Malware Traffic Classification

Evaluation of Algorithms and an Automated Ground-truth Generation Pipeline

Syed Muhammad Kumail Raza
smkumail.raza@imdea.org

Juan Caballero
juan.caballero@imdea.org

ABSTRACT

Identifying threats in a network traffic flow which is encrypted is uniquely challenging. On one hand it is extremely difficult to simply decrypt the traffic due to modern encryption algorithms. On the other hand, passing such an encrypted stream through pattern matching algorithms is useless because encryption ensures there aren't any. Moreover, evaluating such models is also difficult due to lack of labeled benign and malware datasets. Other approaches have tried to tackle this problem by employing observable meta-data gathered from the flow. We try to augment this approach by extending it to a semi-supervised malware classification pipeline using these observable meta-data. To this end, we explore and test different kind of clustering approaches which make use of unique and diverse set of features extracted from this observable meta-data. We also, propose an automated packet data-labeling pipeline to generate ground-truth data which can serve as a base-line to evaluate the classifiers mentioned above in particular, or any other detection model in general.

1. INTRODUCTION

With the introduction of latest networking protocols, network traffic now-a-days is mostly encrypted. On one hand this increases anonymity and privacy, but on the other it becomes a daunting task for incident response teams and network intrusion detection systems to determine its trustworthiness. Traditional approaches of identifying threats, such as deep packet inspection and signatures, are not applicable on encrypted traffic. Breaking the encryption of modern encryption algorithms commonly used in the industry is next to impossible given the limited computational power of today's computers. Moreover, solutions that decrypt network traffic have any number of problems.

1. It compromises the privacy of the users, which was the very essence of introducing reliable, encrypted network flow in the first place.

2. There is no general solution of decrypting data. Moreover, same solutions do not always work on all encryptions.

3. They are extremely computationally expensive.

Furthermore, these solutions rely on the configuration of a cooperating endpoint. This makes it challenging to deploy, and limits its applicability. In this work, we do not propose decryption the network traffic. We instead focus on identifying malware communication in encrypted traffic through passive monitoring, the extraction of relevant data features, and semi-supervised clustering algorithms, namely FishDBC [14], dbscan [9], AutoClass [12] both in hierarchical and non-hierarchical form. We compare the results with the mean-shift clustering algorithm [1]. Also, to solve the problem of evaluating these clustering algorithms as well as to facilitate the possibility of training supervised machine learning models on these features, we propose a pipeline to automatically labelling packets in the network traffic dataset generated from a sandbox or an enterprise network using filtering approaches. The pipeline consists of 1) Data Extraction from a network flow, 2) Data Cleaning and filtering using well known benign and malicious domains and their derivatives. 3) Data Labeling to build the ground truth. The rest of the report contains the previous related work, the experiments on the clustering algorithms on a small dataset and their evaluations, data labelling pipeline results and discussion and lastly the conclusion.

2. RELATED WORK

Malware detection on network traffic has two major directions. Horizontal correlation and vertical correlation. The former use the network traffic of a single host to find evidence of a malware infection. The latter, uses the flow of at least two hosts to find non-benign communication. Horizontal correlation can also detect large-scale, malicious communication graphs. Major techniques are content independent, while others take content into consideration.

2.1 Flow Metadata

Network traffic monitoring systems have been used to collect metadata about network communications such as the IP addresses, ports, number of bytes exchanged and number of packets. The meta-data is important when traffic is encrypted because deep-packet inspection is not practical anymore.

The most common and the easiest way to analyse flow data employs IP address in the blacklists and flow repositories. This idea of data merging is largely used. However it comes with some inherent disadvantages, namely it is fragile and maintenance is difficult. Additional protocol-independent so called temporal features have also been studied, such as the packet lengths and distribution of byte values [18]. This data has also been used to perform application or malware detection [17–19, 29, 35, 36]. One such technique which has gained repute is BotFinder [33] which is a vertical correlation technique that uses only temporal and flow features. Unsupervised machine learning is used to identify clusters that are characteristic of malware communications collected from malware sandboxes. It detects periodic components in malware communications, with a detection rate of 0.8 with a 0.0001 false positive rate. Because BotFinder is content independent, it can operate on encrypted traffic. Our approach uses almost all data features available in a pcap including the temporal as well as categorical ones. This makes it robust agianst evasion techniques in

which the communication patterns doesn't show up in the network flow.

Another important vertical technique is BotHunter [18]. This algorithm works by tracking the comms between internal and external entities, and develops a signature of events that match a state-based infection sequence model. It uses these signatures to recognize the dialog during a malware infection. It is also content-agnostic, and uses Snort [32] signatures and anomaly detection based on content inspection. It employs a novel n-gram statistical payLoad Anomaly Detection Engine called SLADE [31].

2.2 Domain Name System(DNS)

According to IETF, the Domain Name System (DNS) [27] is a hierarchical, decentralized means to provide additional information about domain names, notably a domain name to IP address mapping. Recently, there is a trend of malware families taking advantage of DNS and domain generation algorithms (DGA) [7] to provide a evasive way to operate its command and control centers. Much work has been done in classifying DNS data as malicious or benign [6, 8, 25, 35]. However, none of these works take advantage of the DNS data to make inferences about encrypted network traffic. We use distributions of multiple data features as outlined before.

2.3 TLS

Transport Layer Security (TLS) [15] is a cryptographic protocol that provides privacy and anonymity for applications. TLS is usually implemented as a wrapper on other common web communication protocols such as HTTP or SMTP for email. HTTPS uses TLS over HTTP, i.e ssl encryption applied to the packets for client-server communication and is supported now-a-days by almost all web servers as well as web-browsers. There is an increasing trend noted in the use of TLS by malwares for the last four years from approx. 7% of samples with active network communications in June 2015 increased to around 18% in April 2016 and still increasing [5]. Researchers have looked into the features of malware certificates used in TLS flows [2–5, 35]. Most of this work doesn't examine the differences between the certificates of a malware TLS flow and a benign TLS flow. Also, since the payload of the packet is encrypted, the unencrypted flow metadata would also be effective in these scenarios. Our work uses these unencrypted TLS flows as features in the clustering algorithms in the upcoming sections. Man in the Middle (MITM) [11] approaches have also been proposed in encrypted traffic scenarios. This is not particularly feasible as in MITM network traffic through a security node is to be decrypted and signatures are to be applied to the unencrypted traffic to detect intrusion.

The next section describes the initial dataset as well as the clustering algorithms experimented with along with the results and evaluation.

3. APPROACH

In order to produce signatures that capture malware behavior, our approach comprises 3 phases: feature extraction, clustering and signature generation. To generate signatures for capturing malware behaviour, three stages are required:

1. Relevant feature extraction
2. Clustering connections based on the extracted features
3. Generating signatures from formed clusters.

The feature extraction process receives as input a set of PCAPs to pre-process, and output a dataset in the form of *feature vectors* that are of interest because of either its payload or its destination, that later are passed as input to the clustering phase. Herein, clusters of connections having similar attributes are generated, in such a way that are able to characterize specific malware, and from which signatures are finally produced using some selected features.

For the whole process, we filter two types of connections. First, those that do not contain any payload (i.e. they just perform the TLS handshake, but do not transfer any information after that), as they can not be analyzed by this approach. In order to select mostly malicious connections and to discard those attributes commonly present in benign traffic, we use a list of domains produced by Tranco [23]. This filtering also has the advantage of increasing the overall process performance, because the production of clusters for benign traffic can be omitted. The details on the generation of the Tranco list used in this research can be found in Section 4.

The reminder of this section is as follows: Section 3.1 describe the process to extract features. Section 3.2 contain the details about the clustering generation phase and Section 3.3 comprises of the evaluation and results.

3.1 Feature Extraction

The feature extraction is performed using Bro (Zeek) Network Security Monitor [30]. Table 1 contains all the features extracted from TLS connections within the input PCAPs. This process is widely studied in previous works [3–5]. As is shown in 1, in TLS 1.3 those features that can be used in fingerprint generation will be much more restricted [15]. For that reason we explore the possibility of use mainly the encrypted content to generate them. Having this in mind, instead of single encrypted packets we make use of concatenated messages or *sequences* of packets. As described in [28], a sequence is the concatenation of the packets going to the same direction from one endpoint to another, until a packet in the other direction is received.

In such way, the *TLS conversation* can reveal more information to the analysis, because it delimit in a more natural fashion when a *complete message* begin and end, instead of logging a constant number of packets (that is less instinctive and can possibly induce the lost of important information). The intuition is that two complete messages or sequences (a single sent-received pair) approximate one request-response in a TLS connection between two endpoints.

As long as we know that some connection is produced by some malware, we can label its conversations as malicious. If the endpoints repeatedly make use of the same request-responses, we can characterize and detect *malicious conversations*. As long as both endpoints remain unchanged, i.e. the server preserve its configuration and the client's implementation is the same, this allows to detect such conversations even if the server is replaced or replicated (for instance, when using domain polymorfism).

From all the conversation, we use only the first three sequences in both directions because it is more likely that the conversation diverges as the communication advance. We extract two features from each sequence: the total size (msg_size_*) and the number of packets being concatenated (msg_pkts_*).

3.2 Clustering

With the set of features extracted as described in the previous section, we experiment and evaluate different clustering algorithms on it. This involves fine-tuning some hyper-parameters and refining the set of features. The clustering algorithms chosen are described as follows:

		TLS Version	
Feature	**Class**	1.2	1.3
c_version	Client	✓	legacy_version = 0x0303
c_record_version	Client	✓	✓
c_ciphers	Client	✓	✓(R)
c_comp_methods	Client	✓	legacy_compression_methods = 0
c_curves	Client	✓	✓(R,E)
c_point_formats	Client	✓	✓(R,E)
c_supported_versions	Client	✓	✓(C)
c_alpn_list	Client	✓	✓(E)
c_extensions	Client	✓	✓(E)
c_fake_resumption	Client	✓	✓(C)
c_leaf_cert_validity	Client/Cert	✓	✗
c_leaf_cert_num_SAN	Client/Cert	✓	✗
c_leaf_cert_version	Client/Cert	✓	✗
c_leaf_cert_ext_num	Client/Cert	✓	✗
c_leaf_cert_validation_status	Client/Cert	✓	✗
c_leaf_cert_self_signed	Client/Cert	✓	✗
c_leaf_cert_subj_CN	Client/Cert	✓	✗
c_leaf_cert_subj_O	Client/Cert	✓	✗
c_leaf_cert_subj_OU	Client/Cert	✓	✗
c_leaf_cert_subj_C	Client/Cert	✓	✗
c_leaf_cert_subj_ST	Client/Cert	✓	✗
c_leaf_cert_subj_L	Client/Cert	✓	✗
c_leaf_cert_subj_Email	Client/Cert	✓	✗
c_leaf_cert_iss_CN	Client/Cert	✓	✗
c_leaf_cert_iss_O	Client/Cert	✓	✗
c_leaf_cert_iss_OU	Client/Cert	✓	✗
c_leaf_cert_iss_C	Client/Cert	✓	✗
c_leaf_cert_iss_ST	Client/Cert	✓	✗
c_leaf_cert_iss_L	Client/Cert	✓	✗
c_leaf_cert_iss_Email	Client/Cert	✓	✗
c_leaf_cert_sign_alg	Client/Cert	✓	✗
c_leaf_cert_pubkey_hash	Client/Cert	✓	✗
c_leaf_cert_pubkey_size	Client/Cert	✓	✗
c_num_certs	Client/Cert	✓	✗
s_version	Server	✓	legacy_version = 0x0303
s_record_version	Server	✓	✓
s_cipher	Server	✓	✓
s_comp_method	Server	✓	legacy_compression_method = 0
s_extensions	Server	✓	✓(R,E)
s_dst_port	Server	✓	✓
s_ct_signature	Server	✓	✓(E)
s_alpn_list	Server	✓	✓(E)
s_heartbleed_detected	Server	✓	✓(E)
s_session_ticket_lifetime	Server	✓	✓
s_leaf_cert_validity	Server/Cert	✓	✗
s_leaf_cert_num_SAN	Server/Cert	✓	✗
s_leaf_cert_version	Server/Cert	✓	✗
s_leaf_cert_ext_num	Server/Cert	✓	✗
s_leaf_cert_validation_status	Server/Cert	✓	✗
s_leaf_cert_self_signed	Server/Cert	✓	✗
s_leaf_cert_subj_CN	Server/Cert	✓	✗
s_leaf_cert_subj_O	Server/Cert	✓	✗
s_leaf_cert_subj_OU	Server/Cert	✓	✗
s_leaf_cert_subj_C	Server/Cert	✓	✗
s_leaf_cert_subj_ST	Server/Cert	✓	✗
s_leaf_cert_subj_L	Server/Cert	✓	✗
s_leaf_cert_subj_Email	Server/Cert	✓	✗
s_leaf_cert_iss_CN	Server/Cert	✓	✗
s_leaf_cert_iss_O	Server/Cert	✓	✗
s_leaf_cert_iss_OU	Server/Cert	✓	✗
s_leaf_cert_iss_C	Server/Cert	✓	✗
s_leaf_cert_iss_ST	Server/Cert	✓	✗
s_leaf_cert_iss_L	Server/Cert	✓	✗
s_leaf_cert_iss_Email	Server/Cert	✓	✗
s_leaf_cert_sign_alg	Server/Cert	✓	✗
s_leaf_cert_pubkey_hash	Server/Cert	✓	✗
s_leaf_cert_pubkey_size	Server/Cert	✓	✗
s_num_certs	Server/Cert	✓	✗
enc_duration	Timing	✓	✓
enc_sent_duration	Timing	✓	✓
enc_recv_duration	Timing	✓	✓
c_max_timing	Timing	✓	✓
s_max_timing	Timing	✓	✓
enc_data_size	Content	✓	✓
enc_sent_size	Content	✓	✓
enc_recv_size	Content	✓	✓
enc_num_pkts	Content	✓	✓
enc_sent_pkts	Content	✓	✓
enc_recv_pkts	Content	✓	✓
c_max_seq	Content	✓	✓
c_max_length	Content	✓	✓
s_max_seq	Content	✓	✓
s_max_length	Content	✓	✓
sent_recv_pkts_ratio	Content	✓	✓
sent_recv_size_ratio	Content	✓	✓
msg_pkts_c_0	Content	✓	✓
msg_size_c_0	Content	✓	✓
msg_pkts_s_0	Content	✓	✓
msg_size_s_0	Content	✓	✓
msg_pkts_c_1	Content	✓	✓
msg_size_c_1	Content	✓	✓
msg_pkts_s_1	Content	✓	✓
msg_size_s_1	Content	✓	✓
msg_pkts_c_2	Content	✓	✓
msg_size_c_2	Content	✓	✓
msg_pkts_s_2	Content	✓	✓
msg_size_s_2	Content	✓	✓

Table 1: Features extracted from TLS traffic. (R)=Reduced. (C)=Changed. (E)=Could be Encrypted

3.2.1 Mean-Shift Clustering

Mean-shift is a non-parametric centroid-based technique aimed to delineate arbitrary shaped clusters in complex multimodal feature space [14]. Applying the mean-shift method, it identifies candidates to be the so called centroids locating the fine and dense regions in the feature space. It then defines clusters looking for the data-points assigning them to one of the centroids. The centroids themselves are changed and updated recursively to avoid redundancy. Even though it uses a bandwidth parameter to determine the region to search through, that could be set manually or estimated by a function, Mean-shift doesn't rely on setting any other more involved initial parameters, it doesn't have any initial biases or assumptions. This property makes it particularly valuable for classifying various behaviors performed on different domain names by variant families of malware, a complex real feature-space. It operates on an n-dimensional space, so it could be fitted using set of m-features. However its scalability is limited. Nonetheless, it provides a good base-line to detect clusters and to compare them with other more experimental clustering algorithms.

3.2.2 DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise [26]. It is a popular unsupervised learning method. It is part of a family of clustering algorithms regarded as Density-based algorithms. These techniques works on the principle of grouping dense regions of data that are separated by coarse regions. Density based clustering algorithms are better than partition-based algorithms as they do not have the limitation of forming just sperical clusters but any shape. It is based on the concepts of density reachability and density-connectivity. Input to these algorithms are two parameters: epsilon (eps) and minimum number of points (minPts). Epsilon is the distance around an object that defines its neighborhood. For a given data-point x, when the number of datapoints within its neighborhood is at least minPts, then y is defined as a cluster center. All datapoints within its eps-neighborhood are said to be directly density-reachable from q. Also, a datapoint d is said to be *density-reachable* if it is within the eps-neighborhood of another datapoint that is directly density-reachable or density-reachable from d. Furthermore, datapoints p and q are said to be *density-connected* if a datapoint o exists that both p and q are density-reachable from. The ideas of *density-reachability and density-connectivity* are used to define what the DBSCAN algorithm considers as a cluster. A cluster is defined as the set of datapoints in a data set that are density-connected to a particular core datapoint [26]. All other datapoints which are not in an eps-neighborhood of a core-datapoint are considererd noise. Unlike K-Means and AutoClass, which assign every object to a cluster. DBSCAN can sort data into clusters of varying shapes as well. It works as following:

- Split the dataspace into n dimensions
- For each point in the dataspace, it forms an n dimensional neighbourhood around that data point, and then counts how many data points fall within that neighbourhood.
- This neighbourhood then forms a cluster. Then the cluster is iteratively expanded, by going through each individual point within the cluster, and counting the number of other data points nearby.

3.2.3 FISHDBC

FISH-DBC stands for Flexible, Incremental, Scalable, Hierarchical Density-Based Clustering for Arbitrary Data

			Connections		
Dataset	**Dates**	**Samples**	**All**	**% WP**	**% NIT**
GT	2017-11-02/2018-01-30	28	200	53.0	46.0

Table 2: Initial Dataset used in this research. WP = With Payoad. NIT = Not in Tranco.

and Distance [14]. The usefulness of this kind of clustering cores in flexibility. It works on arbitrary data without transforming categorical data into numeric arrays to let users define their custom distance functions as an alternative. It has the ability to evolve the flat clustering produced into a hierarchy tree, hence hierarchical clustering giving the users flexibility to group or divide clusters in super or subclusters. Overall, the clustering algorithm can be classified still as density based clustering. It is an approximation of the HDBSCAN algorithm [26], which is a variant of DBSCAN [9] which allows for hierarchical clustering. The advantages of using FISHDBC include better space and time complexity as it tries to avoid the quadratic run-time associated with DBSCAN. This makes it scalable for using it with very large data-structures such as network features in our case.

3.2.4 AUTOCLASS

Probabilistic model-based clustering, previously considered in [12], is another traditional but powerful and widely used clustering technique. We use an implementation of a probabilistic model-based clustering technique called AutoClass [12]. This algorithm lets us choose the number of clusters and soft clustering of the data. Soft clusters means that the datapoints can partially belong to more than one clusters. For such probabilistic model, the clustering algorithm determines the number of clusters and the parameters that govern the distinct probability distributions of each cluster. To accomplish this, AutoClass uses the Expectation Maximization (EM) algorithm [10]. The EM algorithm works in two steps: an expectation step and a maximization step. In the first step, the parameters are randomly initialized. The second-step, namely maximization, iteratively re-calculates the mean and standard deviation the parameters continually until they converge to a local maximum. These local maxima are recorded and the EM process is repeated. The process runs until 200 cycles in our results. AutoClass then tries to validate the best probability score for the optimal set of parameters to be used. It is in the form of a Bayesian score which is based on a similarity and dissimilarity index between the clusters. It also penalizes the Bayesian score for models with large number of clusters. This is done as a regularization approach to prevent overfitting.

Initial Dataset.

The initial dataset to test these clustering approaches consist of 28 manually labelled samples from a unlabelled dataset of 342K samples in the sandbox of a security vendor by an average time of 38.5 seconds. The traces contain all the network traffic produced by the sample during execution. which were recorded using man-in-the-middle proxy so we can map a large number of HTTPS connections to their URL. It consists of 28 executions containing 200 TLS flows. The details are shown in Table 2.

. We use the approach described in Algorithm 1 to cluster the dataset, to then compare their results and performance. We use the labelled GT_dataset for evaluation. We apply the algorithm using all the above mentioned clustering approaches using some pre-tunned hyper-parameters and different sets of features, as shown in Table 3.

Algorithm 1 Clustering

1: Generate groups of server names, using the Effective Second Level Domain of the SNI. In case that it is not present use the destination IP instead. ▷ step 1
2: **if** not merge_cert_aggressive **then**
3: Merge groups that contain the same set of certificates ▷ step 2
4: **else**
5: Merge groups that share at least one certificate ▷ step 2.a
6: **end if**
7: Split groups by their number of TLS fingerprints ▷ step 3
8: Run a clustering algorithm on the entire dataset using payload features
9: **for** cluster in clusters **do**
10: **for** group in groups **do**
11: **if** group is a subset of cluster) **then**
12: Add group to the cluster i.e merge them.
13: **end if**
14: **end for**
15: **end for** ▷ step 3.a, step 3.b.

ID	**Method**	**eps**	**minPts**	**Precision**	**Recall**	**F1**
			GT1			
ms1	mean-shift	-	-	87.56	49.92	63.59
ms2	mean-shift, cert aggr	-	-	86.30	55.63	**67.65**
dbs1	dbscan	50	20	24.01	100	38.73
fdbc1	fishdbc	50	20	36.03	100	52.97
as1	autoscan	-	-	51.30	55.21	53.18
fdbc2	fishdbc All feat	50	20	43.81	100	60.93
			GT2			
ms3	mean-shift	-	-	80.12	41.91	55.03
ms4	mean-shift, cert aggr	-	-	76.22	44.61	56.28
dbs2	dbscan	100	20	39.21	100	56.33
fdbc3	fishdbc	100	20	40.12	100	57.26
as2	autoscan	-	-	55.21	62.81	58.76
fdbc4	fishdbc All feat	100	20	48.33	100	**65.16**

Table 3: Results of the clustering processes on both variants of GT_dataset with different hyper parameters and algorithms. The best f1-score obtained are in bold.

The last step in 1 is performed in order to merge groups resulting from step 3 (line 1.7): when all the elements of a group are a subset of one cluster, we merge them with the elements of other groups that are also subsets of the same cluster. This approach have the restriction that some clustering algorithm must run before in order to generate the initial clusters.

One advantage of clustering approach with Algorithm 1 is that it mostly makes use of payload content, and few features that are still available in TLS 1.3, namely SNI extension, server certificate and the TLS fingerprint, derived from client features still available (set of cipher suites, extensions, elliptic curves and point formats).

3.3 Results and Discussion

Table 3 describes the results obtained after different experiments of clustering algorithms on GT_dataset. The DBSCAN and FishDBC (optionally) requires two additional parameters of *eps* and *minPts* as described in the previous section. Multiple experiments were performed by tuning these values to get best results possible. These values were set in the range of (50, 100) and (20,100) respectively. The table shows a subset of experiments which had valid results. The mean-shift algorithm performs best on GT_dataset1 with the F1-score of 67.65% with aggressive certificate merging option enabled which merges the clusters w.r.t to the step 3 of the Algorithm 1.However, the other clustering techniques fall behind with f1-scores close to random chance. DBSCAN particularly noteworthy with f1-score of 38.73 which is much worse. This can be attributed to the fact that DBSCAN

struggles to handle high dimensional data with the so-called 'Curse of Dimensionality' issue described by several authors [16, 20, 21]. On the GT_dataset2 FISHDBC with all features performs the best with f1-score of 65.16. This is comparable to mean-shift with certificate aggressive in the first variant. Other algorithms still perform worst with f1-scores near to random chance including dbscan and fishdbc with higher eps values. The DBSCAN algorithm performs much better with eps=100 and minPts=20, outputting an F1-score of 56.33 as compared to 38.73 before.

Although some clustering algorithms clearly perform better than the others, they all are struggling to approach the desired precision and recall value needed to build a practical classifier. In some experiments the recall value reaches 100% but the precision lags behind and in others both values are low. The reasons for these low f1-scores are outlined below:

1. The dataset used to evaluate these algorithms namely GT, is too small to exploit the full potential of these clustering algorithms. The dataset contains just 28 manually labelled samples consisting of 200 TLS flows. The details are shown in Table 2. Evaluating on just a small dataset is not realistic as in reality a large number of TLS flows are produced in network traffic even if we capture it for a relatively short period of time. As a comparison, the superset of the GT dataset comes from the execution of 342K malware samples in the sandbox of a security vendor, by an average time of 38.5 seconds. This means much larger TLS flows can be captured in less than an hour of network traffic.

2. Although the dataset is small, it has a very high dimension. The ratio between the number of features and the number of data samples is much greater than one. Table 1 described all the features which are used in the dataset. It is indeed a high number. This results in a dataset which is quite feature saturated and with a high dimension. This leads to the so-called 'The Curse of Dimensionality' problem [21]. Since most of the clustering algorithms work on the basis of various distance functions, including the ones used in the experiments above, this distance function for high-dimensional data becomes fairly complex and starts to produce spurious clusters. This is because rejecting a sample from a cluster becomes more and more difficult due very little difference or the same cut off values for each cluster centers [20].

The observations above highlight that the dataset is infact too small to evaluate not just clustering algorithms but any kind of supervised or unsupervised machine learning classifiers to detect malware traffic. Therefore, existence of a labelled large dataset is imperative for the task at hand. However, it's worth noting that like any other large dataset, building a labelled ground-truth is very difficult in terms of time complexity. It may even be extremely difficult in this case because of the existence of high-dimensional feature-rich dataset. This is because the interpretability of high dimensional data (>3D) is inherently difficult for labelling and datapoint belonging to similar clusters will require extreme precision.

4. BUILDING GROUND TRUTH

The 28 manually labelled samples of GT dataset took a lot of time and effort, giving us a realization that manually labelling network flows is not practical and expensive. Therefore, we propose an automated pipeline for building the labelled ground-truth from a large repository of captured network traffic. The goal of this pipeline is to produce a labelled dataset that can serve as a base-line if not complete ground-truth to evaluate supervised and unsupervised machine learning classifiers for classifying different types of malware traffic. Before we tap into the actual pipeline, the following subsections introduce some important famous lists of malware families and their features.

4.1 Tranco

Website popularity ranking is an important information for network and web security researchers to keep track of the well known websites in a represented sample. These kind of lists are important in setting up firewall-policies as well filtering strategies in various intrusion detection systems. However the validity of these lists are questioned because of the interest of adversaries to manipulate and replace malicious domains, a process security researchers often attribute as *domain polymorphism* [13, 22]. Le Pochat et al. [23] find that it is trivial for an adversary to manipulate the composition of these lists and insert malicious domains into these kind of white-lists. Therefore, they develop a reliable pipeline to output a ranking list of popular websites which are NOT malicious to be served as a reliable whitelist. The rankings produced by their method are reproducable [23]. They call this ranking list Tranco. This list is essential for filtering out connections to benign (not malicious) domains. We will use it for the same in our pipeline.

4.2 VirusTotal

VirusTotal [34] owned by Google is a free open-source database and service to identify malicious URLs, analyzed by over 70 antivirus databases. It provides a complete repository of all reported domains in the Virus Total community. This means that the VirusTotal database standing at approx. 20GB of malicious app & benign URLs and file hashes can be used to filter out benign domains from a network capture, leaving behind just the domains which are malicious for sure. Which virustotal being large enough should account for most of the benign domains in the web. We will use the VirusTotal database in a similar fashion in our pipeline.

4.3 Filtering Pipeline

The GT_dataset comes from a larger unlabelled dataset which contains 342K samples. This larger dataset is part of another in-house database which contains all 12M connections and 1.9M TLS flows. This database was populated by running a sandbox of a security vendor for several days. Overall, this dataset comprises of 1.9M TLS flows, of which 70% have a payload and roughly 10%are considered malicious (i.e., not filtered by Tranco). The top threedestination ports for TLS flows are 443/tcp (93.6%), 9001/tcp (2.67%),and 80/tcp (1.87%). The idea is to filter this database for samples with all TLS flows and filtering them to just malicious domains to label. The pipeline consists of the following phases:

1. Dumping pcap files identified by their hashes, features along with the effective second level domain (ESLD) which they connect to.

2. Filtering out pcap files with ESLDs which are in Tranco, which essentially means that removing pcap files that only have benign connections. This doesn't necessarily means removing all the benign connections as Tranco only keeps track of the popular benign domains.

3. Filtering out pcap files with ESLDs which are in VirusTotal(VT) database. Since the VT database is a large

open-source resource, one would assume that most of the benign domains both popular as well as unpopular and discreet ones would be removed.

4. Clustering pcap files which visit the same ESLDs and assigning them a unique name, inherently the name of the ESLD. This serves as partial clustering of the pcaps already which can later be used for evaluation.

5. Assigning malicious family labels to the clusters produced after filtering using a well known list of malicious domains in order to classify them as a particular family. This is done by comparing the ESLDs associated with well known virus families using a list of well known virus families.

Algorithm 2 GT_Cluster: Automated pipeline for filtering, clustering and labelling large repository of pcaps and network flows for building a ground-truth

```
Get all unique file_id, file_hash and ESLD from the in-house database.
for each file do
    if file.ELSD in Tranco then
        Remove the file from the dataset                    ▷ Phase 2
    end if
end for
for each file do
    for each entry in VT_json do
        if file in entry AND entry.positives == 0 then
            Remove the file from dataset                    ▷ Phase 3
        end if
    end for
end for
for each ESLD do
    Add all files to a cluster which visit this ESLD.
    Add this cluster to a list of clusters.
end for
for each cluster.ESLD do                                    ▷ Phase 4
    for each family in list of virus families do
        if cluster.ESLD exists in the list family.eslds then
            label this cluster with the name of this family
        end if
    end for
end for
```

Algorithm 2 shows the psuedocode for the pipeline divided in phases as described in the previous paragraph. The VT database is a very large json file containing entries of URL as well as pcaps reported as malicious. Each entry of the json is a data of the pcap file. In these pcaps if there is a connection to an ESLD which is labelled malicious then the detected flag is set to one in that entry. In this way for each ESLD in our Tranco filtered pcaps we can check for that ESLD in VT with the detected flag equal to zero and remove those files from the dataset. In this way we produce a list of pcap files having connections to only malicious domains with a high degree of confidence [23,34]. The output of the above pipeline is a dataset containing clusters of pcaps with all their features and the virus family they belong to as labels. This will serve as a base-line if not a ground-truth to evaluate any supervised or unsupervised machine learning classifiers which as described was the goal. For unsupervised algorithms, instead of using a label, a unique hash based on the contents of the produced cluster can be used to identify the virus family.

domain	total size of cluster	virus family
friendlyduck.com	9838	Cerber
yolox.net	8391	Bublik
maitikio.com	6132	-
cry-havok.org	5491	Upatre
chip-secured-download.de	4910	-
ax100.net	3806	-
Total # of clusters: 17622		

Table 4: Top 6 clusters in the output based on the number of pcaps out of 17,622.

Port	Conns	% of the total	IANA
443	961150	98.7130	https
9001	4921	0.5054	etlservicemgr
80	3524	0.3619	http
8200	761	0.0782	trivnet1
6983	249	0.0256	-
110	191	0.0196	pop3
6829	175	0.0180	-
16838	166	0.0170	-
12889	146	0.0150	-
10656	143	0.0147	-
5757	140	0.0144	x500ms
16465	136	0.0140	-
4490	125	0.0128	-
21	124	0.0127	ftp
21844	108	0.0111	-
27568	96	0.0099	-
9101	79	0.0081	bacula-dir
23052	64	0.0066	-
1863	54	0.0055	msnp
5119	53	0.0054	-
Total	972405	99.8689	9 known services

Table 5: Top 20 used ports in 17,627 clusters having 973,681 malware connections wiht payload.

Resulting Dataset

We ran the above proposed pipeline on the in-house dataset as described in the previous section and the resulting dataset which was obtained contained 17,622 clusters with the top 15 containing over 3000 pcap files. The distribution of clusters also provide an idea of the top domains visited by the pcaps as well as the virus families. Table 4 shows the top 6 clusters with their ESLDs and the virus family associated with theses clusters. Table 5 shows the top 20 ports and services used by these malware connections in the resulting dataset.

5. DISCUSSION

Malware families are growing everyday and due to domain polymorphism, i.e constant changes in the malware domains it is challenging to detect and classify them even in a partially intelligent network intrusion detection system. Moreover, Lack of large labelled dataset hinders the research in this domain. Our work tries to bridge this gap by proposing an automated pipline for building a base-line ground-truth for evaluation. We believe that the pipeline has a large usage potential mainly due to the following reasons. Firstly, it uses two open-source resources for filtering out pcaps which are benign, namely Tranco and VirusTotal. Conjunction of these two cover most of the benign domains both popular and unpopular ones. Therefore one can be certain to a high degree of probability that the remaining pcaps will all be just malicious. Secondly, it requires a large repository of network capture files which also is not difficult to obtain. Any enterprise network traffic or a sandbox ran in a network environment for a few days would suffice. Moreover there already are many unlabelled network capture datasets available on the internet such as Andrubis [24] etc. which can also be used. Lastly,

it requires a list of known virus families. This list is context or application dependent. However, these are also available in a large number over the internet which are regularly updated and validated. All these points show that the pipeline fulfils a need in the cyber-security research [2,4,35]. Since the clustering approach described in the pipeline uses all network features including temporal ones such as packet length, time to transmit etc. it is not dependent on the TLS version and can be applied to all of them. It is also not just restricted to HTTPS, but works on any other port or service as also shown in Table 5.

We would like to point out that a number of improvements to the pipeline can be made by exploiting several malware and network heuristics, yielding somewhat better data-set or clusters. For example, TLS certificates can be used in addition to SNI as well as ESLDs to merge similar clusters into one. In addition to that multiple ESLDs associated with the same virus family can form a super-cluster (or a hierarchical cluster) which then can be used to compare with hierarchical clustering algorithms directly, without the need to flatten them.

However, the ground-truth that was generated via the pipeline above was unfortunately not used to evaluate the clustering algorithms described in the first section due to lack of time. As this research was part of my internship which had a limited time-frame. Nonetheless, it is being used in a later research (not part of the internship) continuing the same work.

6. CONCLUSION

The report presents experiments using different clustering algorithms with different hyper-parameters and heuristic merging to classify various families of malicious network traffic on a small initial dataset. The evaluation of these algorithms yields a realization that a small dataset fails to capture the inherent features of these malware families and it's ability to generalize. The report then presents an automated labelled ground-truth generation pipeline for malware traffic using any large repository of malicious network traffic or an existing large dataset, Tranco [23] and VirusTotal [34]. The report also presents the application of the pipeline and the features of the resulting dataset and shows it's ability to generalize over a diverse set of virus families. We will also release the code of the pipeline to research community with hope that it will be useful and will bridge the gap in using modern artificial intelligence and machine learning algorithms in network security research. We leave it to future research to exploit the improvements indicated in the previous section to augment the pipeline as well as the clustering algorithm to yield a better and diverse ground-truth dataset as well as an efficient generation pipeline.

7. ACKNOWLEDGEMENT

This work was done as part of my internship as a software research project at IMDEA Software Institute and Polytechnic University of Madrid (UPM), in Madrid. The resources used for this research both hardware and software were provided by institute. I would like to acknowledge Mr. Gibran Gomez of the institute for providing me with the initial dataset which was part of his masters thesis [28] at UPM.

8. REFERENCES

[1] S. Anand, S. Mittal, O. Tuzel, and P. Meer. Semi-supervised kernel mean shift clustering. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1201–1215, 2013.

[2] B. Anderson, A. Chi, S. Dunlop, and D. McGrew. 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*, CODASPY '19, pages 267–278, New York, NY, USA, 2019. ACM.

[3] B. Anderson and D. McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, pages 35–46, 2016.

[4] B. Anderson 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*, pages 1723–1732, 2017.

[5] B. Anderson, S. Paul, and D. McGrew. Deciphering malware's use of tls (without decryption). *Journal of Computer Virology and Hacking Techniques*, pages 1–17.

[6] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *USENIX security symposium*, volume 11, pages 1–16, 2011.

[7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: detecting the rise of dga-based malware. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 491–506, 2012.

[8] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Ndss*, pages 1–17, 2011.

[9] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[10] P. S. Bradley, U. Fayyad, C. Reina, et al. Scaling em (expectation-maximization) clustering to large databases. 1998.

[11] F. Callegati, W. Cerroni, and M. Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[12] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A bayesian classification system. In *Machine learning proceedings 1988*, pages 54–64. Elsevier, 1988.

[13] Y. Cohen and D. Hendler. Scalable detection of server-side polymorphic malware. *Knowledge-Based Systems*, 156:113–128, 2018.

[14] M. Dell'Amico. Fishdbc: Flexible, incremental, scalable, hierarchical density-based clustering for arbitrary data and distance. *arXiv preprint arXiv:1910.07283*, 2019.

[15] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. 2008.

[16] J. H. Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

[17] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. 2008.

[18] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security*

Symposium, volume 7, pages 1–16, 2007.

[19] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

[20] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[21] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[22] P. Kotzias, L. Bilge, P.-A. Vervier, and J. Caballero. Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises.

[23] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, Feb. 2019.

[24] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen, and C. Platzer. Andrubis–1,000,000 apps later: A view on current android malware behaviors. In *2014 third international workshop on building analysis datasets and gathering experience returns for security (BADGERS)*, pages 3–17. IEEE, 2014.

[25] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254, 2009.

[26] L. McInnes, J. Healy, and S. Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.

[27] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Symposium proceedings on Communications architectures and protocols*, pages 123–133, 1988.

[28] G. Montes. Detecting and classifying malicious tls network traffic using machine learning. Julio 2018.

[29] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[30] V. Paxson, R. Sommer, S. Hall, C. Kreibich, J. Barlow, G. Clark, G. Maier, J. Siwek, A. Slagell, D. Thayer, et al. The bro network security monitor, 2012.

[31] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer networks*, 53(6):864–881, 2009.

[32] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[33] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 349–360, 2012.

[34] V. Total. Virustotal-free online virus, malware and url scanner. *Online: https://www. virustotal. com/en*, 2012.

[35] T. van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. van Steen, and A. Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium, NDSS 2020*. Internet Society, 2020.

[36] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*, pages 250–257. IEEE, 2005.