unvisited. Consider all JA3 pre-hash strings to be unmapped, indicating that they do not belong to a cluster yet.
4. Repeat the following until the list of unvisited JA3 pre-hash strings is empty.
    (a) Select a random JA3 pre-hash string from the list of unvisited JA3 pre-hash strings. Remove it from the unvisited list. Find its neighbours.
    (b) If the JA3 pre-hash string has less than *minpts* neighbours, flag it as an outlier. Remove it from the unmapped list. Go back to step (a).
    (c) If there are as many as, or more than, *minpts* neighbours, create a new cluster, and add the JA3 pre-hash string to it. Remove it from the unmapped list. Repeat the following until the currently processed list of neighbours is empty.
        i. Select a random JA3 pre-hash string from the currently processed list of neighbours. If the JA3 pre-hash string has not been visited before, remove it from the unvisited list and find its neighbours. If the JA3 pre-hash string has as many as, or more than, *minpts* neighbours, add the neighbours to the currently processed list of neighbours.
        ii. If the JA3 pre-hash string does not belong to any cluster yet, add it to the currently processed cluster, and remove it from the unmapped list.
        iii. Remove the JA3 pre-hash string from the currently processed list of neighbours.

3.4. Definition of the JAPPI methodology

We will next provide a definition for the JAPPI methodology. In the rest of this article, we will call the procedure of protecting a network using the JAPPI methodology the *JAPPI process*. We first give a high-level explanation for the JAPPI process, and then more detailed definitions for each step of the process. Fig. 3 provides a visual high-level representation of each step of the JAPPI process. The steps of JAPPI are as follows:

1. Establish Source for data,
2. Collect data,
3. Find clusters and label them using metadata, and
4. Utilize and maintain.

3.4.1. JAPPI: High level definition

The Jappi process starts by establishing the Source, a reliable source group for gathering the required information. The Source should consist of a collection of various endpoints, such as personal computers and servers, that initiate new TLS connections in the network. Ideally, the Source should contain a good representation of the various endpoint applications and operating systems that JAPPI is intended to be able to cover.

After the Source is established, the JA3 pre-hash fingerprints and metadata about the source endpoint applications should be collected for each new TLS connection initiated from the Source. This data collection phase will be continued until the collected data is considered sufficient for identifying the clusters. The duration of the data collection period may vary from use case to use case.

The JAPPI clusters are identified using a suitable clustering algorithm with the use of the Levenshtein distance algorithm introduced in Section 3.2. After the different clusters have been identified, they are labelled using the endpoint application metadata. One JA3 pre-hash string, the *centroid*, is selected from each cluster to represent the cluster.

After the JAPPI model has been established it can be utilized on the entire protected network. When a new TLS connection is seen from the protected network, the JA3 pre-hash string is calculated from the Client Hello, and the closest centroid is found. The connection is then categorized based on the cluster which the centroid represents. The connection can also be considered uncategorized, or unknown, if all centroids are further away from the calculated JA3 pre-hash string than some predetermined distance. New data should be continuously collected from the Source, and the JAPPI model should be periodically updated based on new data. The JAPPI process described above from the perspective of the network security solution is summarized in Fig. 4.

3.4.2. Establish the source

During the first step, a source group, or the Source, is established for gathering reliable information. This group should consist of a collection of various endpoints, such as personal computers and servers, that initiate new TLS connections in the network. Ideally, the Source should contain a good representation of the various endpoint applications and operating systems that JAPPI is intended to be able to cover.

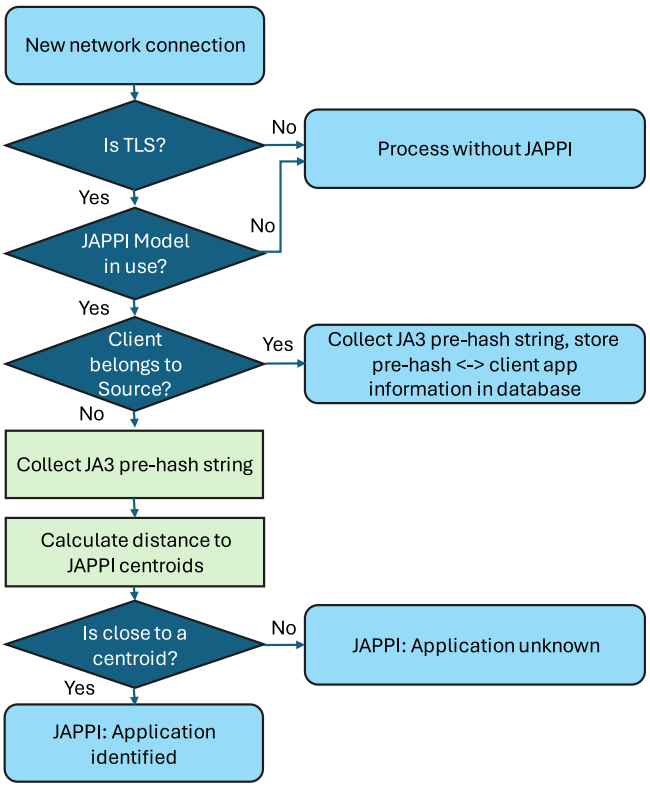


Fig. 4. A flowchart of the JAPPI process from the perspective of the network security solution.

The required information can for example be collected from the endpoints in the Source using a network security solution which is able to collect the JA3 pre-hash strings from the TLS connections, and able to receive endpoint metadata from the endpoints about each new network connection. Alternately, the information can also be collected locally on the endpoints using a suitable tool which is able to collect the same information.

3.4.3. Data collection

During the second step, the required data is being collected from the endpoints in the Source. The information needs to be collected into a database where each JA3 pre-hash string can be mapped to all endpoint applications that have produced the same string. The suitable duration for the initial data collection period may vary depending on the environment and the seasonal variation in the traffic profiles that the environment may have. We recommend that the initial data is collected at least for the period of one day of typical traffic profile. As an example, it is not recommended to perform the initial data collection during a weekend if the environment is less active during that period.

3.4.4. Finding the clusters

After the data has been collected, clustering can be performed on the data using the Levenshtein distance algorithm for pre-hash strings defined in Definition 3.1. JAPPI does not require the usage of a specific clustering algorithm, but two algorithms have been introduced in Sections 3.3.1 and 3.3.2. One sample, a *centroid*, is selected from each cluster to represent it. These centroids represent the JAPPI clusters.

Each cluster should have one or more labels, with an optional primary label. The labels should contain all endpoint applications that have produced a JA3 pre-hash string which is categorized into this cluster. The primary label can be manually added based on expert knowledge, for example to represent the underlying protocol library which these endpoint applications share. As an example, many endpoint applications utilize the Chromium Framework, and will thus produce similar JA3 pre-hashes that will be categorized into the same cluster. The labels should list the names of all the endpoint applications using the Chromium Framework, but the primary label could indicate that this cluster represents the Chromium Framework.

3.4.5. Utilize and maintain

During the last step, the network security solution will start to utilize the JAPPI clusters for categorizing all TLS connections it sees, also from endpoints that are not monitored as part of the Source. When a new TLS connection is seen, the network security solution should calculate the JA3 pre-hash string from the Client Hello message and compare it to the JAPPI centroids that it has. It should select the centroid which is closest to the value, and flag it as the identified JAPPI cluster.

While the current model is being utilized, new data should still be continuously collected from the Source. The clustering step should be rerun periodically to keep the model up to date. The duration of the update cycle may be different from use case to use case.

3.5. Improving Zero Trust models, risk score calculations and threat detection

There are many ways in which a network security solution can leverage the endpoint awareness introduced by the JAPPI methodology. We especially have identified three areas in which JAPPI can bring value for a network security solution: Zero Trust models, calculating a risk score, and threat detection. We will next go through each area and explain the benefit introduced by JAPPI.

3.5.1. JAPPI and Zero Trust models

The concept of Zero Trust and Zero Trust models was first introduced by John Kindervag and the Forrester Market Research organization in 2010 [29]. It introduces a network architecture where no network is considered safe, but instead all networks and all endpoints are considered a potential threat. In a Zero Trust model all requests are authenticated and authorized, and any connection attempts that have not been specifically allowed are terminated.

The authentication and authorization is typically done based on the user and based on the device. This approach does not take into account the fact that there are many client applications installed on the device where an authenticated user has logged in, while only one client application is intended to be able to access a restricted remote resource. This can enable a breach where a malicious software component on the device is able to access the same restricted resources.

With awareness of the endpoint application, a network security solution gains the ability to enforce zero trust on a new level: on the application level. It becomes possible for a network security solution to permit access to restricted resources only for client applications that are specifically intended for accessing the resource, and deny it for others. This becomes possible with JAPPI. A network security solution will be able to permit access only for the connections that are identified with the JAPPI cluster which the desired client application belongs to, and block access for others. It should still be noted that other client applications using the same underlying TLS library can be categorized into the same JAPPI cluster. Nevertheless, JAPPI makes it possible to deny access attempts from all client applications that do not fall into the same JAPPI cluster.

3.5.2. JAPPI and risk score calculation

Risk scores are a common concept in many security solutions, including network security [30,31]. A risk score indicates the risk that an element or event poses to the protected network. It may be applied to a user, a device, or a single network event. It depends on the implementation which factors affect the risk score.

JAPPI introduces a new metric which can be used when calculating a risk score. With JAPPI it is possible to notice when a previously unknown or otherwise suspicious client application is observed in the network. This information can be leveraged with the risk score calculation in many ways. If a suspicious JAPPI identification is made from a single network event, the risk score of the event can be raised. If a device or a user account is seen to perform many network events with suspicious JAPPI identifications, its risk score can be raised. A raised risk score can then affect the actions performed. Network access can be restricted or denied on a highly risky event, user or device, or additional security measures can be applied on the traffic.

3.5.3. JAPPI and threat detection

Threat detection is a core feature in many network security solutions. There are different methods that a network security solution can leverage for detecting threats. One typical method is to perform deep packet inspection on the network traffic, and to see if malicious patterns are seen in the inspected content [32]. However, most threat detection capabilities have a risk of producing false positive and false negative identifications. When a false positive identification happens, the network security solution will terminate a connection that was not actually malicious. A false negative identification indicates that the network security solution was unable to stop something that was actually malicious. When trying to decide whether to trigger a threat detection signature from a traffic pattern where a false positive is highly likely, a network security solution needs to balance between the decisions to terminate something benign, or to let through something that turned out to be malicious.

JAPPI gives a network security solution a tool for making more informed decisions in cases where it is not clear whether the observed traffic patterns are truly malicious or not, and a decision needs to be made whether the connection should be terminated or not. When the network security solution identifies potentially malicious traffic patterns from the traffic, it usually knows which vulnerable endpoint application the traffic would be trying to target. With JAPPI, the network security solution is able to verify whether the endpoint application receiving the traffic would be vulnerable to the attack or not. If the targeted application would not be vulnerable for the attack, the network security solution can make the decision to let the traffic pass. If, however, the JAPPI identification indicates that the endpoint application would be vulnerable for the potential attack, the network security solution can make the decision to terminate the traffic.

4. Experimental results

To validate the accuracy and coverage of JAPPI we performed two real-life traffic experiments and compared the accuracy of JAPPI with our previous methodology. To build the models, we used a subset of a monitored corporate network, where we were able to receive endpoint application metadata from, as the Source. We first constructed a model using our previous methodology based on the JA3 fingerprints. We then formed several JAPPI models using the pre-hash strings and different clustering algorithms. Finally, we collected data sets from two larger network environments with different traffic profiles and compared the coverage of our models on these data sets.

We will first describe the setup of our experiment in Section 4.1, and the data collection steps in Section 4.2. Then in Sections 4.3–4.5 we explain the models that we used during the experiments. In Section 4.6 we present the results of our experiment, and in Section 4.7 we present the results of a lightweight experiment on how to improve threat detection using JAPPI.

Table 1
Examples of collected data.

Sample with endpoint metadata available:

```
"8a6f9c3259c58527963c5c922bd0ea87": {
  "TLS Client Hello JA3 Pre-Hash": "771,4865-4866-4867-49195-49199-
  49196-49200-52393-52392-49171-49172-156-157-47-53,23-65281-10-11-35-5-
  13-18-51-45-43-27-21,29-23-24,0",
  "chrome.exe": {
    "Google LLC": {
      "Windows 10": "Endpoint Operating System",
      "Google Chrome": "Executable Product",
      "103.0.5060.134": "Executable Version"
    }
  },
  "Spotify.exe": {
    "Spotify AB": {
      "Windows 10": "Endpoint Operating System",
      "Spotify": "Executable Product",
      "1.1.99.878": "Executable Version"
    }
  }
}
```

Sample with no endpoint metadata:

```
"8a6f9c3259c58527963c5c922bd0ea87": {
  "TLS Client Hello JA3 Pre-Hash": "771,4865-4866-4867-49195-49199-
  49196-49200-52393-52392-49171-49172-156-157-47-53,23-65281-10-11-35-5-
  13-18-51-45-43-27-21,29-23-24,0"
}
```

4.1. Setup

The experiments were conducted over one month. For data collection and exportation, we used two different networks where Forcepoint SD-WANs had been installed. We selected these network environments due to several reasons. One reason was that the traffic profiles of these environments were quite different. One environment was a corporate network with multiple subnets, spread out geographically between multiple remote sites and connected with VPN tunnels. The corporate network contained network traffic from workstations, internal servers, and automated systems. The other environment was a university network with one subnet and one site, containing network traffic from student housing dormitories. The traffic volumes were also considered adequate in both environments for validation, as the corporate network had approximately 3 million TLS connections per day while the student network had approximately 20 million TLS connections per day. Finally, the ease of setup was considered a strong advantage, as both environments were pre-existing installations meaning that no additional setup work was necessary.

In the corporate network a subset of the protected endpoints had the *Forcepoint Endpoint Context Agent* or *ECA* installed. The Forcepoint ECA sends endpoint application metadata to the Forcepoint SD-WAN for each network connection initiated from the endpoint. This enabled us to map each JA3 fingerprint and pre-hash string (as defined in Section 3.1) accurately to an endpoint application for this subset of endpoints. This subset was selected as the Source. There were no endpoints in the university network where endpoint metadata would have been available, as no endpoints had the Forcepoint ECA installed and reporting to the specific Forcepoint SD-WAN instance monitoring the network. Thus, the Source consisted only of endpoints in the corporate network.

4.2. Data collection

The data we collected from the monitored TLS connections consisted of the JA3 fingerprints and pre-hash strings calculated from the TLS Client Hello messages. The TLS connections produced by the Source also included metadata about the endpoint application which initiated the connection. The data was collected using the Forcepoint SD-WANs,

Table 2
Data sets for forming and validating models.

Initial data set from the Source	
Data collection period in days	1
Monitored TLS connections	184 790
Executables	115
Endpoint Applications	74
Unique JA3 fingerprints	84
Unique JA3 pre-hash strings	73
Unique "JA3 fingerprint, Endpoint application" pairs	179
Corporate data set, excluding the Source	
Data collection period in days	30
Monitored TLS connections	86 283 062
Unique JA3 fingerprints	131 553
Unique JA3 pre-hash strings	97 838
Corporate data set, Source	
Data collection period in days	30
Monitored TLS connections	5 645 593
Executables	265
Endpoint Applications	187
Unique JA3 fingerprints	58 879
Unique JA3 pre-hash strings	29 403
Unique "JA3 fingerprint, Endpoint application" pairs	59 202
University data set	
Data collection period in days	1
Monitored TLS connections	20 938 848
Unique JA3 fingerprints	30 661
Unique JA3 pre-hash strings	30 661

initially stored in a log server, and exported for the experiment using the Forcepoint SD-WAN Management Center. Table 1 presents samples collected from the Source group where endpoint metadata was available, and from the rest of the network where no endpoint metadata was received. The sample from the Source also shows the information received with the endpoint metadata, where two different endpoint applications produced identical JA3 pre-hash strings during the experiment.

In the corporate network we collected data for the entire month. We separated the data from the corporate network into three data sets. The initial data set contained data produced by the Source during the first day of the month. The volume of the traffic from the Source was around 190,000 TLS connections per day, or around 6% of all TLS traffic observed in the corporate network. This data was used for forming the models. The data for the rest of the month was split into two: one set contained data collected from the Source which included endpoint metadata, and the other set contained the rest of the data, which did not include endpoint metadata. From the university network we collected data for the first day of the month. Information about the data sets is presented in Table 2.

There is a difference in the amount of JA3 fingerprints and JA3 pre-hash strings collected from the corporate network. This difference is caused by the fact that the corporate network included several instances of the Forcepoint SD-WAN at different geographical locations, and some instances used an older version of the software which did not include support for reporting the JA3 pre-hash string in addition to the JA3 fingerprint.

From the initial data set we grouped different executables into endpoint applications. We used the endpoint metadata for mapping the executables into endpoint applications according to the "Product name" property, as well as some background knowledge we gained during the data collection and analysis period. As an example, we saw connections from an executable named *default-browser-agent.exe* that was initially reported as endpoint application *Firefox* based on the Product name property, but after further research we learned that these executables were actually *Windows scheduled tasks*, meaning the TLS connection they initiate, and thus the related JA3 fingerprint, should be mapped to the *Microsoft® Windows® Operating System* endpoint application instead of Firefox [33]. In addition, we needed to remove some duplicate values from the data set. Some endpoints in the Source had a local proxy installed on them, which was listed as the source executable for the connections that were passed through it instead of listing the original executable. This caused invalid clustering results, as several different executables got mapped into the same endpoint application.

Table 3
Initial samples and formed categories.

	Number of initial samples	Number of categories
Our previous methodology	84	84
DBSCAN, outliers as centroids	73	31
DBSCAN without outliers	73	15
K-Means, base nodes + random	73	7
K-Means, random	73	9

4.3. Forming a model using our previous methodology

To construct a model according to our previous methodology we collected the JA3 fingerprints and the related endpoint application information into a database. The database consisted of values in the form of "JA3 fingerprint: Endpoint application", where one JA3 fingerprint could be produced by multiple endpoint applications, and one endpoint application could produce multiple JA3 fingerprints. There were 75 unique JA3 fingerprints in the database.

4.4. Forming the DBSCAN models

We formed two models using DBSCAN. The base structure for both models was the same: we looked for all clusters with at least two members. The difference between the two models was how we treated the outliers. For the first model we considered each outlier to be its own category. The rationale was that it is possible for an endpoint application to only produce one unique JA3 pre-hash value during the data collection period, in which case it is valid to consider such an outlier its own category. This model had 31 categories. For the second model the outliers were discarded, and only the clusters were considered categories. This model had 15 categories.

4.5. Forming the K-Means models

We formed three models using K-Means. The difference was in the method of centroid initialization. We formed each model by running the K-Means algorithm once for every group size between 4 and 35 and selecting the one with the best accuracy. To assess the accuracy of a model we verified whether the values produced by one endpoint application were included in the same cluster, and whether the values from certain endpoint applications known to use different TLS libraries were included in different clusters.

For the first model we used random initialization. The most accurate model found using this initialization method had 9 categories. For the second we used a hybrid initialization of random and pre-selected. As the first centroids we selected a few JA3 pre-hash values that we knew were from different endpoint applications and selected the remaining centroids randomly. The most accurate model using this method had 7 categories. For the third model we used the K-Means++ algorithm for initialization. Using this method, we never found a model which would have satisfied our minimum accuracy requirement. It could be that the main clusters are quite close to each other, and some outliers further from them cause sub-optimal initialization with the K-Means++ algorithm.

Table 4
Results obtained from our previous methodology.

	Our previous methodology	%	Our previous methodology without Chromium values	%
Corporate data set, excluding the Source				
Unique JA3 fingerprints	97 838		958	
JA3 fingerprint identified	82	0,08%	31	3,34%
Corporate data set, Source				
Unique JA3 fingerprints	58 879		150	
Unique JA3 fingerprints where app also in initial data set	57 271		150	
JA3 fingerprint identified correctly, app also in initial data set	212	0,37%	84	55,80%
University data set				
Unique JA3 fingerprints	30 661		5864	
JA3 fingerprint identified	60	0,20%	53	0,90%

4.6. Results

Table 3 shows the number of unique JA3 fingerprints and pre-hash values that were collected from the Source network, as well as the number of categories for each method. For our previous methodology, the number of categories equals the number of samples collected from the Source, as there was no similar process of forming the model or grouping the values as there is with JAPPI. For the models based on clustering, the number of categories is the number of clusters for the most accurate model that was found when forming the models. We can see that most categories were in the DBSCAN model where outliers were considered independent categories, and the least number of categories was in the K-Means model using random initialization.

For each model, we present the results for our three data sets: Corporate data set excluding the Source, Corporate data set restricted to only the Source, and the University data set. For the Source we have information about the actual endpoint application, and we consider a value to be identified correctly if the used model mapped it to the correct endpoint application. For the Corporate data set excluding the Source and the University data set, we do not have information about the connection's actual source endpoint application. For our previous model we consider a processed JA3 fingerprint identified if it is present in the initial data set. For the JAPPI models we present the confidence level of the identification for each processed JA3 fingerprint: *Strong*, *Medium* or *Weak*, or if they were considered *Unknown*. The confidence level was decided based on the distance of the processed JA3 pre-hash value from the closest centroid. A distance of less than 10 was defined as *Strong* identification, a distance between 10–19 was defined as *Medium* identification, and a distance between 20–39 was defined as *Weak* identification. If the distance to the closest centroid was 40 or greater, the processed JA3-prehash value was considered *Unknown*.

The results for our previous model are presented in Table 4. We have separated the results into two groups: results for all values, and results for the values that were produced by something else than the Chromium Framework. The reason for this separation was the randomization of the order of the extensions performed by the Chromium Framework, which heavily affects the JA3 fingerprint. We wanted to see how well the model works for values that did not use randomization. To be able to separate the JA3 fingerprints that were produced by the Chromium framework, we utilized the JA3 pre-hash string and Levenshtein distance. We ignored the extensions altogether and looked for the values that without the extensions matched exactly to a value known to be produced by Chromium Framework. The desire to filter out the Chromium values meant that we needed to restrict the number of unique JA3 fingerprints available in the corporate data set excluding the Source: the original number of unique JA3 fingerprints for this data set was 131 553, but only 97 838 of them had also the JA3 pre-hash string available. We can see in Table 4 that there are fewer unique JA3 fingerprints left when the Chromium Framework values are removed. The relative number of non-Chromium values is higher in the University data set than in the corporate data set.

The overall coverage of our previous model is extremely poor. With the Chromium values included, the coverage did not reach even 0,5% with any data set. Removing the Chromium values did not have a significant impact on the coverage either. The biggest impact was seen when the corporate data set was restricted to the JA3 fingerprints produced by endpoint applications that were also included in the initial data set, where the coverage reached 55,8%.

The results for the JAPPI models are presented in Tables 5 (DBSCAN) and 6 (K-Means). There were no notable differences between the two different clustering models or their variations. All JAPPI models were able to identify around 99,1% of the corporate data set with Strong confidence. They were also able to identify around 85% of the University data set with Strong confidence. When considering the number of values that were identified with either Strong or Medium confidence in the University data set, there was a bit more variance in the results for the JAPPI models produced with different clustering algorithms. The DBSCAN models both performed a bit better than the K-Means models, the best being the DBSCAN model with outliers as centroids which reached a coverage of 93,78%. With all models, less than 2% of the TLS connections in the university network and less than 0,4% of the TLS connections in the corporate network were left as Unknown.

Fig. 5 shows a bar chart comparing the coverage results of our previous methodology to two JAPPI models. The selected JAPPI models use DBSCAN where outliers are considered centroids, and K-Means where centroid initialization was done using pre-selected and random centroids. The JAPPI coverage is the sum of values with either Strong or Medium confidence.

4.7. Enhancing intrusion detection

We also performed a lightweight experiment to see what the effect of JAPPI would be on improving threat detection. For this experiment we examined the log events produced by the inspection process from TLS traffic and considered whether they were valid events or false positives. We then considered if using JAPPI could have improved the efficacy and accuracy of the inspection process. For false positives, we considered if we could have prevented the event from triggering with JAPPI. Two example cases where JAPPI could have been used for improving the threat detection efficacy are listed below.

- *Un-trusted server certificates.* We found that Forcepoint SD-WAN had produced a lot of log events about un-trusted server certificates. This information can be useful for the network administrator if the notifications are scarce, as the administrator can for example identify potential phishing attempts. However, in this case it was seen that the number of these events was notably high. When investigating these events, we observed that many of the endpoint applications that were accessing these servers with un-trusted certificates were such that it was acceptable and normal behaviour for the application. For instance, we observed

Table 5
Results obtained from DBSCAN clustering models.

	DBSCAN, outliers as centroids	%	DBSCAN without outliers	%
Corporate data set, excluding the Source				
Unique JA3 pre-hash values	97 838		97 838	
Strong	97 019	99,16%	96 983	99,13%
Medium	313	0,32%	284	0,29%
Weak	202	0,21%	257	0,26%
Unknown	304	0,31%	314	0,32%
Corporate data set, Source				
Unique JA3 pre-hash values	29 403		29 403	
Unique JA3 pre-hash values where app also in initial data set	27 801		27 801	
JA3 pre-hash value identified correctly, app also in initial data set	27 759	99,85%	27 765	99,87%
University data set				
Unique JA3 pre-hash values	30 661		30 661	
Strong	26 397	86,09%	26 097	85,11%
Medium	2357	7,69%	2082	6,79%
Weak	1536	5,01%	2034	6,63%
Unknown	371	1,21%	448	1,46%

Table 6
Results obtained from K-Means clustering models.

	K-Means, base nodes + random	%	K-Means, random	%
Corporate data set, excluding the Source				
Unique JA3 pre-hash values	97 838		97 838	
Strong	96 959	99,10%	96 950	99,09%
Medium	83	0,08%	81	0,08%
Weak	462	0,47%	461	0,47%
Unknown	334	0,34%	346	0,35%
Corporate data set, Source				
Unique JA3 pre-hash values	29 403		29 403	
Unique JA3 pre-hash values where app also in initial data set	27 801		27 801	
JA3 pre-hash value identified correctly, app also in initial data set	27 783	99,94%	27 779	99,92%
University data set				
Unique JA3 pre-hash values	30 661		30 661	
Strong	26 130	85,22%	26 089	85,09%
Medium	1452	4,74%	1429	4,66%
Weak	2533	8,26%	2564	8,36%
Unknown	546	1,78%	579	1,89%

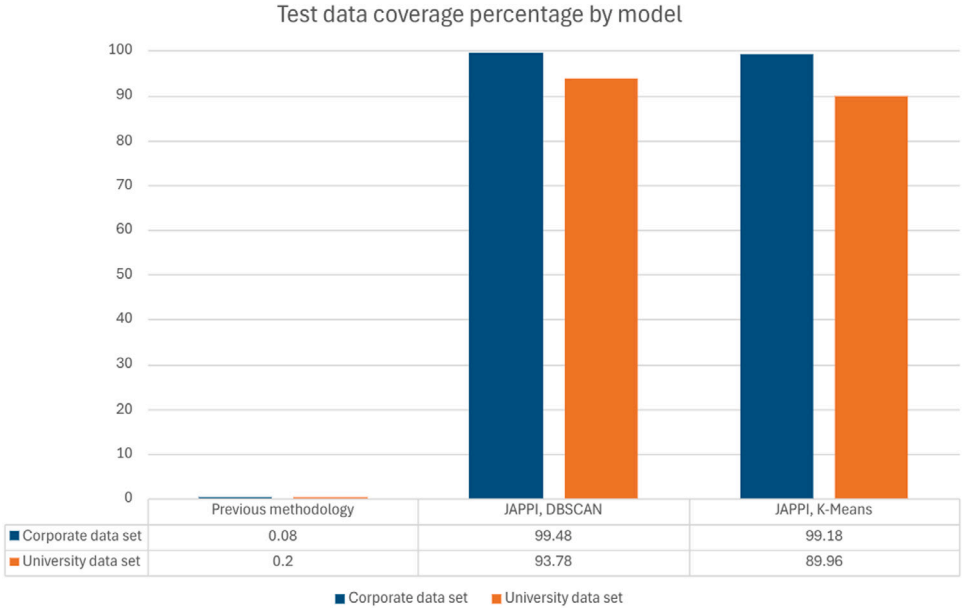


Fig. 5. A bar chart showing the coverage results of our previous methodology, a JAPPI model using DBSCAN, and a JAPPI model using K-Means.

that the Nessus scanner was producing a large amount of these notifications. The Nessus scanner is, among other things, used for identifying vulnerable software versions in the network. There is no need to produce an event if the Nessus scanner finds a server with an untrusted, potentially self-signed, certificate in the network, as this is to be expected. The JA3 pre-hash string produced by the connections was uniquely identified as Nessus by JAPPI, so with our methodology, these events could have been silenced.

- *Usage of anonymous Diffie–Hellman key exchange.* We found several log events about connections using anonymous Diffie–Hellman key exchange. Using anonymous Diffie–Hellman exposes the endpoints to a man-in-the-middle attack, and it is thus highly recommended to avoid using such keys. These connections were uniquely identified as Microsoft Teams by JAPPI, and in some instances this could be verified from the ECA metadata of the connection. It turned out that these matches were in fact triggered from Microsoft Teams traffic due to the usage of *Pseudo-TLS over TCP* [34]. The intention of Pseudo-TLS is to bypass Firewalls and other network security measures by fooling them into thinking that the traffic is TLS. The Pseudo-TLS is technically valid TLS, but in this case, there is no risk for the endpoints from the usage of anonymous keys. Using JAPPI these events could be ignored if the originating endpoint application is identified to be Microsoft Teams.

5. Comparison to other methods

In this section we will go through other existing methods for identifying endpoint application from network traffic presented in Section 2, and compare them to JAPPI. A clear difference between the other presented methods and our previous method and JAPPI is that both our previous method and JAPPI are methodologies defined specifically for a network security solution, including steps for selecting a source for the training material and maintaining the model. The other methods presented are generic methods for identifying endpoint applications from network traffic. To perform the comparison, we will go through the underlying functionality used for identifying the endpoint application presented by each method and the features they use, and compare them to the method used for JAPPI where clustering is used on the collected pre-hash strings.

5.1. Our previous methodology

As described in Section 2.4, our previous methodology used the hash fingerprints themselves instead of the pre-hash string used by JAPPI. The underlying features collected from the traffic are the same, as both are based on the hash fingerprinting algorithms, and specifically with JA3 when considering TLS traffic. The main difference is the flexibility attained by JAPPI from using the pre-hash string, and finding the clusters with machine learning. Due to using the MD5 hash, our previous methodology required an exact match from the traffic for the endpoint application to be considered identified. The method used by JAPPI provides resilience against minor changes in the values presented by the client.

5.2. Method by Husák et al.

The method used by Husák et al. [9] utilized the list of supported cipher suites presented by the client endpoint application. The JA3 algorithm collects this information as well, but it also collects additional information presented in the TLS Client Hello. This means that the JA3 algorithm, used by JAPPI, should be more specific, and thus it is likely able to better distinguish different endpoint applications from each other. In addition, the method used by Husák et al. [9] does not have any flexibility for changes in the values. This introduces the same issue that the original JA3 fingerprint has in that a new version supporting a new cipher suite will not be directly identified. It should still be noted that this method is resilient against the randomization of the order of the extensions implemented by the Chromium Framework [23] as this method does not use any information from the extensions. However, so is JAPPI, as described in Section 3.2.

5.3. Method by Muehlstein et al.

The method used by Muehlstein et al. [10] collects a large feature set from a TLS connection and uses machine learning (support vector machine with radial basis function) on the features. The feature set is much larger than the one used by the JA3 algorithm and thus JAPPI. This means that the method used by Muehlstein et al. [10] is most likely better at identifying small nuances in the traffic and thus has more granularity when categorizing traffic. The usage of machine learning on the feature set most likely also makes it resilient against small changes made during version updates. The downside with this method is that it collects information both from the client and server side, which means that both sides of the connection affect the identification. This is against the desire to only identify one side of the connection, and thus makes the method less usable for the purposes of this article. In addition, the information is collected from the duration of the entire connection. For the purpose of gaining endpoint awareness in a network security solution, the information needs to be available already at the beginning of the connection for the network security solution to be able to utilize it during the connection. For this reason, the method cannot address the problem setting of this article.

5.4. Method by Frolov and Wustrow

The method presented by Frolov and Wustrow [11] is very close to the method used by JAPPI. The researchers collected a feature set from the TLS Client Hello message which is very close to the information collected by the JA3 algorithm, but they collect a few more values as well. In addition to the information collected by the JA3 algorithm, Frolov and Wustrow [11] collect the TLS record level version, list of supported compression methods, signature algorithms and the list of protocols listed in the Application Layer Protocol Negotiation extension. They then take a SHA1 hash from these values, truncated to 64 bits, which is then considered the fingerprint. Unlike JA3, the form of the "pre-hash string" for this fingerprint is not specified.

After initial analysis of their results, the researchers noticed that the same endpoint application can produce several different fingerprints. Due to this, they experimented with using clustering on the collected data. Technically, this is very close to the method used by JAPPI on the pre-hash string, as the researchers used similar pre-hash data for the clustering. As this method collects more information from the TLS Client Hello than the JA3 algorithm, it would be expected that the produced identifications would be more granular than what is gained with the method used by JAPPI. The biggest strength that the method used by JAPPI has over the method used by Frolov and Wustrow [11] is the fact that the JA3 algorithm is a well-defined and popular standard that is already integrated into many network security tools as mentioned in Section 3.1. This makes implementing JAPPI using the JA3 algorithm easier than requiring the implementation of a new fingerprinting algorithm. However, it should be noted that with minor adjustments the JAPPI methodology could be implemented using another fingerprinting algorithm, such as the one introduced by Frolov and Wustrow [11].

5.5. Method by Anderson and McGrew

The method presented by Anderson and McGrew [12] is also very close to the method used by JAPPI. A feature set is collected from the TLS Client Hello message, which includes the version, list of supported cipher suites, and an unspecific set of information collected from the extensions. As opposed to the JA3 algorithm and the fingerprinting algorithm proposed by Frolov and Wustrow [11], this method does not use a specific set of features, but instead says that the fingerprinting process may change if they update their open source code base. The fingerprint syntax is presented as an octet string with the following order: (version)(cipher suites)((extensions)...). As the collected information may change, the syntax for the fingerprint is not well defined like the JA3 algorithm. The researchers do not use clustering on this information, but they do use the Levenshtein distance algorithm on the collected values for observing if a new value is close to an existing one.

The flexibility received from leaving the fingerprinting syntax open may lead to improved identification capabilities as compared to the JA3 algorithm used by JAPPI. The flexibility also brings disadvantages. For one, the information received from an updated fingerprinting algorithm will produce different fingerprints, and backward compatibility is not guaranteed. For another, if the proposed fingerprinting algorithm is used for a methodology such as JAPPI, an update to the algorithm may require updates to other parts of the implementation as well. Both of these features make the method less practical when considering it for gaining endpoint awareness in a network security solution. In addition, like with the method proposed by Frolov and Wustrow [11], a big advantage that the method used by JAPPI has is the fact that the JA3 algorithm is well defined and has been deployed in a range of different network security tools and solutions.

5.6. Method by Gomez et al.

Gomez et al. [13] use a method similar to the ones presented here, but their focus is on identifying different malware families from TLS traffic instead of identifying the different endpoint applications in a more generic sense. Despite the focus, their method should be usable also for generic endpoint application identification. The method collects a large set of features both from the Client and Server side, and from the entire duration of the connection. Clustering is then applied on the feature set to find the various malware families. Due to the large feature set collected both from the client and server during the entire connection, this method is similar to what [10] proposed. It also has the same drawbacks, where both sides of the connection affect the identification, and the identification can only be done after the entire connection has been seen. Due to this, the method cannot address the problem setting of this article.

6. Discussion

We only compared the efficacy of JAPPI to our previous methodology, but not to other similar implementations referenced in Section 2.2. The reason is that we did not have access to instances of relevant vendor devices. Gaining access to such instances and installing them in our test environments was considered out of the scope of our experiment. We did, however, discover that JAPPI brought significant improvements in coverage when compared to our previous methodology. JAPPI was able to cover 99,5% of the observed traffic in the corporate network and 93,8% of the traffic in the university network, compared to 0,37% and 0,2% using our previous methodology.

It is visible from the results of our experiment that the improvements brought by JAPPI upon our original method are far superior. A big reason contributing to this is the randomization of the extension list order by the Chromium Framework mentioned in Section 3, but as we saw from the results, the coverage was only 55.8% even when removing all Chromium values. There can be many reasons for the inferior performance of our original method, one of which could be that the data collection period was too short to properly populate the database. But the main issue is clear: even a small fluctuation in a pre-hash string produces a new string, and thus a new JA3 hash fingerprint. This is the reason an approach that leverages a distance algorithm is needed.

From the results for our previous model, we could see that there were more non-Chromium values in the University data set than in the corporate data set. This is to be expected, as the University data set is expected to be more diverse in both operating systems and endpoint applications. The substantial number of unique JA3 fingerprints produced by the Chromium Framework was still very much visible in both data sets. This is also expected, as randomizing the order of extensions will quickly produce many unique values.

Similarly, as the original JA3 fingerprint, JAPPI can also identify the underlying TLS library but not necessarily distinguish two endpoint applications using the same library. This is especially notable due to the vast usage of the Chromium Framework. When utilizing JAPPI, this limitation needs to be taken into consideration, and the identification from JAPPI should rather be used as additional metadata about the connection. It is especially important not to use the JAPPI identification as the only condition for permitting traffic due to this fact, as a malicious actor may be able to utilize the same underlying library as a legitimate application. However, utilizing the same TLS library heightens the likelihood that two endpoint applications may be vulnerable to the same type of attack, which means that there certainly is value with utilizing JAPPI on enhancing threat detection accuracy.

In our experiment, the Source consisted only of endpoints using the Windows operating system. The coverage of our model was impressive despite this deficiency. It is likely that many of the "Weak" and "Unknown" identifications might belong to another library not monitored during our experiment, potentially only existing for a different operating system. We have, however, previously noticed in [5] that the same endpoint application can produce the same JA3 fingerprint on different operating systems, which could be the reason for the good coverage during our experiment.

A potential issue in a large-scale network could be to have an extensive enough Source to cover the entire target network well enough. Otherwise, the number of connections categorized as "Weak" or "Unknown" rises. In a controlled network, such as a corporate network, this should be doable, but it gets more difficult in a non-controlled network, such as a university student network or a guest network. The better coverage the Source has, the less risk of a missing categorization there is.

It should be noted that only a limited number of values can be considered valid or legitimate for all sections in a hash fingerprinting algorithm. As an example, there are a limited number of cipher suites that are supported with TLS, and different TLS versions support different cipher suites. Still, a malicious actor could formulate TLS Client Hello messages that list an exceptionally large number of bogus cipher suites as supported. This, in turn, might lead to a situation where each run of the recursive Levenshtein algorithm takes up more resources than a legitimate TLS Client Hello message would take. A JAPPI implementation should take this into consideration and perform suitable mitigations when the Levenshtein distance is calculated. The JAPPI methodology has two steps where the Levenshtein distance algorithm is used: when the JAPPI clusters are formed and when JAPPI is utilized on active traffic. The second step is more pressing when the mitigations are considered, as the observed traffic may come from untrusted entities and the distance calculation may be performed in-line. The clusters are formed using data from the Source, which should consist of trusted endpoints. In addition, forming of the clusters is performed separately, and thus the potentially increased resource requirement should not pose a similar risk as with in-line calculations. For the utilization step,

there are a few potential mitigation methods that can be used. One method is to flag such a TLS Client Hello anomalous or malicious and not calculate the Levenshtein distance. Another approach could be removing invalid parameter values from the sections of the pre-hash string before calculating Levenshtein distance. Caching of previously observed pre-hash string sections and their distances to the corresponding sections in the current centroids may also be a suitable option. Caching does, however, not mitigate against attacks where the order of the bogus values is randomized for a section where the order of the values matters, and the values cannot be sorted before calculating the distance.

A concern for privacy might arise when considering the use of the pre-hash string instead of the MD5 hash based fingerprint. The MD5 hash does not, however, bring any notable privacy benefit in the case of the hash fingerprinting algorithms. As mentioned in the previous paragraph, there are only a limited number of valid values for each section, meaning that it would be easy to generate a database for mapping all legitimate hash fingerprints to their pre-hash strings. Because of this, in the context of privacy, taking the MD5 hash of the pre-hash string can be considered an obfuscation step at the most.

7. Conclusion

In this article we have introduced a novel methodology, JAPPI, for identifying endpoint applications from TLS traffic using clustering algorithms on the JA3 pre-hash strings. We performed a large-scale experiment on real-life traffic to validate our methodology and discovered that it performed remarkably better than our previous methodology which relied on simple string matching. Using JAPPI we were able to categorize 99,5% of the traffic in a controlled network environment, and 93,8% in a larger, uncontrolled network environment, compared to 0,1% and 0,2% using our previous methodology. The methodology can considerably improve the efficacy of network security solutions in many areas, such as enforcing a Zero Trust architecture, improving threat detection accuracy, and updating the risk score of a user.

CRediT authorship contribution statement

Jenny Heino: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Christian Jalio:** Validation, Software, Methodology, Data curation, Conceptualization. **Antti Hakkala:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Seppo Virtanen:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jenny Heino reports a relationship with Forcepoint LLC that includes: employment and equity or stocks. Christian Jalio reports a relationship with Forcepoint LLC that includes: employment and equity or stocks. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

The authors wish to thank Dr. Miika Hannula (University of Helsinki, Finland) for assistance in clarifying Definition 3.1.

References

[1] D. Holmes, Take Security To the Zero Trust Edge, Forrester Research, Inc, 2023, https://www.forrester.com/blogs/take-security-to-the-zero-trust-edge/. (Accessed 14 Aug 2023).
[2] Gartner, Inc, Definition of secure access service edge SASE, 2023, https://www.gartner.com/en/information-technology/glossary/secure-access-service-edge-sase. (Accessed 14 Aug 2023).
[3] McAfee Corp, Endpoint intelligence agent, 2019, https://docs.trellix.com/bundle/endpoint-intelligence-agent-v3-2-1-initNull-product/resource/prod-endpoint-intelligence-agent-v3-2-1-cat-product.pdf. (Accessed 11 September 2023).
[4] Forcepoint LLC, Forcepoint one endpoint and how it works, 2023, https://help.stonesoft.com/onlinehelp/StoneGate/SMC/6.10.0/GUID-A392A75D-7EBD-462D-A6FD-0E6F85E533B6.html. (Accessed 11 September 2023).
[5] J. Heino, C. Jalio, A. Hakkala, S. Virtanen, A method for endpoint aware inspection in a network security solution, in: IEEE Access, Vol 10, IEEE, 2022b, pp. 44517–44530.
[6] J. Heino, A. Hakkala, S. Virtanen, Categorizing TLS traffic based on JA3 pre-hash values, in: Procedia Computer Science, Vol 220, Elsevier, 2023, pp. 94–101.
[7] J. Heino, A. Gupta, A. Hakkala, S. Virtanen, On usability of hash fingerprinting for endpoint application identification, in: Proceedings of the 2022 IEEE International Conference on Cyber Security and Resilience (CSR), July (2022) 27–29, Virtual Conference, 2022a, pp. 38–43.
[8] R. Fielding, J. Reschke, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, RFC 7231, Section 5.5.3: User-Agent, Internet Engineering Task Force (IETF), 2014.
[9] M. Husák, M. Čermák, T. Jirsík, P. Čeleda, HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting, in: EURASIP Journal on Information Security 2016, Springer, 2016, pp. 1–14.
[10] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, O. Pele, Analyzing HTTPS encrypted traffic to identify user's operating system, browser and application, in: Proceedings of the 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), January (2017) 8-11, Las Vegas, NV, USA, 2017, pp. 1–6.
[11] S. Frolov, E. Wustrow, The use of TLS in censorship circumvention, in: Network and Distributed Systems Security (NDSS) Symposium, February (2019) 24-27, San Diego, CA, USA, 2019.
[12] B. Anderson, D. McGrew, TLS beyond the browser: Combining end host and network data to understand application behavior, in: Proceedings of the Internet Measurement Conference, October (2019) 21-23, New York, NY, USA, 2019, pp. 379—392.
[13] G. Gomez, P. Kotzias, M. Dell[']Amico, L. Bilge, J. Caballero, Unsupervised detection and clustering of malicious TLS flows, in: Security and Communication Networks, Vol. 2023, Hindawi, 2023, p. 17, Article ID 3676692.
[14] G. Lyon, Nmap network scanning, 2023, https://nmap.org/. (Accessed 11 September 2023).
[15] Tenable, Inc, Active scans, 2023, https://docs.tenable.com/security-center/Content/ActiveScans.htm. (Accessed 7 December 2023).
[16] Cisco Systems, Inc, Encrypted visibility engine, 2023, https://secure.cisco.com/secure-firewall/docs/encrypted-visibility-engine. (Accessed 11 September 2023).
[17] J. Althouse, J. Atkinson, J. Atkins, Open sourcing JA3, 2017, https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41. (Accessed 7 December 2023).
[18] J. Althouse, J. Atkinson, J. Atkins, TLS fingerprinting with JA3 and JA3s, 2018, https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967. (Accessed 11 September 2023).
[19] B. Reardon, Open sourcing HASSH, 2018, https://engineering.salesforce.com/open-sourcing-hassh-abed3ae5044c. (Accessed 7 December 2023).
[20] C. Yu, GQUIC protocol analysis and fingerprinting in zeek, 2019, https://engineering.salesforce.com/gquic-protocol-analysis-and-fingerprinting-in-zeek-a4178855d75f. (Accessed 7 December 2023).
[21] A. Karimishiraz, RDP fingerprinting, 2019, https://medium.com/@0x4d31/rdp-client-fingerprinting-9e7ac219f7f4. (Accessed 7 December 2023).
[22] M.R. Torres, SMBFP SMB fingerprinting zeek package, 2020, https://github.com/micrictor/smbfp. (Accessed 7 December 2023).
[23] Google Inc, Feature: TLS ClientHello extension permutation, 2023, https://chromestatus.com/feature/5124606246518784. (Accessed 11 September 2023).
[24] A.B. Spotify, Open source - spotify, 2023, https://www.spotify.com/us/opensource/. (Accessed 11 September 2023).
[25] T. Anderson, Adobe using google chromium embedded framework for edge tools, 2023, https://www.itwriting.com/blog/6616-adobe-using-google-chromium-embedded-framework-for-edge-tools.html. (Accessed 11 September 2023).
[26] Microsoft, Third party notices for microsoft power BI desktop, 2023b, https://powerbi.microsoft.com/en-us/desktop-3rd-party/. (Accessed 11 September 2023).
[27] U. Heilmeier, Commit f18ee30a3dc8495a3913043aa64c506d3e92fe13: 'TLS: Adding JA3 and JA3s fingerprints', 2021, https://github.com/wireshark/wireshark/commit/f18ee30a3dc8495a3913043aa64c506d3e92fe13. (Accessed 13 September 2023).

[28] Cloudflare, JA3 fingerprint - cloudflare bot solutions docs, 2021, https://developers.cloudflare.com/bots/concepts/ja3-fingerprint/. (Accessed 13 September 2023).
[29] J. Kindervag, Build security into your network's DNA: The zero trust network architecture, 2010, https://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf. (Accessed 11 May 2024).
[30] Zscaler Inc, Understanding user risk score, 2024, https://help.zscaler.com/zia/understanding-user-risk-score. (Accessed 13 May 2024).
[31] Splunk Inc, Use splunk enterprise security risk-based alerting, 2024, https://docs.splunk.com/Documentation/ES/7.3.1/RBA/Analyzerisk. (Accessed 13 May 2024).
[32] Fortinet, Inc, What is deep packet inspection (DPI)?, 2024, https://www.fortinet.com/resources/cyberglossary/dpi-deep-packet-inspection. (Accessed 13 May 2024).
[33] Mozilla Foundation, Default browser agent – firefox source docs, 2023, https://firefox-source-docs.mozilla.org/toolkit/mozapps/defaultagent/default-browser-agent/index.html. (Accessed 26 June 2023).
[34] Microsoft, Pseudo-TLS over TCP, 2023a, https://learn.microsoft.com/en-us/openspecs/office{_}protocols/ms-turn/ae97f19f-88ea-44a7-bb2e-2b23cb645da5. (Accessed 23 Aug 2023).

Jenny Heino is a doctoral researcher at the Department of Computing, University of Turku, Finland. She completed her Master's degree in Mathematical Logic at the University of Helsinki in 2012. After working as a Security Researcher for the Forcepoint Next Generation Firewall for five years, she began her doctoral studies at the University of Turku in 2020, while maintaining her full-time Principal Security Researcher position at Forcepoint. Her research interests currently focus on identifying endpoint applications from network traffic, and ways to utilize this information for improved network security.

Christian Jalio holds a Master of Science degree in Software Development from Aalto University. Currently he is the vulnerability research team lead for the Forcepoint Next Generation Firewall. His current research interests focus on fuzzing, which has resulted in many vulnerability discoveries in products such as Google Chrome and LibreOffice.

Antti Hakkala received his D.Sc.(Tech.) degree in communication systems in 2017 from the University of Turku. He is currently a University Teacher in Communication Systems and Cyber Security at Department of Computing, University of Turku, Finland. He has over 10 years' experience in teaching engineering students on cyber security and communication systems engineering, and has supervised over 100 Bachelor's and Master's theses on cyber security topics. His current research interests include security and privacy in networked information society, wireless network security, and cyber security in autonomous systems.

Seppo Virtanen received the D.Sc.(Tech.) degree in communication systems from the University of Turku, Finland, in 2004. He is currently a Professor of Cyber Security Engineering with the Department of Computing, University of Turku, where he is also the Vice Head of the Department. His current research interests are on the application of artificial intelligence and large language models to network and cyber security, security of smart environments, and cyber security in societal processes.