Using DPI and Statistical Analysis in Encrypted Network Traffic Monitoring

Luca Deri[1], Daniele Sartiano[2]
[1]*ntop/IIT-CNR*
[2]*IIT-CNR/University of Pisa*
Pisa, Italy

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 on an open source toolkit for deep packet inspection (DPI) and traffic analysis developed by the authors. The validation process confirmed that combining the proposed metrics with DPI, it is possible to effectively characterise and fingerprint encrypted traffic generated by home IoT and non-IoT devices, paving the way to next generation DPI toolkit development.

1. Introduction

Network traffic has changed significantly in terms of network protocols and behaviour. Today most of the network traffic is encrypted and the reasons are manyfold:

- Changes in company network topologies with the adoption of multi-cloud architectures require communications to be protected as they are no longer limited to trusted LAN network segments traditionally protected by security devices.

- Devices such as mobile phone and portable PCs are used on public network and WiFi hotspots making compulsory to use encrypted communications in order to safely exchange sensitive data while preserving privacy on potentially hostile networks.

- New multi-language cryptographic libraries such as Amazon s2n and Google Tink made encryption commodity for programers with respect to obsolete libraries such as OpenSSL that were large in size, difficult to use, and affected by severe problems such as Heartbleed.

- Availability of free and automated X.509 certificates issued by non-profit certificate authority Let's Encrypt has driven the adoption of HTTPS to new highs.

- Computational overhead is no longer a problem even on low-end devices, thus even home IoT devices such as virtual assistants and smart home devices relying on cloud-based services need to secure their communication with encryption.

As encryption is becoming pervasive with 87% of the whole Internet traffic in 2019, 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. This is in particular because modern enterprises are rethinking their network security moving off castle-and-moat approaches focusing on defending their perimeter to a new zero-trust model where no user is trusted based on the principle of "never trust always verify" [6]. In home networks the widespread use of IoT and healthcare devices that operate using cloud services has created new security issues pushing towards the zero-trust model 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 [1] to operate on non-TLS (Transport Layer Security) protocols such as SSH, BitTorrent and Skype. Contrary to previous research [2, 3, 4], 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.

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 [5] 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.

In summary, 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 DPI engine 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 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 implemented in a popular open source deep-packet inspection and traffic classification engine named nDPI [7] so that other people can benefit from this work.

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.

2. Related Work

This section first analyses TLS and SSH (Secure Shell), the two leading encryption protocols and it describes various traffic analysis and fingerprint methods. Then it describes how IoT device traffic is analysed and enforced in networks.

2.1. SSL/TLS Fingerprinting

TLS (Transport Security Layer) is the most popular cryptographic protocol used to secure communications on computer networks. TLS has replaced its predecessor SSL (Secure Socket Layer) used for years on the Internet and now deprecated, and it has been designed to provide privacy and data integrity between two communicating applications. TLS uses TCP as transport protocol even though there is also a variant called DTLS (Datagram TLS) mostly used for VPNs and in some mobile applications (e.g. the Signal messaging app) that is similar to the QUIC protocol promoted by Google. 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. In this negotiation phase 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.

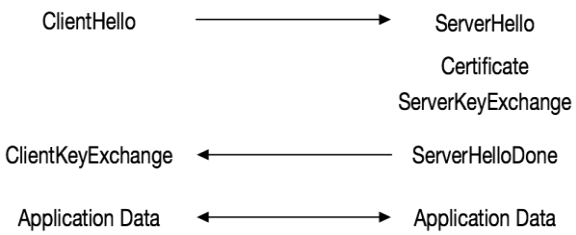

Figure 1. Simplified TLS Handshake (RFC 5246, 2008)

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. For instance, a typical misconception about TLS is that monitoring applications are unable to understand what server name is accessed by the client (e.g. whether the client has accessed www.google.com or maps.google.com) whereas this information named SNI (Server name Indication) is part of the ClientHello message and thus it can be inspected, not to mention that other techniques based on inspection of DNS traffic before TLS exchange might also be used. JA3 is a popular client/server fingerprinting method, hence named JA3C and JA3S, that is based on cryptographic information exchanged in ClientHello and ServerHello packets. Both client and server fingerprints are 32 character hashes of strings obtained concatenating selected fields of the Hello packets. In particular the JA3C string is a concatenation of TLS version, client accepted ciphers, list of TLS extensions, elliptic curves, and elliptic curve formats extracted from the ClientHello packet. Instead the JA3S string is a concatenation of TLS version, accepted cipher, and list of extensions. Both JA3 fingerprints ignore non-cryptographic information such as the SNI string, or certificate information as their goal is basically 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. So in essence even though JA3 is very popular in the security industry being it be used by most IDS (Intrusion Detection Systems) tools, it can be considered as just a feature as it leads to false positives due to multiple matches for the same fingerprint.

2.2. SSH Fingerprinting

SSH is a network protocol used to remotely and securely access a system. Initially designed as a telnet replacement, SSH provides confidentiality and data integrity and thus it is also used to secure other existing protocols by tunnelling traffic such as with X11, a windowing system used by Unix operating systems. The extreme protocol versatility to create encrypted tunnels, has often been used to circumvent security fences and therefore every monitoring system should be able to analyse this traffic in detail. In SSH the two initial messages after the 3WH (three-way handshake) are plain text strings that identify the client and server versions (e.g. SSH-2.0-OpenSSH_7.8). Then peers exchange the SSH_MSG_KEXINIT message for specifying each other the preferences and options for data encryption.

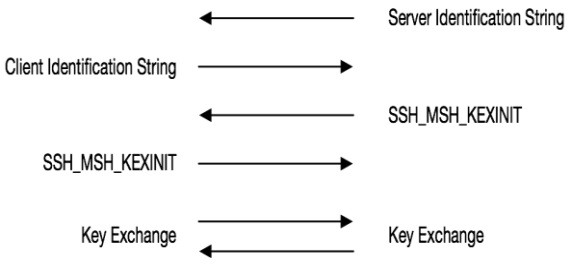

Figure 2. Simplified SSH Handshake (RFC 4254, 2006)

Similar to JA3 for TLS, for SSH there is a fingerprint named HASSH that is compute on the above message to create one hash value for each connection peer. This hash is not used to uniquely identify a client/server but it rather identifies a specific SSH implementation as it only takes in account the list of key exchange methods and encryptions supported by each peer. In fact the unique client/server fingerprint can be obtained by the following two key exchange packets that instead are ignored by HASSH.

2.3. Additional Protocol Fingerprinting

In addition to JA3 and HASSH there are additional fingerprint methods available for additional protocols such as CYU for Google QUIC, or RFDP for the popular RDP (Remote Desktop Protocol) used to remotely connect to Windows hosts.

2.4. Beyond Protocol Fingerprinting

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 can be used to spot malware applications when combined with additional metrics to create a unique fingerprint. This said, 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 IDSs such as Suricata and Zeek that use signatures to identify malware. Mercury [8] does not use standard signatures such as JA3 but a custom fingerprint to recognised applications. Joy, Mercury predecessor, instead used SPLT (Sequence of Packet Length and Arrival Time) and bytes entropy of the first few packets past the 3WH to create malware signatures.

2.5. Statistical Traffic Analysis

Traffic analysis using statistical methods [9, 10] has been used for years. In this work we do not use statistical methods to detect the network application protocol, as DPI [11] proven to be a reliable solution

for this problem. Instead, statistical methods can be used to classify traffic behaviour (i.e. statistical characteristic of the traffic flows) and spot misbehaving communications [12]. In this work we have 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.

Novelty of the work resides in combining data binning techniques with state-of-the-art DPI techniques and some behavioral indicators such as packet payload, packet IAT (Inter Arrival Time), payload bytes entropy, connection duration, and new connection creation frequency. Contrary to similar research in this area [13], our work is not focusing on exactly identifying one specific traffic pattern and detecting in the wild (e.g. identify a communication flow from malware X and label it as malware X) as this requires complex and computationally expensive models that are not really necessary. In the validation section we have analysed several network attack traces from public datasets, and the conclusion is that using this work we can spot misbehaving hosts by leveraging on the score and other methods described in this paper. In general, tools able to analyse encrypted traffic are designed to detect specific patterns and match selected signatures. 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. This is also a novel contribution, namely that this work that implements a lightweight classification model based on the concept of score that does not require specific training or hundred of features to operate, as it detects suspicious behaviour instead of detecting an exact behaviour learnt during the training phase. This feature is very important as it allows unknown cybers-security events to be (partially) detected contrary to other systems that instead are designed only to recognise a set of known behaviours. The following section explains how we have extended visibility to encrypted traffic and monitored IoT traffic successfully. his section first analyses TLS and SSH (Secure Shell), the two leading encryption protocols and it describes various traffic analysis and fingerprint methods. Then it describes how IoT device traffic is analysed.

3. Monitoring Encrypted Traffic

Network administrators need to enforce the specifie policies that include, but it is not limited to:

- Limit the bandwidth of specific protocols (e.g. BitTorrent) and prioritise others (e.g. videoconference).

- Block malicious communications that might flow over encrypted connections. Modern malware such as Danabot use TLS to spread and thus it is compulsory to create mechanisms for recognising bad behaviour in encrypted communications.

As the goal of a deep packet inspection library such as nDPI is to identify application protocols and extract relevant metadata, monitoring applications can use this information to accomplish the above tasks by:

- Fingerprinting network traffic in order to detect if both the protocol (e.g. the certificate) has changed or its behaviour.

- Preventing specific traffic flows (e.g. unsafe TLS communications) to happen on our network.

- Providing metrics for measuring the nature of specific communications (e.g. HTTPS) while not being able to inspect the content with MITM techniques due to the reasons previously discussed.

- Identify malware in network communications for instance comparing fingerprints with a database of known malware fingerprints, or by other means. For instance, a crypto-locker targets Windows systems by encrypting data stored on local or network attached disks. It can be detected by monitoring the traffic towards network storage systems searching for anomalous patters such as creation/deletion of many files in a short amount of time by a single host.

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 decided to take into account the widespread use of IoT and smart devices that nowadays are present in many networks. Most devices such as those based on 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 devices need to communicate (e.g. somebody asks to the home assistant to turn off the light in the bedroom), they do not exchange data directly but the assistant send to the cloud a

message asking to turn off the lights, and the smart lightbulb receives a notification from the cloud to execute the action.

This is a typical example of modern communication patterns where most of these communications with the cloud are encrypted. In this scenario, monitoring tools need to inspect IoT traffic in order to make sure that the devices behave normally. As IoT devices have static communication patterns, this goal can be achieved in two steps:

- Monitor both the pool of peer addresses communicating with the IoT device, and the application protocols used to exchange data 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.

In non-IoT environments the strategy to provide visibility and introspection to encrypted network traffic is somehow similar:

- Use 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 matching the model for that protocol.

Note that the above items can be applied to both plain text and encrypted traffic: the fact that the trend is towards encrypted traffic does not mean that clear-text traffic disappeared and thus that it should be ignored. For instance, even when a host uses DoH (DNS over HTTPS), there is an initial DNS request to resolve the DoH server address and that could be very useful to traffic analysis analysts. The main difference is that with plain text traffic it is possible to dissect the payload to interpret the content, 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, that are not necessarily an indication of compromise or errors, but that are an indicator worth to be analysed by network specialists. This is implemented by the concept of *score* for each entity (flows, hosts, ASs etc.): 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. For this reason, we have used simple bin-based statistics with respect to more accurate yet costly Markov-based indictors. The same applies to the entropy that has been used to understand if a flow is misbehaving, instead of using it to monitor more precise information such as the nature of the information being exchanged. As explained in the validation, this practice is affected by false positives that instead we want to avoid. The follow-up section describes the methodology and metrics used to provide visibility, and that have been implemented in nDPI. It is worth to remark, that this research work is based on practical experience coding various monitoring applications, and extensive validation tests described in the following section.

3.1 TLS-Specific Protocol Fingerprintings

As described in the previous section, JA3 can be used to identify the library used by an application when it connects with a remote peer using the TLS protocol. As already discussed, JA3 fingerprint are not unique across applications, hence and two applications using the same TLS library can have different fingerprints as they have specified different library options. In essence using JA3 as a signature-based indication it is not a good idea, even if it can be used for other purposes such as:

- Regardless of the hash value, JA3C can be used to uniquely identify an application or a web browser plugin. This means that observing the traffic of a host for some time, unless the host software configuration is modified, 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.

- 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.

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) 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. As it will be explained in detail in the following section, 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 and thus further characterise the nature of a specific TLS connection.

Another indicator that can be used to fingerprint communications in particular for IoT devices, are the TLS certificates exchanged by the devices. Up to TLS 1.2, that is by far the most popular TLS version in use today, certificates are exchanged in clear and thus they can by inspected by nDPI; unfortunately, with TLS 1.3 they will be exchanged encrypted hence soon this additional check will not be possible. 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.

3.2 SSH-Specific Protocol Fingerprintings

Similar considerations can be applied to SSH traffic where HASSH replace JA3, and the SSH keys can be used as the TLS certificates.

3.3 Combining Misuse with Anomaly Detection

As already discussed, protocol fingerprints are useful to detect changes in configuration or network protocols: as the use only the initial flow bytes, they are lightweight and predictable in computation costs. The main limitation of fingerprints 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:

- Misuse detection: 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.

- Anomaly detection: 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).

We use 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. Nevertheless, the concept of security risk described later in this section, can be used to spot misuses at flow level, while the score can be used to analyse the behaviour at entity level (e.g., host or network). 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 [14] 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 IP address 172.217.18.98 serving googleads.g.doubleclick.net and pagead2.google syndication.com over TLS is not compulsory to be identical for both services.

- 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 get_static_image.php and get_json_data.php 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 not continuously for the duration of the flow because:

- nDPI already contains algorithms for continuously inspecting traffic over time such as IAT, packet length, and bytes distribution statistics, as well goodput and upload/download metrics. They can be used to spot changes in network behaviour and, for instance, for detecting the nature of a SSH connection (i.e. interactive session, file upload/download, or protocol tunnel) or spotting DoH/DoT (DNS over TLS) on TLS flows.

- Continuous flow metrics computation is not a task for a DPI toolkit: it should analyse only the first few packets of a connection, while providing applications support for computing metrics for the duration of the flow. Applications sitting on top of nDPI are responsible for continuous traffic monitoring by leveraging on the nDPI-provided mechanisms.

- Modern protocols such as HTTP/2.0 and QUIC multiplex multiple services over the same network connection making difficult to create a stable model for the duration of a connection, this unless the two connection peers always exchange the same type of data (e.g. a YouTube video).

The DPI component is used to detect the application protocol and so to label the traffic: for instance, nDPI classifies traffic as TLS. Instagram 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 lengths, past the 3WH for TCP, are grouped in 6 bins of size <= 64 bytes, 65-128, 129-256, 257-512, 513-1024, 1025+. For TLS we have run several experiments to find out whether it was better to use the TLS encryption block length instead of the packet length. Our conclusion is that using the block length is more computationally expensive (as TLS packets needs to be reordered and interpreted in order to extract the block length) than using the packet length, and it does not produce better results in terms of behaviour detection accuracy. For this reason we prefer to use packet length for all protocols being it simpler to compute. This said in case the TLS encryption block length is used, the bin size must be changed as TLS blocks can be as long as 16 Kb whereas network packets are 1514 long unless jumbo frames are used.

- Packet IAT is grouped 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. From our experiments we can report that 4096 bytes are usually enough to reliably compute the flow entropy. hence adding additional bytes do not significantly change the entropy value. As later described in this section, we used the entropy to spot changes in content being exchanged on flows.

The motivation behind choosing a small set of bins is due to the need to have a compact representation that could fit on a 64-byte integer: after normalisation each bin value represents the percentage of the traffic falling in such bin. For instance, the following bin distribution 41,0,5,32,9,14 can be represented as 0x000029000520090E 64-bit integer, leaving the upper two bytes to other uses cases as described below in this section where 8 bins are used. Note that two different flow bins cannot be compared with a simple 64-bit value difference but with other means such as the Euclidean distance of each value byte. In our experiments we have realised that using a larger number of bins for detecting changes in behaviour is not improving the detection: in fact, our goal is not to exactly fingerprint a given communication but rather to understand if the behaviour of such communication is stable over time. Instead, if the goal would be to exactly fingerprint malware X, more sophisticated methods are necessary as they need to take into account other flow characteristics such as the exact sequence of packet length and not just length distribution for which bins are used in this work.

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 uniform distribution where all packets are treated equally. 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 shape, while accounting for differences across flows.

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.

The 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 an order. This means that for instance considering IAT, the following two sequence of values 10,50,10,50,10,50 has the same value of 10,10,10,50,50,50. A simple way to keep track of the order of values is to use a Markov chain as some behavioral IDS do [15]. 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, simple to implement, compact in size, while capturing enough information to model the flow behaviour. Instead, Markov chain should be preferred when modelling more detailed flow properties including detection of bots and malware. In other words Markov chains are useful if you want 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 a d 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.

4. Validation

This work has been developed and validated using various methods:

- Real Internet traffic provided by a regional Italian ISP captured on various networks with both residential and business traffic. This activity lasted for about one year until early 2020. It allowed us to tune nDPI and develop classification techniques described in this paper.
- Over 100 packet traces of network protocols, most of which containing encrypted traffic, used to continuously test nDPI.
- A realistic cyber defines dataset (CSE-CIC-IDS2017/18) that included seven attack scenarios [17]. This dataset has been used to validate metrics for catching cyberattacks such as the heartbleed SSL bug.
- 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 named Sentinel IoT.
- Aposemat IoT-23 dataset, a labeled dataset with malicious and benign IoT network traffic provided by the Stratosphere IPS project. 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, this in addition to live ISP traffic. Most of the IoT datasets containing malicious traffic as those used in this work and in other papers [18] 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, nDPI extracts metadata that can be used for detecting outdated software versions that are good indicators of potential compromise. Such metadata is analysed and used by nDPI to produce a bitmap called *security risk*, where each bit set identifies a potential security risk, that can be used to compute the security score of flow peers. For instance, the string “SSH-2.0-libssh-0.5.2” identifies a library more than 6 years old and with many known vulnerabilities. To date, nDPI security risks include, but are not limited to: HTTP (cross-side scripting, SQL injection, binary application transfer, suspicious user-agent/header, potential remote code

execution attempt), port-based (known protocol on non-standard port), name-based (suspicious DGA, domain generation algorithm, domain), TLS (weak/obsolete cipher, self-signed certificate, TLS not carrying HTTPS), SSH (obsolete protocol version, weak cipher) content-based (malformed packet) risks. As explained later in this work, these risk indicators can be combined with anomaly and behavioral traffic analysis to create a simple yet effective system for analysing encrypted traffic. For instance during our experiments, we have realised that many modern malware such as Dridex, Trickbot and Emotet can be spotted as they trigger many security risks supported by nDPI

4.1. 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 (see Table 1).

Expectedly, 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 application (e.g. a VPN client). This is an interesting property to disclose the nature of TLS communications (i.e. Tor vs. web surfing) that can also be used to improve JA3 fingerprinting reliability. For instance a long-standing TLS connection with no ALPN can be an indication of a non-web related activity such as a VPN or a malware.

Talking about JA3 we have performed some experiments to better understand how JA3C fingerprints are used. In order to do that we have written and eBPF probe for Linux systems based on a home-grown open-source library named libebpfflow. Thanks to the library it has been possible to track JA3C usage according to the application using it.

Table 1. Advertised ALPN and Supported TLS Versions

TLS Client App	ALPN	Supported TLS Versions
git	http/1.1	None
curl	h2, http/1.1	None
wget	None	TLS 1.0, 1.1, 1.2, 1.3
Brave	h2, http/1.1	TLS 1.0, 1.1, 1.2, 1.3, GREASE
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

Table 2. JA3 Fingerprint Distribution per Application

Application	Number of Different JA3 Fingerprints
Dropbox	3
Telegram	1
wget	1
chromium-browser	5
git-remote-http	1
thunderbird	2
cups	1

As shown in the previous table, there are applications having only one fingerprint and others with more than one. Since TLS configuration, and thus JA3 fingerprint, can be manipulated via API calls of the encryption library being used, 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 (see Table 2).

4.2 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 conducted two types of experiments in order to understand if entropy could reveal the nature of the information being exchanged, and if each protocol has a typical entropy value. In the first set of experiments, we have downloaded various files over HTTPS, using the same client and server hosts, in three different format. The following table shows the results with various file types for each format (i.e. PDFs of one page and many pages, with only text or plenty of images etc.).

Table 3. TLS Payload Entropy per File Type

Byte Entropy	PDF	PNG	TEXT
Average	6.426	7.009	7.009
Std Dev	0.007	0.013	0.002

The experiment highlights that while PDF documents can be distinguished from PNG/TEXT files when transferred over TLS, it is not really possible to know whether a PNG or TEXT file is transfer on top of a TLS connection by simply looking at the data entropy, Furthermore, it is worth to remark that when changing the cipher used in the experiment (e.g. transferring the same files over a different type of client and/or server) the entropy values can slightly change making the use of this technique unreliable for this problem with entropy so close in value (see Table 3). For this reason, we believe that using the entropy for detecting the file type is unfeasible, but instead entropy is a good indicator for other use cases as described later in this section for hearthbleed. This is because each protocol, regardless of the cipher being use if encrypted, has a typical entropy that can be used to verify both if the information being transfer really matches the DPI-detected protocol, and speculate about the nature of unknown traffic.

In another set of experiments where we have analysed several hundred of flows and explored whether specific protocols have a typical entropy value.

Table 4. Payload Entropy Distribution

Byte Entropy	DNS	TLS	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. In essence the byte entropy can be used as an indicator for anomalies as well detecting potential DPI invalid classification. For instance, a DNS query with an entropy of 6.5 is definitively suspicious (i.e. it can hide potential data exfiltration), same as a connection with unknown protocol detected and entropy 7.5 can hide a TLS stream. Entropy has been an effective metric for detecting hearthbleed (see Table 4). 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.

Table 5. TLS.OpenVPN Bin and Entropy Distribution Between Two Hosts

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

Table 5 contains the result of this experiment limited to three flows out of several thousand flows: this just as a short example to clarify the concept. 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 byte entropy in each traffic direction.

Using Euclidean distance, nDPI features functions for creating 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> where otherTLS 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 for instance 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. Table 6 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 6. 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.

5. Conclusions

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. The ability to fingerprint protocols also in terms of behaviour, enables better traffic characterisation and detection of changes in traffic behaviour with respect to existing techniques.

The result of this research work has been successfully validated on live Internet traffic as well on various traffic datasets, and integrated in nDPI, an open-source DPI engine developed by the authors, so that the whole Internet community can benefit from it.

6. Future Work

In [19] authors propose a solution named “bag of system calls” for representing and classifying an application behaviour by looking at the sequence of system calls an application performs. A bag is a tuple that contains <syscall id, frequency> and is computed in a sliding time window. When in learning mode, the classifier computes the bag tuples on a “normal system” in a time window and stores them in memory: this iterative process ends as soon as a computed bag is similar to a bag that was previously observed. In running mode, bags are computed on a time window and compared with the list of known bags: if the similarity distance between the bag and the list of bags computed during learning is above a threshold, the system reports this as anomaly. Bags could be represented as bins where each bin slot contains the observed frequency, and where the bag time windows is a flow. A future work item is to evaluate if the classification process using bins, centroids and similarity, could be replaced with bags: instead of having multiple triplet models as already discussed, it should be possible to create a model of bags per SNI or destination IP that could reduce the number of triplets.

7. 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] Y. Bakhdlaghi, “Snort and SSL/TLS Inspection”, SANS Institute, 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] L. Deri, "nDPI: Open-Source High-Speed Deep Packet Inspection", 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2014.

[8] D. McGrew, B. Enright, B. Anderson, S. Acharya, and A. Weller, "Mercury", https://github.com/cisco/mercury, 2019.

[9] J. Zhang, et al., "Robust network traffic classification", IEEE/ACM transactions on networking 23.4 (2014): 1257-1270.

[10] M. Crotti, et al., "A statistical approach to IP-level classification of network traffic", Proceedings of 2006 IEEE International Conference on Communications. Vol. 1. IEEE, 2006.

[11] 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.

[12] L. Shu Yun, and A. Jones, "Network anomaly detection system: The state of art of network behaviour analysis", Proceedings of 2008 International Conference on Convergence and Hybrid Information Technology. IEEE, 2008.

[13] A. Moore, M. Crogan, A. W. Moore, Q. Mary, D. Zuev, D. Zuev, and M. L. Crogan. "Discriminators for use in flow-based classification", Technical Report RR-05-13, Dept. of Computer Science, Queen Mary University of London, Aug. 2005.

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

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

[16] 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.

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

[18] 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.

[19] D. Fuller and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation" Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop. IEEE, 2005, pp. 118–125.

