.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A method for endpoint aware inspection in a Network Security Solution

JENNY HEINO[1,2], CHRISTIAN JALIO[2], ANTTI HAKKALA[1] AND SEPPO VIRTANEN[1] (Senior Member, IEEE).

[1]Department of Computing, University of Turku, 20014 Turku, Finland
[2]Forcepoint LLC, Itälahdenkatu 22 A, 00210 Helsinki, Finland

Corresponding author: Jenny Heino (e-mail: jeahei@utu.fi).

ABSTRACT Due to the surge in remote work after the outbreak of COVID-19, network security has gained an enormous focus. The issue of erroneous inspection decisions in network security solutions has long been criticised, but the importance of the decision accuracy has never been as important as today. In this paper we provide a solution for improving the inspection decision accuracy by specifying a method for endpoint aware inspection in a network security solution capable of performing deep packet inspection. The method utilises a subset of the protected network to gather hash fingerprints from the endpoint application network traffic patterns. The information gathered from this subset is then utilised for gaining endpoint awareness for the rest of the protected network. We use methods that work on the application layer of the protocol stack. This makes the method applicable not only for local implementations, such as NGFWs and IPSs, but also for SaaS and SASE solutions. The method is, however, easily utilised with lower layer information, such as network and transport layer information, for operating system awareness as well. We also present a proof-of-concept case study where we observe that, of the applicable network connections, 100% could be identified when the operating system and endpoint application were present in the source group. To our knowledge, this is the first method to enhance the inspection process accuracy by leveraging a subset of the protected network to gain endpoint awareness.

INDEX TERMS Computer network management, Firewalls (computing), Middleboxes, Network security

I. INTRODUCTION

The network security landscape experienced an abrupt change due to the COVID-19 pandemic, as remote work became the norm within a few months. The acute need for network security solutions put additional pressure on the network security solution providers as well, and problems that might have been prioritized lower before the pandemic quickly became high priority. One of such problems is the accuracy of deep packet inspection related observations.

A network security solution, which is used for providing additional security to the network, may utilise many different features for analysing the network traffic. These features often include deep packet inspection, intrusion prevention systems, network application identification, URL and content categorisation and TLS decryption. Depending on the network security solution, it may be able to terminate content it considers malicious, or it may only provide additional information for the network administrator. Examples of such network security solutions include *Next Generation Firewalls* or NGFWs, *Intrusion Prevention Systems* or IPSs, *Intrusion Detection Systems* or IDSs and *Secure Web Gateways*.

A key element of a network security solution is the ability to identify malicious content from the network traffic. When considering the goodness of entities in a network security solution, such as protocol patterns, files, or URLs, it is often relatively easy to identify the entities that are certainly good or bad. There is an area in the middle, however, a grey area, where it is not as easy to make the distinction. In this grey area, the content can be malicious, or it can be legitimate and just happen to exhibit patterns that may be determined suspicious. It is also possible that a certain pattern in a file or a misuse of a protocol causes one endpoint application to crash, even if the content itself is not intentionally malicious.

This grey area is where *false positives* and *false negatives* happen in a network security solution. False positives and false negatives happen when the network security solution makes an assessment of the goodness or badness of the inspected entity, and that assessment is wrong. When a false

positive happens, the network security solution has made the assessment that the entity is malicious, when in fact it was not. Correspondingly, a false negative happens when the network security solution has made the assessment that the entity is benign, when it was actually malicious. False positives and false negatives can be produced from any part of the network traffic which a network security solution processes. This includes, for example, files, URLs and network protocol patterns.

As the inspection observations become more uncertain, the decision needs to be made by the network security solution whether the potential impact of a false negative outweighs the inconvenience of a false positive. This decision is made more complicated by the fact that a network security solution is usually blind to the local context: what kind of endpoint, and which endpoint application, is the recipient of the traffic.

The grey area of uncertainty is poison for a network security solution. When the decision to block or allow uncertain traffic is wrong, there are only bad consequences. On one hand, if the decision ends in a false positive termination, the administrator needs to do manual work to permit the connection. If false positives are common, the administrator will often make a decision to reduce security in general to ease the workload the false positives create. On the other hand, if the decision ends in a false negative, something malicious is let through, which can have even worse consequences.

To solve the problem of false positives and false negatives, awareness of the local context is needed by the network security solution to form a more conscious decision. Our approach to solving this problem is for the network security solution to become aware of the endpoint application which will be receiving the content. We refer to this as *endpoint aware inspection*. Information on the receiving endpoint application can be significant when considering the grey area of network traffic.

As an example, a new and publicly exploited vulnerability might be discovered in the way one web browser parses a certain file type. Depending on the inspection capabilities of the network security solution, it might not be possible to identify a file exploiting this vulnerability with a 100% certainty. This leaves the network security solution with the choice between a risk of producing false positives, and a risk of accidentally permitting a malicious file through. With visibility into the receiving endpoint application, however, the network security solution can make a much more informed decision about whether to permit the file through or not. When a file considered to potentially exploit this vulnerability is observed, the network security solution can terminate the traffic if the receiving endpoint application would be the vulnerable web browser. Meanwhile, the same file may be permitted through, if the receiving endpoint application is found to be something else.

In this paper, we specify a method for endpoint aware inspection in a network security solution capable of performing deep packet inspection. The method uses a small subset of the protected network to collect verifiable information for mapping traffic patterns to endpoint applications. This information is stored and later utilised to identify endpoint applications from traffic patterns where the endpoint application cannot be verified otherwise. To our knowledge this is the first method to enhance the inspection process accuracy by leveraging a subset of the protected network to gain endpoint awareness.

We use protocol hash fingerprinting algorithms, such as JA3, JA3S, HASSH, CYU and RDFTP, discussed further in Section III-C, to identify the endpoint application. These algorithms were selected due to their simplicity, topicality and their shown promise. However, our method can easily be modified to support any passive fingerprinting method.

Network and transport layer information has often been used for fingerprinting the endpoint operating system [1]. It is noteworthy that the information on these lower protocol levels can, and often will, be different, depending on where the inspection is being performed. A router performing Network Address Translation (NAT) on the traffic will modify both the network and the transport layer information, which means that the identification made based on network and transport layer information will be different on both sides of the router. In addition, this lower layer information can only be used for identifying the operating system, not the endpoint application. Fig. 1 demonstrates an example network configuration, and how the application layer information stays static while the lower layer information changes on the route from the client application to the server application.

Due to using the above mentioned protocol hash fingerprinting algorithms, our method works on the application layer of the protocol stack, on top of the network and transport layer. Because of this, it is not limited to local implementations that are able to see the lower level information untouched, such as *IPS's* and *NGFW's* installed on the perimeter of the protected network. Instead, it can also be applied to cloud security implementations such as *Security as a Service* (*SaaS*) or *Secure Access Service Edge* (*SASE*) solutions. The method is, however, easily modified to utilise lower layer information for operating system identification as well.

In this paper the term *endpoint* refers to the combination of an operating system and endpoint applications installed on top of it. This may include physical endpoints, such as desktops, laptops or server machines, as well as virtual machines. The best results for endpoint aware inspection are gained when all protected endpoints send endpoint application metadata, such as product name and version, to the network security solution on each new network connection that is opened. This way the network security solution can be certain of the endpoint application and make decisions based on accurate information. However, it is often not possible to install an external component, which sends metadata to the network security solution, on all protected endpoints. The reason is usually either that the operating platform is closed and installing such an external component is impossible, or that the network security solution provider does not have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

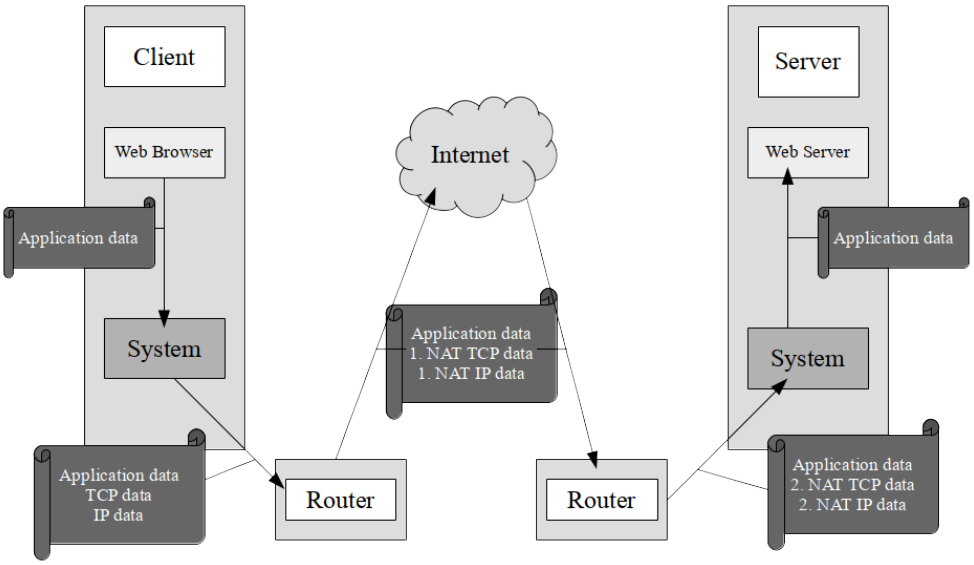

FIGURE 1. *An example network configuration demonstrating application layer data being sent from a client application (Web Browser) to a server application (Web Server). Even though the lower layer data changes several times during the route, as several routers perform NAT on the traffic, the application layer data remains static.*

support for a particular operating system for their external component. Our method acknowledges this and proposes instead that information on the traffic patterns of different endpoint applications is gathered into a database from a subset of protected endpoints where such an external component has been installed and the original endpoint application is known for certain. These traffic patterns are stored in the form of hash fingerprints. This information can then be utilised for identifying the endpoint applications from the endpoints where no external component has been installed, and used when making an inspection based decision.

This method is not a silver bullet solution that entirely removes the problem of false positives and false negatives from a network security solution. Rather, it is a way to extend the context in which the network security solution makes its conclusions. Preliminary results presented in section IV demonstrate, however, that the method has promise, and that it can quite reliably identify the endpoint applications even when the operating system for the source and the target are different.

The rest of the paper is organized as follows. In section II we take a look at existing methods for solving the issue of false positives and false negatives, and consider their limitations. In section III we give a detailed specification for the method and the included components and steps. In section IV we present a proof-of-concept case study where we demonstrate how the method works in a real environment. In section V we discuss different aspects of the proposed method. Finally, in section VI, we make a conclusion of our paper.

II. RELATED WORK

The problem of lacking context in network based threat detection has stimulated a lot of discussion and criticism. There are many attempts to solve this problem, and in this section we present the most common ones. Some methods are better than the others, but each method has its limitations.

The use of active scanners is one of the most common solutions for providing visibility to the endpoints. Active scanners may be installed separately, or they can be an integrated part of the network security system itself. Examples of active scanners are *Nmap* [2] and *Nessus* [3]. Nmap is free to use open-source software. It contains a feature titled "Version detection" [4], which reveals information on the endpoint application based on an active scan. Nmap can currently identify over 8800 different endpoint applications and versions [5]. Nessus on the other hand is a commercial scanner which implements features such as port scanning and vulnerability scanning.

Active scanning has its uses, especially in identifying the server applications. However, client applications cannot be covered with active scanning. In addition, active scanning needs to be performed regularly to keep an up-to-date database of the assets, which can take time. Even then, it is possible that the version or even the endpoint application changes between the scans.

Another active endpoint application detection method is the use of a *Configuration Management Database* or CMDB. A CMDB is a database which contains information on different organisation assets, which can include endpoint applications and endpoint application versions installed on different endpoints. The usefulness of a CMDB in the context of endpoint aware inspection depends heavily on how up-

to-date the database is at any given moment. An actively updated CMDB which includes information on installed endpoint applications and endpoint application versions can be a valuable source for endpoint application information, but active updating may require heavy resources [6].

Different passive identification methods have also been developed. Especially passive operating system fingerprinting is a part of many network security solutions [7]–[9]. The methods for passive OS fingerprinting range from simple string matching on plain-text application layer data (such as HTTP User Agent) to fingerprinting the parameter values sent in the TCP handshake [10]. Passive endpoint application fingerprinting is often more complex and thus not as largely utilised as OS fingerprinting. For some protocols there exists a plain text field revealing the endpoint application, such as User Agent for HTTP. More often than not, however, the endpoint application is not as explicitly expressed and requires more advanced identification methods. Several methods with varying success rates have been proposed especially for identifying the endpoint application from TLS [11]–[13]. None of the passive fingerprinting methods are 100% effective, and all are susceptible to changing when the underlying application is updated. Each passive fingerprinting method also requires a trusted source for information on which endpoint application maps to which fingerprint.

The most accurate method for providing endpoint awareness to a network security solution is if an endpoint sends metadata for each new network connection to the network security solution. This metadata should at minimum include the information on the endpoint application which initiated the connection. This way the network solution will always receive current and exact information on the endpoint application initiating the network connection being inspected. The drawback to this method is that on the endpoint the installation of an external component sending the metadata to the network security solution is required. It is, however, not always possible to install an external component on each protected endpoint. Several network security solutions have implemented these types of external metadata components [14], [15].

Cisco Systems has developed a process of populating a database of TLS fingerprints based on metadata received from endpoints [16], [17]. Based on published information, this process could have similarities to the steps of populating the fingerprint database specified in our method. Cisco has developed a proprietary fingerprinting format and does not use hash fingerprints as our method does. We did not find any reference of improving inspection accuracy with the collected information from the public materials from Cisco, whereas in our method, improving the accuracy of the deep packet inspection process is a core functionality.

III. PROPOSED METHOD FOR ENDPOINT AWARE INSPECTION IN A NETWORK SECURITY SOLUTION

In this section we specify a method for endpoint aware inspection in a network security solution capable of performing deep packet inspection. Utilising this method, a network security solution can have higher confidence in its deep packet inspection based decisions by gaining an awareness of the protected endpoint application. The method consists of four components which are utilised in four steps. The four components and four steps are visualised in Fig. 2. We will first give a high level description of the method, after which we will provide detailed definitions for the concepts needed by the method in sections III-A, III-B and III-C. Based on this we finally provide a thorough definition of the method in section III-D.

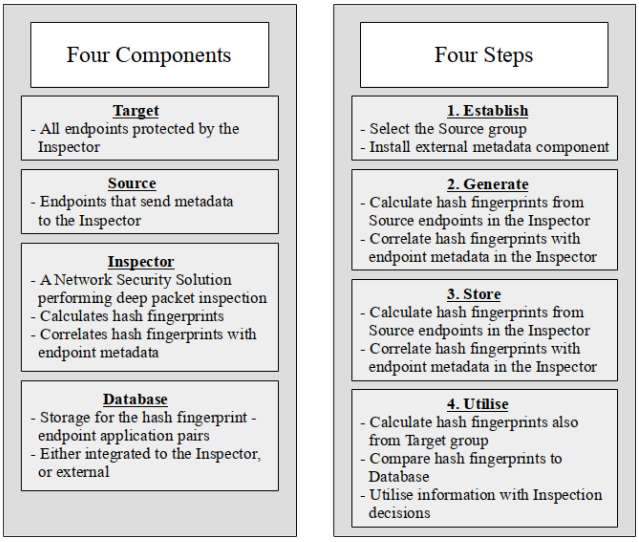

FIGURE 2. *A simplified overview of the four components and four steps defined in the method.*

The four components used by the method are the *Target*, the *Source*, the *Inspector* and the *Database*.

- The *Target*: The entire set of protected endpoints.
- The *Source*: A subset of the endpoints belonging to the Target that send metadata to the Inspector about the endpoint applications for each connection.
- The *Inspector*: A network security solution which performs deep packet inspection on the traffic, is able to calculate hash fingerprints from it, and correlate them with the endpoint application based on the received metadata.
- The *Database*: A storage for the endpoint application - hash fingerprint pairs. The database can be integrated to the Inspector, or it can be an external component.

These four components are then taken into use in four steps: *Establish*, *Generate*, *Store* and *Utilise*.

- 1. *Establish*: Select suitable endpoints to the Source group. Install to the selected endpoints an external component which provides endpoint metadata for the Inspector on each new network connection.
- 2. *Generate*: For a suitable period of time, calculate hash fingerprints in the Inspector for the network connections initiated by endpoints in the Source group. Correlate hash fingerprints with the endpoint application.

- 3. *Store*: Store the correlated hash-endpoint application pairs in the Database. During this period the Database is populated so that enough information has been gathered to have a good coverage of the network traffic for the next step.
- 4. *Utilise*: Start calculating hash fingerprints in the Inspector also for the network connections initiated by the endpoints in the Target group. Deduce the endpoint application based on the information in the database, and use the information when making an inspection decision. During this phase, the Database will continue to be populated from the data received from the Source group to keep it current.

Fig. 3 provides a rough visualization of one use case of the method. On the left it depicts the *Target* group, which consists of all endpoints that are protected by the *Inspector*. A subgroup of the endpoints are tagged with "Source": these endpoints are the *Source* group, providing endpoint metadata to the Inspector. One of the endpoints in the Source group initiates a network connection, and sends metadata to the Inspector that the network connection was initiated by the Firefox web browser. The Inspector calculates a hash fingerprint from the connection, and stores the endpoint application metadata with the hash to the *Database*. When an endpoint in the Target group, that does not belong to the Source group, initiates a new network connection, the Inspector calculates a hash fingerprint from the connection, and checks if the Database has information on the endpoint application for this hash value. In the example the hash is mapped to Firefox. The Inspector can then use this information when formulating inspection decisions for the connection.

A. ENDPOINT APPLICATION

In the context of this paper, we use the term *endpoint application* to reference the software components that are installed on an endpoint and communicate with other software components on other endpoints over a network. Such endpoint applications may be client applications such as web browsers or e-mail clients, but also server components such as web or e-mail servers.

We do not want to restrict the definition to the exact instance of an endpoint application, however. If we identify the endpoint application with something too strict, such as a hash value of the executable, we will lose a large part of the extensibility of the method. This is due to the fact that the network behaviour of an endpoint application can remain constant while other attributes in the endpoint application are changed. The network behaviour can thus remain constant even between different versions of the application. In addition a simple hash value of the executable would restrict the endpoint application identification to a specific operating system and platform, which is unnecessary, as the preliminary results in the case study in section IV demonstrate that the hash fingerprints can remain constant even across different operating systems.

Different operating systems can have different metadata stored for each executable. Windows can store file version information with binary files such as .exe or .dll files [18]. This information may contain the *Product name* property, which defines the product the binary file was distributed with [19]. In most cases, the Product name property is a sufficient way to identify the endpoint application. On most Unix-like operating systems similar information can be gathered based on which package the file was installed with [20], [21]. On Unix-like systems the information about the package corresponds well enough to the product information on Windows. Correlating the same endpoint application between different operating systems may require manual work as the package name is often not exactly the same as the Product name. This correlation is left out of the scope of this paper and is assumed to be done during the implementation.

In addition, we require that the endpoint *trusts* the endpoint application. It is trivial to modify the file version information on Windows or override the file on Unix-like operating systems. However, these operations break the inherent trust the operating system has on the file. On Windows, the executable needs to be signed by a trusted entity, and the signature breaks when the properties are modified. Integrity checks also exist in the package managers on Unix-like systems.

B. ENDPOINT METADATA

Receiving endpoint metadata from a subset of endpoints is a key requirement of this method. With the endpoint metadata it is possible to confidently map the hash fingerprints calculated by the network security solution to an endpoint application. The metadata needs to include enough information to uniquely map each network connection to an endpoint application. For uniquely identifying the network connection, the 5-tuple of source IP address and port, destination IP address and port, and the transport protocol is sufficient. The requirements for identifying the endpoint application are discussed more in the section III-A. The metadata can always include more information as well, such as a hash value of the executable or the version number of the application, but this should not affect the endpoint application identification in the context of this method.

For an endpoint to send metadata to the network security solution, an external component will need to be installed on the endpoint. This often requires that the network security solution vendor develops a specific component for this purpose. Examples of such components include the *Endpoint Intelligence Agent* (*EIA*) for McAfee Network Security Platform [14] and the *Endpoint Context Agent* (*ECA*) for Forcepoint Next Generation Firewall [15].

C. HASH FINGERPRINTS

A *fingerprint* is a short tag for a larger object [22]. In general, a fingerprint is produced by a *fingerprinting algorithm*. In the method presented in this paper we use fingerprinting algorithms that produce an MD5 hash value as the result-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

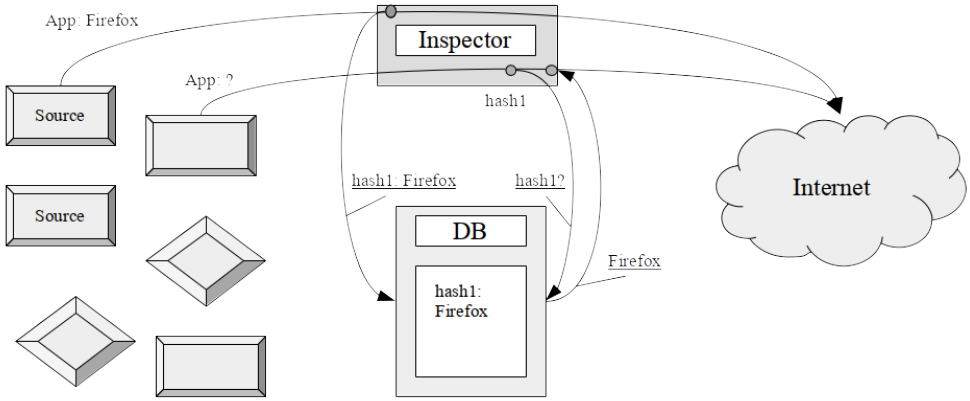

FIGURE 3. *A high level representation of one use case of the method. The Target group is presented on the left, and two endpoints have been selected in the Source group. The network connections initiated by the endpoints in the Target group go through the Inspector. One connection initiated by Firefox in the Source group is displayed. The Inspector calculates the hash fingerprint for this connection and stores it in the Database along with the application metadata. The information is later utilised when an endpoint not belonging to the Source group makes a network connection.*

ing fingerprint [23]. The method can, however, easily be extended to utilise any fingerprinting algorithm.

Several fingerprinting algorithms have recently been developed for fingerprinting endpoint applications from different protocols. The unifying idea between these algorithms is that there are protocol parameters for which different endpoint applications will either select different values, or present the values in a different order. The algorithm defines how these values are stored together in a data structure, and an MD5 hash is calculated from the collected data, comprising the hash fingerprint. Most of the algorithms are so recent that no academic papers have yet been published on their accuracy. In this section we introduce the existing hash fingerprinting algorithms for endpoint application identification. All referenced algorithms are open-sourced, and several open-sourced tools implementing the algorithms exist, such as Zeek [24] and fatt [25].

The *JA3* and *JA3S* fingerprints were the first such hash fingerprints proposed for identifying endpoint applications. JA3 and JA3S fingerprints are used for identifying the client (JA3) and server (JA3S) applications from TLS traffic. The method was developed by Salesforce employees J. Althouse, J. Atkinson and J. Atkins and open-sourced by the company in 2017 [26]. It is based on research by L. Brotherston, presented in 2015 at DerbyCon [27]. The JA3 and JA3S methods utilise clear text information presented during the TLS handshake, such as version, supported cipher suites and extension information, for calculating the hash fingerprint. The JA3 and JA3S fingerprints are the only hash fingerprinting method presented here that has already been academically studied to some extent. The research has, however, mainly focused on identifying malicious connections [28].

Following JA3 and JA3S fingerprints, Salesforce has released similar fingerprinting methods for other protocols as well. In 2018 Salesforce employee B. Reardon published fingerprinting algorithms for SSH client and server identification, called *HASSH* and *HASSHServer* [29]. This method utilises information in the *SSH_MSG_KEXINIT* messages sent in clear text by both the client and the server. In 2019 Salesforce employee C. Yu presented a similar fingerprinting algorithm for identifying endpoint applications from gQUIC traffic called *CYU* [30]. This method again utilises clear text information presented in the Client Hello message.

Complimenting the work by Salesforce, individual researchers have presented hash fingerprinting algorithms for identifying endpoint applications for other protocols. A method for fingerprinting RDP endpoint applications, called *RDFP*, is presented by A. Karimishiraz in [31] published in 2019. The author notes that clients using the Enhanced Security mode can be fingerprinted using the JA3 fingerprinting method, but the RDFP method can be used for identifying the clients that use the Standard Security mode. A method for fingerprinting DHCP clients titled KYD is presented by F. Bannatwala in [32]. The method is based on the work presented by T. Coffeen in [33]. Finally, a method for fingerprinting SMB clients, called *SMBFP*, was introduced in 2020 by M. R. Torres in [34]. The author describes the method as being *incredibly bleeding edge*.

One problem with hash fingerprinting is brought up in the study by J. A Truong in [28] on JA3 and JA3S fingerprints. This problem is collisions, caused by the fact that several endpoint applications use the same underlying TLS libraries to perform TLS connections. Due to these collisions, such a hash fingerprint based endpoint application identification needs to be regarded as a potential identification, rather than a certain one.

D. METHOD DESCRIPTION

In this section we present a formal specification for the method for endpoint aware inspection in a network security solution capable of performing deep packet inspection. First we define the premise for the method, after which we present a detailed description of the method. The method has four components: Target, Source, Inspector and Database, as well

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

as four steps: Establish, Generate, Store and Utilise. The components and steps are presented in a simplified form in Fig. 2.

1) Definitions.

Let N be a network of endpoints ($|N| \geq 2$) protected by a network security solution $\mathcal{W}$. Let $\mathcal{W}$ be able to perform deep packet inspection on the network traffic, and perform security analysis based on the inspected content.

The endpoints $n \in N$ contain endpoint applications; let A be the set of all network applications in all endpoints $n \in N$. The endpoint applications $a \in A$ make network connections; let C denote the set of network connections initiated by all endpoint applications $a \in A$ for all endpoints $n \in N$.

Let H be the set of MD5 checksums; let $h\colon C \rightarrow H$ be the function for calculating a hash fingerprint for a network connection $c \in C$ as described in section III-C. Let $o\colon H \rightarrow \mathcal{P}(A)$ denote the function which returns the set of network applications $a \in A$ that have been mapped to the MD5 hash $d \in H$.

Let P be the set of properties, such as URL, file, or protocol pattern, in a network connection $c \in C$ that may be categorized by the network security solution $\mathcal{W}$; let $g_1\colon C \times P \rightarrow \{0, 1\}$ denote a function for determining whether the network security solution $\mathcal{W}$ will create an inspection match from the property $p \in P$ for connection $c \in C$. $g_1(p)$ will return 0 when a match will not be created, and 1 when a match will be created. Let $g_2\colon C \times P \times \mathcal{P}(A) \rightarrow \{0, 1\}$ denote a similar function, but with endpoint application awareness utilising the o function.

2) Components

Target. The Target group is the entire set N of protected endpoints. An *endpoint* in the Target group refers to the combination of an operating system and endpoint applications installed on top of it. This may include physical endpoints such as desktops, laptops or server machines as well as virtual machines.

Source. The Source group $S \subset N$ is a subset of the protected network. It should form a good sample of the endpoints present in the protected network: if different operating systems are used in the network, the sample group should contain at least one endpoint from each group of operating systems. The preliminary results presented in section IV, however, show that some endpoint applications produce same hash fingerprints between different operating systems, which indicates that the method can be useful even if this requirement cannot be met.

Inspector. The Inspector is the network security solution $\mathcal{W}$ capable of performing deep packet inspection. All network connections $c \in C$ initiated by the endpoints $n \in N$ go through the Inspector. The Inspector is able to calculate hash fingerprints using h from certain network connections $c \in C$ that are supported for hash fingerprinting by $\mathcal{W}$. The network security solution $\mathcal{W}$ may be able to calculate hash fingerprints for one or more network protocols. The Inspector is also able to correlate the hash fingerprints with the endpoint applications $a \in A$ based on the metadata received from the endpoints in the Source group $S \subset N$.

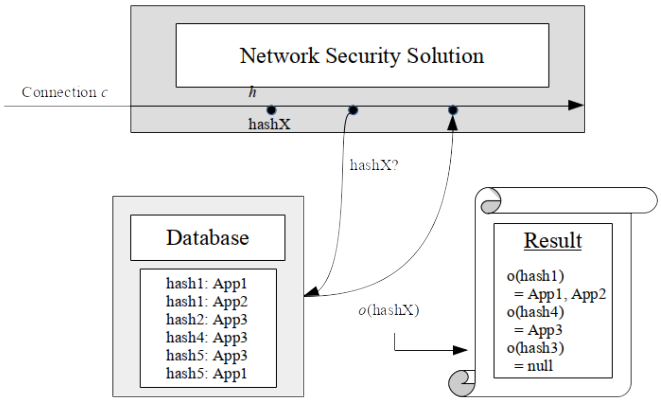

FIGURE 4. *Visualization of the Utilise step. The Result section shows different outputs for the o function for different hash fingerprints.*

Database. The Database is used for storing the hash fingerprints $d \in H$ as well as metadata for the endpoint applications $\{a_1, ..., a_n\} \in A$ that have been associated with the hash fingerprint by the Inspector by the function o. The contents of the database may be limited purely to the list of endpoint applications associated with d by $o(d)$, or it may contain additional metadata as well. This database may either be an integrated database in the network security solution $\mathcal{W}$ itself, or an external database. An integrated database may be easier and faster to implement. An external database may, however, be more useful in case it is desired that data from several different Source groups S from different protected networks N is collected in one database. In very large networks an external, centralised database may thus be more convenient.

3) Steps

Establish. In the fist step, the Source group $S \subset N$ is established. The external components that provide endpoint metadata, as defined in section III-B, for the network security solution $\mathcal{W}$, are installed on the endpoints in the Source group.

Generate. In the second step, the network security solution $\mathcal{W}$ inspects the network connections $c \in C$ initiated by the endpoint applications $a \in A$ in the Source group $S \subset N$, and calculates hash fingerprints using function h from the connections $c \in C$ that are supported for hash fingerprinting by $\mathcal{W}$. These hash fingerprints are correlated to the original endpoint application based on the endpoint metadata by function o.

Store. In the third step, the correlated data, which includes but is not necessarily limited to, the endpoint application and the calculated hash fingerprint, is stored in a database.

Utilise. In the fourth step, the information stored in the database is utilised for the entire Target group N. When the network security solution $\mathcal{W}$ receives a network connection

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

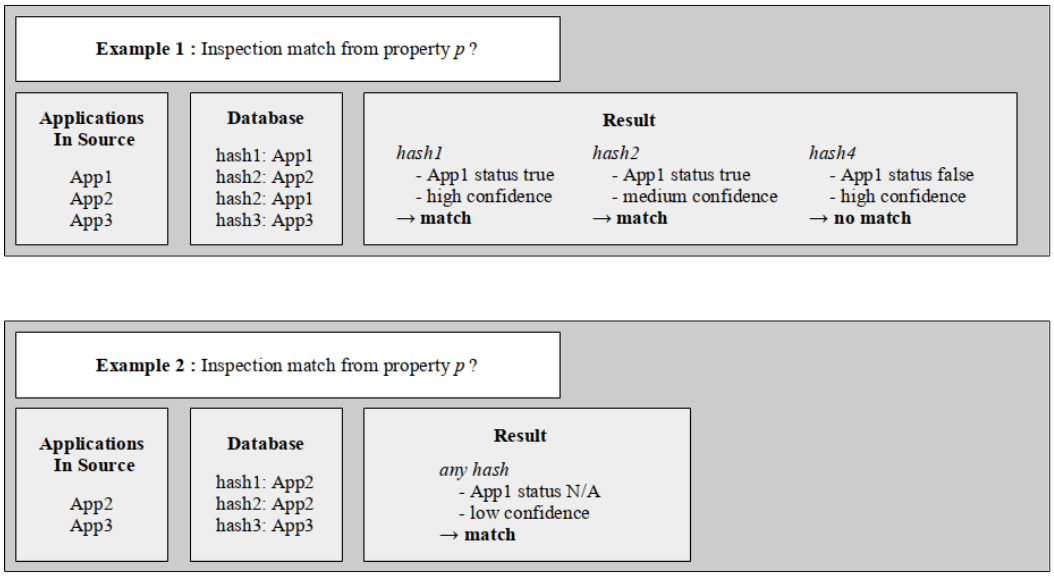

FIGURE 5. *Example demonstrating additional confidence granularity. In this example, the property p has been associated with a vulnerability in endpoint application App1. The decision to make an inspection based match from property p differs depending on whether App1 has been associated with the hash fingerprint calculated from the connection or not.*

c from an endpoint belonging to the Target group N but not to the Source group S, and where the protocol is supported for hash fingerprinting by the network security solution, the hash fingerprint $d \in H$ is calculated for the network connection with function h. The network security solution $\mathcal{W}$ then queries the database for endpoint application information for the calculated hash using function o. Depending on the result of the query, the network security solution can then make a more informed decision when determining whether a deep inspection based match should be produced from the property $p \in P$ or not.

4) Making the deep packet inspection based match using endpoint awareness

When utilising the endpoint application information while making a deep packet inspection based decision, the network security solution will use the result of $g_2(c, p, o(d))$. Below we list the potential results with different values for $o(d)$ and how the network security solution can utilise the information for endpoint aware inspection. The list is also visualized in Fig. 4.

- **No results.** $o(d) = \emptyset$. If no endpoint application in the Source group has previously produced the same hash fingerprint, the network security solution will need to proceed with the inspection decision regarding property p as without endpoint awareness, by calculating $g_1(c, p)$.
- **One result.** $o(d) = \{a\}$. If only one endpoint application in the Source group has produced the same hash fingerprint, the network security solution can assume that the connection currently being inspected has potentially been produced by that endpoint application. It can proceed with the inspection decision regarding property p by calculating $g_2(c, p, a)$.
- **Several results.** $o(d) = \{a_1, ..., a_n\}$. If several endpoint applications in the Source group have produced the same hash fingerprint, the network security solution must assume that the connection currently being inspected has potentially been produced by any of the matched endpoint applications. It can proceed with the inspection decision regarding property p by calculating $g_2(c, p, a_1) \vee ... \vee g_2(c, p, a_n)$.

After endpoint awareness is gained for the inspection process in the network security solution $\mathcal{W}$, it can utilise this information when making inspection decisions. These decisions are related, for example, to categorising the traffic. If the network security solution $\mathcal{W}$ needs to decide whether an inspection match should be produced from the property $p \in P$ of the connection c, it can make the decision in the context of all potentially identified endpoint applications. If the property p of the connection c is especially meaningful for a certain endpoint application $a \in A$, and that endpoint application a has been potentially identified from the connection c, the network security solution $\mathcal{W}$ has more confidence for creating an inspection match. On the other hand, if that endpoint application has not been potentially identified from the connection but something else has, the network security solution has reduced confidence for creating a match.

An additional level of granularity can be achieved when the set A_S of all endpoint applications used by the Source group is stored, and this information is utilised as part of the inspection decision g_2. This provides additional levels into the confidence of the endpoint application match. Fig. 5 visualises this added granularity, as described below.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

Assume that endpoint application *App1* has a vulnerability v, and a property $p \in P$ associated with v is found from the network connection $c \in C$. In the first example, *App1* belongs to A_S and is thus present and actively used in the Source group $S \subset N$. When the network connection c is observed by the network security solution $\mathcal{W}$ from the Target group N, a hash fingerprint d is calculated using h. If the database returns a match for *App1* for $o(d)$, the network security solution $\mathcal{W}$ can have high confidence that the endpoint application is indeed *App1*. It can proceed with the inspection decision regarding property p with this assumption. In this example, the network security solution $\mathcal{W}$ is designed to always produce an inspection match from the property p if the endpoint application *App1* belongs to $o(d)$, even if other endpoint applications have produced the same hash fingerprint as well. If the database does not return a match for *App1*, the network security solution can have similarly high confidence that the endpoint application is not *App1*. In this example, the network security solution $\mathcal{W}$ will not produce an inspection match from the property p if the endpoint application *App1* does not belong to $o(d)$.

In the second example, *App1* does not belong to A_S as it is not used by the Source group $S \subset N$. When the network connection c is observed by the network security solution $\mathcal{W}$ from the Target group N, a hash fingerprint d is again calculated. *App1* will not be returned for this hash fingerprint by the database for $o(d)$. This does, however, not give the network security solution any information on whether the endpoint application might be *App1* or not. Because of this, the network security solution needs to continue with the inspection decision regarding p as without endpoint awareness using function g_1. In this example the network security solution $\mathcal{W}$ is designed to produce an inspection match from p if endpoint awareness is not available. This results in a higher risk of a false positive.

One solution for the problem presented in the second example would be that the administrator additionally defined that *App1* is not used in the protected network. This could be achieved by defining A as the complete set of endpoint applications used in N. The network security solution $\mathcal{W}$ would again be able to use g_2 for deciding whether an inspection match should be created from p, and *App1* will never belong to $o(d)$. This would again give the network security solution $\mathcal{W}$ high confidence that the endpoint application is not *App1*, which in the example would always lead to no inspection match being created from the property p.

IV. CASE STUDY: LOCAL NETWORK SEGMENT WITH ONE EXTERNAL INTERNET CONNECTION

A proof-of-concept case study was performed in a local network segment with one external internet connection. For the scope of this study, seven endpoints in the network were selected as the Target group. These endpoints consisted of five endpoints using Windows 10 OS and two endpoints using Ubuntu Linux OS. The network traffic was inspected using the Forcepoint Next Generation Firewall [35].

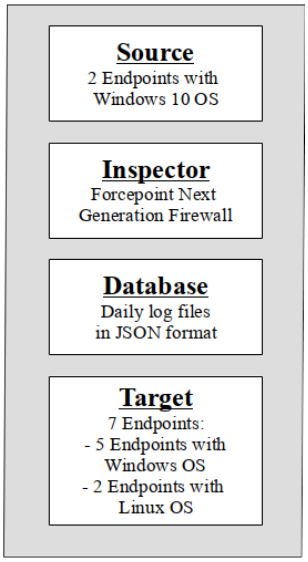

FIGURE 6. *Description of the case study setup.*

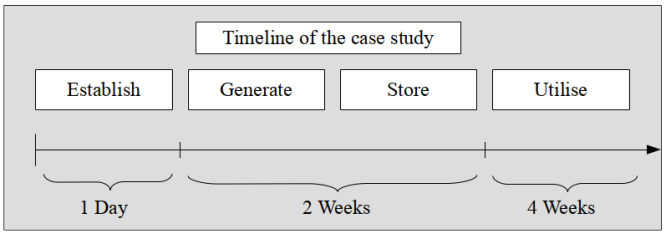

FIGURE 7. *The timeline of the case study.*

The collected hash fingerprints were restricted to JA3 fingerprints. This restriction was done based on the fact that the Forcepoint Next Generation Firewall supports JA3 and JA3S fingerprints, which restricted the hash algorithms to them. As specified in section III-C, JA3 fingerprints are for client applications and JA3S fingerprints are for server applications. As all protected endpoint applications in the selected Target group were client applications, there was no source for JA3S fingerprints in the group, which left JA3 as the selected fingerprint.

The target of the case study was to examine how well the Target group can be covered with information collected from the Source group. The study group is rather small, but our objective was to see if there is potential for scalability. In addition, we were interested to see how well the information collected from one operating system can cover information from another operating system. Because of this, we left all Linux machines outside of the Source group, and present the results also as separated by operating system.

Finally, we conducted a small test to demonstrate how this method could reduce false positives. We selected four files known to trigger vulnerabilities in certain endpoint applications, but not in Firefox web browser. We then accessed these

files using Firefox, and observed the results.

A. SETUP

For the Source group, we selected two Windows endpoints running Windows 10 OS. This left two Ubuntu Linux endpoints and three Windows 10 endpoint outside of the Source group. To verify the results, we collected endpoint metadata also from the endpoints left out of the Source group. We used the Forcepoint Endpoint Context Agent (ECA) component to collect endpoint metadata from the Windows machines, and from the Ubuntu endpoints the endpoint metadata was collected with a custom script based on the BPF Compiler Collection project [36].

As the Inspector we used Forcepoint Next Generation Firewall. The Inspector observed all network traffic from the endpoints and calculated JA3 fingerprints for the relevant connections. For the Windows endpoints the mapping of the endpoint application to the JA3 fingerprint was done by the Inspector using the Forcepoint ECA component. For the Linux endpoints the mapping was done separately by correlating the metadata collected from the endpoints to the log events from the Forcepoint NGFW using the 5-tuple of source IP address and port, destination IP address and port, and the transport protocol. The information was stored in daily JSON files which in the context of this method made up the Database.

The timeline of the case study in presented in Fig. 7. We first had an initial period of two weeks to generate data to the Database from the Source group. Next, we collected data for four weeks from the entire Target group. After this test period was over, we gathered and correlated all collected data and analysed the results.

For the false positive test, we selected two machines from the local network segment. We selected one machine to work as a client and one to work as a server. On the server machine, we created a temporary key and certificate using OpenSSL, and set up a minimalistic HTTPS server using Python. We selected four files we knew to trigger vulnerabilities in certain versions of Internet Explorer, Microsoft Outlook and Android native Web Browser. We then accessed these files from the client using Firefox web browser, which was known not to be vulnerable. The traffic was inspected and decrypted by the Inspector.

B. RESULTS

Table 1 presents the amounts of hashes, endpoint applications and unique hash-application pairs in the Source and Target groups. The Source group data is presented as during the initial two weeks collection period, and as during the entire study period. The Target group data is presented in its entirety, and as restricted only to data collected from endpoint applications that were present in the Source data. In this study, when restricting the Target group data based on data from the Source group, we always consider the Source group data during the entire study period.

From Table 1 we can see that during the entire study period, the Target group contained 80 active endpoint applications, out of which 53 were included in the Source group data. We can also see that the Source group data contained 7 endpoint applications that were not present in the rest of the Target group. The Source group data contained 62 unique JA3 fingerprints whereas the entire Target group data contained 73 unique JA3 fingerprints. After restricting the Target group data only to endpoint applications present in the Source group data, the amount of unique JA3 fingerprints dropped to 56. From Table 2 we can also see that only 3 of these JA3 fingerprints were not observed in the Source group.

We also separated the data collected from the Target group by operating system. The results are presented in Table 3 and Table 4. The difference in the amounts of endpoint applications between Windows and Linux is quite large, as the Linux group contained 15 endpoint applications whereas the Windows group contained 68 endpoint applications during the Utilisation period. In addition it is noteworthy that only 4 out of the 15 Linux endpoint applications were present in the Source group. However, these 4 applications produced most of the hashes in the Linux group: 20 hashes out of 33.

Table 4 shows the hashes produced by the Target group as separated by OS. The amount of hashes from the Windows endpoints diminishes only from 57 to 53 when restricting the endpoint applications to ones present in the Source group. The most noteworthy finding is that all of these 53 hashes were also produced by the Source group. This means that when only considering endpoint applications used in the Source group, the Source group data covered all hashes produced by the Windows endpoints in the Target group. The coverage results for Linux endpoints were not too bad either, as when restricting the results only to endpoint applications present in the Source group only 3 out of 20 hashes were not produced by the Source group.

In Fig. 8 we see the inspection events produced during our false positive test. We can see that the Inspector calculated the JA3 fingerprint from the traffic, which remained same for each of the four connections. We can see that the client machine sent metadata to the Inspector about the endpoint application being Firefox. In addition, we can see that in each case, the traffic was terminated by the Inspector by a situation which, based on its name, is related to a vulnerability in some other endpoint application than Firefox. If our method was utilised for these connections, it would be possible to identify that the recipient endpoint application is not vulnerable for the potential exploit in the file, and if so decided, the file could have been permitted through.

C. REMARKS ON THE CASE STUDY

The decision to leave both Ubuntu machines out of the Source group was deliberate. We wanted to see how well the hashes collected from a Windows environment can apply to a Linux environment. The results showed that the endpoint application coverage is not good, as only 4 out of the 15 endpoint applications on the Linux endpoints were also

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170456, IEEE Access

TABLE 1. Hashes, apps and pairs from Source and Target

	Hashes	Endpoint applications	Hash-App pairs
Source (Collect)	47	43	106
Source (All time)	62	60	157
Target (All apps)	73	80	208
Target (Apps in Source)	56	53	144

The amounts of hashes, endpoint applications and hash-application pairs in the Source group and the Target group. Source group data is presented as during the initial collect period and as during the entire period. Target group data is presented both for all observed endpoint applications and as restricted to the endpoint applications present in the Source group.

Situation	TLS Client Hello JA3 Hash	Action	Executable Original Name
TLS_Client-Hello	3c8ae2c29845cb602d278cb121ab380c	Permit	firefox.exe
File-Text_Microsoft-Scripting-Engine-CVE-2016-0189-Memory-Corruption		Terminate	firefox.exe
TLS_Client-Hello	3c8ae2c29845cb602d278cb121ab380c	Permit	firefox.exe
File-Binary_Microsoft-Outlook-Out-Of-Bounds-Vulnerability-CVE-2018-8587		Terminate	firefox.exe
TLS_Client-Hello	3c8ae2c29845cb602d278cb121ab380c	Permit	firefox.exe
File-Text_Google-Android-Browser-Same-Origin-Policy-Bypass		Terminate	firefox.exe
TLS_Client-Hello	3c8ae2c29845cb602d278cb121ab380c	Permit	firefox.exe
File-Text_Microsoft-Browser-Scripting-Engine-CVE-2016-3382-Type-Confusion		Terminate	firefox.exe

FIGURE 8. *Logs from the Forcepoint Next Generation Firewall during the false positive test.*

TABLE 2. Target hashes

	All apps	Apps in Source
Target hashes	73	56
Target hashes not in Source	20	3

Amounts of JA3 hash fingerprints observed in the Target group. The table presents hashes collected from all endpoint applications in the Target group as well as hashes produced only by endpoint applications that were included in the Source group. It also shows the amount of hashes observed in the Target group that were not observed in the Source group.

present in the Windows endpoints in the Source group. However, for the Linux endpoint applications that were covered by the Source group, the coverage was very good: out of the 20 hashes produced by the Linux endpoint applications that were also present in the Source group, only 3 were not produced by the Windows endpoint applications in the Source group. This is an encouraging result as it indicates that the implementations of an endpoint application for different operating systems are not too different considering their network behaviour. The amount of covered endpoint applications was, however, very small, which means that these results are not conclusive.

The most noteworthy result from this study was that when the Target group data was restricted only to endpoint applications covered by the Source group, all hashes produced by the Windows machines in the Target group were covered by the Source group. This is a very promising result, indicating that as long as the Source group provides great enough coverage of the endpoint applications used in the Target group, this method can provide a proper endpoint awareness for the inspector in the target network.

In this study, 27 out of 80 endpoint applications in the Target group were not present in the Source group. 11 of these endpoint applications were from Linux and 16 were from Windows. The fact that the Source group did not include any Linux endpoints is one reason for this variance. Another reason is that the endpoint applications installed on each endpoint varied greatly based on the preferences of the user of the endpoint. All endpoints in the Target group were mainly used for personal free-time activities, but otherwise were quite heterogeneous regarding installed endpoint applications. The variance in an organisational setting would most likely be smaller, as the tools used for daily tasks are often more homogeneous compared to personal free time preferences.

For the false positive test, we selected files that each were known to exploit a vulnerability in some endpoint application, just not in the web browser we used for accessing them. Thus, it could be argued that the inspection situations that caused the file downloads to terminate were technically not false positives, as the files truly were malicious. One way to approach this is that, no matter which endpoint application is the recipient of the traffic, a malicious file should always be terminated. However, the intention of this test was to demonstrate that our method can be utilised for real life examples. In the case of these particular files, the decision to terminate the connection might stay the same even if the endpoint application was known. But this demonstrates that it is possible for the Inspector to gain better awareness of the inspected content using our method, and thus to make more educated decisions.

TABLE 3. Target by OS

	Hashes	Endpoint Apps	Hash-App pairs
Linux (All apps)	33	15	36
Linux (Apps in Source)	20	4	21
Windows (All Apps)	57	68	163
Windows (Apps in Source)	53	52	140

Amounts of hashes, endpoint applications and hash-application pairs in the Target group, separated by operating system. The data is presented both for all observed endpoint applications and as restricted to the endpoint applications present in the Source group.

TABLE 4. Target hashes by OS

	All Apps	Apps in Source
Linux hashes	33	20
Windows hashes	57	53
Linux hashes not in Source	16	3
Windows hashes not in Source	4	0

Amounts of JA3 hash fingerprints observed in the Target group, separated by operating system. The table presents hashes collected from all observed endpoint applications as well as hashes produced only by endpoint applications that were included in the Source group. It also shows the amount of hashes observed in the Target group that were not observed in the Source group.

V. DISCUSSION

The method presented in this paper utilises hash fingerprinting algorithms that work on application layer data. Because of this, the method is not restricted to local devices, such as NGFWs or IPSs, that are able to see the original network and transport layer content. Instead, it can also be utilised by cloud implementations, such as SaaS and SASE solutions, that may only receive the original application layer data untouched. There are, however, many fingerprinting algorithms that identify endpoint information based on lower layer information as well, that are referenced in section II. The method presented in this paper can easily be applied also to lower layer information.

Without a large scale study it remains unclear how long a time period is needed to gather preliminary information in the Generate and Store steps for the Utilise period to have good enough certainty. In the case study in section IV we selected the period to be two weeks. It is, however, likely that a shorter time period would suffice in an active network. Whether it should be one hour, one day or one week may depend on the size and activity level of the protected network, and requires further study.

It may also be that if the method is used for a long period of time in one environment, it becomes necessary to remove obsolete entries from the database after a certain period of inactivity. This might be necessary to keep the database current. If a hash fingerprint was produced by one endpoint application many years ago and never after that, and is then produced by another endpoint application now, it is safe to assume that any traffic producing this hash fingerprint now would be the new endpoint application instead of the old one. Further study is needed to understand what this period of inactivity would be, and if it needs to be incorporated into the method.

One weakness in the method is in the usage of the passive hash fingerprints. It is trivial for a malicious entity to modify its network patterns to mimic a benign endpoint application. However, we do not consider this weakness a big one. This method provides a means for identifying whether a property in a network connection, such as URL, file or a protocol pattern, is especially meaningful for the identified endpoint application or not. This would often lead to the connection being terminated only when it would be received by certain endpoint application. A malicious endpoint application faking a benign endpoint application would thus at most end up having its traffic terminated due to it being considered harmful for the faked benign endpoint application.

Another weakness in the usage of passive hash fingerprints is in the fundamental existence of such methods that can identify the endpoint application, and potentially even the version, based on network traffic. If the network security solution is able to identify the endpoint application based on the unencrypted protocol properties, so is a malicious third party who is able to eavesdrop the traffic. When a malicious entity is able to observe which endpoint applications, even which versions, are used in the network, they have a much easier task of figuring out a suitable attack.

For this problem to go away, a larger consensus between endpoint application developers would be needed to attack the problem of the fingerprintability of network connections, which we consider not too likely. The best solution would be for the problem to be solved on the protocol specification level. Another solution could be for the network security solution to work as a proxy, initiating a new network connection for each received connection, where the produced hash fingerprint is obscured. This would remove the endpoint application information from any connection going through the network security solution, but letting the network security solution itself continue with this additional context.

It is possible that an endpoint application in the Target group generates a checksum which has not been observed from the Source group, despite the endpoint application in question being in use in the Source group. This may happen if, for example, the endpoint application in the Target group gets updated before any instance of the endpoint application in the Source group. If high confidence is placed on the contents of the database, especially when the proposed technique for higher granularity is used, this can lead to a false assumption that the endpoint application is not the application in question. This could lead to a reduced security

for the specific endpoint application.

We have earlier in the paper indicated that the best security would be achieved if every endpoint in the protected network was able to send metadata to the network security solution, and that this method is aimed at the networks where this is not possible. This method does, however, provide important insights even in the situation where all endpoints provide metadata, as it gives visibility into anomalies in the network patterns for the endpoint applications. When the hash fingerprints for each endpoint application are stored in the database, the network security solution can form an understanding of the common patterns for each endpoint application. Based on this information it can then identify anomalies, such as if one endpoint application suddenly starts producing hash fingerprints that have previously been mapped only to some other endpoint application.

In the context of this work we have proposed that the awareness of the endpoint applications is utilized when making deep packet inspection based decisions. The information could, however, also be used for access based control. This would impose an even higher need for accurate and complete coverage for varying hash fingerprints - otherwise legitimate connections could get dropped if the particular hash fingerprint has not been observed from the Source group. In addition, malicious endpoint applications could get through the network security solution simply by faking a benign hash fingerprint for a white-listed endpoint application. We thus recommend that if used for access-based control, the information is again taken as an added context, and used only to compliment other security layers.

VI. CONCLUSION

In this paper we have presented a method for endpoint aware inspection in a network security solution capable of performing deep packet inspection. This method provides a way to improve the accuracy of deep inspection based decisions. This is achieved by gaining an awareness of the endpoint application initiating the network connection in the network security solution. To our knowledge this is the first method to enhance the inspection process accuracy by leveraging a subset of the protected network to gain endpoint awareness.

On a high level, the method utilizes a subset of the protected endpoints as a source for accurate endpoint application information. This information is correlated to patterns in the network traffic, and the correlated information is then extended to the entire protected network. Using this endpoint awareness, the network security solution is able to perform more conscious deep packet inspection based decisions.

The method utilises application layer data for identifying the endpoint application. The method can be utilised in a local security device, such as a NGFW or an IPS, installed on the perimeter of the protected network. Using application layer data, however, makes the method useful also in cloud security implementations, such as SaaS and SASE solutions. Routing devices may change the network and transport layer data before it reaches a cloud solution, but the application layer data more often remains untouched.

To validate the method we presented a proof-of-concept case study from a small local network segment with one external internet connection. In this case study we saw that 73% of the applicable network connections from the protected network could be mapped back to the endpoint application based on the data from the source group. When we took out the connections that were produced by operating systems that were not present in the source group, the amount of identified network connections went up to 93%. When we further restricted the network connections to those produced by endpoint applications that were present in the source group, the amount of identified network connections was 100%. Finally, we performed a small scale false positive test with real life examples to demonstrate that the method can be used for making a more educated inspection decision. The results of the case study confirm that the method has a solid ground. A larger scale study is, however, needed to verify whether the method remains usable when the sample groups are several magnitudes greater, as they are in the target audience of large organisations.

REFERENCES

[1] Jon Mark Allen. Os and application fingerprinting techniques. Technical report, SANS Technology Institute, USA, 2007.
[2] Gordon Lyon. Nmap Network Scanning. https://nmap.org/, 1997. Accessed 30 December 2020.
[3] Tenable, Inc. Nessus 8.0.x UserGuide. https://docs.tenable.com/nessus/8_0/Content/Resources/PDF/Nessus_8_0.pdf, 2020. Accessed 30 December 2020.
[4] Gordon Lyon. Chapter 7. Service and Application Version Detection. https://nmap.org/book/vscan.html, 1997. Accessed 30 December 2020.
[5] Gordon Lyon. Latest stable Nmap release tarball. https://nmap.org/dist/nmap-7.91.tar.bz2, 2020. Accessed 30 December 2020.
[6] Margaret Rouse. Definition: CMDB (configuration management database). https : / / searchdatacenter . techtarget . com / definition / configuration - management - database, 2020. Accessed 30 December 2020.
[7] Fortinet. FortiOS™ 6.4: Fortinet's Security Operating System. https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiOS.pdf, 2020. Accessed 30 December 2020.
[8] Cisco. Cisco Intrusion Prevention System Sensor CLI Configuration Guide for IPS 6.2. https://www.cisco.com/c/en/us/td/docs/security/ips/6-2/configuration/guide/cli/cliguide/cli_event_action_rules.html, 2013. Accessed 30 December 2020.
[9] Carlo Medas. Cybersecurity fingerprinting techniques for OS recognition. https://www.fing.com/news/cybersecurity-fingerprinting-techniques-for-os-recognition, 2020. Accessed 30 December 2020.
[10] Erik Hjelmvik. Passive OS Fingerprinting. https://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting, 2011. Accessed 30 December 2020.
[11] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. EURASIP Journal on Information Security, 2016(1):1–14, 2016.
[12] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, Ran Dubin, Amit Dvir, and Ofir 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), pages 1–6, Las Vegas, NV, USA, 2017.
[13] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. The security impact of https interception. In Proceedings of NDSS Symposium 2017, pages 1–14, San Diego, CA, USA, 2017.
[14] McAfee Corp. Endpoint Intelligence Agent. https://docs.mcafee.com/download/resource/bundle/endpoint-intelligence-agent-v3-2-1-initNull-

product/raw/resource/enus/prod-endpoint-intelligence-agent-v3-2-1-cat-product.pdf, 2019. Accessed 1 May 2021.
[15] Forcepoint LLC. ECA and how it works. https://help.stonesoft.com/onlinehelp/StoneGate/SMC/6.3.0/GUID-A392A75D-7EBD-462D-A6FD-0E6F85E533B6.html, 2018. Accessed 7 January 2020.
[16] Blake Anderson, Cisco Systems, Inc. Tls fingerprinting in the real world. https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world. Accessed 18th January 2022.
[17] TK Keanini, Cisco Systems, Inc. The future of security analytics. https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2020/pdf/DGTL-BRKSEC-2068.pdf. Accessed 18th January 2022.
[18] Microsoft. FileVersionInfo Class. https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.fileversioninfo?view=net-5.0, 2018. Accessed 3 January 2021.
[19] Microsoft. FileVersionInfo.ProductName Property. https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.fileversioninfo.productname?view=net-5.0, 2018. Accessed 3 January 2021.
[20] Debian. dpkg-query(1) - dpkg - Debian-Manpages. https://manpages.debian.org/buster/dpkg/dpkg-query.1.en.html, 2019. Accessed 3 January 2021.
[21] Apple Inc. pkgutil(1) [osx man page]. https://www.unix.com/man-page/osx/1/pkgutil/, 2011. Accessed 3 January 2021.
[22] Andrei Z. Broder. Some applications of rabin's fingerprinting method. In Renato Capocelli, Alfredo De Santis and Udo Vaccaro (eds.): Sequences II: Methods in Communication, Security and Computer Science., pages 143–152, NY, USA, 1993. Springer, New York.
[23] Ronald Rivest and S Dusse. The MD5 message-digest algorithm, rfc 1321. Internet Engineering Task Force (IETF), 1992.
[24] The Zeek Project. Zeek - An Open Source Network Security Monitoring Tool. https://zeek.org/, 2020. Accessed 4 January 2021.
[25] Adel Karimishiraz. Fatt - Fingerprint all the things! https://github.com/0x4D31/fatt, 2019. Accessed 4 January 2021.
[26] John Althouse, Jeff Atkinson, and Josh Atkins. Open Sourcing JA3. https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41, 2017. Accessed 29 November 2020.
[27] Lee Brotherston. Stealthier attacks and smarter defending with TLS fingerprinting. Conference Presentation, DerbyCon V (2015), Louisville, Kentucky, USA. https://www.slideshare.net/LeeBrotherston/tls-fingerprinting-stealthier-attacking-smarter-defending-derbycon, 2015. Accessed 20 May 2021.
[28] Jessica Ai Truong. Evaluating the detection accuracy of JA3 and JA3S in security monitoring of SSL communication. Master's thesis, Tallinn University of Technology, Estonia, 2019.
[29] Ben Reardon. Open sourcing HASSH. https://engineering.salesforce.com/open-sourcing-hassh-abed3ae5044c, 2018. Accessed 29 November 2020.
[30] Caleb Yu. GQUIC Protocol Analysis and Fingerprinting in Zeek. https://engineering.salesforce.com/gquic-protocol-analysis-and-fingerprinting-in-zeek-a4178855d75f, 2019. Accessed 29 November 2020.
[31] Adel Karimishiraz. RDP fingerprinting. https://medium.com/@0x4d31/rdp-client-fingerprinting-9e7ac219f7f4, 2019. Accessed 29 November 2020.
[32] Fatema Bannatwala. KYD - Know your devices, a method for profiling devices using DHCP. https://github.com/fatemabw/kyd, 2019. Accessed 4 January 2021.
[33] Tom Coffeen. Dhcpv6 fingerprinting and byod. Conference Presentation, NANOG 59, Chandler, Arizona, USA. https://archive.nanog.org/sites/default/files/tues.general.coffeen.fingerprinting.6.pdf, 2013. Accessed 20 May, 2021.
[34] Michael R. Torres. SMBFP SMB Fingerprinting Zeek package. https://github.com/micrictor/smbfp, 2020. Accessed 29 November 2020.
[35] Forcepoint. Forcepoint Next Generation Firewall (NGFW). https://www.forcepoint.com/sites/default/files/resources/brochures/brochure_ngfw_overview_en.pdf, 2020. Accessed 5 April 2021.
[36] IOVisor community. BPF Compiler Collection (BCC). https://github.com/iovisor/bcc, 2015. Accessed 7 February 2021.

JENNY HEINO is a doctoral candidate 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 utilise 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 (S'00 - M'04 - SM'09) received the D.Sc.(Tech.) degree in communication systems from the University of Turku, Finland, in 2004. He is currently an Associate Professor of cyber security engineering with the Department of Computing, University of Turku, where he is also the Vice Head of the Department. He has supervised 5 PhD theses and more than 130 Master's theses to completion. His current research interests are on cyber security in smart environments, secure network and communication technology and security technologies for IoT.