Monitoring IoT Encrypted Traffic with Deep Packet Inspection and Statistical Analysis

Luca Deri
IIT/CNR, ntop
Pisa, Italy
Email: luca.deri@iit.cnr.it, deri@ntop.org

Daniele Sartiano
IIT/CNR, University of Pisa
Pisa, Italy
Email: daniele.sartiano@iit.cnr.it

***Abstract*— The pervasive use of encrypted protocols and new communication paradigms based on mobile and home IoT devices has obsoleted traffic analysis techniques that relied on clear text analysis. This has required new monitoring metrics being able to characterise, identify, and classify traffic not just in terms of network protocols but also behaviour and intended use. This paper reports the lessons learnt while analysing traffic in both home networks and the Internet, and it describes how monitoring metrics used in experiments have been implemented in an open source toolkit for deep packet inspection and traffic analysis. The validation process confirmed that combining the proposed metrics with deep packet inspection, it is possible to effectively characterise and fingerprint encrypted traffic generated by home IoT and non-IoT devices.**

Keywords—Encrypted Traffic Analysis, Internet of Things, Cybersecurity, Deep Packet Inspection, Open Source.

I. INTRODUCTION

Network traffic has changed significantly in terms of network protocols and behaviour. Today most of the network traffic is encrypted. As encryption is now pervasive in Internet traffic, it is becoming important to provide network visibility in this new changed scenario where clear-text protocols are used less frequently even though they are still relatively popular in LAN networks where obsolete operating systems and outdated IoT devices will be used for some more years. This means that we need to complement existing techniques with new measurements metrics able to inspect and characterise encrypted traffic for the purpose of identifying threats and changes in network traffic behaviour. In home networks the widespread use of IoT and healthcare devices that operate using cloud services has created new security issues as users no longer interact directly with the device but only through cloud services. This trend towards cloud-based security is present also on products manufactured by leading firewall vendors that can be accessed solely using a cloud console and no longer connecting to the firewall sitting on the company premises.

Providing network visibility is the base on which security of modern networks works, as it is compulsory to implement mechanisms to enforce network policies that enable zero-trust and modern home networks to operate. This has been the motivation behind this work, being decryption of encrypted traffic not practical for various reasons including, but not limited to, ethical and technical issues that prevent MITM (Man In The Middle) techniques to operate on non-TLS (Transport Layer Security) protocols such as SSH, BitTorrent and Skype. Contrary to previous research [8, 9, 10], goal of this paper is not to define new methods for identifying specific threats but rather to classify network traffic in a generic way without searching specific traffic or malware fingerprints. As specified later in this paper, this approach is able to classify traffic using specific protocol metrics and also detect changes in network behaviour. This fact is effective in particular on IoT and home networks, where the device behaviour should not change unless it is reconfigured or compromised. One of the gol of this research work is to develop a tool able to run on devices of limited power and memory (e.g. NanoPi R1S that features 512 MB RAM), hence we have decided not to use in this work machine/deep learning techniques [1] that are computationally expensive hence unable to run low-end devices. Another objective that has motivated this work, is the definition of new metrics and techniques to be used with encrypted traffic similar to those used with clear text. For instance, in HTTP the User Agent has been used [16] to classify devices and identify malware: how can this be implemented with encrypted traffic? In essence, identify properties in encrypted traffic analysis equivalent to those used for years in clear text traffic so that it is possible to have the same level of visibility without decoding the encrypted traffic payload.

The main contribution of this paper is to show in practice how existing network visibility methods and algorithms have been enhanced to take into account encrypted traffic and to promote the creation of a next generation deep packet inspection (DPI) engine (i.e. a techniques that analyses the complete packet payload) that does more than just identifying network protocols decoding a few packets. The novelty of this work is the combination of existing protocol fingerprint techniques coming from DPI with new traffic behavioural indicators based on data binning techniques that allow traffic not only to be recognised in terms of application protocol, but also to be checked for compliance with the expected behavioral model. Doing this it is possible to improve application protocol detection, and at the same time spot suspicious traffic behaviour in a simple way with respect to what popular IDSs can do in a significantly more complex fashion [6]. This is to create a comprehensive set of algorithms and metrics that can be effectively used to monitor both large and home/IoT networks as well Internet traffic. As described later in this paper, the results of this research have been merged into a popular open source DPI and traffic classification engine named nDPI [8] whose code is publicly available at https://github.com/ntop/nDPI. This is to enable researchers to benefit from this work and reproduce the results we are reporting on this paper.

The rest of the paper is structured as follows. Section 2 analyses encryption protocols and standard traffic fingerprint techniques used to classify encrypted traffic. Section 3 covers the proposed monitoring methodology, metrics and approach. Section 4 discusses the tool implementation and experiments, and finally Section 5 concludes the paper.

ICITST ISBN: 978-1-913572-21-1
WorldCIS ISBN: 978-1-913572-24-2
WCST ISBN: 978-1-913572-25-9
WCICSS ISBN: 978-1-913572-26-6

II. Related Work

This section first analyses fingerprints for encrypted protocols, then describes how IoT device traffic is analysed and enforced in networks by popular IDS (Intrusion Detection System) applications.

A. Traffic Fingerprinting

TLS is the most popular cryptographic protocol used to secure communications on computer networks. TLS communications flow over an encrypted, bidirectional network tunnel that is encrypted using some cryptographic keys based on shared secrets negotiated at the start of the session named TLS handshake. During handshake the two communicating peers agree on algorithms, exchange certificates and cryptographic options before starting encrypted data exchange. The TLS client sends a ClientHello message that contains a list of supported ciphers, compression methods and various parameters including options on elliptic-curve cryptography used by TLS. The server responds with a ServerHello message that contains the chosen TLS protocol version, ciphers and compression methods selected out of the various options offered by the client in the ClientHello message. Then the server sends an optional certificate message containing the public key used by the server. Handshake messages are exchanged in clear, so they can be decoded by dissecting packets, with the exception of the server certificate that in TLS 1.3 is encrypted. Decoding the initial handshake packets allows applications to inspect how data is exchanged and disclose information about both the client and server configuration as well fingerprint and identify client applications. Part of the ClientHello message there is the SNI (Server Name Indication) that contains the server name is accessed by the client. JA3 [4] is a popular client/server fingerprinting method, hence named JA3C and JA3S. Both JA3 fingerprints ignore non-cryptographic information such as the SNI string, or certificate information: their goal is to fingerprint the cryptographic libraries used by the two TLS peers rather than to create a unique client/server fingerprint. This means that if applications A, B, and C use the exact same version of OpenSSL they will have the same JA3 fingerprint even though they can be different in nature. The consequence is that methods based on JA3 fingerprint databases (e.g. https://ja3er.com) are "nice to have" but they cannot be reliably used for instance to discriminate malware from benign applications, or fingerprint a web browser. In addition to JA3, similar techniques exist: HASSH is used for SSH, CYU for Google QUIC, and RFDP for the popular RDP (Remote Desktop Protocol) used to remotely connect to Windows hosts.

B. Encrypted Traffic Analysis in IDSs

As stated earlier in this section, these protocol fingerprints are not designed to uniquely identify an application using it, but their intended use is to have a quick way to calculate a fingerprint that when combined with additional metrics it can be used to identify an application with high confidence. Fingerprints are a way to identity communications originated by the same (set of) application(s) by inspecting the first initial packets and they are often used by popular IDSs such as Suricata (https://suricata-ids.org) and Zeek (https://zeek.org) that use signatures to identify malware. Mercury [7] does not use standard signatures such as JA3 but a custom fingerprint to recognise applications. Joy [2, 3], Mercury predecessor, instead used (Sequence of Packet Length and Arrival Time) and bytes entropy of the first few packets past the 3WH to create malware signatures.

In general, tools able to analyse encrypted traffic are designed to detect specific patterns and match signatures, as they are mostly focusing on security. This means that such tools are unable to analyse connection traffic past the few initial connection packets, and implement visibility looking at the big picture instead to analyse a single-flow. In IoT networks for instance, traffic patterns are rather static thus host misbehaviour can be detected by comparing current with past traffic values and not just looking at individual flows. The following section explains how we have extended visibility to encrypted traffic in order to monitor IoT traffic successfully.

III. Monitoring Encrypted and IoT Traffic

Security monitoring applications can use DPI information in order to:

- Fingerprint network traffic in order to detect if both the protocol (e.g. the certificate) has changed its behaviour.
- Prevent specific traffic flows (e.g. unsafe TLS communications) to happen on our network.
- Identify malware in network communications for instance comparing fingerprints with a database of known malware fingerprints, or by other means.

This in essence requires monitoring applications being able to monitor traffic overtime and spot changes in behaviour that might indicate changes in the remote peers configuration or a malware infection. During this research work we have decided to focus on home IoT and smart devices that nowadays are present in many networks. Most devices such as Amazon Echo and Google Home do not interact directly with the local network but only through the cloud. This means that:

- IoT devices installed in the home network are permanently connected to the cloud services they use.
- When two local devices need to communicate they do not exchange data directly. For instance when the home assistant turns off a light, it sends a message to the cloud asking to turn off the light, and the home lightbulb receives a command from the cloud to turn the light off.

Cloud communications are encrypted, and monitoring tools need to inspect IoT traffic in order to make sure that the devices behave normally. As IoT devices have static communication patterns, we claim this goal can be achieved in two steps:

- Monitor both the pool of host names communicating with the IoT device, and the application protocols used to talk with the cloud: they should not change overtime.
- Identify some encrypted traffic metrics useful to verify that data exchanged by the IoT device with the cloud does not change in nature and behaviour.

The strategy we used in this work to provide visibility and introspection to encrypted network traffic is manyfold:

- Uses DPI techniques to characterise traffic and extract relevant metadata that can be used to further classify the traffic.

- Compare traffic fingerprints to both databases of malicious fingerprints in order to speculate about the nature of the communication and detect when host fingerprints change.
- Use traffic metrics to understand whether known traffic is still matching the previously observed behaviour, and if DPI detected application protocols are still matching the model for that protocol.

This approach can be applied to both plain text and encrypted traffic. The main difference is that with plain text traffic it is possible to dissect the payload to interpret the content as IDSs do, whereas with encrypted traffic this is not possible and thus it is compulsory to use alternative techniques for achieving the same goal. This research work has combined the use of fingerprints as traffic indicators (i.e. not for blocking/alerting traffic in case of a match with a fingerprint blacklist) with behavioral traffic analysis used to spot changes with respect to past or expected behaviour. They are not necessarily an indication of compromise or errors, but rather an indicator worth to be analysed by network specialists. This is implemented by the concept of *score* for each entity such as flows and hosts: a non-negative number indicating that such entity has been affected by an unexpected behaviour. A zero flow score means that no issue has been reported, whereas a positive value indicates the relevance of the issue detected. The flow score is then used to increase the host peers score, that will then increase the AS score such hosts belong to, and so on This way we can easily identify and cluster unexpected behaviour not just at flow level, but also at entity level, easing for instance the work of network analysts that have to interpret data. For instance a network/port scan at flow level can look like an anomalous individual flow, but when correlated with the flow score to the host/network, this fact becomes evident without having to implement costly data structures that keep track of ports or peers being involved in the scan. It is worth to remark that the concept of score does not require perfect metrics that might be computationally/memory expensive, but reliable indicators are enough feed it. In this work we have used data binning techniques to group traffic dimensions (e.g. packet length and inter-arrival-time) into specific bins. Using Euclidean-distance comparison it has been possible to cluster similar behaviour as detect when a known behaviour deviates from what the system considers as normal.

This rest of section describes the methodology and metrics used to provide visibility, and that have been implemented in this work that is based on practical experience coding various monitoring applications, and extensive validation tests described in the following section.

C. TLS-Specific Protocol Fingerprints

As described in the previous section, JA3 cannot be used as a unique fingerprint for identifying a single application but rather a family of application sharing the same TLS encryption configuration. This said:

- In this work we have used JA3 to spot changes in applications as the list of known JA3C fingerprints must be static. Any new JA3C signature means that there is a new/unknown application running or that an existing application has been modified, or perhaps compromised. As on non-IoT devices this can happen when an application is installed or updated, on IoT devices any change is an indication of compromise unless the device firmware changed.
- As the TLS client specifies what encryption options are available, unless the server configuration has changed, it must always use the same JA3S for a given JA3C, so the tuple <JA3C, JA3S> for the same client and server must be static. If JA3C does not change but JA3S does, then the server software configuration has been modified [12].
- In essence JA3C is the new HTTP User-Agent for TLS, as it can be used to fingerprint HTTPS client applications same as the User-Agent for plain text HTTP, as well to

The previous statement triggers another question: how can we differentiate a generic TLS connection from HTTPS? This is a very important question to answer as TLS can be used for non-web usage such as for implementing VPNs for instance, or applications that are not web browsers such as malware, and that connect to web servers for compromising them. In the ClientHello packet there is a TLS extension (not effectively used by JA3) named ALPN (Application-Layer Protocol Negotiation) [24] that it is used by the client to tell the server the list of application protocols supported such as HTTP/1.1 and HTTP/2.0. Non-web applications such as a VPN will not declare any HTTP protocol in the ALPN. Another TLS extension named supported_versions that specifies the list of supported TLS versions by the client, can be combined with ALPN to fingerprint the web client application [9] and thus further characterise the nature of a specific TLS connection. A further indicator that can be used to fingerprint communications in particular for IoT devices, are the TLS certificates exchanged by the devices available up to TLS 1.2. Same as the tuple <JA3C,JA3S> discussed earlier, certificate fingerprints can be used as change indicators not in terms of encryption options but rather of client/server configuration.

D. Catching Unexpected Traffic Behaviour

As protocol fingerprints use only the initial flow bytes, they are lightweight and predictable in computational costs. Their main limitation is that they have not been designed to analyse traffic behaviour, and thus they need to be complemented with additional metrics. When classifying network behaviour there are in essence two main strategies [11]:

- Classify good (normal operations) and bad behaviour (e.g. malware) and match the current behaviour against the model. This approach has a few limitations such as being able to recognise only what the system has been trained for, and also requiring traffic annotation that is not something network specialists usually like to do.
- Classify past traffic with a comprehensive list of metrics, and check if the current traffic matches the traffic model that it has been built for a given device. The limitation of this approach, is that traffic is classified as good if it's a "déjà vu", and bad if there is a new traffic pattern in the network that needs to be checked for maliciousness (i.e. an expected behaviour is not malicious per se).

This research work uses the second classification strategy as it fits well with IoT devices where their behaviour is mostly static, this contrary to a laptop where traffic patterns and visited sites are less predictable. As in an encrypted stream it is not possible to inspect the content, the idea is to map key connection properties in order to create a traffic model for a

device traffic. This information could be used to complement MUD [10] profiles, that describe the intended service a device can use/provide, specified in terms of IP addresses and ports. Such model is not general to a device but is based on the tuple <IP source, IP destination, L3 protocol, L7 protocol, SNI or host name> because:

- The SNI and host name further characterise the protocol that might behave differently according to the service the client is connecting to. For instance the traffic model of an Android device talking with two different server over TLS is not compulsory to be identical.
- As with clear text traffic, the traffic model for encrypted traffic differs based on the service being requested. This means that in HTTP for instance, two requests will also behave differently as the type of data can be different. For this reason, the model should take this fact into account by creating a single model for the above tuple.

The model is created only on the initial connection packets and leveraging on DPI for continuously inspecting traffic over time such as IAT (Inter Arrival Time), packet length, bytes distribution statistics, as well goodput and upload/download metrics. They can be used to spot changes in network behaviour and, for detecting the nature of a connection (i.e. interactive session, file upload/download, or protocol tunnel). The DPI component is used to detect the application protocol and so to label the traffic: for instance TLS.Instagram is used when observing TLS traffic whose SNI ends with cdninstagram.com. For each connection, the following metrics are computed for both client-to-server and server-to-client on the first 256 packets to reduce the computational cost of periodically recomputing it until the end of the connection:

- Packet payload, past the 3WH for TCP, are divided into 6 bins of size <= 64 bytes, 65-128, 129-256, 257-512, 513-1024, 1025+.
- Compute packet IAT and also divide it in 6 bins <= 1 ms, 1-5, 6-10, 11-50, 51-100, 100+.
- Payload bytes entropy: create a vector of 256 integers, and for each byte of the payload increment the corresponding element. The entropy is then calculated on this vector. A high value means that the bytes are more spread (high variance) with respect to low values where data is more predictable.

After a few experiments, we have decided to use a non-uniform bin size distribution that focuses on the bottom size (i.e. short packets and those with small IAT) as they map better traffic properties with respect to more heaven distribution. Bins are exported after normalisation, i.e. the bin value is reported as percentage with respect to the total. This allowed us to keep the detail of the time/packet length simple, while accounting for differences across flows. It is possible to represents the bins as a 64 bit value where the power 6 bytes of the number is the value for the i-th bin, and the upper two bytes are set to 0. Note that the 64 bit value is a compact way to represent the bin value, but two different flows cannot be compared with a simple 64 bit value difference but with other means such as the Euclidean distance of each value byte. In addition to the above metrics, for each tuple <host, L7 protocol> there are two additional bins defined:

- Connection duration divided in 8 bins, <= 1 sec, 2-3, 4-5, 6-10, 11-30, 31-60, 61-300, 300+.
- New connection creation frequency also divided in 8 bins with the same bin distribution.

These last two metrics can be used to detect changes in behaviour. For instance a host that suddenly changes its usual connection duration/creation rate to many short-living flows is an indication of a possible network/port scan. The use of bins is basically a compact way to classify and compare traffic properties without taking into account the sequence of events. A simple way to keep track of the order of values is to use a Markov chain as some behavioral IDS do [13]. In our case the matrix size will be a 6x6 grid where each cell contains the number of transitions with respect to consecutive connection packets. While a Markov chain approach is more accurate than binning to report about the flow behaviour, this work relies on simple bins as they are efficient to compute, while capturing enough information to model the flow behaviour. Instead Markov chains are useful whenever it is necessary to detect an exact behaviour, (but this will move our work towards signature-based detection that is not the path we want to take). This has been also the motivation for selecting a few bin classes with respected to having many more classes: when we need to decide whether an observed behaviour matches the expected model, a few bin classes are enough, whereas for exactly fingerprinting a given behaviour many more classes and additional methods are necessary. In summary we have preferred a binning approach as in this work we do not want to create an exact flow fingerprint useful to spot a specific malware application, but rather model traffic to create a flow score that describes how the observed behaviour is far from the expected model. The following section describes how the proposed methods have been validated with real traffic, and how they have been evaluated with both IoT/non-IoT traffic.

IV. Validation

This work has been developed and validated using various methods:

- A dataset provided by NIST that contains network traffic of 16 different types of popular home IoT devices This dataset has been complemented with additional IoT traces [15] named Sentinel IoT.
- Aposemat IoT-23 dataset, a labeled dataset with malicious and benign IoT network traffic provided by the Stratosphere IPS project available at https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/. Unfortunately also this dataset has little TLS traffic.
- A dataset captured in June 2018 on a “smart home” with several home IoT devices such as smart speakers, home assistant, and smart kitchen equipment. This dataset is interesting as it allowed us to compare current IoT traffic with the one captured two years ago. This is very important to validate this idea against devices such as home assistant that were already available years ago but with a very different hardware and software setup.

The different nature of the above scenarios is important as it allowed results to be evaluated in different scenarios, with both IoT and non-IoT traffic and benign and malicious traffic. In total the traffic traces stored in pcap format exceeded 100 GB. Most of the IoT datasets containing malicious traffic as

ICITST ISBN: 978-1-913572-21-1
WorldCIS ISBN: 978-1-913572-24-2
WCST ISBN: 978-1-913572-25-9
WCICSS ISBN: 978-1-913572-26-6

those used in this work and in other papers [5] contain non-TLS attacks such as scans or spoofing, easy to spot with the new connection frequency and connection duration bins already discussed in this section. Furthermore, the DPI engine extracts metadata that can be used for detecting outdated software versions that are good indicators of potential compromise and thus they can contribute to the host score. For instance the string "SSH-2.0-libssh-0.5.2" identifies a library more than 6 years old and with many known vulnerabilities.

E. Protocol Fingerprint Evaluation

TLS traffic is about 90% based on TLS 1.2 for Internet traffic. Looking at IoT devices the percentage decreases to about 50% with half of the traffic TLS 1.0 in Sentinel that has been captured in 2018, whereas on the more recent NIST dataset TLS 1.2 is about 90% as in live ISP traffic. TLS 1.3 slowly but steady increasing in terms of adoption. Looking at the ALPN flags in live ISP traffic 60% of the client advertise only HTTP 1.1 and 40% also support HTTP2, whereas going back to 2018 in the Sentinel or Stratosphere datasets the HTTP2 protocol is not advertised at all even also due to the limited support of ALPN in TLS traffic. The TLS extension advertising the supported TLS version is less popular than ALPN, and it can be found only in recent 2019 live traffic. The following table show some statistics about the above TLS extensions.

TABLE I. ADVERTISED ALPN AND SUPPORTED TLS VERSIONS

TLS Client App	ALPN	Supported TLS Versions
git	http/1.1	None
Firefox	h2, http/1.1	TLS 1.0, 1.1, 1.2, 1.3
Chrome	h2, http/1.1	TLS 1.0, 1.1, 1.2, 1.3
Safari	h2,h2-14,h2-15,h2-16,spdy/3,spdy/3.1	None
OpenVPN	None	TLS 1.0, 1.1, 1.2, 1.3
AnyConnect	None	None

As expected, non-web-based applications such as the AnyConnect and OpenVPN client do not advertise any ALPN, whereas all the other applications do with the exception of wget whose source has not been refreshed in a while. This confirms that when ALPN is specified (as this is its purpose being it designed to advertise the protocol that will be used over TLS), the client is a web-based application whereas when ALPN is not present, nothing can be said about the nature of the application that can either be an outdated client as wget or a non-web applications (e.g. a VPN client). This is an interesting property to disclose the nature of TLS communications (i.e. Tor vs. a web browser) that can also be used to improve JA3 fingerprinting reliability.

TABLE II. JA3 FINGERPRINT DISTRIBUTION PER APPLICATION

Application	Number of Different JA3 Fingerprints
Dropbox	3
Telegram	1
chromium-browser	5
thunderbird	2

Talking about JA3 we have performed some experiments to better understand how JA3C fingerprints are used using a system probe that enabled us to track JA3C usage according to the application using it. As shown in the previous table, there are applications having only one fingerprint and others with more than one. Multiple fingerprints might indicate that there are different entities issuing requests. In the case of a web browser, for example, add-on and third-party extensions might generate this behaviour. This means that not only multiple applications can share the same fingerprint, but also that one application can have multiple fingerprints. The consequence is that while JA3 can be used as indicator of change when the JA3C is modified, the experiments confirm that it cannot be used as a reliable fingerprint being it affected by false positives.

F. Traffic Behaviour Evaluation

When the JA3 fingerprints do not change, we also need to check if the flow behaviour is unchanged with respect to the past. Instead of interpreting the protocol messages, complicated activity for proprietary protocols such as WhatsApp, we have used the entropy value computed on the raw packet payload. We have analysed several hundred of flows, and explored whether specific protocols have a typical entropy value. The is in order to understand if entropy could reveal the nature of the information being exchanged, and if each protocol has a typical entropy value.

TABLE III. PAYLOAD ENTROPY DISTRIBUTION

Byte Entropy	DNS	AmazonVideo	NetFlow	Skype VoiceCall
Average	4.285	7.789	4.079	5.963
Std Dev	0.272	0.231	0.533	0.055

The results reported in the previous table are interesting as each protocol has a typical value whose variance is limited in range. This makes it possible to combine DPI application protocol discovery with the entropy value to further enforce detection and spot outliers and thus potential anomalies. Note that AmazonVideo, tested with various different movies, is delivered over TLS and that entropy values around 7.5 are typical of TLS and not of this specific video streaming application. Entropy has been an effective metric for detecting attacks such as OpenSSL hearthbleed. Under attack the victim host reported for TLS a <client, server> entropy of <7.9, 0.0> compared to <7.9, 7.8> when not under attack. In another experiment we combined entropy information with additional behaviour indicators including:

- DPI application protocol (e.g. TLS.Amazon).
- TLS SNI or host name (e.g. android.clients.google.com).
- Client-to-server and server-to-client payload bin and entropy values. These values are computed on the first 256 packets of a flow. Flows with less than 10 packets are not considered. The bin values have been normalised in order to make them comparable with other flows regardless of the number of packets.

The following table contains the result of this experiment limited to a three flows out of several thousand flows: this just as a short example to clarify the concept.

TABLE IV. TLS.OPENVPN BIN AND ENTROPY DISTRIBUTION

PacketLen Bin Distribution %	Packet TimeDiff Bin Distribution %	Entropy Cli-to-Srv	Entropy Srv-to-Cli
50,9,0,9,18,14	41,0,5,32,9,14	7.402	7.312

45,9,0,14,18,14	41,0,5,32,9,14	7.399	7.294
50,9,0,9,18,14	41,0,5,32,9,14	7.388	7.304

The first column is the packet length bin normalised to 256 (decimals are not depicted as values have been rounded) and the second the normalised packet time difference bin. The last two columns represent the entropy mean and stddev.

Based on the Euclidean distance, we have implemented functions for computing the bin centroid (i.e. the arithmetic means of the bins) and the maximum distance between the centroid and the bins, i.e. <centroid, max distance, otherTLS> . Note that for TLS traffic the otherTLS fields contains additional metrics such as ALPN, JA3C/JA3S, certificate fingerprint that will be empty for non TLS communications. This is the expected fingerprint, for this communication: we expect that future communications will honour this fingerprint and discrepancies will be considered as anomalies. As the use of bins is very lightweight with respect to a machine learning model, it is possible to create a fingerprint for each triplet <client IP, server IP+SNI+Certificate, destination port>. The use of SNI and of the certificate fingerprint is very important as the destination IP can serve multiple SNIs whose behaviour can be very different. The table below shows a GoogleHome device that contacts a remote google service whose SNI is clients.google.com served by host 172.217.7.206 whose traffic was part of the NIST dataset containing over 800 flows generated by this device.

TABLE V. GOOGLE HOME CONTACTING SNI CLIENTS.GOOGLE.COM.

TLS Certificate Fingerprint	ALPN	PacketLen Bin Centroid Distribution %
None	h2;h2-16;h2-15;h2-14;spdy/3.1;spdy/3;http/1.1	54,17,10,5,3
DC:30:BA:11:56:E5:65:7F:CE:40:33:FF:14:2E:6E:D2:C2:33:4E:E4	h2;h2-16;h2-15;h2-14;spdy/3.1;spdy/3;http/1.1	0,0,15,43,30

The centroid has been computed using the Euclidean distance of the individual bin values as computed by nDPI. As you can see, the centroid is very different as the TLS certificate fingerprint changes; this even though the server IP, SNI and destination port and JA3C are the same. This means that with our approach we can fingerprint traffic per triplet and detect when observed traffic does not match the fingerprint as its max distance exceeds the one set in the model. A disadvantage of this approach is that it cannot generalised for instance to all TLS traffic going towards all Google SNIs as each service has its own fingerprint. This is not necessarily a limitation of this work as a single comprehensive model would use many more resources, thus jeopardising the advantage of having resource effective and fine grained models.

While the NIST and our home-dataset do not contain malicious activities, they have been used to tune our techniques and make sure that our tool does not report false positives when analysing traffic behaviour. Instead we have used IoT-23 to verify that the techniques we have implemented in the DPI library are effective to detect attacks typical of IoT devices often based on unencrypted protocols. In both cases the developed tool has not reported any false positive. All attacks families contained in the IoT-23 dataset have been successfully detected by exploiting the host score described in section III, based on the metadata information generated by the DPI library. As the dataset contains evident attacks, any reasonable score threshold can be used to detect such attacks, whereas in real life some threshold tuning is necessary as we do not expect attacks to be always so brute and thus simple to detect.

V. CONCLUSION

This paper has demonstrated that it is possible to effectively characterise and fingerprint encrypted network traffic by leveraging on existing methods complemented with novel techniques described in this paper without using computationally expensive machine learning-based techniques that are unfeasible to use on IoT devices. The ability to fingerprint protocols also in terms of behaviour, enables traffic characterisation and detection of changes in traffic behaviour. The result of this research work has been successfully validated on live Internet traffic as well on various traffic datasets, and integrated an open source DPI engine used in many open source applications for network traffic monitoring and security.

REFERENCES

[1] S. Rezaei, L. Xin, "Deep learning for encrypted traffic classification: An overview." IEEE communications magazine 57.5 (2019): 76-81.

[2] B. Anderson, D. McGrew, "Identifying encrypted malware traffic with contextual flow data", Proceedings of the 2016 ACM workshop on artificial intelligence and security, 2016.

[3] D. McGrew and B. Anderson, "Enhanced Telemetry for Encrypted Threat Analytics", Proceedings of ICNP NetworkML Workshop, IEEE, 2016.

[4] J. Althouse, "TLS Fingerprinting with JA3 and JA3S", https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967, Salesforce, 2019. (Access Date Nov. 2020).

[5] M. Grill, M. Rehák. "Malware detection using http user-agent discrepancy identification", 2014 IEEE International Workshop on Information Forensics and Security (WIFS). IEEE, 2014.

[6] S. Patel, A. Sonker. "Internet protocol identification number based ideal stealth port scan detection using snort." Proceedings of CICN Conference, IEEE, 2016.

[7] D. McGrew, B. Enright, B. Anderson, S. Acharya, and A. Weller, "Mercury", https://github.com/cisco/mercury, 2019. (Access Date Nov. 2020).

[8] L. Deri, "nDPI: Open-Source High-Speed Deep Packet Inspection", 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2014.

[9] L. Deri, "Effective TLS Fingerprinting Beyond JA3", https://www.ntop.org/ndpi/effective-tls-fingerprinting-beyond-ja3/, February 2020. (Access Date Nov. 2020).

[10] E. Lear, R. Droms, D. Romanascu, "Manufacturer Usage Description Specification," RFC 8520, March 2019.

[11] S. García, A. Zunino, M. Campo, "Survey on network−based botnet detection method", Security and Communication Networks 7.5 (2014): 878-903.

[12] J. Ai Truong, "Evaluating the dissection accuracy of JA3 and JA3S in security monitoring of SSL communication", Master's Thesis, Tallin University of Technology, 2019.

[13] S. Garcia. "Modelling the network behavior of malware to block malicious patterns." The Stratosphere Project, DOI 10 (2015).

[14] I. Sharafaldin, A. H. Lashkari, Ali A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", Proceedings of ICISSP. 2018.

[15] M. Miettinen et all, "IoT sentinel: Automated device-type identification for security enforcement in IoT", proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017.

[16] A. Hamza, et all, "Detecting volumetric attacks on loT devices via SDN-based monitoring of MUD activity." Proceedings of the 2019 ACM Symposium on SDN Research. 2019.