End-node Fingerprinting for Malware Detection on HTTPS Data

Tomáš Komárek
Czech Technical University in Prague
Faculty of Electrical Engineering
komartom@fel.cvut.cz

Petr Somol*
Cisco Systems (Czech Republic)
Advanced Threat Group
psomol@cisco.com

ABSTRACT

One of the current challenges in network intrusion detection research is the malware communicating over HTTPS protocol. Usually the task is to detect infected end-nodes with this type of malware by monitoring network traffic. The challenge lies in a very limited number of weak features that can be extracted from the network traffic capture of encrypted HTTP communication. This paper suggests a novel fingerprinting method that addresses this problem by building a higher-level end-node representation on top of the weak features. Conducted large-scale experiments on real network data show superior performance of the proposed method over the state-of-the-art solution in terms of both a lower number of produced false alarms (precision) and a higher number of detected infections (recall).

CCS CONCEPTS

•Networks → Network security; *end-nodes;* **•Security and privacy → Intrusion detection systems;**

KEYWORDS

HTTPS data, Malware detection, Supervised learning

ACM Reference format:
Tomáš Komárek and Petr Somol. 2017. End-node Fingerprinting for Malware Detection on HTTPS Data. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 7 pages.
DOI: 10.1145/3098954.3107007

1 INTRODUCTION

Increasing number and sophistication of attacks against critical enterprise computing infrastructure drives the need to deploy increasingly more sophisticated defense solutions. An essential component of the defense are Network Intrusion Detection Systems (NIDS) analyzing network traffic that crosses the defense perimeter and looking for evidence of ongoing malicious activities. Many current NIDS use supervised learning algorithms to analyze traffic captures for possible malware infections. Such systems are first trained on reference samples of malicious and benign activity to be later able to detect incidents of similar characteristics automatically. These systems can analyze various network traffic properties (NetFlow records, web proxy logs, etc.).

This paper focuses solely on analysis of web proxy logs that are containing information about the web traffic generated by end-nodes in the monitored network. Recently developed malware typically tunnels the communication over HTTP/HTTPS protocol and makes the malicious traffic masquerade as web traffic, since this essential channel is left open in the majority of networks. Because of that, proxy logs have been shown to contain enough information to enable detection of various forms of exploitations, malicious command & control channels and other attacks that are difficult or impossible to discover using standard content-analysis-based security solutions [7, 8, 12, 13]. These high-level log records represent relatively lightweight source of data that can face scalability issues caused by constantly rising volume of network traffic. Contrary, solutions based on deep packet inspection (DPI) are often computationally prohibitive and/or not applicable on large networks.

NIDSs built around proxy logs usually extract a set of features from each HTTP/HTTPS request recorded in the data to create a feature vector that is subsequently classified as either malicious or benign. Increasing usage of encrypted web communication via the HTTPS protocol, however, makes this approach ineffective, since the amount of information that can be extracted from HTTPS log records is heavily reduced. The majority of the most informative HTTP related log fields (URL, User-Agent, MiMe type, etc.) becomes inaccessible after the encryption and the remaining ones, namely the timestamp, sent/received bytes and the duration of the communication, are very weak for a reliable malware detection. Malware writers are aware of this fact and use communication via HTTPS as an effective detection evasion technique.

One way to deal with this problem is to intercept HTTPS communication on the proxy, decrypt it, log the information needed and encrypt it again (MITM, man-in-the-middle). While this mechanism enables to use the same detection technique as in the case of HTTP data, it suffers from two main drawbacks. MITM is computationally expensive and the privacy of users is compromised. Both issues render the mechanism problematic for a practical deployment and alternative ways are of a great need.

Recent work of Kohout *et al.* [9] demonstrates that many applications exhibit unique communication patters observable through timing and transfered bytes that can be learned, modeled and identified in network traffic. These patterns are detectable across multiple log records as opposed to single records. To enable their analysis with standard machine learning tools, whole sets of log records are transformed into single feature vectors using histogram-based fingerprinting. This strategy has been successfully used to detect domains related to malicious activity [10, 11] and infected end-nodes [4].

*Petr Somol is also with Institute of Information Theory and Automation, Czech Academy of Sciences.

ARES '17, Reggio Calabria, Italy

978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3107007

The main contribution of this paper is a novel fingerprinting method that we suggest as an alternative to the histogram-based one. The second contribution is an empirical large-scale evaluation on real network data that shows that the proposed solution achieves significantly better efficacy results. The evaluation is done on the task of detecting infected end-nodes as proposed by Čech *et al.* [4].

The paper is organized as follows. Section 2 describes the setup of the problem in detail. Section 3 presents our solution to that problem and Section 4 the state-of-the-art solution. Both the new and the old HTTPS-specific solutions together with the traditional approach adopted from the detection on non-encrypted HTTP data are compared in Section 5. Section 6 concludes the paper.

2 PROBLEM SETUP

The aim is to detect end-nodes (i.e., network hosts) of a computer network that are infected with malware communicating over HTTPS protocol. It is assumed that the only accessible data for the detection are high-level log records about the requests of individual network end-nodes for establishing their HTTPS connections.

Let $\mathcal{E}^i$ denote i^{th} end-node of a computer network. Within a given period of time, each end-node can initiate a set of connections. The requests for establishing connections are typically monitored by a proxy server located on the perimeter of the computer network and captured in the form of logs.

$$\mathcal{E}^i = \{\mathbf{x}_1^i, \mathbf{x}_2^i, \dots\}, \tag{1}$$

where vector $\mathbf{x}_j^i \in \mathbb{R}^n$ represents j^{th} log record of i^{th} end-node. Elements of the vector then correspond to features that are extracted from fields of j^{th} log record.

A traditional approach to build a detection system $\mathcal{H}$ searching for infected end-nodes is the following. First, training set $\mathcal{D}$ of feature vectors along with their label $y \in \{0, 1\}$ indicating whether a particular log record is related to malware communication ($y = 1$) or not ($y = 0$) is collected.

$$\mathcal{D} = \{(\mathbf{x}_1^1, y_1^1), (\mathbf{x}_2^1, y_2^1), (\mathbf{x}_1^2, y_1^2), \dots\}. \tag{2}$$

Second, supporting classification model $h(\mathbf{x}) = y$ is inferred from the training set. For this purpose, any common supervised learning algorithm such as Neural Networks, Random Forests or Support Vector Machines can be used [6]. Finally, the detection system operating on the level of end-nodes is constructed.

$$\mathcal{H}(\mathcal{E}^i) = \max_j h(\mathbf{x}_j^i), \tag{3}$$

where end-node $\mathcal{E}^i$ is classified as infected when at least one of the feature vectors extracted from its traffic is classified as malicious.

Although this approach has been shown to be successful on HTTP logs (i.e., traffic captures of non-encrypted communication), it does not achieve a satisfying performance on HTTPS data. This is implicitly presumed in previous works [4, 10, 11] and verified by our experiments in Section 5. The reason is that individual HTTPS log records do not carry enough discriminatory information in contrast to HTTP ones. The information available from HTTPS records is limited just to transfered bytes and timestamps because of the encryption.

Kohout *et al.* [9] propose to extract four features (i.e., $n = 4$) from each log entry, namely: **uploaded bytes** x_{up} from the end-node to a target server, **downloaded bytes** x_{down} by the end-node from the server, **duration** x_{dur} (in seconds) for which the communication was active and **inter-arrival time** x_{iat} (in seconds) elapsed between two consecutive requests for establishing communication between the same end-node and server.

$$\mathbf{x} = (\hat{x}_{\text{up}}, \hat{x}_{\text{down}}, \hat{x}_{\text{dur}}, \hat{x}_{\text{iat}}), \quad \hat{x} = \log(1 + x). \tag{4}$$

The logarithmic transformation is applied to suppress noise and decrease ranges of the features. The same set of four features is used in the work of Čech *et al.* [4] and in this paper.

The challenge of HTTPS traffic analysis is emphasized by comparison to the richness of information content in unencrypted HTTP. If available, HTTP logs allow for extracting much richer sets of features including URL string textual features, User-Agent features, request parameter key-value features, etc. In contrast, none of this information is available in this setup.

To make HTTPS data with the weak features applicable for malware detection, the individual log records can no longer be treated separately. Therefore in this paper, similarly to Čech *et al.* [4], one classification sample is defined as a set of all log entries that belong to the same end-node and that were generated during a five minute time interval (Equation 1). The training set then has the form

$$\mathcal{D}' = \{(\mathcal{E}^1, y^1), (\mathcal{E}^2, y^2), \dots\}, \quad y^i = \max_j y_j^i, \tag{5}$$

where an end-node is considered to be infected if it contains at least one request issued by a malware binary. Otherwise, the end-node is considered to be clean.

This definition of classification samples preserves the information about repeatedly established connections, co-occurring connections or other possibly interesting contextual information. On the other hand, this way defined samples can not be directly used for training a classification model, because standard learning algorithms require samples represented by single vectors of a fixed dimension and not sets of vectors. A simple concatenation of vectors would not work, since the number of generated log records varies across individual end-nodes.

A possible solution to this problem of variable end-node body, is to define a *fingerprinting* function

$$\mathcal{F}(\mathcal{E}) = \mathbf{e} \in \mathbb{R}^d \tag{6}$$

transforming end-nodes $\mathcal{E}$ into vectors $\mathbf{e}$ of d real numbers. Using this function, every end-node can be mapped from the set space to the vector space. Consequently, any off-the-shelf algorithm for supervised learning can be used to train the detection system on top of that new vectorial representation.

$$\mathcal{H}(\mathcal{E}^i) = h(\mathcal{F}(\mathcal{E}^i)). \tag{7}$$

The crucial part of this approach, however, remains in the definition of the fingerprinting function. It has to be designed carefully such that it extracts maximum useful information for the subsequent classification.

3 PROPOSED SOLUTION

In this section we propose a fingerprinting method that encodes variable sets of four-dimensional vectors (i.e., end-nodes) into single vectors of a fixed dimension.

The method is based on counting strategy. Given a fixed number of pre-selected regions in the feature space, every end-node is represented as the number of its log records residing at each region. In this way it is ensured that the fingerprint dimensionality, which equals to the number of regions, remains the same for an arbitrarily large set of log records.

As illustrated in Figure 1, the regions are two-dimensional parallel-axis rectangles. These 2D rectangles are part of lattices that are quantizing individual 2D sub-spaces of the original 4D feature space. The reason behind using 2D sub-spaces instead of 1D, 3D or 4D is a practical balance between expressiveness and robustness of the representation. 2D regions already capture relations between two features, yet the relations are not affected by any changes in other two features. For example, the first five elements of the fingerprint in Figure 1 that capture the end-node activity from transferred bytes viewpoint, would remain unaffected, even if values of time related features were markedly shifted. In the case of 3D or 4D regions, the log records would be, however, residing at different regions after the shift, which would result in a completely different fingerprint. Since the shift of time related features can be caused independently on the end-node behavior, e.g., by a higher load of intermediate nodes in the network, and at the same time we do not want to lose the information from these features completely, we found this solution of 2D sub-spaces to be useful.

The next two sub-sections describe the fingerprinting function and the procedure for selecting the rectangles in detail.

3.1 Fingerprinting using 2D projections on parallel-axis rectangles

The proposed transformation $\mathcal{F}$ encodes end-node behavior $\mathcal{E}$ into compact vector (fingerprint) $\mathbf{e}$ of size d.

$$\mathcal{F}(\mathcal{E};\mathbf{\Phi}) = \mathbf{e} \in \mathbb{N}^d. \tag{8}$$

The encoding process is parametrized by $\mathbf{\Phi}$ representing the pre-selected 2D rectangles, each defined by a pair of feature indexes and value intervals specifying boundaries on the two indexed features. $\mathbf{\Phi}$ is thus an ordered list of length d occupied by 6-tuples $\mathbf{T}_t$.

$$\mathbf{\Phi} = (\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_d),\ \mathbf{T}_t = (f_{\text{idx}}, f_{\text{low}}, f_{\text{up}}, s_{\text{idx}}, s_{\text{low}}, s_{\text{up}})_t. \tag{9}$$

The individual tuples $\mathbf{T}_t$ represent rectangles in a 2D space. The 2D space is always a sub-space of the original four-dimensional feature space (Equation 4). Edges of these rectangles are aligned with coordinate axises of particular 2D sub-spaces. Therefore, each tuple $\mathbf{T}_t$ has values $f_{\text{idx}}, s_{\text{idx}} \in \{\hat{x}_{\text{up}}, \hat{x}_{\text{down}}, \hat{x}_{\text{dur}}, \hat{x}_{\text{iat}}\}, f_{\text{idx}} \neq s_{\text{idx}}$ determining the first and the second feature of a chosen 2D sub-space, respectively. Furthermore, to determine an exact position of a rectangle in the chosen 2D sub-space, the tuple also has values $f_{\text{low}}, s_{\text{low}} \in \mathbb{R}$ and $f_{\text{up}}, s_{\text{up}} \in \mathbb{R}$ specifying lower and upper boundaries on the selected features, respectively. A procedure for obtaining $\mathbf{\Phi}$ is described in Section 3.2.

Given a list of such rectangles together with a sample of an end-node behavior that is sought to be encoded, the fingerprinting process counts for every rectangle the number of log entries inside the sample that fit to that rectangle. As a result, elements of the output vector (i.e., fingerprint) correspond to these counts. See Algorithm 1 for the description of this process in the form of pseudo-code.

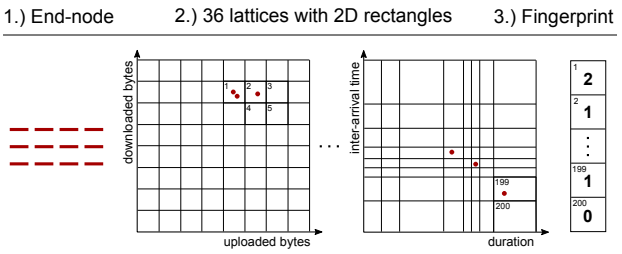

Figure 1: The proposed fingerprinting function in three phases. The first phase shows an example of end-node activity with three log records, where each log record is described by four features. In the second phase, it is verified whether some log records fit to any of 200 pre-selected 2D rectangles. These rectangles are part of 36 2D lattices quantizing the original 4D feature space from various aspects. The lattices differ in the chosen sub-space, applied discretization method or in the density at which the particular sub-space is quantized. The final fingerprint is then constructed as the number of log records in each of the selected rectangles.

3.2 Procedure for obtaining rectangles

In this sub-section we describe a procedure that constructs list of rectangles $\mathbf{\Phi}$ based on properties of the training data. It is a top-down approach consisting of two phases. In the first phase — *candidate rectangles generation* — an exhaustive set of rectangles covering the feature space with many overlapping rectangles of various sizes is generated. In the second phase — *candidate set optimization* — the set is reduced to produce required list $\mathbf{\Phi}$ of rectangles covering only those regions that are important for the classification.

Candidate rectangles generation The aim of this phase is to generate rectangles that could potentially serve as new discriminative features in the resulting vectorial representation. As it is not known at this stage, which regions can produce useful features, the intention is to generate many rectangles of various sizes with the hope that some of them will produce the discriminative features. The complete set of possible rectangles would be infinite. In practice we therefore resort to approximate the complete set of candidate

Algorithm 1: Fingerprinting Method

Input : End-node $\mathcal{E}$ and list of 6-tuples $\mathbf{\Phi}$
Output: Vector $\mathbf{e}$ representing behavior of end-node $\mathcal{E}$

Function *fingerprinting* $\mathcal{F}(\mathcal{E};\mathbf{\Phi})$
  $\mathbf{e}[1 \ldots \text{length}(\mathbf{\Phi})] \leftarrow 0$
  foreach $(f_{\text{idx}}, f_{\text{low}}, f_{\text{up}}, s_{\text{idx}}, s_{\text{low}}, s_{\text{up}})_t \in \mathbf{\Phi}$ **do**
    foreach $\mathbf{x} \in \mathcal{E}$ **do**
      if $f_{\text{low}} \leq \mathbf{x}[f_{\text{idx}}] \leq f_{\text{up}}$ *and* $s_{\text{low}} \leq \mathbf{x}[s_{\text{idx}}] \leq s_{\text{up}}$ **then**
        $\mathbf{e}[t] \leftarrow \mathbf{e}[t] + 1$
  return $\mathbf{e}$

rectangles with quantization of 2D sub-spaces into several lattices with intervals estimated from the training data.

Specifically, for each feature pair out of six possible combinations

$$(\hat{x}_{\text{up}}, \hat{x}_{\text{down}})_1, (\hat{x}_{\text{up}}, \hat{x}_{\text{dur}})_2, (\hat{x}_{\text{up}}, \hat{x}_{\text{iat}})_3, \\ (\hat{x}_{\text{down}}, \hat{x}_{\text{dur}})_4, (\hat{x}_{\text{down}}, \hat{x}_{\text{iat}})_5, (\hat{x}_{\text{dur}}, \hat{x}_{\text{iat}})_6, \tag{10}$$

several lattices that quantize the selected 2D sub-space is constructed. Internal regions of these lattices then serve as the candidate rectangles. The number of such lattices is determined by the number of different methods used for the quantization and by the number of different densities at which the features (i.e., coordinates of the 2D sub-space) are discretized with each of these methods.

In particular with regard to our experiments in Section 5, for each feature pair both feature ranges are divided into either 8 or 16 intervals. These two densities of quantization form two different types of lattices with 64 and 256 rectangles. We employ three discretization methods (described in the next paragraph) to capture various aspects of patterns in the data. In total, there are 6×2×3 = 36 distinct lattices including $6 \times (64 + 256) \times 3 = 5760$ candidate rectangles.

The first employed method — *uniform quantization* — partitions the feature space uniformly. It divides a range of a given feature into a desired number of equally sized bins. It is a pure unsupervised method that use the training data only for estimating range bounds of the feature. The next two methods — *quantile quantization on positive/negative data* — take into consideration probability of log occurrence with respect to the class label. The higher the probability is, the finer partitioning is performed. This is achieved by dividing the feature range according to estimated quantiles on the data of both classes separately.

Candidate set optimization In this phase the final list of rectangles is to be selected as a subset of the candidate set so as to maximize the amount of discriminative information extracted by the eventual fingerprinting method.

Once the candidate rectangles get collected, they can be used to transform training set $\mathcal{D}'$ into fingerprints. The new vectorial representation would, however, expectedly contain a mix of useful features with irrelevant ones. To avoid the danger of over-fitting and to reduce the computational time during both the training and the testing part, we propose to select only discriminative features from these fingerprints that are generated with the usage of the candidate set. Thus, the candidate set optimization problem can be viewed as a form of standard feature selection (FS) problem [3], which can be solved by an existing FS method.

For the purpose of this paper we verified that even a simple FS method such as Mutual Information (MI) [14] is a viable option. MI assigns a score to each feature according to measure of the mutual dependence in between the feature and the class labels. Therefore, the final list of rectangles Φ is determined on the basis of a graph showing the sorted features according to the assigned scores (Figure 2).

4 RELATED WORK

This section is dedicated to related works that take advantage of supervised learning in combination with high-level network traffic log records in order to detect malware infections.

4.1 Detection on HTTP data

A vast majority of works employs the traditional approach, where the detection is done on per-log record basis [7, 12, 13]. While in that case the feature vectors (as extracted from log records) are used directly as the input vectors for the classification, Bartoš *et al.* [1] show that a classification model trained on fingerprints rather than individual low-level vectors achieves higher recall at comparable precision. The particular transformation is designed so that the output representations are invariant under shifting and scaling of the input feature values and under permutation and size changes of the input sets. However, the transformation can be used to represent individual connections only. By a connection, it is understood a set of records related to the same end-node and server instead of a number of servers. Thus, the information about repeatedly established connections or their co-occurrence is not preserved in the classification samples. Furthermore, the proposed features can be extracted from log records of non-encrypted HTTP communication only.

4.2 Histogram-based fingerprinting

As the closest prior art we consider the work of Kohout *et al.* [9]. Their statistical fingerprinting method relies on four already described features in Equation 4 that can be extracted from HTTPS log records. Individual log records within a given classification sample are treated as realizations of a four-dimensional random variable. To represent a distribution of that variable as a vector, a joint four-dimensional histogram is constructed. Specifically, a smoothed histogram is constructed, in which contribution of each log entry is distributed among multiple bins instead of just one. This intervention helps to smooth out small disturbances in the data. Bins of that histogram are centered at equidistant points of four-dimensional lattice

$$L = \{0, \ldots, 11\}^4. \tag{11}$$

These histograms are represented as sparse feature vectors (i.e., fingerprints) with the dimension $d = 11^4 = 14,641$. The technique is unsupervised and does not require optimization of any parameters.

The histogram-based fingerprinting is demonstrated on the task of grouping web servers that are hosting similar applications [9]. Recently, the same representation has been used to detect domains related to malicious activity [10, 11] and to identify infected end-nodes [4]. Therefore, the histogram-based fingerprinting represents the state-of-the-art method that the herein proposed solution is compared to in Section 5.

Before presenting results from the comparison, we would like to highlight several differences between the histogram-based and the herein proposed fingerprinting. First, the histogram-based method quantizes the feature space into 4D bins, while the proposed method into 2D bins (i.e., rectangles). As mentioned in Section 3, the advantage of 2D rectangles over 4D bins is the robustness to arbitrarily large changes in one or two input features. Second, the 4D bins of histograms are obtained by quantizing the feature space with one 4D lattice (Equation 11), whereas the proposed solution tries to minimize the risk of an inappropriately discretized feature space by quantizing each of the six 2D sub-spaces several times with various 2D lattices. Third, elements of a histogram-based fingerprint sum

Table 1: Dataset Specifications

Dataset	Training	Testing
Time period	March 2016	April 2016
Infected batches	2,800	1,578
Clean batches	43,813,135	31,607,364
Positive log records	17,574	17,938
Negative log records	1,346,888,662	895,445,264

to one because of the normalization condition. This property might suppress the information about malicious behavior when a benign activity strongly dominates in a particular time window. It is a well-known evasion technique when malware tries to intentionally communicate in the same time window as a legitimate process in order to make an eventual detection harder. In the case of the proposal, there is no such property and the elements are stored as non-negative integers.

5 EXPERIMENTS

In this section we present results from an experimental evaluation on the task of detecting end-nodes that are infected with malware communicating over HTTPS. The evaluation is performed on real network data described in the next section. The herein proposed solution is compared to the state-of-the-art fingerprinting method as well as to the traditional approach classifying end-nodes based on analysis of isolated log records.

5.1 Dataset

The used dataset contains HTTPS log records from 15 computer networks of international companies of various types and sizes. They are collected during the time period of two months, namely March and April of 2016, using Cisco Cognitive Threat Analytics[1]. In total, there are over two billions records produced by more than half million of unique end-nodes. See Table 1 for details about the dataset. The first month is used as the training dataset, whereas the later one serves for testing purposes only.

Every log record consists of end-node's identifier, server IP address, number of sent and received bytes, duration of the connection and the timestamp indicating when the connection has started. We adopted the set of four features (Equation 4) from [9] that are extracted from each log entry.

Besides the mentioned fields, some logs are provided with a hash of the process that has initiated the HTTPS connection. These hashes are compared against the database of malware hashes maintained by VirusTotal service[2]. If a hash is identified by at least 20 anti-viruses used by VirusTotal, the log record is considered to be positive (i.e., malicious). In other cases, the log is treated as negative (i.e., benign).

The availability of these hashes depends on whether Cisco AnyConnect Secure Mobility Client[3] is running on end-nodes, which does not have to be so in all cases. Although the networks are supposed to be well maintained, we admit, therefore, that in the dataset there still might remain undiscovered malicious records that are actually mislabeled with the negative label.

Similarly to Čech *et al.* [4], the end-nodes are analyzed in five minute batches. Consequently, one classification sample $\mathcal{E}^i$ corresponds to the set of end-node's HTTPS logs generated in a particular five minute time period. A batch is infected when it includes at least one positive record. Otherwise, the batch is considered to be clean.

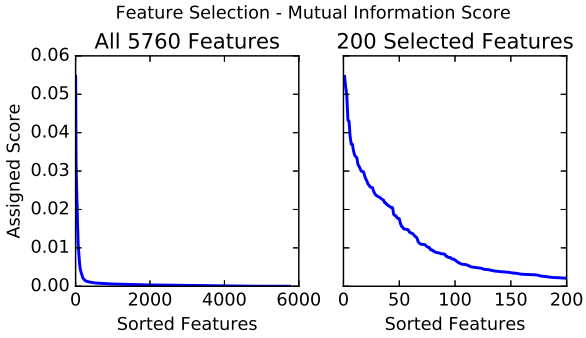

Figure 2: Feature Scores. Both graphs show the same data, the left view covers all features, the right view zooms to the first 200 features. The 200 features induce the selected rectangles from the generated candidate set (Section 3.2).

5.2 Experimental setting

The following three methods are evaluated on the presented dataset.

1. Isolated logs This method represents the traditional approach where a classifier is trained on isolated feature vectors as extracted from log records. Batches are then classified as infected when at least one of inner records is identified as positive (Equation 3).

2. Histograms Histogram-based fingerprinting is the state-of-the-art approach on HTTPS data described in Section 4. This technique encodes every batch into a vector of 14,641 real numbers.

3. Proposal The novel technique for end-node fingerprinting is implemented following the description in Section 3.

The initial set of the 5,760 candidate rectangles is reduced to 200 using Mutual Information feature selection. As can be seen from Figure 2, there is an exponential decay in the assigned scores by the FS method. Therefore, only the first 200 effective rectangles are utilized, resulting in fingerprints of 200 natural numbers.

Figure 3 illustrates a distribution of the selected rectangles with respect to the 36 parent lattices created on six feature pairs by using three discretization methods and two densities. It can be observed that even though every lattice produces some useful rectangles, the most used ones on average are those generated with — *quantile quantization on positive data* — at the finest density of 256 rectangles per a lattice. It can be also seen that the sixth feature pair F6 (i.e., the sub-space of time related features) has a lower number of informative rectangles than the first feature pair F1 (i.e., the sub-space of transferred bytes). We attribute this difference to the

[1] https://cognitive.cisco.com/
[2] https://www.virustotal.com/
[3] https://ist.mit.edu/cisco-anyconnect

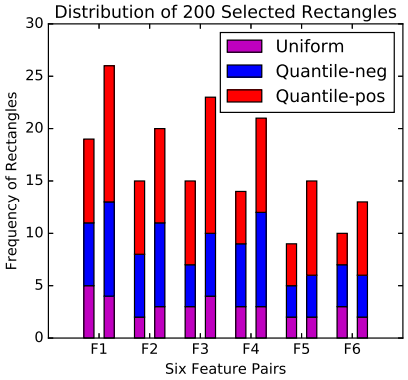

Figure 3: Properties of selected rectangles. The sequence of six feature pairs respects the order in Equation 10. Colors indicate the used quantization method. Left and right bars reveal the density (64 and 256, respectively) of the parent lattice.

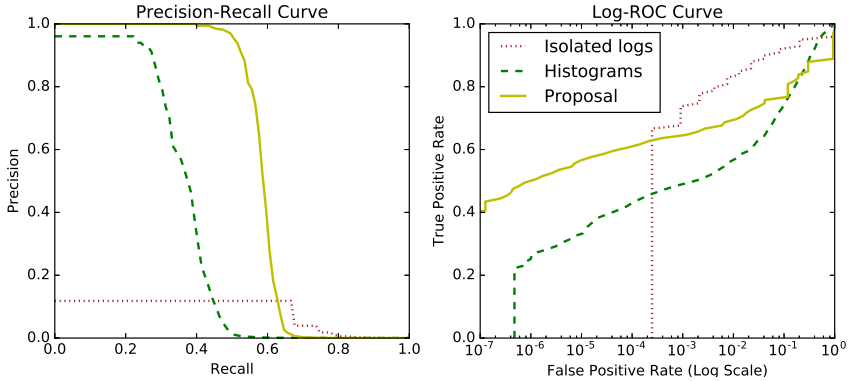

Figure 4: Results from an experimental evaluation on the testing set presented in terms of Precision-Recall and ROC curves. Please note that the ROC curve has a logarithmic scale on the x-axis to magnify the area of interest with low false positive rate.

instability of time related features mentioned in Section 3.

All three methods produce classification samples in the form of vectors. To compare the methods, a classification model is trained and evaluated on respective samples of each method separately.

We opt for Random Forests (RF) [2] as this model is known to perform well in the majority of practical learning problems [6]. The number of trees to grow and their maximum depth are fixed to 128 and 30, respectively. The only hyper-parameter of the model that is tuned for each method separately is the number of randomly selected attributes at each split. The appropriate values are selected from a list specific for each method: *Isolated logs* - $\{1, 2, 3, 4\}$, *Histograms* - $\{10, 100, 500, 1000\}$, *Proposal* - $\{10, 100, 150, 200\}$; by measuring out-of-bag error on the training data. The rule of thumb for setting this parameter on classification tasks is square root of the dimension.

All compared methods are trained using 2,800 infected batches and two millions of clean ones. The non-infected batches are uniformly sampled from the complete set in order to avoid out of memory issues during the training. Although we were able to use the complete training set in the case of the proposed method, since the fingerprints are compact and can be effectively stored in memory using only 8 and 16 bit unsigned integers, we have not observed any significant improvements in efficacy. Therefore, we consider the used subset of clean batches as a representative sample of end-node benign activity.

5.3 Efficacy results

The results are presented in Figure 4 in terms of Precision-Recall (PR) and Receiver Operating Characteristic (ROC) curves [5]. The PR curves show whether every positive prediction is indeed infected batch (precision) and at the same time whether the model detects as many infected batches as possible (recall also called true positive rate). Thus perfect recall (i.e., the score equals to 1.0) means that all infected batches are detected, while perfect precision means that there are no false alarms. The ROC curves then indicate how much are network administrators flooded by the individual systems with meaningless false alarms (false positive rate) at various levels of model recall. Considering the strong prevalence of clean end-nodes, the emphasis in the network security domain is put on extremely low false positive rate and high precision at a reasonable recall [15].

As can be seen from Figure 4, the traditional technique — *Isolated logs* — adapted from the detection on HTTP data does not achieve a satisfying efficacy. It produces too many false alarms at any operational point. Network administrators of such system would spend an enormous amount of expensive time by examining many reported incidents only to eventually determine that they mostly reflect benign end-node activities.

The classification model based on the histogram fingerprinting exhibits false positive rate reduced by two orders of magnitude when compared to the traditional approach at 20% recall, which is remarkable. This emphasizes the advantage of the fingerprinting approach.

The proposed solution, however, improves the detection capabilities even further. It enables to detect about twice as many infected batches at the same or even higher precision in the area with low false positive rate. The large decrease in precision after 50% recall indicates an inability to reliably detect more than a half of infections contained in the dataset. Considering the fact that only four simple features are used for the detection, it is reasonable to not expect the approach to detect all infections with a high confidence.

Overall, the proposed solution exhibits an improved detection performance in precision as well as in recall when compared to the state-of-the-art method.

6 CONCLUSION

This paper addresses the challenging problem of detecting end-nodes that are infected with malware binaries communicating over HTTPS protocol. Since only a limited number of features can be extracted from HTTPS data without using MITM (man-in-the-middle) mechanism or DPI (deep packet inspection), we introduce a solution based on the idea of modeling communication patterns

observable through timing and transfered bytes. These patterns are primarily detectable across multiple connection requests issued by individual end-nodes. To encode the variable end-node activity into a single vector, enabling its direct use in supervised learning, we propose a novel fingerprinting method. The method is designed such that the output fingerprints are robust to changes in one or two input features. In the real-world cases, such changes can be caused independently on the end-node behavior, e.g., by a variable load of intermediate network nodes. In the experimental section of this paper we demonstrate that the proposed method outperforms the prior art technique.

ACKNOWLEDGMENTS

This research has been supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/235/OHK3/3T/13 and also by Czech Science Foundation project (GAČR) 15-08916S.

REFERENCES

[1] Bartos, K., Sofka, M., and Franc, V. Optimized invariant representation of network traffic for detecting unseen malware variants. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 807–822.

[2] Breiman, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001), 5–32.

[3] Brown, G., Pocock, A., Zhao, M.-J., and Luján, M. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research 13* (2012), 27–66.

[4] Čech, P., Kohout, J., Lokoč, J., Komárek, T., Maroušek, J., and Pevný, T. *Feature Extraction and Malware Detection on Large HTTPS Data Using MapReduce.* Springer International Publishing, Cham, 2016, pp. 311–324.

[5] Davis, J., and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning* (New York, NY, USA, 2006), ICML '06, ACM, pp. 233–240.

[6] Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research 15* (2014), 3133–3181.

[7] Franc, V., Sofka, M., and Bartos, K. *Learning Detector of Malicious Network Traffic from Weak Labels.* Springer International Publishing, Cham, 2015, pp. 85–99.

[8] Grill, M., and Rehák, M. Malware detection using http user-agent discrepancy identification. In *Information Forensics and Security (WIFS), 2014 IEEE International Workshop on* (2014), IEEE, pp. 221–226.

[9] Kohout, J., and Pevný, T. Automatic discovery of web servers hosting similar applications. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (May 2015), pp. 1310–1315.

[10] Kohout, J., and Pevný, T. Unsupervised detection of malware in persistent web traffic. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (April 2015), pp. 1757–1761.

[11] Lokoč, J., Kohout, J., Čech, P., Skopal, T., and Pevný, T. *k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach.* Springer International Publishing, Cham, 2016, pp. 131–145.

[12] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Learning to detect malicious urls. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 30:1–30:24.

[13] Machlica, L., Bartos, K., and Sofka, M. Learning detectors of malicious web requests for intrusion detection in network traffic. *CoRR abs/1702.02530* (2017).

[14] Peng, H., Long, F., and Ding, C. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 8 (Aug. 2005), 1226–1238.

[15] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *In Proceedings of the IEEE Symposium on Security and Privacy* (2010).