Hindawi
Security and Communication Networks
Volume 2023, Article ID 3676692, 17 pages
https://doi.org/10.1155/2023/3676692

Research Article

Unsupervised Detection and Clustering of Malicious TLS Flows

Gibran Gomez,[1] **Platon Kotzias**,[2] **Matteo Dell'Amico**,[3] **Leyla Bilge**,[2] **and Juan Caballero**[4]

[1]*IMDEA Software Institute, Universidad Politécnica de Madrid, Madrid, Spain*
[2]*Norton Research Group, Paris, France*
[3]*University of Genoa, Genoa, Italy*
[4]*IMDEA Software Institute, Madrid, Spain*

Correspondence should be addressed to Gibran Gomez; gibran.gomez@imdea.org

Received 25 October 2021; Revised 15 December 2022; Accepted 21 December 2022; Published 12 January 2023

Academic Editor: Vincenzo Conti

Malware abuses TLS to encrypt its malicious traffic, preventing examination by content signatures and deep packet inspection. Network detection of malicious TLS flows is important, but it is a challenging problem. Prior works have proposed supervised machine learning detectors using TLS features. However, by trying to represent all malicious traffic, supervised binary detectors produce models that are too loose, thus introducing errors. Furthermore, they do not distinguish flows generated by different malware. On the other hand, supervised multiclass detectors produce tighter models and can classify flows by the malware family but require family labels, which are not available for many samples. To address these limitations, this work proposes a novel unsupervised approach to detect and cluster malicious TLS flows. Our approach takes input network traces from sandboxes. It clusters similar TLS flows using 90 features that capture properties of the TLS client, TLS server, certificate, and encrypted payload and uses the clusters to build an unsupervised detector that can assign a malicious flow to the cluster it belongs to, or determine if it is benign. We evaluate our approach using 972K traces from a commercial sandbox and 35M TLS flows from a research network. Our clustering shows very high precision and recall with an $F1$ score of 0.993. We compare our unsupervised detector with two state-of-the-art approaches, showing that it outperforms both. The false detection rate of our detector is 0.032% measured over four months of traffic.

1. Introduction

Transport layer security (TLS) is the most popular cryptographic protocol, widely used to provide confidentiality, integrity, and authentication to applications such as Web browsing, email, and instant messaging [1, 2]. Its security properties and wide availability make TLS also appealing to malware, which can abuse it to hide its malicious traffic among benign traffic of a myriad of applications while preventing examination of its application payload. In February 2020, 23% of malware was using TLS [3].

Network detection of malicious TLS flows is important, but it is a challenging problem. It allows us to protect whole networks by monitoring their traffic, regardless of whether endpoint security has been deployed. An important property of scalability and privacy is that network detection should not require decrypting the content; i.e., it should avoid man-in-the-middle (MITM) interception [4]. Another important property is to cluster detected flows with other similar malicious flows, providing valuable threat intelligence to the analysts that investigate detection.

Anderson and McGrew proposed supervised machine learning (ML) detectors for malicious TLS flows [5–7]. A limitation of binary-supervised detectors (e.g., [7]) is that they try to distinguish any malicious TLS flow, regardless of the malware family producing it. This is problematic because different families may exhibit significant differences in TLS usage (e.g., TLS versions, cipher suites, extensions, and certificates). To cover all those differences, the generated model tends to become too loose, thus introducing errors. In addition, a binary detector does not provide contextual information about similar malicious flows. In follow-up work, they have also proposed combining TLS features with HTTP and DNS features to improve the binary detection

model [5]. However, some families may only use TLS (e.g., HTTPS, but no HTTP) and may connect using IP addresses instead of domain names. Furthermore, HTTP and DNS features may contain sensitive user information, thus lowering user privacy. In their original work, Anderson et al. also evaluated a multiclass-supervised classifier, where each class corresponds to a different malware family [7]. This approach better models the TLS traffic of individual families and classifies detected flows. However, it requires clean family labels to train classifiers. A common labeling method is a vote on the families present in the AV detection labels of a sample [8]; unfortunately, this approach is ineffective for many samples. As a concrete example, Anderson et al. [7] could only label this way 27% of 20.5K samples using TLS; no classifier could be built for the families of the 73% unlabeled samples, and thus, their malicious traffic could not be detected. Recent works have proposed anomaly based intrusion detection systems using neural network autoencoders [9, 10]. These works do not specifically target TLS flows but can be applied to them as they do not require access to unencrypted payload. These type of models are built on benign traffic, so they do not require labeled malicious traffic during training. But they suffer false positives when the benign traffic changes greatly.

In this paper, we present a novel unsupervised approach to detect and cluster malicious TLS flows. Our approach respects user privacy as it only requires access to encrypted TLS flows and not to their unencrypted payload or any other unencrypted traffic. Our approach takes input traces of network traffic generated by executing suspicious samples in a sandbox. From each TLS flow in traces, it extracts 90 features that capture characteristics of the TLS client, TLS server, server's certificate, and encrypted payload. After filtering benign TLS traffic, the remaining vectors are clustered so that each cluster contains similar traffic belonging to a (potentially unknown) malware family. Since malware families may use different types of traffic (e.g., *C* & *C* and updates from the download server), multiple clusters can be output for a family. The use of clustering removes the requirement for family labels as it allows detecting any malicious traffic similar to training flows, even if training flows could not be annotated with a known family name, i.e., flows from the 73% samples that Anderson et al. [7] had to remove from their training. The clusters are input to the unsupervised detector that can be deployed at the boundary of a network to identify malicious TLS flows. The detector measures the distance of a given flow to all clusters and outputs the closest cluster. If no cluster is close, the flow is determined to be benign. If family labels are available for the identified cluster, the analyst also obtains the family of the detected flow. When family labels are not available, detection is not affected: the flow is associated with the random identifier of its cluster but still provides contextual information about samples generating similar flows.

To evaluate our approach, we use 972K network traces provided by a commercial sandbox vendor and 35M TLS flows collected at the boundary of a research network over seven months. We identify, for the first time, how the sandbox can significantly impact the collected TLS traffic if it runs old OS versions (e.g., Windows 7 and Windows XP). Those operating systems use TLS 1.0 by default instead of currently dominating TLS 1.2 and 1.3 versions. Thus, training a classifier with malicious traffic from a single sandbox using an old OS could incorrectly capture that TLS 1.0 traffic is malicious and TLS 1.2 and 1.3 are benign. This is an important finding because popular sandboxes (e.g., VirusTotal [11]) run decade-old OS versions, and a common belief is that the lack of newer OS defenses makes it easier for malware to run and manifest its behaviors. Our results highlight the importance of using a variety of OS versions in malware sandboxes. Furthermore, this issue could have impacted previous work that identified significant differences between malware and benign program TLS client characteristics (e.g., malware using older cipher suites and fewer extensions) [5, 7], as those same authors have recently concluded that TLS client features are not enough by themselves [1], in contrast with their earlier claims.

Our clustering achieves an $F1$ score of 0.993. We observe that 31% of the produced clusters only contain samples for which the state-of-the-art AVclass labeling tool [8] is not able to obtain family names: supervised multiclass approaches would not work for those samples. We also observe that our clustering is able to group TLS 1.3 flows from multiple samples of the same family, even when no server name indication (SNI) header is present. TLS 1.3 is a challenging case, not evaluated in prior works, as certificates are encrypted and client and server features are greatly reduced.

We compare our unsupervised detector with the two state-of-the-art approaches. Our comparison with Joy [12], the state-of-the-art supervised binary detector by Anderson et al. [7], shows that when applied to the same dataset, our approach achieves an $F1$ score of 0.91 compared to 0.82 for Joy. We also compare our approach with Kitsune [10], the state-of-the-art autoencoder-based anomaly detector. Our approach achieves an $F1$ score of 0.99 compared to 0.59 for Kitsune. We also evaluate our detector over long windows of time to estimate its false detection ratio (FDR). Over one week of traffic from the research network, our approach achieves an FDR of 0.031%. Over four months, the FDR remains almost the same at 0.032%, highlighting the stability of the detection model.

This paper provides the following contributions:

(i) We present a novel unsupervised approach to detect and cluster malicious TLS flows. Compared to binary-supervised detectors, our clustering approach models separately TLS characteristics from different families. This results in tighter models that improve detection and can provide contextual information about the cluster to which a detected flow belongs to. Compared to multiclass classifiers, our approach can detect samples for which family labels are not available.

(ii) We observe that training malicious TLS detectors on traces from a single sandbox that uses old OS versions can significantly bias the detector. This highlights the importance of using a variety of OS versions in malware sandboxes.

(iii) We evaluate our approach using 972K network traces from a commercial sandbox and 35M TLS flows from a research network. Our unsupervised detector achieves an $F1$ score of 0.91, compared to 0.82 for the state-of-the-art supervised detector, and an FDR of 0.032% over four months of traffic.

The remainder of this paper is organized as follows: Section 2 motivates our research problem. Section 3 details our novel unsupervised approach to detect and cluster malicious TLS flows. Section 4 describes the datasets used. Section 5 evaluates our approach and compares it with state-of-the-art approaches. Section 6 presents prior related work. Section 7 discusses limitations and avenues for improvement. Finally, Section 8 concludes. For the reader's benefit, Table 1 details the acronyms used in this work.

Table 1: Acronyms used in the paper.

Acronym	Full name
AE	Autoencoder
ALPN	Application layer protocol negotation TLS extension
API	Application programming interface
APT28	Fancy Bear Russian cyber espionage group
C&C	Command-and-control
CA	Certification authority
CDN	Content delivery network
CT	Certificate transparency
DHCP	Dynamic host Configuration protocol
DN	Distinguished name
DNS	Domain name system
e2LD	Effective second-level domain
FDR	False detection ratio
FN	False negative
FP	False positive
HDBSCAN	Hierarchical density-based spatial clustering of applications with noise
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol secure
IP	Internet protocol
Mcs	Minimum cluster size
MITM	Man-in-the-middle
ML	Machine learning
OS	Operating system
PUP	Potentially unwanted programs
RMSE	Root mean square error
RRP	Request-response pair
SB	Sandbox
SAN	Subject alternative name TLS extension
SNI	Server name indication TLS extension
SSL	Secure socket layer
TF-IDF	Term frequency-inverse document frequency
TCP	Transmission control protocol
TLS	Transport layer security protocol
TP	True positive
URI	Uniform resource identifier
VT	Virus Total

2. Motivation

Our goal is to detect malicious TLS flows (TLS sessions) between an infected host in a protected network, e.g., an enterprise or a university network, and a remote malicious server. A prerequisite is that, to respect user privacy, only TLS flows are accessible from the protected network. This implies that the application payload of the TLS flows should not be accessed, i.e., no MITM and that unencrypted traffic should not be needed for detection. Thus, detection features should exclusively come from TLS flows.

Our intuition is that it is possible to capture TLS characteristics of a specific type of malware family traffic (e.g., *C* & *C* and updates), but it is very hard to capture TLS characteristics that distinguish any malware using TLS from any benign application using TLS. For example, certificates and domains in SNI headers are clearly family specific. Similarly, encrypted payload features such as packet sizes are specific to protocols (e.g., *C* & *C* and update) used by each family [13].

Thus, rather than building a supervised binary classifier, we propose an unsupervised detector that clusters similar malicious TLS flows and then detects new TLS flows by measuring the distance of the clusters. A benefit of our clustering approach is that the model can assign detected flows to the cluster that led to detection. Some clusters will be labeled with a recognizable family name such as Upatre, Zbot, or Bublik. For others, the family may be unknown, but the cluster still provides important contextual information in the form of samples that generate similar TLS flows. In contrast with multiclass-supervised classifiers (e.g., [7]), family labels are not required for input samples so that detection can still work for a large fraction of samples that may miss them.

2.1. Ethical Considerations. Passive data collection performed at the research network was approved and performed according to the institutional policies. To protect user privacy, it covers only the collection of encrypted TLS flows and excludes personally identifiable information such as client IP addresses. Access to the data is limited to employees of the institution. This research made no attempt to decrypt TLS flows. Our goal is to enable the detection of malicious TLS traffic while maintaining user privacy.

3. Approach

Our approach takes input network traces produced by running suspicious executables in a sandbox. It outputs an unsupervised detector that can be deployed on a network to detect malicious TLS flows. The input network traces are annotated with the hash of the sample whose execution produced it. Our approach comprises four steps, as illustrated in Figure 1. Section 3.1 describes feature extraction that produces a feature vector for each TLS flow in the input network traces. Section 3.2 presents filtering that removes feature vectors corresponding to benign traffic. Section 3.3 details clustering, which groups similar feature vectors. Finally, Section 3.4 describes the unsupervised detector, which gives the feature vector of a previously unseen TLS flow and classifies it as malicious (with its corresponding cluster) or benign.

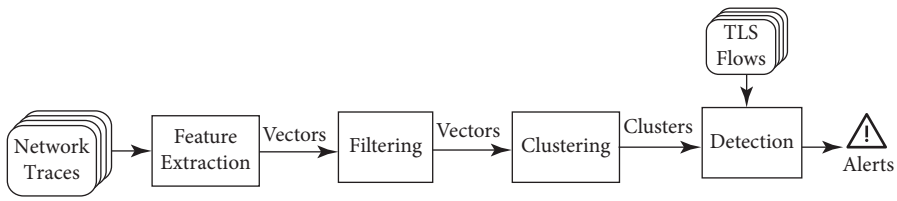

Figure 1: Approach architecture.

3.1. Feature Extraction. TLS fingerprints are applied to the first few payload bytes of each TCP connection to identify TLS flows [14], regardless of the ports used for communication. For the identified TLS flows, the TCP connection is reassembled, the full TCP payload is extracted, and then, the early part of the TCP payload corresponding to the TLS handshake is separated from the application data using the value of the content-type field of TLS records. TLS flows that have no application data in either direction are removed.

We extract 90 features from the remaining TLS flows. Of these, 67 features are new, while the other 23 features have been used in prior works. The features can be grouped into four categories. Client, server, and certificate features are extracted from the TLS handshake, while encrypted payload features are instead extracted from the encrypted application data. Features are either numerical or categorical. To build feature vectors, numerical features are normalized using their z-score, i.e., by subtracting the mean and dividing by the standard deviation. Categorical features are first applied to one-hot encoding, and the result is multiplied by the term frequency-inverse document frequency (TF-IDF) of values.

3.1.1. Client Features. These 11 features, summarized in Table 2, are extracted from the client hello message. They capture the functionality of the TLS client software. Programs either use the default configuration of a cryptographic library or OS and API or configure them with their preferences. Client features identify programs whose TLS functionality is configured similarly. The features correspond to the highest supported TLS and record versions, the list of supported TLS versions (extension added in TLS 1.3), the list of supported ciphers, compression methods, elliptic curves, and point formats, the list of extensions included in the message, the domain name in the SNI extension, and the list of supported application protocols in the application layer protocol negotiation (ALPN) extension (e.g., HTTP/0.9, SPDY/1). The fake resumption feature is explained later in the resumed sessions paragraph.

3.1.2. Server Features. The 9 server features in Table 3 correspond to the destination port and features extracted from the server hello message. These features capture the TLS functionality of the server software, i.e., the parameters that the server selects for the TLS session after intersecting the client TLS support with its own TLS support. Server features identify servers configured similarly. The server hello features are the selected TLS version, record versions, cipher, and compression method, the list of extensions in the message, the selected application protocol in the ALPN extension, and the lifetime in the session ticket extension. The last feature captures the signature in the signed certificate timestamp extension that a server may use to transmit signed proofs of the server's certificate presence in the certificate transparency (CT) logs [16].

Table 2: Client features. (R) = reduced. (C) = changed. (E) = could be encrypted.

Feature	Type	Prior work	TLS 1.2	TLS 1.3
c_version	Cat	[7]	✓	✓ (C)
c_record_version	Cat	✗	✓	✓
c_supported_versions	Cat	✗	✗	✓ (C)
c_ciphers	Cat	[7]	✓	✓ (R)
c_comp_methods	Cat	✗	✓	✗
c_curves	Cat	[1]	✓	✓ (R, E)
c_point_formats	Cat	[1]	✓	✓ (R, E)
c_extensions	Cat	[7]	✓	✓ (R, E)
c_server_name	Cat	[15]	✓	✓ (E)
c_alpn_list	Cat	✗	✓	✓ (E)
c_fake_resumption	Cat	✗	✓	✓ (C)

3.1.3. Certificate Features. These 24 features, summarized in Table 4, are extracted from the certificate chain sent by the server. These features capture the number of certificates in the chain and fields of the leaf certificate. The focus is on the leaf certificate because that certificate is specific to the service, while other certificates in the chain belonging to the certification authorities (CAs) are used and thus may be common to many unrelated services. Nine leaf certificate features correspond to certificate parameters, namely, the version, validity period, number of subject alternative names (SANs) and extensions included, validation status on the day we first process a flow, whether the certificate is self-signed, signature algorithm, public key length, and public key hash. Another seven features correspond to fields of the subject-distinguished name (DN), and the remaining seven correspond to the same fields in the issuer DN. In the rare case where client certificates are used, 22 additional analogous features are extracted from the client's certificate chain.

3.1.4. Encrypted Payload Features. Another 24 features, summarized in Table 5, are extracted from the encrypted application data transferred after the TLS handshake has been completed. These features capture the application protocol used for communication. The intuition is that the

Table 3: Server features.

Feature	Type	Prior work	TLS 1.2	TLS 1.3
s_dst_port	Cat	[7]	✓	✓
s_version	Cat	✗	✓	✓ (*C*)
s_record_version	Cat	✗	✓	✓
s_cipher	Cat	[7]	✓	✓ (*R*)
s_comp_method	Cat	✗	✓	✗
s_extensions	Cat	[7]	✓	✓ (*R*, *E*)
s_alpn_list	Cat	[1]	✓	✓ (*E*)
s_session_ticket_lifetime	Cat	[1]	✓	✓
s_ct_signature	Cat	✗	✓	✓ (*E*)

Table 4: Certificate features.

Feature	Type	Prior work	TLS 1.2	TLS 1.3
(c|s)_num_certs	Cat	[7]	✓	✗
(c|s)_leaf_cert_version	Cat	✗	✓	✗
(c|s)_leaf_cert_validity	Cat	[7]	✓	✗
(c|s)_leaf_cert_num_SAN	Cat	[7]	✓	✗
(c|s)_leaf_cert_ext_num	Cat	✗	✓	✗
s_leaf_cert_validation_status	Cat	✗	✓	✗
s_leaf_cert_self_signed	Cat	[7]	✓	✗
(c|s)_leaf_cert_sign_alg	Cat	[7]	✓	✗
(c|s)_leaf_cert_pubkey_hash	Cat	✗	✓	✗
(c|s)_leaf_cert_pubkey_size	Cat	[7]	✓	✗
(c|s)_leaf_cert_subj_cn	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_o	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_ou	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_c	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_st	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_l	Cat	✗	✓	✗
(c|s)_leaf_cert_subj_email	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_cn	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_o	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_ou	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_c	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_st	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_l	Cat	✗	✓	✗
(c|s)_leaf_cert_iss_email	Cat	✗	✓	✗

protocol changes infrequently because protocol updates require both client and server software updates and need to be thoroughly debugged to maintain compatibility. In particular, prior work has shown that the command and control (*C* & *C*) protocol used by malware changes much slower than its communication endpoints (i.e., domains, IP addresses, and ports) [17]. Similarly, we expect the protocol to also change less frequently than TLS configuration parameters.

Traffic analysis approaches often leverage the number, direction (i.e., client-to-server or server-to-client), and sizes of packets as features to identify encrypted content (e.g., [18, 19]). However, packets do not always accurately capture the underlying protocol because application messages can be fragmented into multiple packets by the transport and IP protocols. To address this issue, we define a sequence as all consecutive packets sent in one direction until another packet is sent in the opposite direction. The concatenation of the payload of all packets in a sequence is a good approximation of an application message [13]. The use of variable-length sequences avoids the need to select a threshold.

We call two consecutive sequences in the opposite direction a request-response pair (RRP). The number of RRPs to consider is a hyperparameter of payload features. The intuition to select this parameter is that the initial part of communication is more commonly related to the protocol, while later parts may be more related to the transferred content (e.g., files sent). In this work, we consider the first three RRPs, thus we analyze a total of six sequences from each flow. In our sandbox traces, we observe that 95% of TLS flows have a single RRP, 3% have two, and 2% have at least three RRPs.

For each sequence, two features are extracted: the size of the concatenated payload (e.g., msg_size_c_0) and the number of packets in the sequence (e.g., msg_pkts_c_0). These sequence features correspond to half of the 24 payload features. The other payload features correspond to the total byte size of the encrypted payload (and its split in sent and received bytes), the total number of packets (and its split in sent and received packets), the size of the larger sequence in each direction, the maximum number of packets in a sequence in each direction, the ratio of packets sent over packets received, and the ratio of bytes sent over bytes received.

In conclusion, payload features identify TLS flows with similar content, despite polymorphism of the domain and IP address. We have also experimented with timing features but found them too sensitive to the network setup (e.g., server location and congestion), and thus, we do not use them.

3.1.5. Resumed Sessions. TLS resumption allows us to quickly re-establish a prior TLS session using a shorter handshake [20]. The server encapsulates the session state into a ticket sent to the client. Later, the client can resume the previous session by sending the corresponding ticket to the server. The shorter handshake does not include client or server certificates. To avoid leaving certificate features empty (which may make resumed sessions look alike), feature extraction tracks session tickets sent from servers. When a TLS session is resumed, it uses the ticket sent by the client to identify the original TLS session (containing the same ticket in the opposite direction) and extracts the missing certificate features from the original session. If the original session cannot be identified, the fake resumption client feature is set to indicate that it may not be a real resumed session. Fake resumptions are used to avoid confusing middleboxes that do not support TLS 1.3 [21]. We also observe them by using Ultrasurf, an Internet censorship circumvention tool, which establishes TLS 1.2 flows without certificates.

3.1.6. TLS 1.3. Most features presented so far are available up to TLS 1.2. However, TLS 1.3 changes the protocol to reduce the information available in the TLS handshake. In particular, some fields become encrypted, and other fields have fewer values to choose from. These changes are

Table 5: Payload features.

Feature	Type	Prior work	TLS 1.2	TLS 1.3
enc_data_size	Num	✗	✓	✓
enc_sent_size	Num	✗	✓	✓
enc_recv_size	Num	✗	✓	✓
enc_num_pkts	Num	✗	✓	✓
enc_sent_pkts	Num	✗	✓	✓
enc_recv_pkts	Num	✗	✓	✓
c_max_seq	Num	✗	✓	✓
c_max_length	Num	✗	✓	✓
s_max_seq	Num	✗	✓	✓
s_max_length	Num	✗	✓	✓
sent_recv_pkts_ratio	Num	✗	✓	✓
sent_recv_size_ratio	Num	✗	✓	✓
msg_pkts_c_0	Num	✗	✓	✓
msg_size_c_0	Num	[13]	✓	✓
msg_pkts_s_0	Num	✗	✓	✓
msg_size_s_0	Num	[13]	✓	✓
msg_pkts_c_1	Num	✗	✓	✓
msg_size_c_1	Num	[13]	✓	✓
msg_pkts_s_1	Num	✗	✓	✓
msg_size_s_1	Num	[13]	✓	✓
msg_pkts_c_2	Num	✗	✓	✓
msg_size_c_2	Num	[13]	✓	✓
msg_pkts_s_2	Num	✗	✓	✓
msg_size_s_2	Num	[13]	✓	✓

captured in the TLS 1.3 columns in Tables 2–5. In particular, client features such as the list of ciphers, elliptic curves, and point formats provide less information in TLS 1.3, as many values have been removed for increased protection, e.g., against downgrade attacks. However, as long as clients still support TLS 1.2, we expect the removal of those values to happen slowly over time. Furthermore, certificates are encrypted so that their features cannot be extracted directly from the network trace. On the other hand, payload features are not affected. Despite these changes, our approach is able to produce accurate TLS 1.3 clusters.

3.2. Filtering. Sandbox traces may include benign traffic from different sources. One source is background traffic generated by the OS and other benign programs installed in the virtual machine. Another source is benign samples executed in the sandbox. Yet another source is produced by malicious samples when connecting to benign services,for instance, to test Internet connectivity. Some of that benign traffic may use TLS. To identify benign traffic in the network traces, we use the Tranco list of the top 1M popular domains [22, 23]. Removing benign TLS flows is important to avoid generating clusters of benign traffic for scalability. We also filter vanilla Tor traffic used by some malware samples, which we identify using a previously proposed fingerprint [24]. Note that for better scalability, flows without application data in either direction were already removed prior to feature extraction. Of course, any filtering can be incomplete; e.g., some unpopular benign domains could remain. An advantage of an unsupervised model is that any remaining benign traffic would produce its own cluster. Once that benign traffic is identified, the cluster can be removed, without requiring retraining of the whole model, which would be needed by supervised approaches.

3.3. Clustering. The goal of the clustering is to group similar feature vectors that correspond to the same type of malicious TLS traffic. Each cluster comprises feature vectors generated by the same or different samples. Feature vectors from the same sample may end up in different clusters if the sample produces different types of communication such as *C* & *C* communication or communication with an updated server. When a cluster contains TLS flows from different samples, those samples should belong to the same malware family. However, samples from the same family could end up in different clusters, e.g., when a subset of samples of the family exhibits the same type of traffic and a different subset of family samples exhibits a different type of traffic.

We use a hierarchical density-based clustering algorithm based on HDBSCAN [25]. It does not require the number of clusters to be specified, recognizes clusters of arbitrary shape and variable density, scales to large datasets, and allows working with any kind of data by defining arbitrary distance functions. It distinguishes between clusters and noise, i.e., scattered data points that should not be considered to be part of any cluster. It has three hyperparameters: the number of elements that should be close to a central one to define a dense zone (mpts), the minimum cluster size (mcs), and a parameter (m) that tunes the density of linkage in the data structure it uses for neighbor search. Our evaluation searches for the best values for these parameters. The distance function used divides the features into two sets: numerical and categorical sets. First, it calculates the Euclidean distance of the numerical features and multiplies it by the fraction of numerical features over all features. Then, it computes the cosine distance between the categorical features, multiplying it by the fraction of categorical features. The final distance is the sum of these two values.

For each produced cluster, the domains in the SNI header and the leaf certificates of the flows in the cluster are collected so that they can be added to blacklists. In addition, our approach tries to assign a human-interpretable label to each cluster by applying the AVclass [8] labeling tool to the samples that produced the flows in the cluster. We detail labeling in Section 4.4. The cluster label corresponds to the family that has a majority in the cluster followed by a suffix to differentiate multiple clusters from the same family. For 31% of the clusters, AVclass cannot obtain a family for any sample, but in contrast to multiclass-supervised classifiers, this does not affect our unsupervised detector, affecting only the availability of human-readable family names to identify clusters.

3.4. Detection. The unsupervised detector leverages the produced clustering model to decide whether a previously unseen flow belongs to a known cluster. Detection consists in searching for the closest node to a given flow if any. Otherwise, the flow is considered an outlier (i.e., benign). For this, we evaluate two different methods. The variable threshold method determines the density of a cluster as the

largest distance of a node in that cluster from its closest neighbor; if the distance of the vector to a node in a cluster is below this threshold, the vector is considered to be belonging to that cluster. The fixed threshold method instead defines a fixed threshold for all clusters so that the unseen element should be close enough to a cluster's node to be part of it. Regardless of the method, if the given flow is assigned to a cluster, then it is labeled as malicious, and the cluster identifier is considered output. If no cluster is close enough, the flow is labeled as benign. The detection threshold is the main parameter that controls false positives (FPs) and false negatives (FNs). If the detection threshold is set very tight, then false positives are minimized, at the expense of increasing false negatives. When it is relaxed, false positives increase, while false negatives reduce. We evaluate this effect in Section 5.2.

4. Datasets

To perform clustering and build detectors, we use 972K sandbox network traces provided to us by a security vendor, summarized in Table 6. To evaluate the produced detector, we use seven months of TLS traffic collected at the boundary of a research network, summarized in Table 7.

4.1. Sandbox Network Traces. We use two datasets (SB-small and SB-medium) of network traces obtained by a security vendor from the execution of suspicious samples in their sandbox. Each sample was run for one minute, and sleeps introduced by the sample were skipped. The network trace contains all the traffic produced by the sample. Each sample has a single network trace. All traces contain TLS flows because this was the selection criteria used by the security vendor to share executions with us.

The union of the SB-small and SB-medium datasets produces the SB-all dataset. SB-all contains before filtering 12.9M flows on 527 destination ports, with the top three being 443/TCP (93.6%), 9001/TCP (2.67%), and 80/TCP (1.87%). Of 12.9M TLS flows, 24.5% have a payload and 8.2% (1M) remain after filtering and are used in clustering. We explain the reasons for this significant drop in Section 4.3. Those 1M TLS flows are almost exclusively for 443/TCP (99.8%) and originated by 28% of the 972K samples. Of all samples, 9% exclusively communicate through TLS (excluding DNS and DHCP), while 91% use HTTP in addition to TLS. Thus, HTTP traffic could not be used for detecting 9% of the samples, even if user privacy was not a concern. We also observe three times more effective second-level domains (e2LDs) contacted through TLS than HTTP. 6% of e2LDs are contacted both via HTTP and HTTPS, likely due to HTTP redirections towards HTTP URIs.

4.2. Research Network Traffic. To evaluate false positives (FPs), we use logs of TLS flows collected at the boundary of a research network. Since no infections were detected in this network during the monitoring period, we assume that traffic is benign. For privacy reasons, the logs consist exclusively of TLS flows. The traffic is split into five datasets, summarized in Table 7. The first three comprise over seven months of traffic originated by 1,216 source IP addresses that produce 34M TLS flows. For the comparison with Joy [12], we collected two additional short captures, where we run Joy in parallel with our TLS log collection.

4.3. TLS Versions. There exist two significant differences between SB-all sandbox traffic and benign traffic from the research network. First, in SB-all, only 24% of the TLS flows exchanged any application data. This is in stark contrast with 96% of the research network flows having application data. Second, 93% of the sandbox flows after filtering use TLS 1.0, 5.6% use TLS 1.2, and less than 1% use TLS 1.3. This TLS 1.0 dominance is in stark contrast with the traffic from the research network where 80–86% of the flows after filtering use TLS 1.2, 12%–18% use TLS 1.3, and less than 1% use TLS 1.0. These differences cannot be due to the different dataset time frames as SB-medium and Benign01 partially overlap in the second half of 2019.

We believe both differences are rooted in the sandbox using almost exclusively Windows 7 virtual machines, which we have verified by applying the p0f passive fingerprinting tool to the network traces [26]. Windows 7 system libraries by default use TLS 1.0. TLS 1.1 and 1.2 are supported but have to be manually enabled by the user [27]. We believe that the two differences are caused by the majority of the malware using default Windows TLS functionality rather than a cryptographic library statically linked in the executable. If malware was using a statically linked cryptographic library, then, in 2017–2018, TLS 1.2 would be used by default. To use TLS 1.0, malware developers would have to configure the used cryptographic library to specifically use TLS 1.0, which seems unlikely as this happens for samples from many families. 6.5% of flows using TLS 1.2 and 1.3 likely correspond to malware using a statically linked TLS library or invoking the default browser in the sandbox to open a webpage.

The low fraction of TLS flows with application data is also likely due to the use of the default TLS functionality in Windows 7. Malware tries to connect to servers using TLS 1.0, but many servers no longer support TLS 1.0, or the offered cipher suites, and reject it. Note that the PCI Council has suggested that organizations migrate from TLS 1.0 to TLS 1.1 or higher before June 30, 2018 [28]. This conclusion is supported by 37% of TLS flows having a protocol_version alert, i.e., no TLS version can be agreed, and another 16% having a handshake_failure alert, i.e., no cipher suite can be agreed.

This analysis highlights the impact that sandbox configuration can have on the collected data. Configuring the sandbox with old software may be beneficial to observe some behaviors like exploitation, but it can negatively impact other aspects such as TLS behavior. Running the sample on multiple OSes is arguably the best alternative, but it impacts scalability. Note that we do not control the sandbox and thus cannot configure it with newer OSes. This is a common scenario where data collection and analysis are performed by different teams. The main impact of the sandbox in our work is that flows with no application data are filtered, and

Table 6: Summary of network traces collected from a commercial sandbox.

Dataset	Start	End	Samples	TLS destinations			TLS flows			TLS version			
				SNI	IP	Ports	All	Payload (%)	Filtered (%)	Resumed (%)	1.0 (%)	1.2 (%)	1.3 (%)	Other (%)
SB-small	2017-11-01	2018-01-30	342.1K	49.8K	17.9K	268	1.9M	70.1	8.8	19.7	87.5	12.3	0	0.2
SB-medium	2019-05-30	2019-10-30	630.5K	65.8K	22.4K	339	11.0M	16.6	8.1	15.0	95.4	2.9	1.7	0.0
SB-all	2017-11-01	2019-10-30	972.6K	113.5K	37.6K	527	12.9M	24.5	8.2	15.7	93.5	5.6	0.9	0.0

Table 7: Summary of collected research network traffic.

Dataset	Start	End	SRC	TLS destinations			TLS flows			TLS version			
				SNI	IP	Ports	All	Payload (%)	Filtered (%)	Resumed (%)	1.0 (%)	1.2 (%)	1.3 (%)	Other (%)
Benign01	2019-10-08	2019-11-24	1.0K	108.6K	85.5K	824	9.6M	96.0	8.9	37.3	0.9	86.8	12.2	0.1
Benign02	2019-12-10	2020-01-31	1.0K	127.2K	100.3K	1.1K	13.2M	96.2	9.8	42.5	0.8	80.7	18.4	0.1
Benign03	2020-02-01	2020-04-30	1.0K	120.8K	98.4K	2.9K	11.6M	96.0	7.9	43.4	0.9	81.7	17.4	0.0
Comp.train	2020-10-23	2020-10-25	845	6,257	8,409	37	879.9k	66.5	27.9	34.9	0.7	65.5	33.7	0.1
Comp.test	2020-10-28	2020-10-28	496	7,277	9,603	1.0K	396.2k	74.9	18.7	36.9	0.3	69.0	30.3	0.4

eventually, some network traces may be removed because they do not contain any flows with application data. Thanks to the large size of our dataset, even after filtering, we are still left with 1M TLS flows from 272K malicious samples. We do not think other selection biases are introduced as the TLS version has little influence on the clustering results according to the feature analysis process.

4.3.1. TLS 1.3. Since its release on August 2018, TLS 1.3 has seen fast adoption [2, 29]. Holtz et al. measured that by April 2019, it was used in 4.6% of TLS flows [29]. In our research network, one year later, we observed 17.4% TLS 1.3 flows, nearly a fourfold increase. In the sandbox dataset, after filtering, modest 0.9% of flows from 6.8K samples use TLS 1.3. This shows that a minority of malware authors try to keep their TLS traffic as secure as possible, avoiding the default TLS versions offered by the OS and using instead a statically linked cryptographic library they can customize. Most TLS 1.3 flows belong to three malware families: Sofacy, an alias for the Fancy Bear (APT28) Russian cyber espionage group, Lockyc, an imitator of the Locky ransomware, and Razy, which steals cryptocurrency wallets [30]. The main impact of TLS 1.3 is that certificates are not observable. Additionally, the SNI can be transmitted encrypted by using new extensions. We also expected a reduced number of cipher suites, but due to backward compatibility, we observe 14–78 cipher suites being offered, much higher than five standard TLS 1.3 cipher suites. Despite the reduced feature set, our evaluation shows that our approach successfully handles TLS 1.3.

4.4. Family Labeling. We query the hashes of all 972K sandbox samples to VirusTotal (VT) to collect their AV labels. Among those, 64% (623K) were known to VT. We feed the VT reports to the AVclass malware labeling tool [8]. AVclass outputs the most likely family name for each sample and also classifies it as malware or PUP based on the presence of PUP-related keywords in the AV labels (e.g., adware and unwanted). Overall, AVclass labels 59% (574K) of the samples. For the remaining samples, no family was identified because their labels were generic. The 272K samples in the filtered SB-all dataset are grouped into 738 families, 545 malware and 193 PUPs. For the interested reader, Table 8 shows the top 20 families.

To establish the malware family responsible for a cluster, we apply a majority vote to the AVclass labels of the samples in the cluster. However, for 35% of clusters, AVclass does not assign a family to any of the samples in the cluster (i.e., they only have generic labels), and thus, the cluster cannot be labeled. Note that, in contrast to multiclass-supervised classifiers, our unsupervised detector does not use family labels and thus can still detect flows for the 41% of samples and 35% of clusters that AVclass cannot label. In detection, even when the cluster to which the detected flow belongs does not have an associated family name, it still provides important contextual information to analysts by capturing other malicious samples that generate similar traffic.

4.5. Ground Truth. To evaluate the clustering results, we use a manually labeled subset of flows from SB-all, summarized in Table 9. The AVclass results were used to randomly select

Table 8: Top 20 families by samples after filtering.

Family	Type	SNI	IP	TLS flows	Samples
ClipBanker	Malware	7	10	319,185	88,680
Upatre	Malware	251	347	311,687	64,860
Shiz	Malware	12	18	41,578	19,959
Zbot	Malware	120	116	46,437	5,502
Pcchist	Malware	1	1	4,713	4,653
InstallMonster	PUP	61	75	4,537	4,163
Sofacy	Malware	1	1	3,381	3,381
Xetapp	PUP	2	2	3,908	2,828
OxyPumper	PUP	7	502	3,051	2,431
Bublik	Malware	36	33	30,741	2,240
Vkontaktedj	PUP	2	12	2,060	1,599
Khalesi	PUP	8	7	1,787	1,591
Playtech	PUP	8	17	2,912	1,497
Multiplug	PUP	2	2	9,223	1,055
Adposhel	PUP	2	28	958	958
Razy	Malware	88	120	10,579	945
Lockyc	Malware	2	3	724	709
NoobyProtect	Malware	12	22	1,133	632
Download Assistant	PUP	50	63	574	573
Delf	Malware	76	114	1,127	542

Table 9: Manually generated ground truth.

Family	Total flows	Samples	Clusters
ClipBanker	10,183	9,718	3
Shiz	10,105	9,250	4
Upatre	20,352	8,738	3
Bublik	140	57	5
—	185	170	12
Ekstak	20	20	1
Miancha	9	9	1
Total	40,994	27,962	29

samples from four of the largest families (ClipBanker, Upatre, Shiz, and Bublik), a couple of small families (Ekstak and Miancha), and a variety of samples that AVclass cannot classify. Note that the AVclass family is not used as ground truth by itself as it is not available for all samples, may be incorrect for some samples, and does not separate different types of traffic from the same sample. Still, family labels help ensure that a variety of malware is included. Then, benign traffic was filtered, and features were extracted (e.g., certificates, domains in SNI headers, and payload sequences). In addition, additional information was obtained that is not available to our detector such as VT reports that contain file properties and behavioral information, other features in the network traces not used by our approach (e.g., non-TLS traffic and destination IP addresses), and the AVclass family. By examining static, dynamic, and shared indicators in the reports and network traffic, 29 clusters were identified.

5. Evaluation

This section first presents the clustering results in Section 5.1 and then the detection results in Section 5.2.

5.1. Clustering Results. We leverage the ground truth to assess the clustering accuracy along three dimensions: features, clustering algorithm parameters, and z-score normalization for numerical features. To evaluate clustering accuracy, we use precision, recall, and $F1$ score, which are common metrics for evaluating malware clustering results [31]. These metrics do not require or use cluster labels. They measure structural similarity between obtained clustering and ground truth clustering.

Table 10 summarizes the results obtained for 9 clustering configurations. First, we evaluate how different sets of features affect the results, using default parameters for clustering and z-score normalization. We evaluate 7 sets of features. FD1 corresponds to using all features and acts as a baseline. The other feature sets correspond to an ablation study where we remove some features and measure how much the accuracy metrics are reduced with respect to the FD1 baseline. To evaluate the impact of each feature category, we build four feature sets (FD2–FD5), each excluding the features in one feature category (e.g., excluding client features in FD2). To evaluate the impact of payload features, we exclude features from the other three categories (client, server, and certificate) in FD6. To evaluate the impact of our novel features, we exclude them from FD7, leaving only those already present in prior works. This search shows that server and payload features provide most information (largest drop when excluded) and that client features also provide some information. However, certificate features are not useful (excluding them may help achieve the best results) because they make clustering split true clusters into subclusters due to the prevalence of certificate polymorphism and certificates from free CAs. The only useful information in free certificates is the domain name, which often overlaps the SNI feature. In addition, some CDNs like CloudFlare are abused by multiple families, and the similarity of their certificates does not capture a real relationship. Based on these results, when exploring other dimensions, we exclude the certificate features. The results from FD7 show that the new features proposed in this work increase the clustering accuracy, raising the $F1$ score to 0.993, compared to 0.973 when using only the features proposed by prior works. Next, we evaluate the z-score normalization of the numerical features by observing that normalization improves results. Finally, we perform another search to optimize clustering hyperparameters. The table does not show every parameter value configuration evaluated. It only shows the results with the best parameter configuration, which turns out to be the default parameters. The best clustering is FD9, which does not use certificate features, applies z-score normalization for numerical features, and uses hyperparameters mpts = 2, mcs = 2, and $m = 10$. It achieves a precision of 0.996, a recall of 0.990, and an $F1$ score of 0.993.

5.1.1. SB-All Clusters. Table 11 shows the clustering results of the 1M flows in SB-all using the best clustering (FD9). It produces 18,569 clusters, of which 49% contain multiple flows and the rest are singletons. 36% of clusters contain flows from multiple samples, 12% contain more than one

Table 10: Clustering results of the ground truth. "Singl." refers to singleton clusters (i.e., only one element).

Clustering				Clusters		Cluster size					Accuracy			Time
ID	Features	Param.	ZS	All	Singl.	Min	Max	Med.	Mean	PStdev	Prec.	Recall	*F*1	
FD1	All	Default	✓	27	0	2	9,978	65	1,518.3	3,121.6	0.993	0.872	0.928	18.0
FD2	No client	Default	✓	25	0	2	9,990	65	1,639.8	3,257.3	0.992	0.877	0.931	16.9
FD3	No server	Default	✓	10	0	21	20,213	354	4,099.4	6,274.6	0.747	0.905	0.819	16.6
FD4	No cert	Default	✓	27	0	2	10,003	26	1,518.3	3,503.6	0.996	0.990	0.993	13.6
FD5	No payload	Default	✓	9	0	51	10,290	4,951	4,554.9	4,196.0	0.982	0.855	0.914	13.1
FD6	Payload	Default	✓	20	0	4	10,008	78	2,049.7	3,940.1	0.994	0.990	0.992	10.6
FD7	Prior	Default	✓	24	1	1	10,065	51	1,708.1	3,575.9	0.989	0.958	0.973	19.2
FD8	No cert	Default	✗	27	0	2	9,991	21	1,518.3	3,467.4	0.995	0.982	0.988	12.8
FD9	No cert	Best	✓	27	0	2	10,003	26	1,518.3	3,503.6	0.996	0.990	0.993	13.6

Table 11: SB-all clusters using the best clustering (FD9).

All	Singl.	Min	Max	Med.	Mean	PStdev	Time (h)
18,569	9,548	1	335,026	1	57.2	2,644.2	4.3

server leaf certificate, and 3% contain more than one SNI. These results show that clustering is able to group multiple samples that belong to the same family in the same cluster despite domain and certificate polymorphism that malware authors may apply to avoid blacklists. We also observe that 31% (5,847) of clusters (2,076 nonsingleton clusters) only contain samples without an AVclass family label. Samples in those families could not be detected using supervised multiclass classifiers since no labels existed for training their models.

We manually analyze clusters and report some observations and example clusters. The largest cluster has 335,026 flows from the ClipBanker family. There are 1,118 clusters without domain information; i.e., all flows lack an SNI header. One such cluster has 8 flows, each from a different sample. The flows connect to three servers (i.e., destination IP addresses) that use two certificates, one for *.zohoassist.com and the other for *.zoho.com. There are 8 content sequences, all with RRPs of similar sizes. AVclass fails to provide a family label for all eight samples. This is an example of clustering being able to group multiple samples despite the absence of domain information and malware using multiple certificates. It is also an example of a cluster where AVclass fails to label samples and supervised multiclass classifiers do not work.

A total of 50 clusters (33 nonsingletons) have TLS 1.3 flows, which lack certificates and may also lack an SNI header, making them challenging to detect and label. Of the 33 nonsingleton clusters, 30 contain only TLS 1.3 flows and three contain multiple TLS versions. Further examination shows that these 33 clusters likely belong to two families. According to AVclass, four TLS 1.3 clusters correspond to the Sofacy family. Each sample of this family produces a single TLS flow to domains huikin.host or w.huikin.host, both hosted at IP address 18.197.147.148, but clustering splits traffic into subclusters with different encrypted payload sequences. Of the 3,418 samples in these clusters, 37 have no AVclass family but are still correctly clustered with their family peers. For the other 29 nonsingleton TLS 1.3 clusters, we observe that all flows lack an SNI header but go to the same destination IP address 3.123.117.231. Note that the destination IP address is not a feature of our clustering. This indicates that grouping is correct although multiple clusters are obtained based on differences in the encrypted payload. In the future, we plan to evaluate the destination IP address as a feature, which we originally thought to be problematic due to IP reuse. Many samples in these clusters have an AVclass family label, but those labels correspond to 17 families, so it is not possible to identify the correct family. This highlights how our clustering can group samples not only with missing labels but also with conflicting ones, which would be problematic for supervised approaches.

We observe multiple families abusing free certificates such as Let's Encrypt and Tencent Cloud's TrustAsia certificates. One such cluster consists of 12 flows, each from a different sample. It contains three domains: atendimentostore-al.com, atendimentostore-ac.com, and buricamiudos-al.com. The similarity between the domain names indicates that the cluster is correct. Each domain resolves to a different server and has its own Let's Encrypt certificate. All samples have generic detection labels, so AVclass does not output a family for any of them. The payload consists of four content sequences, all of them with two RRPs. The first RRP has a request of 160 bytes and a response of 7088–7280 bytes. The second has a request of 96–112 bytes and a response of 5.18 MB–5.38 MB, potentially corresponding to a downloaded file. Clustering is able to group the 12 samples despite the use of different domains and certificates.

We also observe CloudFlare being abused by many families. One such cluster consists of five flows from three samples. Each flow is for a different domain and goes to a different CloudFlare server using different CloudFlare-issued certificates. The similarity, in this case, comes from the client and payload features. Overall, we observe that clustering is able to split different families abusing CloudFlare infrastructure into their own clusters.

5.2. Detection Results. This section evaluates our unsupervised detector. First, we select the best configuration. Then, we measure the false detection rate (FDR) of the selected configuration using benign traffic from the research network. Finally, we evaluate false negatives (FNs).

5.2.1. Unsupervised Detector Configuration. For selecting the best unsupervised detector configuration, we examine which threshold selection method produces a lower FDR with the best clustering configuration (FD9) and whether removing small clusters with few flows can significantly improve the FDR. Table 12 shows the results of applying each detector configuration to 95K flows from one day of benign traffic. We first evaluate the variable threshold selection method. We compare results when keeping and removing clusters with fewer than 50 flows. Removing them reduces the FDR from 1.8% to 0.11% but at the cost of not being able to detect traffic that matches those clusters. Then, we evaluate the fixed threshold selection method using different threshold values. We start with threshold 0.2 because the vast majority of false alarms occur at that distance or greater distances in the variable method. Then, we halve that threshold a couple of times to observe effects. The results show that the fixed threshold achieves the lowest FDR and that smaller thresholds make detection stricter and thus reduce FPs. In the limit, a threshold of zero would make the detector flag only flows with identical feature vectors to those in the cluster. Therefore, we select 0.05 as the best threshold as it still detects small modifications, as shown later in the FN evaluation.

5.2.2. FDR. To determine the real FDR of the unsupervised detector, we first apply the selected configuration (0.05 fixed threshold) to one week of traffic from the Benign02 dataset, observing a total of 119 alarms, for an FDR of 0.031%. To evaluate degradation over time, we then apply the detector again to almost four months of benign traffic (111 days corresponding to the union of the Benign02 and Benign03 datasets in Table 7), observing 708 alarms produced by 5 clusters, corresponding to a measured FDR of 0.032%. Thus, the FDR remains stable even after several months.

5.2.3. False Negative Evaluation. We also perform an experiment to validate that the chosen configuration of the unsupervised detector is not too strict and still detects variations of flows in clusters. For this experiment, we apply clustering, using the same configuration as FD9, on 90% randomly sampled flows of each cluster in ground truth, reserving the other 10% as testing data never seen by the model. We build an unsupervised detector using the previously selected configuration of a fixed 0.05 threshold. We use the produced model to make predictions on the 10% unseen flows. In this scenario, true positives (TPs) are flows assigned to a cluster by the closest neighbor that shares the same manually assigned cluster in ground truth. If a flow is not assigned to any cluster (no nearest neighbor), it is an FN as we know it is malicious. When the nearest neighbor of the flow is in a different ground truth cluster than the evaluated flow, we consider it an FP since the output cluster is wrong. We perform 10-fold cross-validation, sampling different testing data each time. Five runs have 100% TPs, and there are a total of 12 FNs on all 10 runs, for an FN rate of 0.029%. No FPs are observed. This experiment confirms that the unsupervised detector is capable of detecting previously unseen flows with low FNs and that the selected clustering configuration is not too strict.

Table 12: Comparison of unsupervised detector configurations on 95K flows from one day of benign traffic.

Model	Thres. sel.	Thres.	MCS	FDR (%)	Alarms
FD9	Var	—	—	1.8	1,698
FD9	Var	—	50	0.11	109
FD9	Fixed	0.20	—	0.4	389
FD9	Fixed	0.10	—	0.08	73
FD9	Fixed	0.05	—	0.002	2

5.3. Comparison with Prior Works. This section compares our approach with the two state-of-the-art publicly available detection tools: Joy [12], a binary-supervised classifier, and Kitsune [10], an autoencoder (AE)-based anomaly detector. Both tools can detect malicious traffic but do not classify detected traffic into families. Thus, the comparison focuses on the detection goal.

5.3.1. Joy. We compare our unsupervised detector with Joy [12], publicly available implementation of the binary-supervised detector by Blake and McGrew [5, 6]. We focus on the comparison with malicious flow detection since the implementation of the multiclass-supervised classifier by the same authors [7] is not publicly available. One limitation of Joy is that it considers all network flows in the input network traces as malicious. However, as shown in Section 4, much traffic in sandbox traces is benign. To understand the impact of this choice by Joy, as well as to make a fair comparison between Joy and our unsupervised detector, we build three different Joy logistic regression models. The joy-polluted model applies Joy without any modifications. Thus, all flows in the sandbox network traces are considered malicious. The joy-unpolluted-exc model excludes the training benign flows identified by our filtering step. The joy-unpolluted-inc model labels the benign flows identified by our filtering step as negative. This last model explicitly tells Joy that the sandbox traces indeed contain benign traffic. Finally, the supervised detector corresponds to our unsupervised detector using filtering and the best configuration identified in Section 5.2.

For training, we use 90% of the network traces from the SB-small dataset as the positive class (minus benign flows for models with filtering) and three days of traffic from the research network as the negative class. For the research network traffic, we run Joy in parallel with TLS log collection since Joy requires network traces as input and cannot process directly our TLS logs. For testing, we use the remaining 10% traces of SB-small and an extra day of traffic from the research network. Two small differences in feature extraction between Joy and our approach are that Joy is not able to process a portion of SSLv2 flows and that it classifies any flow, even those that do not complete the TLS handshake. To make the comparison as fair as possible, we

Table 13: Comparison of Joy models with our approach.

Model	TP	FP	TN	FN	Prec.	Recall	$F1$
Joy-polluted	19,950	142,703	284,836	11	0.12	0.99	0.22
Joy-unpolluted-exc	19,951	142,163	285,376	10	0.12	0.99	0.22
Joy-unpolluted-inc	15,515	2,154	425,385	4,446	0.88	0.77	0.82
Unsupervised	18,359	2,241	425,294	1,601	0.89	0.92	0.91

exclude both cases so that the comparison can be made for the same flows.

Table 13 summarizes the comparison. Using Joy without modifications (joy-polluted) produces a very large number of FPs and an overall low $F1$ score of 0.22. This indicates that Joy is learning to differentiate sandbox traffic (mostly TLS 1.0) from traffic from the research network (mostly TLS 1.2 and 1.3). This is confirmed by the difference between the joy-unpolluted-exc and joy-unpolluted-inc models. When, during training, Joy is told that benign traffic in the sandbox is not necessarily malicious (joy-unpolluted-inc), the accuracy greatly improves as many FPs are removed, pushing the $F1$ score from 0.22 up to 0.82. However, the number of FNs increase significantly because Joy is forced to produce aless-strict model that now has to account for some non-malicious sandbox traffic. In contrast, our unsupervised approach captures different types of traffic from a family in their own clusters, producing a tighter model. Our unsupervised detector outperforms all Joy models, achieving an $F1$ score of 0.91, compared to 0.82 for Joy's best model.

5.3.2. Kitsune. We also compare our approach with Kitsune [10], a state-of-the-art AE-based anomaly detector. Kitsune's approach does not require malicious traffic during training. It builds an ensemble of neural network AEs from input benign traffic and derives a detection threshold by examining the maximum root mean square error (RMSE) observed in input benign traffic. Given test traffic, Kitsune computes the RMSE value. If the value is larger than the inferred threshold, then traffic is considered anomalous, and an intrusion is flagged. Similar to Joy, Kitsune takes input network traces in the PCAP format, and thus, we cannot use our TLS logs from the research network for this evaluation. Unlike Joy, we could not run Kitsune in parallel with our TLS log collection, since our collection had been discontinued by the time we performed this evaluation. To evaluate both tools on the same inputs, we leverage the sandbox traces. We first filter out all non-TLS traffic from the traces by selecting the packets to/from port TCP/443 and then separate benign traffic from malicious traffic using our filtering (Section 3.2).

First, we tried to produce the Kitsune model using all the benign traffic in the SB-small dataset, but after one week of running, the model had not finished, and we stopped it. Kitsune was designed to run on low-resource network devices, and its design minimizes the amount of memory used. However, it runs on a single thread, making runtime the bottleneck. Thus, its model training does not scale to large amounts of traffic. Because of this, we use a small set of ground truth traces to train and test both tools. More concretely, we use 4,125 ground truth traces for training each model. Then, we test both tools on the same traffic that was not part of training of either of the two approaches. In particular, we use the benign traffic along with the malicious traffic of 5,156 traces to build the testing dataset. We use the Kitsune configuration suggested by its authors [10]: the detection threshold is the maximum RMSE value seen during the training phase, and the maximum number of features allowed per AE is 10.

Another key difference is that Kitsune operates at the packet level, while our approach operates at the flow level. To compare the results of both approaches, we first create a mapping from each packet to the flow to which it belongs. When Kitsune flags a packet as an intrusion, we use the mapping to identify the flow to which the packet belongs and mark the whole flow as malicious. Flows where no packet has been flagged as an intrusion are considered benign. To build the packet-to-flow mapping, we produce a flow identifier for each packet using a set of six values: hash of the trace where the packet appears, source IP, source port, destination IP, destination port, and protocol. To assign the same identifier to both directions of the flow, we first sort the six values lexicographically and then hash them to produce the flow identifier. After applying Kitsune's feature extractor to the ground truth traces, we add two additional fields to the Kitsune vectors: the flow identifier and the SNI field for the flow. The SNI is used by filtering to determine if a flow is benign or malicious. We also modify the feature extraction of our approach to include the flow identifier.

Table 14 shows the results of both models. Kitsune achieved a precision of 0.54, a recall of 0.65, and an $F1$ of 0.59. For the same dataset, our approach achieves a precision of 0.99, a recall of 0.99, and an $F1$ score of 0.99. The low precision of Kitsune is due to a large number of FPs (13,137). False positives are a common problem with anomaly detectors since any significant deviation from profiled benign traffic is flagged as an intrusion, while benign testing traffic may contain natural changes that are not related to malicious behavior. Kitsune also introduces 8,323 FNs, likely due to malicious traffic with similar packet sizes and interarrival times to benign traffic, which cannot be easily distinguished using the benign traffic model. Our approach can ameliorate that problem by using three additional categories of TLS-specific features (clients, servers, and certificates).

6. Related Work

Table 15 summarizes the most related work. It includes three works that specifically explore the detection and classification of TLS flows [5–7] and two general anomaly-based

Table 14: Comparison with Kitsune for ground truth TLS traffic.

Model	TP	FP	TN	FN	Prec.	Recall	$F1$
Kitsune (flow level)	15,681	13,137	3,527	8,323	0.54	0.65	0.59
Unsupervised	30,297	1	22,317	94	0.99	0.99	0.99

intrusion detection systems that do not specifically target TLS flows but can be applied to them as they do not require access to unencrypted payload data [9, 10]. The bottom row shows the approach presented in this paper. The table characterizes each work according to its goal, approach, and features used. The goal can be detecting malicious flows, as well as classifying those malicious flows by originating family. The approach shows whether the work uses supervised ML models, unsupervised clustering, and anomaly detection based on autoencoders. It also illustrates approach granularity, i.e., whether it works at the flow (F) or packet (P) level. The features used can be TLS-specific (clients, servers, and certificates), specific to other protocols such as DNS and HTTP, and examine the payload and interarrival times of packets in a flow in a protocol-agnostic manner. Next, we compare these 5 works with our approach.

Anderson et al. [7] first proposed a binary (logistic regression)-supervised detector using features from TLS flows. In addition, they trained a logistic regression multiclass classifier for 18 malware families but failed to generate a classifier for 73% of their input samples, for which they could not obtain a family label. We compare our unsupervised detector with the public implementation of their binary-supervised classifier. Their implementation assumes that all TLS flows in the input network traces are malicious, leading to a very low $F1$ score. After fixing that issue, the $F1$ score raises to 0.82, compared to 0.91, for our unsupervised detector on the same data. Compared to their multiclass classifier, our unsupervised detector does not require family labels and detects flows from samples without a family while still assigning a family to detected flows if available.

In follow-up work [5],the authors add to their TLS detector model features extracted from DNS responses and HTTP flows. In contrast, our approach operates solely on TLS flows and does not require plain-text protocols, increasing user privacy. Furthermore, our unsupervised approach can assign detected flows to their family clusters and handle samples that do not use HTTP or connect to their C & C servers using IP addresses.

Blake and McGrew [6] also evaluated six supervised learning classifiers for detecting malicious TLS flows despite inaccurate ground truth and nonstationarity of network data. In their experiments, the random forest performed best, but the quality of its results decreased significantly over time. In comparison, the FDR of our unsupervised detector does not degrade over a four-month period.

Mirsky et al. [10] presented Kitsune, an anomaly detection approach that takes benign traffic as input and uses an ensemble of autoencoders to detect anomalies that indicate intrusions. Using an anomaly-based approach removes the need for a malicious training dataset but prevents it from addressing the classification goal. Kitsune uses packet sizes and interarrival times as features so that it can be applied to any protocol including TLS flows. It is designed to be efficient so that it can run on network devices that have limited resources (e.g., Raspberry PI). We compare our approach with the publicly available implementation of Kitsune, showing that our approach significantly improves the accuracy.

Bovenzi et al. [9] presented H2ID, an intrusion detection system that addresses both the detection and classification goals. Similar to Kitsune, it first uses an autoencoder approach to detect anomalies as intrusions and uses payload size and interarrival timing as features so that it can be applied to different protocols, including TLS flows. However, it adds a second phase where it applies a supervised ML model to classify the detected anomalies into known attacks or unknown if they do not match the model. In contrast, our approach does not require a labeled training dataset and can cluster similar malicious flows even when their family is unknown.

6.1. TLS Fingerprints. A large body of work has built TLS fingerprints to identify the applications that initiate TLS flows. Fingerprints have been used to analyze several aspects of the TLS ecosystem, including the impact of HTTPS interception by middleboxes and antivirus products [32], the evolution of TLS clients over time [1, 2], and the TLS implementations of popular censorship circumvention tools [33]. Prior works also built TLS fingerprints for detecting malware and PUP families [34, 35]. However, a recent work has shown that malware TLS fingerprints generate high FPs in real networks [1]. All these approaches build TLS fingerprints solely using features extracted from the client hello message such as TLS versions, supported cipher suites, extensions, and elliptic curves point formats. Ede et al. [36] proposed FlowPrint, a semisupervised approach for fingerprinting mobile apps from their TLS traffic by also using destination features such as the server certificate, IP address, and port. In comparison, our clustering also uses client and certificate features but enhances detection by further including server and encrypted payload features.

6.2. Malware Clustering. There has been extensive work conducted on malware clustering techniques, using a variety of features such as system calls, system changes, and network traffic [13, 31, 37–42]. Most similar to our approach are clustering approaches based on network traffic, which may build detection signatures [39–41] or unsupervised detectors [13] using produced clusters. Some of these works propose generic payload features that capture message sizes [13, 41], but none of these works uses TLS-specific features. In future work, we would like to combine our TLS features with other network and behavioral features to build a malware family classifier for samples executed in a sandbox.

Table 15: Related work on detection and classification of malicious TLS flows.

Work	Year	Goal		Approach			Features							
		Detection	Classification	Supervised	Unsupervised	Anomaly detection	Granularity	TLS client	TLS server	TLS certificate	Payload size	Interarrival	DNS	HTTP
APM [7]	2016	✓	✓	✓	✗	✗	*F*	✓	✓	✓	✓	✓	✗	✗
AM16 [5]	2016	✓	✗	✓	✗	✗	*F*	✓	✓	✓	✓	✓	✓	✓
AM17 [6]	2017	✓	✗	✓	✗	✗	*F*	✓	✗	✗	✓	✓	✗	✗
Kitsune [10]	2018	✓	✗	✗	✗	✓	*P*	✗	✗	✗	✓	✓	✗	✗
H2ID [9]	2020	✓	✓	✓	✗	✓	*F*	✗	✗	✗	✓	✓	✗	✗
This work	2022	✓	✓	✗	✓	✗	*F*	✓	✓	✓	✓	✗	✗	✗

7. Discussion

This section discusses limitations and avenues for improvement.

7.1. Potential Biases. We identify two potential sources of bias in our methodology. First, we discard almost 64% of the sandbox TLS flows in our malware dataset because they try to use TLS 1.0, which was already deprecated by many servers at the time of execution. This happens because most malware uses Windows TLS functionality, and by default, Windows 7, the sandbox OS version, uses TLS 1.0. Of 9.3M discarded connections, 23% would have been anyway removed later by filtering. Second, we filter out benign domains, which may result in excluding some malware families from our study. For example, click-fraud malware generates revenue by falsifying clicks on pay-per-click campaigns and typically contacts numerous benign destinations. Other malware families may abuse benign content, hosting services such as code repositories (e.g., Github and Bitbucket), document editors (e.g., Google Docs and Pastebin), cloud storage services (e.g., Dropbox and Google Drive), or even social media (e.g., Twitter). Despite excluding some malware families, our dataset still encompasses a large part of the malware ecosystem, containing 738 different families (see Section 4.4). A third bias could be introduced by our ground truth not being representative of the whole SB-all dataset. We try to ameliorate this by including multiple families, limiting the number of samples of each family, and including samples without a known family.

7.2. Detection through Other Protocols. Usage of TLS by malware keeps increasing, having more than doubled from 10% in 2016 [7] up to 23% in 2020 [3]. However, malware may still use unencrypted traffic or mix unencrypted and encrypted protocols, as illustrated by 91% of samples using plain HTTP in addition to TLS. Prior work suggested using unencrypted protocols such as HTTP and DNS to assist in the detection of TLS malware [5]. However, we already observe 9% of the samples using TLS but not other unencrypted traffic beyond DNS; we expect this ratio to keep increasing over time. Moreover, samples not using (or encrypting) a TLS SNI header (the one case where DNS helps) could also connect to their C & C using IP addresses. Furthermore, purely TLS-based detection improves user privacy, and technologies such as DNS over HTTPS could further hamper DNS utility. While detection via other protocols is still a possibility for a significant fraction of samples, which is not the case for all samples, and the situation is likely to get worse.

8. Conclusion

Detecting malicious TLS traffic is important, but it is a challenging problem. Binary-supervised detectors are limited and try to distinguish any malicious TLS flow, regardless of the malware family producing it. This produces models that are too loose, introducing errors. In addition, they do not provide contextual information about detected flows. Multiclass-supervised classifiers produce tighter models for specific malware families and can classify detected flows. However, they require family labels to train classifiers, which are not available for a large fraction of samples.

We have proposed a novel unsupervised approach for detecting and clustering malicious TLS flows. It first clusters similar malicious TLS flows without requiring family labels. Then, it builds an unsupervised detector that measures distance of the clusters to determine if a given flow is malicious (belongs to a cluster) or benign (no cluster is close enough). We have evaluated our approach using 972K traces from a commercial sandbox and 35M TLS flows from a research network. Our unsupervised detector achieves an $F1$ score of 0.91, compared to 0.82 for the state-of-the-art supervised detector, and an FDR of 0.032% over four months of traffic.

In future work, we plan to evaluate additional features such as the destination IP address and timing-related features. For example, the IP address that we originally thought to be problematic due to IP reuse happens to be useful in some cases according to our evaluation. Timing-related features have been used by prior works (e.g., [7, 9, 10]), but our examination found them too sensitive to the specific network setup, so we believe there is a need for a systematic evaluation of their usefulness. Furthermore, we would like to explore how to generalize our approach to handle other types of network traffic beyond TLS.

Data Availability

This work uses two data types: MALWARE TRACES are network traces from likely malicious samples executed in a sandbox and BENIGN TRACES are network traces from a research network containing traffic which is considered benign. MALWARE TRACES data used to support the findings of this study have not been made available because they come from a commercial sandbox and are considered proprietary data for the commercial service. BENIGN TRACES data used to support the findings of this study have not been made available because they contain sensitive user data that cannot be made publicly available by the research institution.

Disclosure

This study is also available as an Arxiv preprint [43].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by the Comunidad de Madrid S2018/TCS-4339 (BLOQUES-CM) and by the Spanish Government MCIN/AEI/10.13039/501100011033/ERDF through the SCUM Project (RTI2018-102043-B-I00), the PRODIGY Project (TED2021-132464B-I00), and an FPI grant (PRE2019-088472). These projects are cofunded by the European Union European Social Fund, HORIZON EUROPE European Innovation Ecosystems and NextGeneration funds.

References

[1] B. Anderson and D. McGrew, "TLS beyond the browser: combining end host and network data to understand application behavior," in *Proceedings of the Internet Measurement Conference*, Amsterdam, Netherlands, October 2019.

[2] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of age: a Longitudinal Study of TLS Deployment, Longitudinal Study of TLS Deployment," in *Proceedings of the Internet Measurement Conference 2018*, Boston, MA, USA, October 2018.

[3] L. Nagy, "Nearly a Quarter of Malware Now Communicates Using TLS," 2020, https://news.sophos.com/en-us/2020/02/18/nearly-a-quarterof-malware-now-communicates-using-tls/.

[4] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-Middle attack to the HTTPS protocol," *IEEE Security and Privacy Magazine*, vol. 7, no. 1, pp. 78–81, 2009.

[5] A. Blake and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, Vienna, Austria, October 2016.

[6] A. Blake and D. McGrew, "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*, Halifax, NS, Canada, August 2017.

[7] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 195–211, 2018.

[8] M. Sebastian, R. Rivera, P. Kotzias, and J. Caballero, "AVClass: a tool for massive malware labeling," in *International Symposium Research in Attacks, Intrusions, and Defenses,* Springer, Berlin, Germany, 2016.

[9] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in IoT scenarios," in *Proceedings of the GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Taipei, Taiwan, December 2020.

[10] Y. Mirsky, D. Tomer, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, 2018.

[11] F. Diaz, "Revamping In-House Dynamic Analysis with VirusTotal Jujubox Sandbox," 2019, https://blog.virustotal.com/2019/10/in-house-dynamic-analysis-virustotal-jujubox.html.

[12] D. McGrew, B. Anderson, P. Perricone, and B. Hudson, "Joy: a package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring," 2020, https://github.com/cisco/joy.

[13] C. J. Dietrich, C. Rossow, and N. Pohlmann, "CoCoSpot: clustering and recognizing botnet command and control channels using traffic analysis," *Computer Networks*, vol. 57, no. 2, pp. 475–486, 2013.

[14] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, "Dynamic application-layer protocol analysis for network intrusion detection," in *15th USENIX security symposium*, pp. 257–272, USENIX Association, Berkeley, CA, United, 2006.

[15] P. Prasse, L. Machlica, T. Pevný, J. Havelka, and T. Scheffer, "Malware detection by analysing network traffic with neural networks," in *Proceedings of the 2017 IEEE Security and Privacy Workshops*, San Jose, CA, USA, May 2017.

[16] E. K. B. Laurie and A. Langley, "RFC 6962 - Certificate Transparency," 2013, https://tools.ietf.org/html/rfc6962.html.

[17] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, Alexandria, Virginia, USA, October 2007.

[18] S. Chen, R. Wang, X. F. Wang, and K. Zhang, "Side-Channel leaks in Web applications: a reality today, a challenge tomorrow," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2010.

[19] Q. Sun, D. R. Simon, Y. M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted Web browsing traffic," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2002.

[20] S. Joseph, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," 2008, https://tools.ietf.org/html/rfc5077.

[21] A. Ghedini, "You Get TLS 1.3! You Get TLS 1.3! Everyone Gets TLS 1.3!," 2018, https://blog.cloudflare.com/you-get-tls-1-3-you-get-tls-1-3-everyone-gets-tls-1-3/.

[22] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: a research-oriented top sites ranking hardened against manipulation," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, 2019.

[23] Tranco: a research-oriented top sites ranking hardened against manipulation, Retrieved 20/05/2019 from https://tranco-list.eu/list/GKYK, 2019.

[24] S. Matic, C. Troncoso, and J. Caballero, "Dissecting tor bridges: a security evaluation of their private and public infrastructures," in *Network and Distributed System Security Symposium,* The Internet Society, Reston, Virginia, United States, 2017.

[25] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-asia Conference on Knowledge Discovery and Data Mining,* Springer, Berlin, Germany, 2013.

[26] M. Zalewski, "p0f v3: passive fingerprinter," 2020, https://lcamtuf.coredump.cx/p0f3/README.

[27] A. Marshall, "Security Protocol Support by OS Version," 2020, https://docs.microsoft.com/es-es/security/engineering/solving-tls1-problem.

[28] L. K. Gray, "Are you ready for 30 June 2018? Saying goodbye to SSL/early TLS," 2017, https://blog.pcisecuritystandards.org/are-you-ready-for-30-june-2018-sayin-goodbye-to-ssl-early-tls.

[29] R. Holz, J. Amann, A. Razaghpanah, and N. Vallina-Rodriguez, "The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods," 2019, https://arxiv.org/abs/1907.12762.

[30] T. Seals, "Razy Malware Attacks Browser Extensions to Steal Cryptocurrency," 2019, https://threatpost.com/razy-browser-extensions-theft/141181/.

[31] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, 2009.

[32] Z. Durumeric, Z. Ma, S. Drew et al., "The security impact of HTTPS interception," in *Network and Distributed System Security Symposium*, San Diego, CA, USA, 2017.

[33] S. Frolov and E. Wustrow, "The use of TLS in censorship circumvention," in *Network and Distributed System Security Symposium*, San Diego, CA, USA, 2019.
[34] B. Lee, "TLS fingerprinting," 2019, http://github.com/LeeBrotherston/tls-fingerprinting.
[35] Salesforce, "JA3-A Method for profiling SSL/TLS clients," 2018, https://github.com/salesforce/ja3/tree/master/lists.
[36] T. Ede, R. Bortolameotti, A. Continella et al., "FlowPrint: semi-supervised mobile-app fingerprinting on encrypted network traffic," *Network and Distributed System Security Symposium*, vol. 27, 2020.
[37] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of Internet malware," in *International Symposium on Research in Attacks, Intrusions and Defenses,* Springer, Berlin, Germany, 2007.
[38] A. Oprea, L. Zhou, W. Robertson, and A. S. Buyukkayhan, "Lens on the endpoint: hunting for malicious software through endpoint data analysis," in *International Symposium on Research in Attacks, Intrusions, and Defenses,* Springer, Berlin, Germany, 2017.
[39] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: clustering analysis of network traffic for protocol and structure independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, USA, 2008.
[40] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," *Networked Systems Design and Implementation*, Springer, Berlin, Germany, 2010.
[41] M. Zubair Rafique and J. Caballero, "FIRMA: malware clustering and network signature generation with mixed network behaviors," in *International Symposium on Research in Attacks, Intrusions and Defenses,* Springer, Berlin, Germany, 2013.
[42] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assesment,* Berlin, Germany, 2008.
[43] G. Gomez, P. Kotzias, M. Dell'Amico, L. Bilge, and J. Caballero, "Unsupervised Detection and Clustering of Malicious TLS Flows," 2021, https://arxiv.org/abs/2109.03878.