Extending C2 Traffic Detection Methodologies: From TLS 1.2 to TLS 1.3-enabled Malware

Diogo Barradas
University of Waterloo

Carlos Novo
INESC TEC, HASLab & DCC FCUP

Bernardo Portela
INESC TEC, HASLab & DCC FCUP

Sofia Romeiro
INESC-ID / IST, Universidade de Lisboa

Nuno Santos
INESC-ID / IST, Universidade de Lisboa

ABSTRACT

As the Internet evolves from TLS 1.2 to TLS 1.3, it offers enhanced security against network eavesdropping for online communications. However, this advancement also enables malicious command and control (C2) traffic to more effectively evade malware detectors and intrusion detection systems. Among other capabilities, TLS 1.3 introduces encryption for most handshake messages and conceals the actual TLS record content type, complicating the task for state-of-the-art C2 traffic classifiers that were initially developed for TLS 1.2 traffic. Given the pressing need to accurately detect malicious C2 communications, this paper examines to what extent existing C2 classifiers for TLS 1.2 are less effective when applied to TLS 1.3 traffic, posing a central research question: *is it possible to adapt TLS 1.2 detection methodologies for C2 traffic to work with TLS 1.3 flows?*

We answer this question affirmatively by introducing new methods for inferring certificate size and filtering handshake/protocol-related records in TLS 1.3 flows. These techniques enable the extraction of key features for enhancing traffic detection and can be utilized to pre-process data flows before applying C2 classifiers. We demonstrate that this approach facilitates the use of existing TLS 1.2 C2 classifiers with high efficacy, allowing for the passive classification of encrypted network traffic. In our tests, we inferred certificate sizes with an average error of 1.0%, and achieved detection rates of 100% when classifying traffic based on certificate size, and over 93% when classifying TLS 1.3 traffic behavior after training solely on TLS 1.2 traffic. To our knowledge, these are the first findings to showcase specialized TLS 1.3 C2 traffic classification.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Malware traffic detection, command and control, encrypted traffic

ACM Reference Format:
Diogo Barradas, Carlos Novo, Bernardo Portela, Sofia Romeiro, and Nuno Santos. 2024. Extending C2 Traffic Detection Methodologies: From TLS 1.2 to TLS 1.3-enabled Malware. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024), September 30–October 02, 2024, Padua, Italy.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3678890.3678921

1 INTRODUCTION

Transport Layer Security (TLS) has been the most predominant protocol for malware command and control (C2) communications at least since 2021, with a report from Sophos News stating that TLS was used in 45.7% of malware's outbound communications [26]. According to WatchGuard, TLS was used by 55% of malware in the fourth quarter of 2023 [36], and in the first quarter of 2023, this number reached peaks of 96.4% [35]. Malware's adoption of TLS is linked to their operators' attempts to conceal communications. Since TLS encrypts communication payloads, it generally prevents malware detection and Intrusion Detection Systems (IDS) from conducting deep packet inspection to determine the traffic's true nature. Moreover, C2 traffic making use of TLS can now more easily blend in with an overwhelming portion of legitimate Internet traffic.

Nevertheless, some detection methods still provide the ability to detect C2 traffic within TLS connections. Some detection methods resorted to simple rules based on plaintext information. Examples include IP blocklisting, widely used for spam prevention [20] and malware detection, but also certificate blocklisting [27], which took advantage of server certificates being openly exchanged in TLS up until version 1.2. Other methodologies propose traffic classification based on metadata. One example is certificate analysis, e.g., checking if certificates are self-signed, their validity dates, or the domain present in their Common Name [34]. A different example includes machine-learning classifiers trained on multiple characteristics of TLS connections, such as sequences of packet lengths [42] and inter-arrival times, TLS extensions and parameters [45], contextual information [6], host behavior [25], or certificate and first application data packet lengths [56].

However, the Internet has been moving to TLS 1.3, as well as adopting Encrypted Client Hello (ECH), and DNS over TLS (DoT) or HTTPS (DoH), which erode the effectiveness of existing detection techniques for TLS-encrypted C2 traffic. TLS 1.3 communications now use a revised handshake protocol and TLS record format, changing the typical record sequence and hiding record types and contents early on in the handshake process. Most of the handshake is now encrypted, including server certificates, hindering common C2 traffic detection methods. Nonetheless, despite the introduction of an encrypted handshake, TLS 1.3 metadata like TLS record sizes and timings can still be analyzed. This raises the central research question tackled by this paper: *can TLS 1.2 C2 traffic detection methodologies be adapted to TLS 1.3 flows?*

To account for the upcoming loss of information, modern IDS approaches have considered detection techniques that can potentially be applied to both TLS versions. These rely only on a subset of features (e.g., packet lengths) that will remain available in TLS 1.3 [25]. However, since *certificate information, handshake and alert-related messages, and general application behavior are all sent intertwined in encrypted format*, it is difficult to ensure that a traffic classifier trained on TLS 1.2 traffic can be directly applicable to TLS 1.3 traffic.

This paper builds on the insight that certificate sizes and application behavior patterns can be *reasonably inferred*. Indeed, even if partly encrypted, the TLS handshake protocol follows a fixed sequence of exchanged messages and a well-behaved structure. As such, one should be able to infer certificate sizes from the observed encrypted data records, and isolate application data record sizes from other encrypted messages, both within low margins of error.

We present two novel, complementary techniques for inferring information from TLS 1.3 flows, and two classifiers based on said information: certificate size, and patterns in application behavior.

Certificate size classification relies on a common characteristic of earlier C2 traffic transmitted via TLS 1.2, in which the connections rely on short, self-signed certificates. This is a correlation that can be expected to be directly transposed to TLS 1.3, as malware is commonly used in an off-the-shelf manner by *script kiddie* attackers, a prevalent and ever-increasing threat to modern systems [16]. We propose a method to infer the size of the exchanged certificate in TLS 1.3 connections, by analyzing the sizes of the first server-to-client records in light of the TLS 1.3 Handshake protocol; and C-Cert, a binary C2 traffic classifier based on this single feature.

Alternatively, we demonstrate that C2 traffic has a distinct application behavior, with specific payload lengths in the initially exchanged messages. This is a correlation that would persist even if countermeasures to C-Cert were to be employed – e.g. artificially increasing certificate sizes. Avoiding it would require more considerable technical expertise, as it would entail fundamentally changing malware code to modify its communication patterns. We propose a method to isolate application behavior in TLS 1.3 flows, and C-Behav, a C2 classifier based on application data. Behavior isolation is based on identifying and discarding presumable Handshake and Alert messages from in-between TLS 1.3 application data. The classifier, trained on application data extracted from TLS 1.2, can then classify TLS 1.3 sequences using our extraction method.

We validate the effectiveness of our TLS 1.3 classifiers by devising multiple experiments across different datasets we sourced both from public repositories and from in-house traffic captures collected by ourselves. In these experiments, we trained our classifiers resorting to TLS 1.2 traffic only, and attempted to classify a set of samples comprised of benign and malicious TLS 1.3 traffic. Since real-world traffic is unbalanced towards benign traffic, we train our classifiers under similar settings and highlight that a "good" classifier must necessarily achieve a high true negative rate (TNR) to minimize false alerts and become practical as part of an IDS.

Our results revealed that in two datasets of legitimate TLS 1.3 traffic – one public, one collected in-house – C-Cert was able to achieve a true negative rate of 99.48% and 99.91%, respectively, while achieving true positive rates (TPR) of 100% on our TLS 1.3 C2 traffic. Similarly, a validation of the C-Behav classifier resulted in a TNR of 99.42% and 99.78% in the same datasets, while achieving a TPR of 93.87%. We also evaluate both classifiers' performance when dealing with TLS 1.3 C2 traffic produced by malware that artificially increases certificate sizes to avoid detection. Our results show that while this practice severely impairs the predictions of C-Cert, the predictions of C-Behav remain largely unaffected, still achieving high TNR values (over 93%) on such traffic.

To further assess the feasibility of our classifiers, we extended our evaluation through the analysis of a novel dataset comprising TLS traffic collected within a real university campus network. This dataset contains roughly 10 million unlabeled TLS flows, more than half being TLS 1.3. In this dataset, C-Cert (resp. C-Behav) identified 99.13% (resp. 96.35%) of all TLS 1.3 flows as benign. While we cannot assess the actual TNR of the classifiers because we cannot be sure of which TLS flows are indeed C2 or non-C2, our results suggest that multiple flows predicted as C2 traffic do include several suspicious domains. Interestingly, an IP address associated with C2 activity over TLS 1.3 (as predicted by our classifiers) was at first not perceived as malicious upon lookup on VirusTotal [40], but after a few days this same IP address was (and is currently) established to be associated with malicious activity. This demonstrates the capability of our approaches to identify C2 traffic in the wild.

To summarize, this paper features the following contributions:

- Two novel techniques for TLS 1.3 certificate size inference and application behavior isolation, as well as two new TLS traffic classifiers that use the features extracted by these techniques.
- Two "in-house" datasets of legitimate (>500K flows) and C2 traffic (>790K flows) using two TLS versions.
- A "real-world" unlabeled dataset of TLS 1.2 and TLS 1.3 traffic (>10M flows) collected within a university network.

Our classifiers, datasets and experiments are open-source [47].

2 TECHNICAL OVERVIEW

In this section, we start by introducing TLS 1.3, showing how it differs from previous versions and how these differences may hinder existing detection methods (§2.1). Afterwards, we describe the challenges involved in classifying TLS 1.3 traffic by leveraging an intentionally oversimplified dataset containing both TLS 1.2 and TLS 1.3 traffic (§2.2). Finally, we summarize the benefits of our server certificate and application behavior inference procedures when detecting TLS 1.3 C2 traffic in our simplified dataset (§2.3).

2.1 Background on TLS 1.3

The latest version of the TLS protocol, TLS 1.3, is specified in RFC 8446 [53]. The RFC states some of the major differences between TLS 1.2 and TLS 1.3, including: no support for legacy encryption algorithms, encryption of all handshake messages after the Server Hello, and restructured handshake state machine. TLS 1.3 saw an unprecedentedly fast adoption due to market concentration – a few large providers were responsible for pushing the new TLS version into the market early on and sustaining its growth [32, 33]. Furthermore, a draft was submitted proposing Encrypted Server Name Indication (ESNI) for TLS 1.3, which has the client sending the Server Name Indication (SNI) extension that identifies the domain name that the client is connecting to, encrypted with a server public key obtained with a DNS query.

The aforementioned draft later evolved into ECH [54], in which encryption is not only applied to the server domain but also to other privacy-sensitive extensions and parameters in the Client Hello, like the *Application Layer Protocol Negotiation* (ALPN) list, preventing unintended information leaks. The server domain and other TLS handshake features, such as the number of client extensions or signature algorithms, have been used for malware traffic detection [25] and may be obscured by ECH. For instance, when ECH is used, TLS Handshakes between a client and servers behind the same service provider should all look identical, thus hiding which server is terminating the connection. The ECH standard is still a work-in-progress [49], albeit with some adoption [59]. We will assume its usage, which further hides information from traffic classification services, to consider stronger adversaries and ECH's expected standardization and widespread adoption. With the same reasoning, we also assume the usage of DNS encryption through DoT or DoH, which also hides some helpful contextual information.

Prior to TLS 1.3, authentication certificates can be extracted by searching for TLS records of type *Handshake* (type 22) and handshake type *Certificate* (type 11). However, in TLS 1.3, an observer can only infer certificate sizes from the size and nature of the TLS records observed. Client and Server Hello messages have also become significantly less informative with TLS 1.3, especially if using ECH. While the Client Hello may preserve some information for backwards-compatibility reasons (supported TLS versions, cipher suites and extensions), once the server acknowledges the client's support for TLS 1.3, all sensitive or identifiable information should be sent encrypted after the Server Hello. An observer of a Server Hello record can extract the chosen cipher suite, but not much more, which is insufficient for most C2 traffic detection methodologies.

2.2 Challenges in Classifying TLS 1.3

In contrast to C2 traffic exchanged over TLS 1.2, malicious traffic exchanged over TLS 1.3 benefits from additional protections that make it difficult for IDSs to accurately identify said traffic. The main *technical challenges* tied to this difficulty entail:

- the refined Handshake protocol modifies the sequence of records exchanged between clients and servers;
- no certificate information is available for classification;
- application behavior is no longer clearly isolated from protocol-related messages for outside observers.

To showcase these challenges, we analyze a small illustrative example that describes how the blind application of TLS 1.2 C2 traffic classifiers fails when used in TLS 1.3. For succinctness, we abstract away the details about the datasets and classifiers mentioned in this example but will properly address them in the following sections.

The simplified dataset. We consider genuine TLS traffic captured from websites featuring the Tranco [37] list and malicious TLS traffic generated using Metasploit. Its composition is as follows:

(1) 10,000 genuine TLS 1.2 traffic flows;
(2) 10,000 genuine TLS 1.3 traffic flows;
(3) 75 malicious TLS 1.2 traffic flows;
(4) 75 malicious TLS 1.3 traffic flows;
(5) 50 malicious TLS 1.3 flows with increased certificate size.

The reasoning for (4) and (5) is that, while one expects the majority of C2 TLS 1.3 traffic to originate from off-the-shelf malware (4), we explicitly consider a more challenging scenario where an attacker manipulates server certificates to avoid detection (5).

Metrics of interest. Intrusion detection is a scenario where the dataset is significantly unbalanced towards benign traffic, which often leads to a base rate fallacy [10] in result analysis. As such, we are interested in ensuring a high true negative rate, such that classifiers are not prone to flood the IDS with false alerts. Thus, we assess the quality of a classifier by measuring the TNR on benign traffic – to check the rate of false classifications – and the true positive rate on malicious traffic – to check the efficacy of the classifier in actually identifying C2 flows.

A straw man C2 traffic classifier for TLS 1.3. Towards showcasing the difficulties in transposing the detection capabilities of TLS 1.2 traffic classifiers to TLS 1.3 traffic, we start by identifying sets of features which are currently useful for detecting malicious TLS 1.2 traffic but which could also potentially be applicable to TLS 1.3 traffic. Namely, we extract the sequence of TLS record sizes, directions and content types of TLS 1.2 flows, as has been done in prior work (more details on §3.2). Then, we train a classifier – C-Basic – on these sequences (more details on §4.1).

As expected, C-Basic performs competently when applied to the TLS 1.2 traffic comprising our example, achieving both a TPR and TNR of 100.0%. Feature analysis reveals that predictions over TLS 1.2 traffic are heavily influenced by the size of the TLS record that carries the server certificate, suggesting that server certificate sizes play a major role in C2 traffic detection. Additionally, we observed distinct patterns in the sizes of application data records of TLS 1.2 C2 traffic when compared to benign traffic, suggesting that one might infer the type of traffic from the sizes of such records.

In turn, the application of C-Basic on TLS 1.3 traffic results in a drastic drop of performance, effectively identifying all traffic as benign. These results suggest that TLS 1.3's obfuscation of certificate sizes and of the relationship between handshake and application data messages significantly thwarts the straightforward adaptation of TLS 1.2 traffic classifiers to TLS 1.3 deployments, thus requiring more advanced traffic analysis techniques.

Challenges out of scope. TLS record padding and Pre-Shared Key (PSK) based authentication pose additional challenges in traffic classification that are considered outside the scope of our work.

2.3 The Need for Inference on TLS 1.3

The results described in the previous section revealed that features such as server certificate and application data record sizes are useful for detecting C2 traffic over TLS 1.2. While we hypothesize these features could also be useful for identifying TLS 1.3 C2 traffic, they remain obscured through the inner workings of TLS 1.3.

Here, we briefly assess the performance of the two new classifiers we propose when applied to our oversimplified dataset. Both classifiers leverage features that are not directly accessible, but which are instead *inferred* from TLS 1.3 traffic – recall that C-Cert is based on the inferred server certificate size, and C-Behav is based on the inferred sequence of application-data records. We relegate a thorough explanation of the inner workings of these classifiers to §4 and inference procedures to §5, and showcase how these allow for the development of improved TLS 1.3 C2 traffic classifiers.

More reliable C2 traffic classification for TLS 1.3. In the aforementioned simplified dataset, C-Cert achieves a TPR of 75% on malicious flows with and a TNR of 99.86% on all flows, revealing the strong contribution of certificate sizes for distinguishing C2 traffic from benign traffic. Indeed, when removing the 50 flows with artificially increased certificate size, we achieve a TPR of 100.0%, suggesting that C-Cert can be avoided by artificially padding certificates used in C2 communications. Alternatively, C-Behav achieves a TPR of 93.5% and a TNR of 99.83%. It shows that one can also detect C2 traffic from application behavior patterns, despite intentional modifications to the sizes of C2 servers' certificates.

In the following sections, we demonstrate the feasibility of our approach in more refined and realistic scenarios. §3 describes our methodology and datasets. §4 defines C-Cert and C-Behav, demonstrating the patterns in which their classifications are based on. §5 describes the aforementioned adaptation to TLS 1.3 flows in further detail, whereas §6 evaluates the efficacy of our classifiers.

3 METHODOLOGY AND DATASETS

We begin by introducing our assumptions and threat model, and by establishing definitions on TLS flows and corresponding methods pre-processing for feature extraction. Afterwards we provide an in-depth description of the datasets used throughout the paper.

3.1 Assumptions and Threat Model

We make the following assumptions regarding the TLS connections that we are processing and classifying.

TLS 1.3 with ECH. We assume a widespread adoption of TLS 1.3 with ECH. Server certificates, domain names and other potentially sensitive fields are hidden. Plaintext TLS records are mostly uninformative. Other records are encrypted and their content type is hidden. Notice that in this scenario, flows can still be isolated; passive observers performing TCP reassembly can see individual TLS records and take note of their plaintext data, direction (*client-to-server*, C2S, or *server-to-client*, S2C) and size. These records can be split into two groups: a first set of S2C records, and the remaining records. We perform server certificate size inference over the record lengths of the first group, providing us with an additional feature for traffic classification. The remaining records are filtered to exclude handshake messages, isolating application behavior.

No deviation from TLS specification. We expect clients and servers not to deviate from the standard TLS protocol specification.

No errors. We assume no protocol errors or failures. We discard flows containing "Hello Retry Request" messages due to limitations in our network capture processing software.

No lost packets. We assume that all flow packets are seen by our capturing process, which is important for TCP reassembly.

No padding. We assume that clients and servers will not use record padding. This is an option contemplated in TLS 1.3's standard but often met with reluctance, given its toll on bandwidth, possible delays and the need for a deliberate adoption.

Server certificate-based authentication. We assume no PSK-based authentication, nor clients certificate-based authentication.

§7.1 discusses the impact of not meeting these assumptions.

Threat model. Our threat model is that of a standard intrusion detection scenario. A client machine is compromised with malware, which will attempt to communicate with a C2 server located in some external network using TLS 1.3. Our classifiers will play the role of a firewall/IDS, retrieving and processing the messages exchanged in the network as a passive eavesdropper and then label traffic as malicious/benign. We assume that our classifiers run in a trusted environment, i.e., that the malware infecting the client cannot prevent our classifiers from eavesdropping network packets.

3.2 TLS Flows and Feature Extraction

With the typical TLS-over-TCP traffic model, each TLS connection is layered over a TCP connection. Different applications can then be layered on top of TLS: application data is broken into *TLS records* which are then secured, broken into TCP segments, and encapsulated into IP packets. TLS records may carry TLS messages instead of application data, but information about *content type* is hidden. For traffic analysis purposes, we group IP packets into *flows* – sets of IP packets identified by a five-tuple of client and server IP addresses, transport ports, and transport protocol [58]. We use these *flows* as our classification object.

Throughout our experiments, we process "raw" network traffic captures into tabular data by extracting certain metadata fields from each TLS flow. We use Cisco's Joy [17] for this, which can follow TLS connections and extract flow metadata. Some of these fields are extracted from *Client Hello* and *Server Hello* messages: the TLS record versions, and the *server_name*, *supported_versions*, *session_ticket* and *pre_shared_key* extension values. We extract the *HandshakeType* of the first TLS record observed to filter out TLS connections for which the *Client Hello* was not observed. We also check if the server has sent a *HelloRetryRequest*, which makes the client send a new *Client Hello* message. These extensions/parameters are all externally observable. Combined, they account for 14 values/rows. These are not used for classification, but rather to identify and filter out flows, to determine their negotiated TLS version, and whether PSKs are being used for authentication (bypassing certificate exchange in the handshake process). For TLS 1.2 flows, we also extract server certificate sizes, when observed.

Feature extraction. We also use Joy to extract features from each flow, namely details on individual TLS records: their payload size (in bytes), their direction (client-to-server, C2S, or server-to-client, S2C), their record type, and their inter-record time (in milliseconds). We do not use timing features since they are affected by the distance between client and server, by network conditions, or by the client machine resources.[1] We extract the size, direction and type of each flow's first 20 TLS records, accounting for a total of 60 features per flow – in every dataset we consider (see § 3), over 70% of the included TLS flows do not reach the 20 records mark. Flows with fewer than 20 records are padded. These features are either fed directly to classifiers, or further processed (e.g., filtered by record type), depending on the classifier and flow TLS version. Next, we present further details on the TLS traffic datasets used in our work.

[1]As our traffic comes from multiple capture environments with different class distributions, timing information could introduce a significant bias to distinguish the originating dataset (*leakage*), which would not capture a realistic application scenario.

Table 1: Dataset composition and classifiers overview (● → dataset is used for training and testing; ◑ → testing only).

	MTA	Tranco	MS	DoHBrw	UCNet
TLS ⩽ 1.2	✓	✓	✓		✓
TLS 1.3	★	✓	✓	✓	✓
TLS⩽1.2 flows	36K	282K	396K		4,50M
TLS 1.3 flows	0.5K	224K	395K	247K	5,77M
C2 traffic	✓		✓	†	?
non-C2 traffic	✓	✓		✓	✓
Public / In-House / Real-world	P	IH	IH	P	RW
`C-Basic`	●	●	●	◑	◑
`C-Cert`	●	●	◑	◑	◑
`C-Behav`	●	●	●	◑	◑

★ – labeled with certificate fingerprinting: TLS 1.3 is unlabeled and thus not used.
† – malicious DoH usage, traffic tunneled through a legitimate DNS server (not used)
? – campus traffic is mostly benign, but might include some malware infections

3.3 Dataset Synopsis

At the time of writing, despite TLS being extensively used by malware, there is not a precompiled labeled dataset featuring malware traffic over TLS. Some popular malware traffic datasets either: a) feature unencrypted traffic, for which there are other efficient detection methods; b) are outdated, and thus potentially no longer accurately represent current threats; or c) contain only select features, already extracted from pre-processed captures, preventing others from applying general processing techniques over "raw" traffic captures. To accurately represent our scenario, we gather benign and C2 TLS traffic by combining multiple sources. Table 1 summarizes our datasets, the contained TLS versions, associated labels, and the experiments used on. Succinctly, they are as follows.

- Malware Traffic Analysis (MTA) — traffic captures scraped from a public website, filtered for TLS and labeled using certificate fingerprinting (C2 and non-C2 traffic, TLS 1.2 only).
- Tranco — in-house captures of automated web browsing-related network traffic (non-C2 traffic, both TLS versions).
- Metasploit (MS) — in-house captures of C2 traffic, generated with a Metasploit Framework-based infrastructure (C2 traffic only, both TLS versions).
- DoHBrw — benign TLS 1.3 traffic from the public CIRA-CIC-DoHBrw-2020 dataset [46] (non-C2 traffic, TLS 1.3 only).
- UCNet — traffic captured on a university campus network (both TLS versions).

The datasets produced by in-house captures are available in [47].

Classification setup. TLS 1.3 traffic is never used for training – all TLS 1.3 traffic is placed into a '*test*' set from the beginning, with our premise being that a reliable groundtruth for TLS 1.3 traffic and TLS 1.3 C2 datasets are (at the moment) still difficult to obtain. The MTA and Tranco datasets were used for classifiers' training and validation: These datasets are split into *training/validation* and *test* sets using stratified group k-fold [55] (SGKF) – maintaining the percentage of samples for each class (C2 vs. non-C2) while making sure that the same group (file from which the flow was obtained) does not appear in both sets. The *training/validation* set was further split using SGKF into 10 folds, on which we perform cross-validation (CV) for parameter tuning. After determining the best parameters, classifiers are trained on the whole *training/validation* set, and evaluated on the test set. These *test* metrics are the ones presented throughout the paper. The *test* set is never used for training nor parameter tuning, only for the final classifier evaluation.

The MS dataset is generally used for testing only, but also partially for C-Basic and C-Behav's training: 50% of the 'MS TLS 1.2 short-certificate' files are reserved for testing only. Out of the remaining 'TLS 1.2 short' files, 90% of flows are also randomly reserved for testing, leaving only 10% (of 50%) for training/validation. This small percentage of flows roughly matches the numbers of common malware families within the MTA set. These are combined with the MTA and Tranco training/validation sets, with 10-fold SGKF being used here too. The DoHBrw and UCNet datasets are only ever used for testing. Our code is based on Python 3.10, and uses Pandas (2.1.1), NumPy (1.26.1) and scikit-learn (1.3.2) modules.

3.4 Publicly-Available Datasets

Malware Traffic Analysis (MTA). This dataset was constructed from traffic captures corresponding to malware infections, by scraping the MTA website [21]. This website contains multiple posts with traffic captures, typically generated by running malware samples in a *sandbox* environment; these captures usually depict the initial stages of a malware infection and its C2 communication, often also including the download, C2 and actions of additional "follow-up" malware downloaded by the first sample (e.g. *spambot* activity, *denial-of-service* attacks, network discovery, etc.). MTA is a frequently updated source of "raw" malware traffic captures, which has been used by multiple recent works focused on malware traffic detection [11, 30, 38, 39]. 2345 files we obtained via this extraction process, dated from 2013 until late 2020.

To establish our groundtruth, we filter these captures by TLS traffic, and identify C2 communication based on server certificates. To match with C2 server certificates, we used the SSL Blacklist project's (SSLBL) [1] list of certificates that are known to be employed by botnet C2 servers. We have also identified additional C2 server certificates manually, by crossing information from the MTA captures (such as IP addresses, certificate Common Names and Issuer Distinguished Names) with information from MTA itself and from VirusTotal [40]. This resulted in 4795 certificates from SSLBL corresponding to 11243 TLS flows, and 115 manually identified certificates associated with 6910 TLS flows.[2]

Our preliminary analysis has shown that recent years have seen a great increase in TLS traffic, but a small number of TLS flows identified as C2 via SSLBL, suggesting a mismatch between MTA and SSLBL. As such, our database does not include: a) recent MTA captures (roughly from 2021 onward) which had a negative impact on the classifiers' learning, indicative of ground truth errors; and b) TLS connections for which server certificates cannot be obtained, because they are either running TLS 1.3 or performing PSK-based authentication, a mechanism formerly known as *session resumption*. These two sources of potential C2 TLS flows were excluded due to their unreliable ground truth.

DoHBrw. For another source of TLS 1.3 benign traffic, we use the CIRA-CIC-DoHBrw-2020 (DoHBrw) dataset [46], which was built with the purpose of detecting and characterizing DoH traffic. This dataset contains raw traffic captures of non-DoH and benign DoH traffic, generated by visiting websites using HTTPS in browsers appropriately configured. It also contains malicious DoH traffic

[2]Our manual labeling is not exhaustive. This is because we cannot confidently blocklist certificates for which we have little associated traffic and no other information. Despite being associated with a malware infection, not all MTA traffic is necessarily C2.

generated with DNS tunneling tools. We don't use the malicious DoH traffic in our training as we're not focused on detecting malicious use of DoH: this type of traffic is a very specific context of a legitimate service being abused for malicious communication. Instead, we intend to make more generic C2 detectors. We use this dataset as a source of benign TLS 1.3 traffic for testing purposes.

3.5 In-House Datasets

Tranco. To augment our TLS dataset, we use Browsertime [57], a web browsing automation tool. We browse websites from the Tranco list generated on 28 January 2024 to obtain benign (non-C2) TLS traffic. Out of the 1 million domains included in the Tranco list, all the first 10K were visited. Then, 1-out-of-10 were randomly selected from the following 240K domains, 1-out-of-20 from the next 250K, and 1-out-of-50 from the last 500K. This resulted in a total of 56500 domains. Each website is visited twice: once using TLS 1.2 and once using TLS 1.3, by changing the browser's configurations. We use Cloudflare's "Malware and Adult Content Blocking" DNS resolver [18] to avoid malware-related websites. This capture resulted in 505938 TLS flows: 282227 TLS 1.2, and 223711 TLS 1.3.

We have also used curl [19] to capture web requests to the Alexa Internet TOP 1000 websites, using both TLS 1.2 and TLS 1.3. We saved the TLS pre-master secrets, allowing us to later decipher the traffic and the TLS 1.3 handshakes in particular. This dataset was not used to train or test classifiers directly, but rather to compare both TLS versions and to better analyze TLS 1.3 traffic, contributing to the Certificate Size Inference process and related statistics (§5.2).

Metasploit (MS). For malicious traffic, and given the lack of confirmed malicious TLS 1.3 C2 traffic on public datasets, we generate traffic using Metasploit, and the Python version of Meterpreter [52], with some adaptations to allow for TLS version specification and for pre-master secret storage for later analysis. Meterpreter has been associated with recent attacks [23], and Metasploit has been identified as the third most frequently seen malware family of 2021 and 2022 [43]. Unlike other more popular malware families, both MS and Meterpreter are open-source, making them good resources for generating traffic illustrative of C2 communication. The communication between the Meterpreter executable and the MS framework acting as C2 server uses HTTP over TLS. The executable establishes short-lived TLS connections to poll the C2 server for commands and to send back responses. Despite the encryption provided by TLS, the HTTP requests and responses are additionally encrypted.

By default, when listening for executables connecting over TLS, the MS framework will generate a new server certificate each time it is run. This certificate is self-signed, and filled with random information for city, state, organization name, and organizational unit. MS authors claim that such certificates are realistic-looking and reduce the chance of signature-based detections by IDSs [51]. Being self-signed, these certificates are not expected to be trusted by the victim's operating system, nor by traffic inspection tools between the victim and C2 server. Naturally, direct certificate validation will be unfeasible after switching to TLS 1.3 since certificates are end-to-end encrypted. However, our proposed techniques will include certificate size inference from the TLS 1.3 traffic flow, which will play a key role in this context. No additional changes were made to MS and Meterpreter besides modifying the server certificate generator, fixing the desired TLS version and storing pre-master secrets. As such, the traffic we generate should be similar to other deployments using unaltered code and default options. An obvious downside is the relative predictability of MS's C2 traffic, and its impact on detection performance. However, we do not train classifiers to simply distinguish MS traffic from other types of traffic; instead, we use MS as an example of a new malware family.

We captured traffic using both TLS versions and with short and long server certificates – i.e. single self-signed or chained certificates, respectively. This resulted in 132.2K (1.2/short), 131.7K (1.3/short), 264.1K (1.2/long), and 263.1K (1.3/long) TLS flows.

3.6 Real-World Datasets

UCNet. To test our methodology in a non-simulated environment with real users, we created a new dataset consisting of anonymized network traces collected from the university campus network (UCNet) at the host institution of some co-authors of this study. We concentrated on monitoring specifically the wireless and VPN networks as we wanted to capture i.) numerous TLS flows, therefore leaning us toward high user-density networks as opposed to other internal campus networks, and ii.) collect TLS traffic exchanged with the Internet, which most likely entails C2 communications. Both the wireless network and the VPN network are designed to serve 14K users each, including students, professors, and staff.

Data collection process. Given the sensitivity of analyzing traffic from a real network, we have strictly followed all the guidelines dictated by our Institutional Review Board (IRB) to guarantee that the collected dataset does not reveal privacy-sensitive information about users, services, and network infrastructure (see §9). For data collection, we use an on-campus capture machine positioned above the edge router that serves as the entry/exit point to the Internet for the campus network. This machine is a Dell Inc. PowerEdge R650xs server running Linux Debian Bullseye. Traffic is captured with an Intel Ethernet Controller X710 10GbE SFP+ network interface. The configuration for the capture interface is 10Gbase-SR/Full Duplex with no auto-negotiation and a link speed of 10Gbps, and it is connected to a switch that mirrors the inbound and outbound traffic from the campus router. Using `tcpdump`, we collect raw network traces and forward them to a second machine for processing, through another network interface. After a capture is finished, it is processed in two main steps: i.) Joy parses the captures into TCP flows and anonymizes the extracted metadata, producing a JSON-like output; and ii.) a Python3 script filters flows for TLS connections, and extracts relevant features. LibreNMS is used to gather traffic statistics via SNMP requests to multiple network devices.

Monitored traffic and dataset composition. We monitored the campus network from March 2 to April 2, 2024. During this time, we conducted captures lasting 1 hour and 30 minutes, starting at 3 pm every day from Monday to Friday. The average time to produce each trace, including capture and processing, is roughly 5 hours, with an average capture size of 263GB. Some traces were excluded due to memory-related errors during collection or interrupted connections between our machines. Ultimately, we successfully collected 10 traces from various days, which were then assembled into a single dataset of 10.1M flows. Specifically, 4,337,589 correspond to TLS 1.2 flows, and 5,769,412 relate to TLS 1.3. Appendix A, provides additional insights on the data, such as statistics on traffic volume.

Table 2: Detection results for C-Basic, trained on the sets mentioned in section 3. Since all MS traffic is C2, only TPR percentages are shown; all Tranco and DoHBrw traffic is non-C2, thus only TNR percentages are shown. Metrics refer to 'test' sets only.

Classifier	MTA		Tranco (TNR)		MS TLS 1.2 (TPR)		MS TLS 1.3 (TPR)		DoHBrw
(Config.)	TNR	TPR	TLS 1.2	TLS 1.3	short	long	short	long	(TNR)
C-Basic (DTC)	96.87	98.82	100.00	99.98	100.00	0.00	**0.00**	0.00	99.97
C-Basic (RF)	96.68	98.92	100.00	100.00	100.00	0.00	**0.00**	0.00	99.98

4 TLS 1.2-BASED TRAFFIC CLASSIFICATION

In this section, we first aim to expand our initial understanding (see §2.2) on the detection of TLS 1.2 C2 traffic across datasets with different characteristics. These results can then be used as a baseline for comparing the performance of classifiers aiming to detect TLS 1.3 C2 traffic. Second, we perform feature importance analyzes and explore how to leverage the resulting information for building TLS 1.3 traffic classifiers, while considering changes from TLS 1.2 to 1.3.

We build on the insights of Anderson et al. [5, 8], using TLS record lengths as features for our classifiers. With TLS layered over of TCP, sequences of TLS record lengths are more informative than sequences of TCP segment sizes, since the latter may require packet reassembly. In TLS 1.2, protocol-related messages are easily identified using the record type: 21 for Alert messages, 22 for Handshake messages, and 23 for Application Data. Handshake messages are not encrypted, allowing for information extraction; Alert messages are encrypted, but their type is externally observable.

For classifiers, we use single decision tree classifiers (DTCs) [13] and Random Forests (RFs) [12]. DTCs are easier to interpret, facilitating the selection of the most relevant features, and their decision logic can be easily converted into rules for IDSs. RFs are ensembles of DTCs, which have also been used for network traffic classification tasks, but they are less interpretable than single DTCs [8].

4.1 Features: All TLS Records — C-Basic

For the first experiment, we train classifiers on sequences of TLS records sizes, directions and types. We perform grid search with CV to adjust model parameters and select the ones leading to the best F1-score, to ensure a balance between classifier precision and recall. For the DTC classifier, this results in: maximum tree depth (m.t.d.) of 16, considering all features, using the Gini impurity as split criterion, and no class weighting. For the RF classifier: 70 estimators, m.t.d. of 15, all features considered, and no class weighting as well.

The main purpose of this step is to identify key features in the classification of C2 traffic. The feature importance is obtained for each training CV split, with the 5 most important features for the DTC classifier being, on average, `Size0` (0.502), `Size2` (0.278), `Size6` (0.122), `Size1` (0.042) and `Size4` (0.014). These values are identical to the ones obtained for the RF classifier, and suggest that `Size0` and `Size2` are the most influential to the classification of C2 traffic. These are the sizes of the first and third TLS records, typically corresponding to the Client Hello and Server Certificate messages respectively. According to this insight, we prioritize the consideration of these messages. Intuitively, *Hello* messages reflect the selected cipher suites/extensions, while certificate chain sizes suggest a pattern in the certificates used by C2 servers.

Table 2 demonstrates the results obtained by these classifiers. TPRs of 98.82% and 100.0% are obtained for the MTA and 'MS TLS 1.2 short' test sets when using DTCs. However, no 'MS TLS 1.2 long' flows were identified as C2. Testing against TLS 1.3 traffic – Tranco, MS and DoHBrw – one can observe that it is almost entirely classified as benign. We can thus conclude that, while C-Basic detects MS traffic using default (short) server certificates and default options, no MS flows are detected when using chained certificates or when simply upgrading to TLS 1.3 (Tab. 2, in bold).

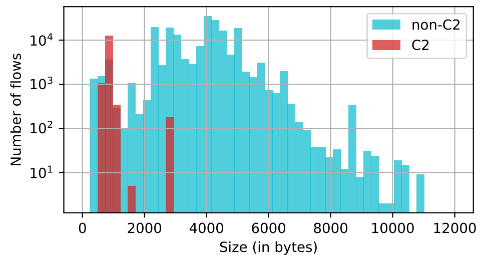

Figure 1: Histogram of the server certificate chain sizes across the two classes for the MTA and Tranco datasets.

This strongly suggests that i.) C-Basic is ineffective with TLS 1.3 traffic, potentially due to the fundamental differences of the protocols, classifying it mostly as negative/benign; and ii.) server certificate chain sizes greatly influence the performance of C-Basic.

Finding 1: Sequences of record sizes can be extracted for both TLS 1.2 and TLS 1.3 flows, but are not equivalent to one another. Classifiers trained on TLS 1.2 traffic are unreliable on TLS 1.3.

4.2 Feature: Certificate Chain Size — C-Cert

Building on the relevance of certificate chain size for C2 traffic classification, we depict the distribution of certificate chain sizes for the MTA and Tranco dataset on Figure 1. It suggests a correlation between C2-related TLS sessions, and smaller certificate sizes. This can be explained by the usage of self-signed certificates on connections established by malware, an uncommon practice among legitimate services[3]. Importantly, this is something that can be easily identified when Server Certificates are sent in plaintext.

Considering this key feature, we introduce the classifier C-Cert. This classification works by extracting the certificate chain size in bytes from each TLS session, and using this size as the single feature to distinguish between C2 and non-C2. Grid search with CV is used to adjust classifier parameters, this time selecting the ones leading to the highest TPR score. For the DTC classifier, this results in: m.t.d. of 1, using the Gini impurity as split criterion, and class weights inversely proportional to class frequencies. For the RF classifier: 40 estimators, m.t.d. of 5, using entropy as the split criterion, and class weights inversely proportional to class frequencies. We artificially add some error to C-Cert's training,

[3]While self-signed certificates may be used for legitimate purposes, they are commonly associated with 'test' or internal deployments, or services not meant to be publicly accessible. Network administrators may add the expected internal services to an allowlist, and then verify the exceptions.

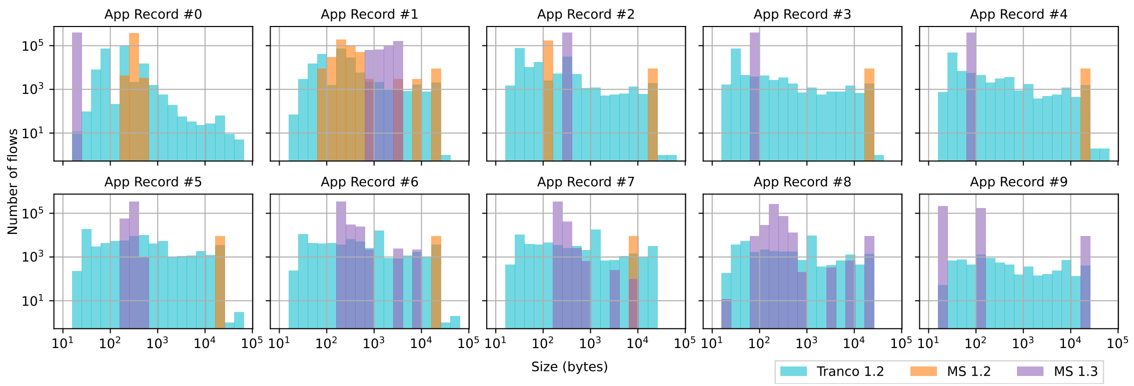

Figure 2: Histogram of the first 10 record sizes in both directions (C2S and S2C), filtered by *Application Data* (content type 23).

Table 3: Detection results when considering server certificate chain size (C-Cert) and application data records (C-Behav).

Classifier (Config.)	MTA TNR	MTA TPR	Tranco TLS 1.2	MS TLS 1.2 short	MS TLS 1.2 long
C-Cert (DTC)	56.61	98.89	99.73	100.00	**0.00**
C-Cert (RF)	76.69	99.77	99.74	100.00	**0.00**
C-Behav (DTC)	93.90	95.64	99.93	98.97	98.85
C-Behav (RF)	91.58	97.59	99.85	99.16	99.03

to make it more robust to small variations and to errors that we anticipate for certificate size inference on TLS 1.3.

Table 3 (top) shows that discriminating traffic based on this feature alone yields performance rates identical to C-Basic, except for the TNR on MTA's test set. The TNR results of MTA might intuitively indicate that the classifier is regularly identifying non-C2 traffic as C2. However, upon further inspection, multiple supposedly non-C2 flows of MTA have been identified to actually correspond to C2 traffic. Given MTA's non-exhaustive groundtruth, one must take these results with a grain of salt, which motivates the need for multiple TNR result assessments on different datasets depending on the reliability of their groundtruth. These values are nonetheless aligned with our previous thesis that certificate sizes are playing a significant role in C-Basic's classification. When applying C-Cert to MS traffic, all of MS TLS 1.2 'short' traffic is correctly identified as C2. However, note that traffic using 'long' certificate chains bypasses detection completely (Tab. 3, in bold).

Certificate classification results could likely be improved if we additionally considered certificate contents [34], e.g., through subject and issuer analysis, checking the certificate validity date, or analyzing the Common Name for Domain Generation Algorithms. Unfortunately, this information cannot be reasonably extracted in TLS 1.3. Certificate sizes, however, can potentially be inferred.

Finding 2: There are patterns in server certificate sizes that can be used for basic traffic classification. Smaller certificates are consistent with the usage of self-signed certificates, which is the *off-the-shelf* behavior of emulating a malware on Metasploit.

4.3 Features: Application Data Only — C-Behav

Alternatively, we consider a classification methodology focused on size and direction of *Application Data* records (type 23). This classifier is named C-Behav. By ignoring protocol-related messages and metadata, we attempt to detect malware C2 based solely on the application behavior reflected in its TLS communication. Figure 2 depicts the distribution of sizes of the 10 first records of Tranco TLS 1.2 and MS, after simply filtering for application data (record content type 23). These histograms showcase a noticeable pattern in MS's application behavior, which may allow for its detection. Observe that this pattern is completely unrelated to the server certificate. As such, if this detection approach is successful, then modifying certificate characteristics, or even TLS extensions or parameters, would have reduced impact on detection, unlike fingerprint-based detection, which can be susceptible to such changes. Furthermore, classifiers based on behavior are expected to more reliably detect *off-the-shelf* C2: modifying application behavior would imply rewriting malware application code, something that cybersecurity threats such *script-kiddies* often abstain from doing. Indeed, despite more challenging, application behavior-based detection may be more directly transferable to TLS 1.3, as it upgrades TLS 1.2 by enhancing privacy of communication metadata, but does not impose any changes on how the underlying applications communicate.

Classifiers are trained on directions and sizes of the first 10 application data records of each flow (20 features in total). We use grid search with CV to adjust model parameters for F1-score. For the DTC classifier: m.t.d. of 16, considering all features, using entropy as split criterion, and no class weighting. For the RF classifier: 70 estimators, m.t.d. of 20, maximum number of features equal to the square root of the total, and class weights inversely proportional to class frequencies. We also artificially add some error to C-Behav's training, in an attempt to make it robust to small variations and errors. We empirically chose a non-central chi-square distribution with 3 degrees of freedom and a non-centrality parameter of 10. Feature importance is obtained for each training CV split, with the 4 most important features being, on average, `Size0` (0.705), `Size1` (0.156), `Direction1` (0.050) and `Size2` (0.033) for the DTC classifier, and `Size0` (0.371), `Size1` (0.187), `Size2` (0.100) and `Direction0` (0.085) for the RF classifier. This suggests that the first application data records are the most influential for C2 traffic classification.

Table 3 (bottom) presents the results. The number of samples is slightly smaller for this experiment, as some flows were discarded for not containing any application data records. By comparing results with C-Basic, C-Behav's lower efficacy highlights the impact of removing the information conveyed by handshake-related records. However, contrary to previous results, performance on MS TLS 1.2 is now similar for both server certificate chain sizes. Note that, similarly to C-Basic, C-Behav was trained on a small portion of 'MS TLS 1.2 short' traffic. However, only when removing handshake-related records was the classifier able to detect 'MS TLS 1.2 long' traffic. This suggests that a behavioral pattern exists.

While TLS 1.3 does not fundamentally change how application-data records are protected, these cannot be easily distinguished from protocol-related records. This can also be observed in Figure 2, where 'MS TLS 1.2' and 'MS TLS 1.3' "application data" records (type 23) exhibit different distributions, despite corresponding to the same application. This is because the TLS 1.3 ones may now be carrying handshake messages. As such, these classifiers cannot be adequately used on TLS 1.3 without first discarding protocol-related records — Trying to do so yields a TPR of 0% on MS TLS 1.3 traffic, both for DTCs and RFs, similarly to what happens with C-Basic.

Finding 3: Patterns in application behavior can be used for a C2 traffic classification that is independent of certificate size. However, an "application behavior extraction" procedure is needed for TLS 1.3, since some application data records are, by design, carrying handshake messages.

5 TLS 1.3 TRAFFIC ANALYSIS

We now analyze the structure and typical sequence of TLS 1.3 records, and propose methodologies to extract additional information from traffic flows. These will be used to refine C-Cert, and to effectively apply C-Cert and C-Behav classifiers to TLS 1.3 traffic. All information related to TLS 1.3 specifications is based on [53].

5.1 TLS 1.3 Record and Handshake Dissection

Only Client Hello and Server Hello messages are sent in plaintext. The following messages – further handshake messages, alerts, and application data, that may contain privacy-sensitive information – are wrapped in TLS 1.2 application records. Figure 3 illustrates how TLS 1.3 encrypts these messages, which become opaque encrypted communication. TLS 1.3 adds one extra byte to indicate the actual record type, and 16 extra bytes for an authentication tag generated by TLS 1.3's authenticated encryption algorithms.

This format employed by TLS 1.3 has other consequences for traffic classification: application behavior is harder to isolate from protocol-related behavior. However, the Handshake process still follows a fixed structure in the first TLS records after the TCP handshake, which can be used to infer certificate sizes.

First, the Client sends a Client Hello with supported cipher suites, extensions, and sharing keys. The Server responds with a Server Hello, selecting a cipher suite and also sharing keys. Optionally, it can also send a *Change Cipher Spec* (CCS) record. From then on, exchanged records are encrypted. Then, the Server sends multiple handshake messages: one for Encrypted Extensions (mandatory, even though extensions themselves are optional); one for a Client Certificate Request (optional); one for Server Certificate (optional); a Certificate Verify record (mandatory when server certificate is sent); and a final Finished message (mandatory). The client responds with an optional CCS record, indicating that its following TLS records are encrypted, and a mandatory Finished record. Optional Client Certificate and Certificate Verify messages may be sent beforehand, following the same logic as those for the server certificate. Finally, if the handshake was successful, application data can be sent in both directions. Additional opaque protocol-related messages may be interleaved with opaque application data records.

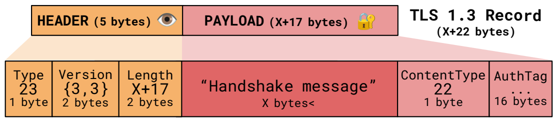

Figure 3: TLS 1.3 record format with illustrative fields/sizes.

Caveats. When present, the server certificate is always the third to last handshake message sent by the server. These messages are somewhat independent of TLS records and may be sent all together in one record, or spread across multiple records; a single message may be fragmented over multiple records, adding uncertainty to certificate size inference, or application behavior isolation.

An extension has been proposed enabling support for certificate compression in TLS 1.3 [28]. From our observation, compressed non-self-signed certificates are still larger than uncompressed self-signed certificates. Indeed, the procedure for certificate size inference is mostly unaffected, since the *Compressed Certificate* message directly replaces the *Certificate* message. The inferred size will then correspond to the (smaller) length of the compressed certificate.

5.2 Certificate Size Inference

Given the previously outlined handshake process, and considering the most commonly observed TLS 1.3 traffic in our curl captures (server certificate sent, no client authentication), we propose a method for server certificate size inference. This is based on the insight that, although in encrypted form, this certificate must be sent in a well-defined interval: necessarily after the Server Hello record (or CCS record, when present); and necessarily before the client's CCS record (optional) or application data records.

Towards this goal, we isolate the records that meet these criteria. If a single record is observed, then it must contain all of the multiple handshake messages mentioned previously. While the server certificate cannot be isolated from the other messages in ciphertext, its size may be estimated by discounting the size of the other messages – the size of the other messages is also an estimation, but with less variation. Otherwise, if multiple records are observed, then either: i.) the messages are fragmented across multiple records, and their payload sizes must be added together, subtracting the 17 extra bytes per record, and dealt with as if they were a single record; or ii.) each handshake message is sent in its own individual record with only the certificate possibly fragmented. This can be detected by checking if the last record of the set has a size compatible with a *Finished* message. In the latter case, the certificate message is likely in the third to last record, or fragmented across the second and the third to last records of the set. This means that subtracting the size of other handshake messages is not necessary, since handshake messages are already sent separate TLS records. This certificate inference logic is illustrated in Figure 4 — boxes in light orange represent encrypted handshake records, sent in the S2C direction after the Server Hello, and before any C2S application data records. Record sizes are observable, but not their content.

Handshake messages. We now describe the encrypted handshake messages sent by the server [53], as well as the sizes we observed:

- *Encrypted Extensions (EE) message*: 6 bytes for type and length indications, followed by optional extensions sent in response to a client request. Our curl captures suggested the following three extensions to be the most frequently used. `Server_name` is an empty (all zeros) extension taking up 4 bytes, found in 80.0% of our TLS 1.3 sessions. `Supported_groups` is chosen from a set sent by the client, taking 6 bytes for the header and 2 bytes per group. It was found in 13.4% of our sessions, commonly selecting 2 groups: 10 bytes. `ALPN` is chosen from a list sent by the client and takes up either 9 or 15 bytes on a header and a string, depending if the application layer protocol is HTTP/2 or HTTP/1.1. It is very common in our observations, found in 96.9% of sessions, and mostly HTTP/2 resulting in 9 bytes.

- *Certificate message*: an 8 bytes header, considering type and length indication as well as an empty certificate request context. This is followed by a chain of certificates, usually with 5 extra bytes for each certificate in the chain. This results in a size of $8 + N + 5 \times C$, where N is the certificate chain size in bytes and C is the number of chain certificates.
- *Certificate Verify message*: a 6 bytes header to indicate type, length and signature algorithm, and a signature. The signature's length depends on the algorithm. In our observations, this is 70 to 72 bytes for ECDSA (`ecdsa_secp256r1_sha256`, used in 39.8% of sessions), and 256 (58.8%) or 512 (1.4%) bytes for RSASSA-PSS (`rsa_pss_rsae_sha256`) and depending on the RSA key length. No other signature lengths were observed.
- *Finished message*: 36 or 52 bytes, depending on selected cipher suite. This includes a 4 bytes header, and 32/48 bytes of verify data (SHA256/348). The algorithm used is directly observable in plaintext, in the Server Hello record.

Given the public parameters of the Client Hello, we may further narrow down our guess of the overhead introduced by the TLS handshake. E.g., if the client uses the ALPN extension, the server will likely include ALPN in its EE and select the client's preferred (first) application layer protocol. Otherwise, if ALPN is missing, it should not appear in the server's EE either. Using fewer extensions and algorithm possibilities in the Client Hello will lead to smaller, more predictable server authentications, facilitating fingerprinting.

Inference process and error assessment. Given the sizes stated before, a minimum of 6+8+6+70+36 = 126 bytes must be discounted from the total handshake messages size. Preliminary evaluation has shown that 132 bytes is a value leading to a low inference error, due to the combination of extensions, signature and hash algorithms most commonly present in server responses. 17 bytes must also be discounted from each TLS record, accounting for the content type and authentication tag. After filtering traffic for valid TLS 1.3 sessions, we isolate the first application data records (type 23) in the S2C direction before any C2S application data records are seen. These presumably contain the Handshake messages.

If one single application data record is isolated, it should contain all Handshake messages; in that case, we discount 132+17 bytes from that record size to obtain the inferred certificate size. If more than one record was isolated but the first one is larger than 300 bytes, we probably have multiple handshake messages sent together as a single message, but fragmented in multiple records. In that case we sum the sizes of all the record fragments we isolated and discount 17 bytes per record plus the 132 extra bytes. Otherwise, if the first record is not 300 bytes long, then each handshake message was probably sent in its own record, with the certificate possibly fragmented. In that case, we may discard the *Finished* record based

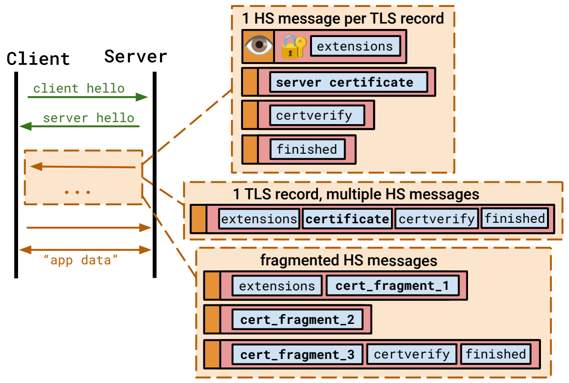

Figure 4: Certificate size inference process. Arrows depict TLS records, orange arrows depict *Application Data* records. Encrypted Handshake (HS) messages may be divided into application data records in one of multiple ways.

on its size, and every record after it. The 300 bytes threshold was empirically chosen to distinguish an isolated EE message from a fragment of a larger message. Then, if we're left with more than 3 records, we sum the presumable certificate fragments, discounting each record overheads. If we're left with 3 records or fewer, we simply return the size of the largest one, also discarding the record overhead. This process is depicted as a diagram in Appendix B.

5.3 Application Behavior Extraction

To enable an external detection based on application behavior, we must look at the sequence of TLS records that carry application data in isolation. While isolating application records in TLS 1.2 is as simple as looking at the TLS record metadata, in TLS 1.3 the record types are encrypted, and actual application data records are not immediately discernible from handshake and other non-application records. Our process of isolating application behavior in TLS 1.3 is twofold: first we discard handshake records by leveraging the previous characterization used for certificate size inference. then, we try to identify and discard records carrying Post-Handshake and Alert messages, since they are also protocol-related and thus not actual data explicitly sent by the application.

Handshake records are removed by discarding records until the first application data record in the C2S direction, which presumably contains the client's *Finished* handshake message. If no application data records are found in the C2S direction, the flow is ignored for lack of confirmation of a successful handshake. Flows are also ignored if no records are left after discarding the handshake.

Regarding Alert messages, we discard the last record of each sequence if they are exactly 19 bytes long, since that is the expected size of a *Close Notify* alert message. Other alerts are less common as they result from errors and lead to an abortive closure of the connection. We then focus on Post-Handshake Messages, which may be sent at any point after the Finished handshake messages. They are frequently used to issue *New Session Tickets* (NSTs) that allow for session resumption. From our observation, NSTs are usually sent in the first S2C records, often in pairs, with sizes ranging from 57 to 281 bytes each. A single record may carry multiple NSTs.

We tag records as likely NSTs if they appear near the beginning of a TLS session in the S2C direction, and if they're not the only response to a previous client request. Specifically, if a S2C record is seen before any C2S application data record, it is tagged for removal together with all other consecutive S2C records of the same size. Otherwise, if we see a C2S request for which there are multiple S2C records in response, the first response is marked as NST, together with other consecutive records of the same size. The exceptions are if those S2C records are <60 or >600 bytes, in which case no records are discarded. This range accounts for the possibility of two NSTs being in a single record. There are other Post-Handshake Messages besides NSTs, used for client authentication or cryptographic key updates, but these have not been observed in our captures.

5.4 Results

Using both of the methods we propose – Certificate Size Inference and Application Behavior Extraction – our previous classifiers C-Cert and C-Behav can now be applied to (parsed) TLS 1.3 traffic.

Impact of certificate size inference. By inferring server certificate sizes from the TLS 1.3 test flows, and then feeding them to the C-Cert classifiers, we obtain the results on Table 4 (top). While there seems to be a decline in performance for Tranco traffic, particularly for the DTC classifier, the RF classifier presents TNRs of over 99% on Tranco and DoHBrw traffic while also reliably identifying the MS TLS 1.3 'off-the-shelf' malware (short certificates) traffic. These results show that classification performance is similar to the one obtained for TLS 1.2 (Table 3), confirming that a useful classification feature is being successfully extracted from TLS 1.3 traffic. This is achieved without sacrificing TNR, an important characteristic for real-world applicability of IDS systems.

However, as expected, MS traffic using 'long' certificate chains is misclassified. Recall that these flows were added to assess classifier efficacy in a scenario where a knowledgeable adversary artificially increases certificate sizes to avoid detection.

Finding 4: Server certificate sizes can be reasonably inferred for most TLS 1.3 traffic. This feature can then be used for efficient TLS 1.3 traffic classification, with very high TNR values.

Impact of application behavior extraction. Our results when applying application behavior extraction to TLS 1.3 traffic, followed by classification using C-Behav, are depicted in Table 4 (bottom). Performance on the Tranco and DoHBrw test sets yields TNRs over 99% for the RF classifier. As expected, performance on the MS test set is slightly lower, with TPRs of around 93%. When compared to the results from Table 3 we see a performance toll of around 6%, which can be attributed to the errors and uncertainties inherent to the application behavior extraction procedure.

Most importantly, C-Behav's performance on TLS 1.3 is mostly independent of the server certificate sizes, allowing for 'MS TLS 1.3 long' to be correctly identified. We highlight that only a few 'MS TLS 1.2 short' flows were used for training, as previously explained. This demonstrates that C-Behav can be used to effectively circumvent countermeasures to C-Cert that artificially increase the size of certificates. Intuitively, and as is common in intrusion detection systems, we expect the efficacy of C-Behav to drop if additional manipulation is done at the application level, e.g. adapting the malware to pad messages erratically.

Finding 5: After discarding protocol-related records, we can feed TLS 1.3 application data to classifiers trained on TLS 1.2 application data, yielding promising results for real-world applicability.

Table 4: Detection results for C-Cert and C-Behav, after certificate size inference and application behavior extraction.

Classifier (Config.)	Tranco TLS 1.3	MS TLS 1.3 short	MS TLS 1.3 long	DoHBrw (TNR)
C-Cert (DTC)	94.91	100.0	0.00	99.61
C-Cert (RF)	99.91	100.0	0.00	99.48
C-Behav (DTC)	98.64	93.93	93.54	97.27
C-Behav (RF)	99.78	93.87	93.21	99.42

6 REAL-WORLD TRAFFIC CAPTURE RESULTS

To find out whether our proposed classifiers and inference methods can be used on real-world Internet traffic, we test them on UCNet. We assume that UCNet's traffic is all non-C2 related, which might not be completely true – some campus users might have malware-infected machines. We present values for (true) negative rates only. We do not show results for C-Basic applied to UCNet, since we have already established that it is not useful on TLS 1.3 traffic, classifying it mostly as benign (see §4.1). Its high TNR would be an artificially good result on UCNet, with little relevance.

Applying C-Cert and certificate size inference. C-Cert is applied to TLS 1.2 traffic directly, or to TLS 1.3 traffic by means of certificate size inference. From the over 4.3M TLS 1.2 flows of UCNet, server certificates could not be extracted on roughly 0.6M. The other 3.7M flows whose certificates were successfully extracted were classified using C-Cert (DTC/RF), both with 99.89% TNR.

Using certificate size inference on TLS 1.3 traffic, we were able to obtain certificate sizes for 3.2M flows (other 1.4M flows used PSKs, nearly 1M had an "Hello Retry Request" and certificate size inference failed for 144K flows). The inferred sizes were given to C-Cert (DTC/RF), resulting in TNRs of 99.07%/99.13% respectively. The roughly 30K flows positively identified by the DTC classifier were found to correspond to 1412 clients and 656 server IPs. Almost 27K of these flows feature no SNI, and the remaining ones correspond to 165 SNIs, with a distribution that is far from uniform: e.g., only 37 SNIs have more that 2 flows. These TNR results are similar to the inverse false-positive rates achieved by other IDS systems [22, 61], which suggests their potential for practical applicability.

When analyzing positive results, the *sysadmin* conducting the data capture found multiple SNIs consistent with Domain Generation Algorithms (DGAs), typically associated with malware activity. These are likely true positives. Interestingly, the RF classifier identified fewer DGA-related domains. This highlights the difficulty in selecting the "best" classifier when facing a noisy groundtruth.

Applying C-Behav and application behavior extraction. Once again, C-Behav can also be applied to TLS 1.2 traffic directly, or to TLS 1.3 traffic after using application behavior extraction. 192K TLS 1.2 flows and 8.8K TLS 1.3 flows were discarded for having no application data records. 258K TLS 1.3 flows were also discarded for having no application data after applying behavior extraction. C-Behav (DTC/RF) had negative rates of 90.39%/95.69% on TLS 1.2 traffic, and 92.28%/96.35% on TLS 1.3 traffic. Interestingly, the metrics for TLS 1.2 seem to closely lag behind the ones for TLS 1.3. This suggests that the poor metrics are not necessarily related to our application behavior extraction, but rather to some legitimate UCNet traffic not being accurately represented by our training data. Note that classifiers were not trained on UCNet traffic.

Incident discovery. An interesting outcome of this experimental validation was the discovery of a malware incident within UCNet's traffic, while analyzing the positives pointed out by the C-Cert (DTC) classifier. After noticing the seemingly DGA-related SNIs, server domains and IP addresses were checked against VirusTotal [40], a malware intelligence aggregator. Initially the server IP was not identified as malicious. A few days later, however, it had been flagged by multiple security vendors (Antiy-AVL, Criminal IP, and SOCRadar), according to the *sysadmin* in charge of the data capture. C-Cert's results, combined with an expert's analysis, allowed for early detection of this malicious activity, which took place over TLS 1.3. Further details can be found in Appendix C.

7 DISCUSSION

We now discuss the performance of our inference methods and classifiers on the curl captures and UCNet dataset.

Inference performance. For certificate size inference, we discount 132 bytes only when the multiple handshake messages cannot be externally isolated, as discussed in Appendix B. This is a conservative estimate based on our curl captures, close to a minimum of 126 bytes. It allows us to obtain the correct certificate size for more than 50% of the curl TLS 1.3 sessions, within a small error margin ($\leqslant$27 bytes) for over 80% of those sessions, with an average error of 1.0%. The mean/maximum error observed was 34.33/467 bytes, with inferred sizes always equal to or greater than actual certificate sizes. As errors inflate sizes, they usually lead to false negatives.

The range of 60 to 600 bytes for records carrying NSTs also stems from our curl capture: after discarding Handshake and Close Notify messages, 76.67% of TLS 1.3 flows still had extra records to be removed. On average, these flows had 1.42 extra records each, with a size of 391.48 bytes/record. The NST record discarding methodology was correct for 79.7% of flows, with the remaining 13.8%/6.5% having too many/not enough records removed, respectively. On average, 1.08/1.53 too many/few records were removed. Errors in this procedure will cause issues in behavior-based classification. However, not using this *best-effort* approach would, on average, cause a larger drop in classification performance.

UCNet results discussion. Looking at C-Cert's predictions, some flows were confirmed to be actual false positives, and blocking them would likely have caused disruption to legitimate services. Nonetheless, legitimate domains and IP addresses can be added to an "allowlist" to prevent unnecessary alerts. Given the non-uniform distribution of positive flows per IP addresses and SNIs, a succinct allowlist could have substantial impact on the number of false alerts.

C-Behav also flagged as positive many UCNet flows belonging to non-malicious apps that were not appropriately represented in our training dataset. Improved results are expected when training on some of the target network's traffic. In particular, the "malware incident" was not flagged by C-Behav, as its behavior was not consistent with the trained patterns – our C2 ground truth mostly started with application data records in the C2S direction, while most of this malware's flows started with a S2C application record.

7.1 Limitations

Errors and failures. Protocol errors or handshake failures lead to connection termination, but TLS records exchanged until then can still be processed/used for classification. Servers may also send a "Hello Retry Request" when more information is needed to proceed with the handshake. The client will then send a new Client Hello on the same connection. Due to limitations in our network capture processing software – which only extracts metadata from the first Client Hello, ignoring the second one – we discard flows with "Hello Retry Request" messages. This issue stems from a limitation on the network capture processing, not from the proposed methodology.

Lost packets. TCP segments sent/received by the communicating parties but unseen by the capture process may cause some of the flow's TLS records to be lost, negatively affecting classification. This is an engineering issue that we do not address.

Deviation from TLS specification. The software we use may fail to process flows from non-standard TLS implementations. These might get flagged by non-TLS-specific traffic anomaly detectors [15].

Padding. TLS record padding [53] is the most direct countermeasure to our approach, as it masks the handshake process by hiding the size of the server's handshake messages, and/or introducing artificial variance in the sent application data. Padding handshake records is expected to reduce the efficacy of C-Cert. In turn, padding application data records would disrupt C-Behav.

Randomly padding messages would be a countermeasure to C-Behav as is, but one might be able to train the classifier with a richer ground truth to consider this variation. As a more sophisticated alternative, one might use padding to mimic the behavior of popular applications and evade classifier adaptation. This should be effective, albeit much more complex to implement by an adversary. One should also keep in mind that padding detection via statistical analysis is possible [24, 60], and in encrypted traffic analysis, padding countermeasures have been shown to be avoidable altogether [14]. We believe that, while padding is an effective C2 adaptation to the current version of the classifiers, countermeasures against said padding are also a reasonable future optimization.

PSK-based authentication. The usage of PSK-based authentication can be detected by outside observers, such as IDSs. A PSK, established out of band or issued by the server in a previous connection, allows for skipping most of the encrypted handshake. Server certificate size inference would make no sense, but application behavior extraction should be largely unaffected. If the flow carrying the initial session was also observed, perhaps both could be correlated; otherwise, sessions using PSKs could be blocked altogether, avoiding the potential risks of a weaker handshake analysis at the expense of additional overhead, or potential disruption.

Client authentication. Certificate-based client authentication can be performed during the handshake, or afterwards using Post-Handshake Messages. Authenticating during handshake increases the size of S2C handshake messages (due to the "Certificate Request" message), increasing the error for server certificate size inference. A larger client response (which then contains "Certificate" and "Certificate Verify" messages) can be used as an indicator of client authentication usage, triggering adjustments for the server certificate size inference. Post-Handshake Authentication can be performed at any time after the "Finished" messages, and would be more challenging to isolate from application behavior. However, given that this feature was not available in previous TLS versions, using it requires deliberate adaptation.

8 RELATED WORK

Traffic fingerprinting. TLS fingerprinting aims to identify applications or libraries by comparing their observable behavior with well-known fingerprints [31]. Shen et al. [56] propose traffic classification for TLS 1.2 flows based on lengths of the certificate and the first application data packet combined, using clustering and Markov chains. C-BEHAV expands on this insight, considering instead multiple application data packet lengths, but follows a different classification methodology. Exploring the use of Markov chains and clustering of n-grams of application data length is an interesting future work, as an alternative to C-BEHAV. Regarding TLS 1.3, Anderson [4] suggests that fingerprinting will still be effective in a foreseeable future, due to its backwards compatibility with TLS 1.2. A frequently used method for TLS fingerprinting is JA3/JA3S [3], which considers metadata such as TLS versions, cipher suites and extensions. Alternatively, fingerprinting may also be applied to encrypted traffic, being typically based on machine learning and using TLS flows as classification objects [2, 48]. Mavroudis and Hayes [44] show that one can infer the specific webpage being accessed by a user on a website over TLS 1.3, given a previously collected labeled set of TLS 1.3 webpage captures. Recently, Xue et al. [62] demonstrated fingerprinting in a protocol-agnostic manner. Their techniques are complementary to C-Behav, suggesting novel methodologies for characterizing our isolated application behavior.

Malware TLS traffic. To detect the usage of TLS by malware, several works [7, 8, 50] propose machine learning algorithms and features including sequences of packet lengths and packet inter-arrival times, and TLS metadata. In 2018, malware's TLS usage had been found to be distinct from benign usage regarding TLS metadata and parameters, e.g., extensions, key lengths [9]. A recent study on TLS-based malware detection [34] found 10 approaches using server certificate analysis for detection. However, these methods cannot be applied to TLS 1.3- or ECH-using flows. Additional approaches [25, 42] do not consider stream attributes that are blocked by TLS 1.3. Our work shows that separating these attributes in TLS 1.3 flows is not trivial, and propose complementary methods to these classifiers towards efficient malware detection on TLS 1.3 flows.

Specifically for TLS 1.3 flows, Lin et al. [41] propose a classification method capable of dealing with TLS 1.3 traffic. However, they use a small evaluation dataset that contains mostly unencrypted or non-TLS traffic. Gomez et al. [29] study clustering-based detection of malicious TLS flows, considering TLS 1.3. The authors follow an unsupervised approach, while we follow a supervised approach instead. However, given that their feature extraction and classification methodology are closed source, and they used proprietary datasets for validation, we cannot assess the efficacy of their approach on our datasets, or compare our classifiers in their experimental setting. Our early experiments with clustering-based detection suggest that our methods for certificate inference and behavior extraction can also be used as a complement to unsupervised methodologies.

Anderson and McGrew [8] are among the makers of the Joy tool used for feature extraction, which also has classification capabilities. However, an early evaluation of Joy's classification on our data yielded metrics lagging behind the approach followed in our paper: almost no MS traffic detected in conditions similar to C-Basic (TPR under 1%), and many false positives for DoHBrw (TNR of 87.16%).

9 ETHICAL CONSIDERATIONS

Responsible data collection. Our "in-house" datasets were captured in controlled environments without real users. For UCNet, special care was taken to protect the privacy of campus users. Traffic classification is performed on anonymized data with no identifiable/personal information. Permission for data capture and classification was obtained from the campus Department of Informatics Services. The entire data collection, anonymization and treatment occurs on two restricted-access campus machines. Only an authorized subset of IT staff can access these machines. Source and destination IP addresses are anonymized using Joy, and raw data always stays within the campus network. The collected data was solely used for validating our previously trained classifiers. Disclosed SNIs are publicly accessible, within top-level domains. Private domains from non-propagated DNS zones are not disclosed.

Risk of malicious certificate generation. The certificate manipulation we introduced in Metasploit may bypass simple (or poorly-implemented) detectors based on certificate size or certificate chain length. We argue that certificate-based detection should be used as additional information, but not as a network's sole defense.

Risk of tracking usage. Our classification methodology exploits patterns revealed by the TLS 1.3 communication protocol. While similar approaches have been exploited in the past for tracking or surveillance purposes [53] (E.3), our techniques are mostly focused on distinguishing malicious from benign traffic through the inference of self-signed certificates' characteristics, and we do not attempt to infer other latent information (e.g., user browsing habits).

10 CONCLUSION

We propose methods for certificate size inference and application behavior isolation on TLS 1.3 encrypted flows. Our results suggest the feasibility of these methods, and their usefulness for malware C2 traffic classification. The proposed classifiers achieve TPRs of 100% and 94% with certificate- and behavior-based detection respectively. We validate that these classifiers do not sacrifice TNR, both showcasing values of roughly 99% on existing datasets. Furthermore, the same classifiers achieve values of 99.15% and 96.35% on a real-world dataset collected on a university campus, respectively. Additionally, these approaches for information extraction in TLS 1.3 flows can also be used as a building block for other traffic analysis tasks, such as stream- or graph-based network analysis.

ACKNOWLEDGMENTS

We express our gratitude to our shepherd and the anonymous reviewers for their insightful comments. Special thanks to Alexandre Francisco, head of Network and Systems Administration at IST, and his team members Rodrigo Cunha and Jorge Matias, for their invaluable cooperation and assistance in collecting the UCNet dataset. This work was supported by Fundação para a Ciência e Tecnologia (FCT) grants UIDB/50021/2020 and 2021.08532.BD, by IAPMEI grant C6632206063-00466847 (SmartRetail), by NSERC grant RGPIN-2023-03304, and co-financed by Component 5 of core funding for Technology and Innovation Centres (CTI), integrated in the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021 - 2026.

REFERENCES

[1] abuse.ch. 2023. SSL Blacklist (SSLBL). https://sslbl.abuse.ch/

[2] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2021. A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 1 (Feb. 2021), 1–26. https://doi.org/10.1145/3447382

[3] John Althouse. 2019. TLS Fingerprinting with JA3 and JA3S. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967/

[4] Blake Anderson. 2019. TLS Fingerprinting in the Real World. https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world

[5] Blake Anderson, Andrew Chi, Scott Dunlop, and David McGrew. 2019. Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy* (Richardson, Texas, USA) *(CODASPY '19)*. Association for Computing Machinery, New York, NY, USA, 267–278. https://doi.org/10.1145/3292006.3300025

[6] Blake Anderson and David McGrew. 2016. Identifying Encrypted Malware Traffic with Contextual Flow Data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, Vienna Austria, 35–46. https://doi.org/10.1145/2996758.2996768

[7] Blake Anderson and David McGrew. 2016. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*. 35–46.

[8] Blake Anderson and David McGrew. 2017. Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1723–1732. https://doi.org/10.1145/3097983.3098163

[9] Blake Anderson, Subharthi Paul, and David McGrew. 2018. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques* 14, 3 (Aug. 2018), 195–211. https://doi.org/10.1007/s11416-017-0306-6

[10] Stefan Axelsson. 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)* 3, 3 (2000), 186–205.

[11] Ofek Bader, Adi Lichy, Chen Hajaj, Ran Dubin, and Amit Dvir. 2022. MalDIST: From Encrypted Traffic Classification to Malware Traffic Detection and Classification. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE Press, Las Vegas, NV, USA, 527–533. https://doi.org/10.1109/CCNC49033.2022.9700625

[12] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[13] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees*. Taylor & Francis.

[14] Jonas Bushart and Christian Rossow. 2020. Padding ain't enough: Assessing the privacy guarantees of encrypted DNS. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*.

[15] Brian Caswell, Jay Beale, and Andrew Baker. 2007. *Snort intrusion detection and prevention toolkit*. Syngress.

[16] Minhaz Chowdhury, Nafiz Rifat, Shadman Latif, Mostofa Ahsan, Md Saifur Rahman, and Rahul Gomes. 2023. ChatGPT: The Curious Case of Attack Vectors' Supply Chain Management Improvement. In *2023 IEEE International Conference on Electro Information Technology (eIT)*. IEEE, 499–504.

[17] Cisco Systems. 2019. cisco/joy: A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring. https://github.com/cisco/joy

[18] Cloudflare. 2024. 1.1.1.1 — The free app that makes your Internet faster. https://one.one.one.one/family/

[19] curl. 2023. command line tool and library for transferring data with URLs. https://curl.se/

[20] Christian J. Dietrich and Christian Rossow. 2009. *Empirical research of IP blacklists*. Vieweg+Teubner, Wiesbaden, 163–171. https://doi.org/10.1007/978-3-8348-9283-6_17

[21] Brad Duncan. 2024. Malware-Traffic-Analysis.net. https://malware-traffic-analysis.net

[22] Yebo Feng, Jun Li, and Devkishen Sisodia. 2022. Cj-sniffer: Measurement and content-agnostic detection of cryptojacking traffic. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*. 482–494.

[23] Fraunhofer FKIE. 2023. Meterpreter (malware family) – malpedia. https://malpedia.caad.fkie.fraunhofer.de/details/win.meterpreter

[24] Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. 2003. On effectiveness of link padding for statistical traffic analysis attacks. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings*. IEEE, 340–347.

[25] Zhuoqun Fu, Mingxuan Liu, Yue Qin, Jia Zhang, Yuan Zou, Qilei Yin, Qi Li, and Haixin Duan. 2022. Encrypted malware traffic detection via graph-based network analysis. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*. 495–509.

[26] Sean Gallagher. 2021. Nearly half of malware now use TLS to conceal communications. https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/

[27] Ibrahim Ghafir, Vaclav Prenosil, Mohammad Hammoudeh, Liangxiu Han, and Umar Raza. 2017. Malicious SSL Certificate Detection: A Step Towards Advanced Persistent Threat Defence. In *Proceedings of the International Conference on Future Networks and Distributed Systems* (Cambridge, United Kingdom) *(ICFNDS '17)*. Association for Computing Machinery, New York, NY, USA, Article 27. https://doi.org/10.1145/3102304.3102331

[28] Alessandro Ghedini and Victor Vasiliev. 2020. TLS Certificate Compression. RFC 8879. https://doi.org/10.17487/RFC8879

[29] Gibran Gomez, Platon Kotzias, Matteo Dell'Amico, Leyla Bilge, Juan Caballero, and Vincenzo Conti. 2023. Unsupervised Detection and Clustering of Malicious TLS Flows. *Security and Communication Networks* 2023 (jan 2023), 17 pages. https://doi.org/10.1155/2023/3676692

[30] Joonseo Ha and Heejun Roh. 2021. Experimental Evaluation of Malware Family Classification Methods from Sequential Information of TLS-Encrypted Traffic. *Electronics* 10, 24 (Jan. 2021), 3180. https://doi.org/10.3390/electronics10243180 Number: 24 Publisher: Multidisciplinary Digital Publishing Institute.

[31] Habdul Hazeez. 2022. What is TLS fingerprinting? https://fingerprint.com/blog/what-is-tls-fingerprinting-transport-layer-security/

[32] Ralph Holz, Johanna Amann, Abbas Razaghpanah, and Narseo Vallina-Rodriguez. 2019. The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods. arXiv:1907.12762 http://arxiv.org/abs/1907.12762

[33] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. 2020. Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. *ACM SIGCOMM Computer Communication Review* 50, 3 (July 2020), 3–15. https://doi.org/10.1145/3411740.3411742

[34] Kinan Keshkeh, Aman Jantan, Kamal Alieyan, and Usman Mohammed Gana. 2021. A Review on TLS Encryption Malware Detection: TLS Features, Machine Learning Usage, and Future Directions. In *Advances in Cyber Security*, Nibras Abdullah, Selvakumar Manickam, and Mohammed Anbar (Eds.). Vol. 1487. Springer Singapore, Singapore, 213–229. https://doi.org/10.1007/978-981-16-8059-5_13 Series Title: Communications in Computer and Information Science.

[35] WatchGuard's Threat Lab. 2023. Internet Security Report: Q1 2023. https://www.watchguard.com/wgrd-resource-center/security-report-q1-2023

[36] WatchGuard's Threat Lab. 2024. Internet Security Report: Q4 2023. https://www.watchguard.com/wgrd-resource-center/security-report-q4-2023

[37] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2024. Tranco list with ID 662LX. https://tranco-list.eu/list/662LX

[38] Ivan Letteri, Giuseppe Della Penna, Luca Di Vita, and Maria Teresa Grifa. 2020. MTA-KDD'19: A Dataset for Malware Traffic Detection.. In *Itasec*. CEUR, Ancona, 153–165.

[39] Adi Lichy, Ofek Bader, Ran Dubin, Amit Dvir, and Chen Hajaj. 2023. When a RF beats a CNN and GRU, together—A comparison of deep learning and classical machine learning approaches for encrypted malware traffic classification. *Computers & Security* 124 (Jan. 2023), 103000. https://doi.org/10.1016/j.cose.2022.103000

[40] Chronicle Security Ireland Limited. 2023. VirusTotal. https://www.virustotal.com/

[41] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini (Eds.). ACM, Austin, Texas, 633–642. https://doi.org/10.1145/3485447.3512217

[42] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 1171–1179.

[43] Mandiant. 2023. M-Trends reports. https://www.mandiant.com/m-trends

[44] Vasilios Mavroudis and Jamie Hayes. 2023. Adaptive Webpage Fingerprinting from TLS Traces. arXiv:2010.10294 [cs.CR]

[45] David McGrew and Blake Anderson. 2016. Enhanced telemetry for encrypted threat analytics. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, Singapore, 1–6. https://doi.org/10.1109/ICNP.2016.7785325

[46] Mohammadreza MontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari. 2020. Detection of DoH Tunnels using Time-series Classification of Encrypted Traffic. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, online, 63–70. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00026

[47] Carlos Novo. 2024. CarlosANovo/extending12to13: Code for "Extending C2 Traffic Detection Methodologies: From TLS 1.2 to TLS 1.3-enabled Malware". https://github.com/CarlosANovo/extending12to13

[48] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *Comput. Surveys* 54, 6 (July 2021), 1–35. https://doi.org/10.1145/3457904

[49] Christopher Patton. 2020. Good-bye ESNI, hello ECH! http://blog.cloudflare.com/encrypted-client-hello/

[50] Paul Prasse, Gerrit Gruben, Lukas Machlika, Tomas Pevny, Michal Sofka, and Tobias Scheffer. 2017. Malware Detection by HTTPS Traffic Analysis. (2017), 10 pages.

[51] Rapid7. 2023. cert_provider.rb – Metasploit framework (source code). https://github.com/rapid7/metasploit-framework/blob/6.3.17/lib/msf/core/cert_provider.rb#L41

[52] Rapid7. 2023. metasploit-payloads. https://github.com/rapid7/metasploit-payloads/tree/master/python/meterpreter

[53] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[54] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2023. *TLS Encrypted Client Hello.* Internet-Draft draft-ietf-tls-esni-16. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-tls-esni/16/ Work in Progress.

[55] scikit-learn developers. 2024. sklearn.model_selection.StratifiedGroupKFold. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedGroupKFold.html

[56] Meng Shen, Mingwei Wei, Liehuang Zhu, and Mingzhong Wang. 2017. Classification of Encrypted Traffic With Second-Order Markov Chains and Application Attribute Bigrams. *IEEE Transactions on Information Forensics and Security* 12, 8 (Aug. 2017), 1830–1843. https://doi.org/10.1109/TIFS.2017.2692682 Conference Name: IEEE Transactions on Information Forensics and Security.

[57] sitespeed.io. 2023. Browsertime. https://github.com/sitespeedio/browsertime

[58] Brian Trammell and Elisa Boschi. 2008. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103. https://doi.org/10.17487/RFC5103

[59] Johannes B. Ullrich. 2022. Encrypted Client Hello: Anybody Using it Yet? https://isc.sans.edu/diary/Encrypted+Client+Hello%3A+Anybody+Using+it+Yet%3F/28792

[60] Wei Wang, Mehul Motani, and Vikram Srinivasan. 2008. Dependent link padding algorithms for low latency anonymity systems. In *Proceedings of the 15th ACM conference on Computer and communications security.* 323–332.

[61] Konrad Wolsing, Eric Wagner, Antoine Saillard, and Martin Henze. 2022. IPAL: breaking up silos of protocol-dependent and domain-specific industrial intrusion detection systems. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses.* 510–525.

[62] Diwen Xue, Michalis Kallitsis, Amir Houmansadr, and Roya Ensafi. [n. d.]. Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes. ([n. d.]).

A ADDITIONAL UCNET DETAILS

Figure 5 shows the traffic observed throughout the entire month (on the left) and on a specific day, March 12 (on the right) on the university campus network where the UCNet dataset was captured. One of the capture periods is highlighted on the right. The figure presents the IPv4 traffic volume and the total traffic volume (also including IPv6). Over the month, IPv4 traffic constituted 72.93% compared to 27.07% for IPv6, with IPv4 throughput peaking at 1.3 times that of IPv6. This IPv4 predominance is largely due to the prevalence of wireless traffic, which mainly uses IPv4 addresses. The total traffic volume for this period was 868.82TB, with 633.60TB being IPv4. On a single day, IPv4 traffic accounted for 68.81%, and IPv6 for 31.19%. The total traffic volume for that day was 36.05TB, with 24.81TB attributed to IPv4.

For wireless traffic, we gathered data on the number of active clients and the generated traffic volume: Throughout the month, the average number of connected clients was 1,111, with a peak of 5,273. The traffic volume for this period amounted to 1.25TB. On the weekends, the average number of clients dropped to about 500. On March 12, the average was 1,864 connected clients, with a peak of 5,221 at 3:30 p.m. The highest concentration of connected clients was observed between 2 p.m. and 5 p.m., aligning with our data collection times.

B CERTIFICATE INFERENCE PROCEDURE

Figure 6 depicts the server certificate size inference procedure as a diagram. In it, `PHS` (**P**resumable **H**andshake records' **S**izes) is a list of record sizes (positive integers). These sizes correspond to TLS records with an outer/opaque type of *Application Data* (type 23, as defined in [53], section 5.2) sent in the S2C direction, before any C2S application data records are seen. One `PHS` sequence is extracted from each TLS session.

The `Len()` function takes a list and returns its number of elements – `Len(PHS)` corresponds to the number of records fitting our previous criteria. `PHS[i]` stands for the i^{th} element (starting from 0) of the `PHS` list. `Sum()` takes a list and returns the sum of its every element, and `Max()` takes a list and returns the value of its largest element. The 'discard' operations update the `PHS` list, removing elements from it. A possible `PHS` could be `[120,3000,200,53,10]` in which case `Len(PHS)==5` and `PHS[0]==120`; discarding records with size 53 and everything after results in `PHS==[120,3000,200]` and a new `Len(PHS)==3`; then, `Max(PHS)==3000` and the return value for our inference process would be 2975.

C SEEMINGLY MALICIOUS SNIS

Unfortunately, IP addresses cannot be revealed, but permission has been granted to reveal some of the seemingly malicious SNIs:

- `www[.]657slky5koy37mdi4zsr[.]com`
- `www[.]qrhiw2rpj7llszas3adqr[.]com`
- `www[.]invclnriexetkhj2fhn[.]com`
- `www[.]ccm74fubv7o647avj27[.]com`
- `www[.]n4s7bgnu53yio6ecmhoy[.]com`
- `www[.]prrvbot4oqggr[.]com`

These domains are public, and do not seem to have been registered, further highlighting the suspicious behavior.

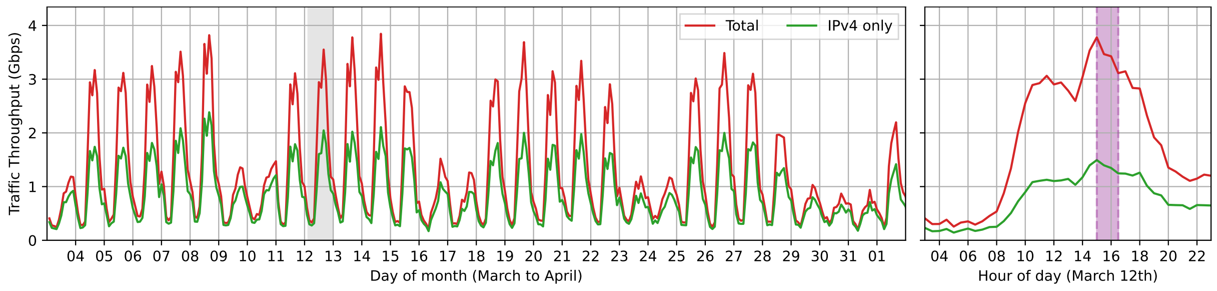

Figure 5: Graph of the monthly traffic throughput (left), and a zoom-in on a specific day (right). The zoomed-in area is highlighted in gray on the left. Highlighted in purple on the right is the time during which a daily traffic capture took place.

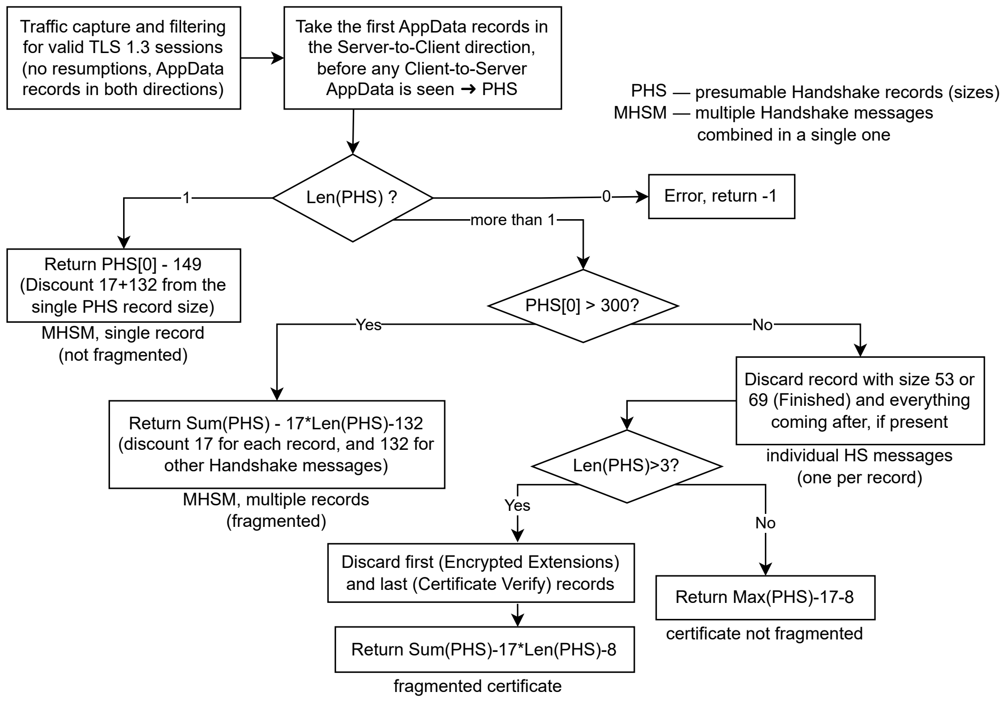
Traffic capture and filtering for valid TLS 1.3 sessions (no resumptions, AppData records in both directions)
Take the first AppData records in the Server-to-Client direction, before any Client-to-Server AppData is seen ➜ PHS
PHS — presumable Handshake records (sizes)
MHSM — multiple Handshake messages combined in a single one
Len(PHS) ?
1
0
more than 1
Error, return -1
Return PHS[0] - 149 (Discount 17+132 from the single PHS record size)
MHSM, single record (not fragmented)
PHS[0] > 300?
Yes
No
Return Sum(PHS) - 17*Len(PHS)-132 (discount 17 for each record, and 132 for other Handshake messages)
MHSM, multiple records (fragmented)
Discard record with size 53 or 69 (Finished) and everything coming after, if present
individual HS messages (one per record)
Len(PHS)>3?
Yes
No
Discard first (Encrypted Extensions) and last (Certificate Verify) records
Return Sum(PHS)-17*Len(PHS)-8
fragmented certificate
Return Max(PHS)-17-8
certificate not fragmented

Figure 6: Certificate size inference process illustrated as a diagram.