A Markov Random Field Approach to Automated Protocol Signature Inference

Yongzheng Zhang[1], Tao Xu[1,2], Yipeng Wang[1(✉)], Jianliang Sun[1,2], and Xiaoyu Zhang[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{zhangyongzheng,wangyipeng,zhangxiaoyu,xutao9083,sunjianliang}@iie.ac.cn, yipeng.wang1@gmail.com
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. Protocol signature specifications play an important role in networking and security services, such as Quality of Service(QoS), vulnerability discovery, malware detection, and so on. In this paper, we propose ProParser, a network trace based protocol signature inference system that exploits the embedded contextual correlations of n-grams in protocol messages. In ProParser, we first apply markov field aspect model to discover the contextual relations and spatial structure among n-grams extracted from protocol traces. Next, we perform keyword-based clustering algorithm to cluster messages into extremely cohesive groups, and finally use heuristic ranking rules to generate the signature specifications for the corresponding protocol. We evaluate ProParser on real-world network traces including both textual and binary protocols. We also compare ProParser with the state-of-the-art tool, ProWord, and find that our approach performs more accurately and effectively in practice.

Keywords: Protocol signatures · Markov random field · Network security

1 Introduction

Protocol signatures are a set of unique byte subsequences that can be used to distinguish the network traces of individual protocols. Protocol signature specifications play an important role in networking and security services, such as Quality of Service(QoS), Intrusion Detection and Prevention Systems(IDSes/IPSes), malware detection, vulnerability discovery, and so on [5, 15, 16, 25, 27]. To be specific, Internet Service Providers(ISPs) uses protocol signature specifications to understand the components of protocol traffic passing through their networks. With an in-depth analysis of the composition of protocols, ISPs can

This research was supported by the National Natural Science Foundation of China under grant numbers 61402472, 61572496, 61202067 and 61303261, and the National High Technology Research and Development Program of China under grant numbers 2013AA014703 and 2012AA012803.

B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 459–476, 2015.
DOI: 10.1007/978-3-319-28865-9_25

impose meaningful and appropriate policies on protocol traces to provide a better service experience in practice. Furthermore, protocol signature specifications are also crucial for IDSes/IPSes. IDSes/IPSes match the packet payload against the protocol signatures to discover abnormal behaviors or activities in protocol traffic. Besides traffic monitoring and IDSes/IPSes, protocol signature specifications are also helpful for vulnerability discovery. For example, existing penetration testing tools often need protocol signatures to generate the protocol traces for vulnerability detection.

Prior arts for protocol signature inference are generally divided into two categories: reverse engineering-based approaches and network trace-based approaches. In this paper, we concern the problem of automated protocol signature inference based on the packet payload of protocol traces. Notice that many network trace-based approaches have been proposed in prior arts, such as Discoverer [1], ACAS [2], Veritas [20], ProDecoder [3], ProWord [21, 26] and so on. The most recent and relevant work is ProWord [21, 26] proposed by Zhang et al. ProWord is an elegant solution for network trace-based protocol signature specification inference. ProWord has two key modules, and it works as follows: ProWord first breaks packet payload into candidate words based on a modified Voting Experts algorithm. Then, ProWord infers protocol signature specifications by a ranking algorithm that selects the highest ranked words as protocol feature words. However, ProWord has two major limitations. 1). to infer protocol signatures, ProWord breaks the packet payload into a set of candidate words. However, this naive solution ignores the spatial coherence of candidate words in protocol messages, and thus leads to a reduced performance on accuracy in practice. For example, message "MAIL FROM" is a protocol signature of SMTP (Simple Mail Transfer Protocol). However, ProWord often breaks the above signature "MAIL FROM" into two parts, "MAIL" and "FROM". Note that the divided messages "MAIL" and "FROM" are not true protocol signatures for SMTP. 2). The computational efficiency of ProWord presents one of its main limitations. For examples, the memory space requirement in ProWord is very high due to the construction of a prefix tree in the VE algorithm.

In this paper, we propose ProParser, which performs automated protocol signature inference based on the network traces of application protocols. The input of ProParser is the network traces of a given protocol, and the output is the protocol signatures, where each protocol signature is represented by a set of n-grams. ProParser has four functional modules in practice: n-Gram Extraction, Keyword Inference, Message Clustering, and Signature Generation. Specifically, we first extract n-grams from the packet payload of protocol traces. Next, we use a Markov field aspect model to infer protocol keywords, which are used to define protocol signature specifications. Then, we utilize a hierarchical clustering algorithm called sequential Information Bottleneck(sIB) algorithm [6] to group similar protocol messages into clusters of the same type according to their protocol keywords. Finally, we generate the final protocol signatures using heuristic ranking rules that find the invariant field among messages in each cluster. The key novelty of ProPaser lies in its exploitation of the spatial coherence of

keywords in protocol messages that is usually missed under the previous conditional independence assumptions. Therefore, ProPaser is a more robust network trace-based system for automated protocol signature inference.

In order to test and verify the effectiveness of ProParser, we apply ProParser on a set of real-world application traces, including a text protocol SMTP and a binary protocol DNS, and then we utilize precision and recall as the metrics to evaluate our experimental results. The experimental results show that ProParser has precisely parsed the protocol signatures with an average recall of 98% and an average precision of 98.5%. In summary, our contributions are highlighted as follows.

- We introduce and present a Markov random field approach to extract the protocol keywords from the packet payload of protocol traces. The proposed approach considers the spatial coherence of keywords in protocol messages that would be missed under the previous conditional independence assumptions.
- We design a system called ProParser, which can automatically infer the protocol signatures of a specific protocol from its real-world traces with no prior knowledge about the protocol specification. We propose a new technique to extract protocol keywords that is independent of the type of the target protocol.
- ProParser is able to handle both textual and binary protocols. Compared to the state-of-the-art method ProWord, our approach performs better experimental results on effectiveness and efficiency.

The rest of the paper is organized as follows. We state our problem scope and review related work in Section 2. We describe the design and technical details of ProParser in Section 3. We present datasets, evaluation methods, experimental results in Section 4. Finally, we conclude the paper in Section 5.

2 Related Work

Prior arts for protocol signature inference can be generally divided into two categories: reverse engineering-based approaches [10, 11, 12, 19] and network trace-based approaches [1, 2, 8, 9, 17-22]. In the remainder of this section, we introduce some typical prior arts of the two categories.

2.1 Reverse Engineering-Based Methods

The reverse engineering based methods implement the executable code and analyze the received application messages to infer protocol signature. Caballero et al. proposed Polyglot, an automatic protocol reverse engineering approach by using dynamic binary analysis [10], they implemented executable codes and monitored the data to extract the protocol signature. Lim et al. implemented an analysis tool which extracts network packet formats by means of working on executable

binary code [11]. Cui et al. presented Tupni in [12], a reverse engineering method by analyzing a set of input, including record sequence, record types and input constraints. These researches have some limitations. First, unknown protocols' executable code is often difficult to obtain, even if it is available, it still suffers from tedious manual effort and harsh operating condition. Additionally, reversing process is significantly difficult once the executable code uses code obfuscation or code compression. As a consequence, we assume the executable code of protocols is not available and focus on network trace based methods.

2.2 Network Trace-Based Methods

Cui et al built Discoverer, which automatically extracted protocol signatures from network traces [1]. Discoverer first separated messages into tokens and classified them into clusters based on the token pattern of the messages. Then, Discoverer implemented recursive clustering to divide the clusters into similar clusters with same message formats. Finally, it merged the clusters using sequence alignment to avoid over-classification. However, the predefined delimiters using in tokenization phase are obviously invalid for binary protocols. In the meanwhile, the sequence alignment algorithm is time-consuming and not that necessary to be a part of signature inference. We notice that the protocol signatures represented by n-gram sets are more efficient. By contrast, ProParser does not depend on delimiters and it uses some heretics ranking rules to extract protocol signatures.

Ma et al. built a statistical and structural content model to identify protocol from network traces automatically [14], they believed that the first 64-bytes can approximately draw a complete distribution of the entire session. However, this assumption often does not hold in reality especially for binary protocols. Haffer et al. proposed ACAS, which explored automatically extracting application signatures from IP traffic payload contents [2]. They also regarded first 64 bytes of each TCP Flow as feature vectors leading to the information loss. ProParser use whole bytes of flows to infer protocol signature.

Zhang et al. proposed ProWord [21], an unsupervised approach to extract protocol signature. They built a word segmentation algorithm to generate candidate feature words and then used a ranking algorithm to select the top-k words. Their work achieves decent accuracy and conciseness while suffers from some obvious drawbacks. ProWord broke payloads into candidate words to discover semantics information while its precision and recall are barely satisfactory, especially for binary protocols. By contrast, ProParser can handle these problems because it does not rely on word boundaries. In addition, the signature pruning phase of ProWord needs manual efforts while ProParser is fully-automated.

Finamore et al proposed KISS in [8], which first extract statistical features from network traces, and then build a support vector machines(SVM) based classifier. Zhang et al. vectored captured protocol traces and employed K-means algorithm to cluster them in [17]. Xie et al. proposed a multi-classifier SubFlow using statistical features from network traces in [18]. However, these methods

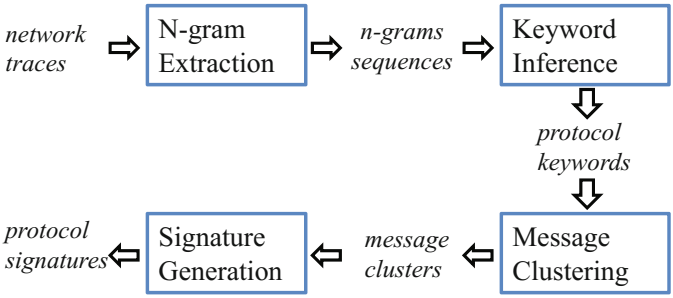

Fig. 1. Architecture of ProParser

suffer from relatively low accuracy caused by protocol behavior confusion or payload byte stuffing. ProParser can address these problems because it has redundant information of protocol messages by using both sematic correlation and statistical distribution.

3 ProParser

The input of ProParser is the network traces of a given protocol, and the output is the protocol signatures, where each protocol signature is represented by a set of n-grams. As shown in Fig. 1, ProParser has four functional modules in practice: n-gram extraction, keyword inference, message clustering and signature generation. Next, we provide the technical details for each module.

3.1 *n*-Gram Extraction

The input to this n-Gram Extraction module is a set of packet traces of the same protocol, and the output to this module is protocol messages, where each protocol messages is denoted by a sequence of n-grams. An n-gram is defined as a subsequence of n elements contained in a given sequence of at least n elements. For example, considering messsage "\x48\x7e\x0a\x3c\x0d" in BitTorrent protocol, we can decompose it into 3-grams as follows: "\x48\x7e\x0a", "\x7e\x0a\x3c", "\x0a\x3c\x0d". More generally, given a byte sequence "$c_1c_2\cdots c_m$", we break it into n-grams as follows, "$c_1c_2\cdots c_n$", "$c_2c_3\cdots c_{n+1}$", $\cdots$, "$c_{m-n+1}c_{m-n+2}\ \cdots c_m$". In practice, we note that a larger value of n will generate a tremendous set of n-grams and the execution time is also high, while a smaller value of n will introduce noise data and further identify inaccurate protocol keywords. Therefore, we give a tentative value in this paper, and we set the value of n to be 3.

In addition, we should also consider the total number of n-grams considered in the n-gram vocabulary. Theoretically, a given n-gram collection may involve approximately 256^n items. In reality, more items can provide more semantics information for the protocol under analysis. However, this enormous amount of items causes prohibitively expensive time consumption. In this paper, we select

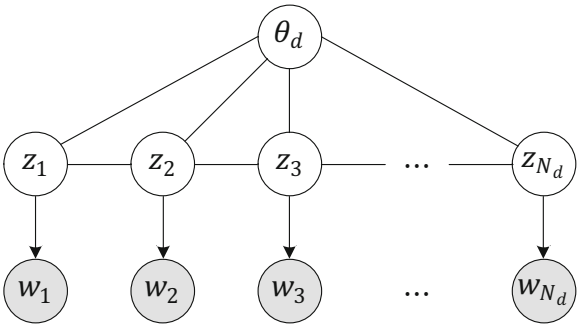

Fig. 2. Markov Random Field Dirichlet Allocation

a P-percent subset from the origin n-gram collection with high frequency of n-gram occurrence to make a trade-off. We vary the range of $P = \{40\%, 60\%, 80\%\}$ to find the most appropriate value for protocol signature inference.

3.2 Keyword Inference

In the module keyword inference, we aim to identify the protocol keywords that are in the given network traces of an application protocol. The input to this module is a sequence of n-grams extracted by the previous module, and the output to this module is a distribution of protocol keywords inferred by ProParser. In prior work, ProDecoder [3] uses a model called Latent Dirichlet Allocation (LDA) to infer protocol keywords from the network traces of individual protocols. However, the basic LDA model is based on a bag-of-words assumption. In other words, LDA model assumes that its n-grams are drawn independently from the keyword mixture $\boldsymbol{\theta}_m$, and thus it ignores the spatial structure of the packet. In practice, we notice that the Markov Random Field model (MRF) can reflect such local interactions for spatial contiguity.

Basic of Markov Field Aspect Model. Given a protocol packet corpus $D \equiv \{\{w_{m,i}\}_{i=1}^{N_m}\}_{m=1}^{M}$ of M packets, where $w_{m,i}$ represents the i-th n-gram in packet m, and N_m is the number of n-grams considered in packet m. Remember that in the basic LDA model, each n-gram $w_{m,i}$ corresponds to a specific keyword indicator $z_{m,i}$. More specifically, each packet m is modeled as a probability distribution of protocol keywords, denoted by $\boldsymbol{\theta}_m = p(\boldsymbol{z}|m)$, where each keyword $z = k \in \{1, \cdots, K\}$ is in turn a probability distribution over the n-gram terms $\boldsymbol{t} = \{u\}_{u=1}^{W}$, denoted by $\boldsymbol{\varphi}_k = p(\boldsymbol{t}|k)$. To improve the spatial coherence for keyword inference, we consider to move from a multinomial distribution over hidden variables $\boldsymbol{z}$ to a representation of Markov random field as shown in Fig. 2. Our Markov random field based inference model can be formulated as a product of MRF and LDA over protocol keyword $\boldsymbol{z}$, and thus the proposed model can be called Latent Dirichlet Markov Random Field model (abbr. LDMRF).

The detailed mathematical derivation of our target posterior distribution $p(\boldsymbol{z}|\mathcal{M}, \boldsymbol{w})$ can be found as follows,

$$\begin{aligned}
\underbrace{p(\boldsymbol{z}|\mathcal{M}, \boldsymbol{w})}_{\texttt{Posterior Distribution}} &= \underbrace{p(\boldsymbol{z}|\mathcal{M})}_{\texttt{MRF}} \cdot \underbrace{p(\boldsymbol{z}|\boldsymbol{w})}_{\texttt{LDA}} \\
&= \underbrace{\prod_{m=1}^{M} \frac{1}{Z} \psi_c(\boldsymbol{z}_m|\mathcal{M})}_{\texttt{MRF}} \cdot \underbrace{\prod_{m=1}^{M} \boldsymbol{\theta}_m}_{\texttt{LDA}} \\
&= \underbrace{\prod_{m=1}^{M} \frac{1}{Z} \prod_{i=1}^{N_m} \psi_c(z_{m,i})}_{\texttt{MRF}} \cdot \underbrace{\prod_{m=1}^{M} \prod_{i=1}^{N_m} \theta_{m,i}}_{\texttt{LDA}} \\
&= \frac{1}{Z} \prod_{m=1}^{M} \prod_{i=1}^{N_m} (\psi_c(z_{m,i}) \cdot \theta_{m,i}) \\
&= \frac{1}{Z} \prod_{m=1}^{M} \prod_{i=1}^{N_m} (\exp\{-E_c(z_{m,i})\} \cdot \theta_{m,i})
\end{aligned} \tag{1}$$

where $\mathcal{M}$ corresponds to the states of Markov random field, and Z is a normalization constant. In addition, ψ_c is a potential function, and E_c denotes clique potential in the Potts model. From Equation 1, we clearly find that the transition of raw LDA to LDMRF is equivalent to placing a Markov random field prior on the probability of keyword $\theta_{m,i}$. Note that determining keyword indicator $\boldsymbol{z}$ of LDMRF model is the core problem of learning the proposed protocol keyword model. By using $\boldsymbol{z}$, we can easily calculate the two types of distributions: (1) the n-gram distribution for each keyword k, denoted $\boldsymbol{\varphi}_k$, and (2) the keyword distribution for each packet m, denoted $\boldsymbol{\vartheta}_m$. In the rest of this paper, we use parameter sets $\Phi = \{\boldsymbol{\varphi}_k\}_{k=1}^{K}$ and $\Theta = \{\boldsymbol{\vartheta}_m\}_{m=1}^{M}$ to denote the above two types of distributions, respectively.

Approximate Inference. Next, we would like to discuss about estimating the parameter $\boldsymbol{z}$ in LDMRF. Remember that our target posterior distribution is $p(\boldsymbol{z}|\mathcal{M}, \boldsymbol{w})$, and it can be formulated as follows,

$$p(\boldsymbol{z}|\mathcal{M}, \boldsymbol{w}) = \frac{p(\boldsymbol{z}, \boldsymbol{w}) \cdot p(\boldsymbol{z}|\mathcal{M})}{p(\boldsymbol{w})} \tag{2}$$

Note that exact inference of the target distribution $p(\boldsymbol{z}|\mathcal{M}, \boldsymbol{w})$ in the LDMRF model is particularly difficult. Thus, in this paper, we obtain an approximate inference result through Gibbs sampling, an example of Markov Chain Monte Carlo (MCMC) algorithm [7]. Gibbs sampling is an iterative algorithm, where in each iteration the value of each variable is updated by a value drawn from the target distribution of that variable conditioned on the rest of variables. To

estimate the parameter $\boldsymbol{z}$ in the LDMRF model, the updating rule for Gibbs sampling algorithm is as follows,

$$p(z_{(m,i)} = k|\boldsymbol{z}_{\neg(m,i)}, \boldsymbol{w}, \mathcal{M}) \propto \frac{n_k^{(t)} - 1 + \beta}{\sum_{i=1}^{W} n_k^{(t)} - 1 + W\beta} \cdot \frac{n_m^{(k)} - 1 + \alpha}{\sum_{k=1}^{K} n_m^{(k)} - 1 + K\alpha} \cdot \exp\left(\sum_{i \sim j} \Delta \Lambda(z_{m,i}, z_{m,j})\right), \tag{3}$$

where $n_k^{(t)}$ is the number of times that n-gram term t is assigned to keyword k, and $n_m^{(k)}$ denotes the number of times that an n-gram from the packet m has been assigned to keyword k. Λ is an indicator function, which decides if the keyword indexes for neighbors $z_{m,i}$ and $z_{m,j}$ are the same. Δ is a strength parameter, and a positive value of Δ awards configurations where neighboring nodes have the same label. After a sufficient number of iterations, the Gibbs sampling algorithm converges, and we can obtain keyword assignments for $\boldsymbol{z}$, which are then used to estimate the two parameter sets Θ and Φ according to the following equations:

$$\varphi_{k,t} = \frac{n_k^{(t)} + \beta}{\sum_{t=1}^{W} n_k^{(t)} + W\beta} \tag{4}$$

$$\vartheta_{m,k} = \frac{n_m^{(k)} + \alpha}{\sum_{k=1}^{K} n_m^{(k)} + K\alpha} \tag{5}$$

Perplexity. In order to ensure that the Gibbs sampling algorithm in LDMRF has converged and that the LDMRF model with the estimated parameter sets θ and ϕ is generalizable, we employ *perplexity* as the metrics to quantify the quality of our estimation. Perplexity, which is defined as follows, is a well-known measure of the ability of a model to generalize to unseen data [24].

$$perplexity(D) = \exp\left\{-\frac{\sum_{m=1}^{M} \log p(\boldsymbol{w}_m)}{\sum_{m=1}^{M} N_m}\right\} \tag{6}$$

where N_m is the total number of n-grams in message m. In ProParser, we prefer a lower perplexity score as a lower perplexity score denotes better generalization performance in practice. Perplexity also allows us to determine the right number of keywords for the given corpus of messages.

3.3 Message Clustering

The Message Clustering module aims to partition the messages which contain identified keywords into multiple clusters. The fact that an application protocol

always has many types of signatures representing different protocol grammar, so it is critical to guarantee the purity of each cluster, that is, one cluster can contain messages from only one protocol while messages from one protocol can be partitioned into multiple clusters. Take a message set of HTTP as an example. For the following cluster of three messages, the first two messages should be partitioned into one cluster and the third one should be set alone.

1) GET /activity.ini HTTP/1.1
2) GET /stat.xml HTTP/1.1
3) POST / HTTP/1.1

Algorithm 1. Sequential Information Bottleneck

Input: Clustering threshold K; Feature vector X; Maximum iteration M; Convergence multiplier θ.
Output: A partition C of X into K clusters.

function sIB(K, X, M, θ)
  Partition $C \leftarrow \phi$
  for i^{th} period of partition C_i **do**
    $C_i \leftarrow$ random partition $c_1, c_2, \cdots, c_k$ from X
    $changeFlag \leftarrow 0$, $itFlag \leftarrow 0$
    while $itFlag < M$ **and** $changeFlag > \theta|X|$ **do**
      $ifFlag \leftarrow itFlag + 1$
      for j **from** 1 **to** $|X|$ **do**
        pop x from c_j
        $d(x, c_{new}) \leftarrow argmin_{c \in C} d(x, c_{new})$
        if $d(x, c_{new}) < d(x, c_j)$ **then**
          insert x into c_{new}
          $changeFlag \leftarrow changeFlag + 1$
        else
          insert x into c_j
        end if
      end for
    end while
    $C \leftarrow argmax_{c \in C} Score(c)$
  end for
end function

Taking probability correlated keywords as message features, we adopt the sIB clustering algorithm to accomplish this task. This method aims to obtain the relevant information of the messages sharing the same message format, denoted by a cluster. sIB has two objective progresses comparing with aIB, a hierarchical clustering algorithm been used by ProDecoder. First, as an agglomerative clustering method, aIB is irreversible and cannot guarantee the global optimum, sIB performs multiple reruns and multiple iterations in each run to avoid losing the optimal solution. Second, it is observed that sIB has more rapid convergence to global optimum so that decrease the execution time.

The input of sIB is cluster threshold K and the joint probability distribution $p(x, y)$ where the random variable X denotes the message feature vector and random variable Y denotes the relevant features of X. The output of sIB is a partition C with K clusters. Initially, we randomly divide the feature vectors in X into a partition C with K clusters, i.e. $C = \{c_1, c_2, \cdots, c_k\}$. Then we step into a loop. Iteratively, we choose every object $x \in X$ out of its current cluster $c(x)$ and reallocate it to a new cluster C_{new} which satisfies $C_{new} = argmin_{c \in C} cost(x, c)$. The cost function is defined as follows:

$$d(x, c) = (p(x) + p(c)) * JS[p(y|x), p(y|c)], \tag{7}$$

where $p(x)$, $p(y)$ represent cluster prior probabilities, and JS is Jensen-Shannon divergence that represents the possibility of $p(x)$ and $p(y)$ derived from the same distribution and can be calculated by the following equations:

$$D_{kl}(p\|q) = \sum_{x \in X} p(x) \log \frac{p(x)}{p(y)}. \tag{8}$$

$$JS_{\pi_1, \pi_2}(p\|q) = \pi_1 D_{kl}(p\|r) + \pi_2 D_{kl}(q\|r). \tag{9}$$

More details about the above equations can be seen in [6]. There are two stop conditions of the above loop: maximum iterations $maxL$ and convergence multiplier θ, that is, when the time of iteration is greater than $maxL$ or the changed elements in the current loop are less than $\theta * |X|$, the loop is terminated. Now, we obtain a converged partition C^*. We calculate its score $F(C^*) = I(Y; C)$, where $I(Y; C)$ denotes the mutual information between C and Y. The $I(Y; C)$ can be calculated by the following equation:

$$I(Y; C) = \sum_{y,c} p(y, c) \log \frac{p(y, c)}{p(y) p(c)}. \tag{10}$$

In order to find out an optimal partition of X, we run sIB n times with random initialization. As a consequence, we will get a partition set $S = \{C_1, C_2, \cdots C_n\}$, and their corresponding scores $F = \{F(C_1), F(C_2), \cdots, F(C_n)\}$. Finally, we select the partition C^*, which satisfies the equation $C^* = argmax_{C \in S} F(C)$. In ProParser, we heuristically set the cluster threshold K to 1.5 times the number of keywords in keyword inference module. Up to this point, we acquire message clusters with extremely cohesive set of messages.

3.4 Signature Generation

Given the clusters of highly related messages, the main goal of this module is to discover protocol signature represented by the 3-grams, i.e., the invariant part among messages. As shown in Fig. 2, the input of this module is the messages in each cluster and the output is the common subsequence represented by 3-grams. To this end, we exploit a ranking method to identify 3-grams that are most

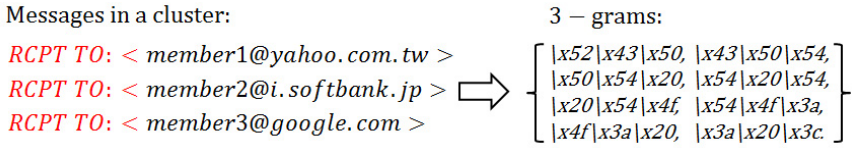

Fig. 3. An Example of Signature Generation

likely to be the protocol signature. We first grade the possible keywords of each cluster and choose the higher ones as candidates relatively. Then we combine the candidates together and prune the redundancy.

1) 3-*grams Ranking:* Considering messages in a specific cluster, protocol signature can be a set of 3-grams with any length and location, it is important to develop a strategy to identify the proper ones. Inspired by the information retrieval heuristics proposed in [23] and aggregation methods proposed in [21], we build several ranking rules and adapt the heuristics to protocol reverse engineering to choose the accurate 3-grams from aforementioned clusters. The ranking rules consist of frequency rule, location rule and position rule.

Frequency Rule. Given a candidate set S of a cluster, we define the number of occurrence of $s \in S$ as **gram frequency**, the number of message which contains $s \in S$ as **message coverage**. We would like to give a higher score to s with higher gram frequency. If the gram frequency is the same, we appreciate the s with higher message coverage. The intuition of this rule is that we believe the 3-grams with substantial amount of appearance is more likely to be the protocol signature.

Location Rule. We define the specific location with maximum number of occurrence of $s \in S$ as **max location**, the number of message which contains $s \in S$ in the max location as **location coverage**. We are willing to give a higher score to s with higher location coverage. For max location calculation, we count all candidates in every possible location in the message and choose the maximum. The intuition of this rule is we believe that s occurs in several locations in a message while the location with maximum occurrence is more valuable. Also, considering the situation where $s1, s2 \in S$ have the same value of message coverage, $s1$ appears at a fixed location while $s2$ scatters at several locations, we prefer $s1$ apparently.

Position Rule. We define the message byte offset of $s \in S$ as **gram position**. The gram position at the beginning or the end of a message deserves a higher score. The intuition of this rule is that we find the bytes that occur at such position is more likely to be used as a protocol signature.

We compute the scores of candidates using the rules of frequency, location and position separately and combine them by multiplication. This aggregation method is proved to obtain proportional fairness of multipliers in [23].

2) 3-*grams Combination:* Based on the ranking method, we get many 3-grams corresponding to the clusters. Note that the 3-grams may be separated

into several clusters, we need to combine them according to their ranking scores. We normalized the scores of 3-grams in each cluster due to its different capacity, then sum them up and sort them in the decreasing order.

3) 3-*grams Pruning:* Note that the quality of the extracted 3-grams directly affects the accuracy of whole system, we establish several pruning rules to eliminate the unreasonable 3-grams.

Length Scale. Message format is designed to exchange data among network hosts, a long signature will burden the data transmission while a short signature lacks the ability to distinguish protocols. In this paper, we control the protocol signature in a reasonable range of [3, 10] for eliminating.

Score Threshold. Based on our ranking method, even candidates with low frequency or coverage can get a score. To increase the compactness of extracted 3-grams, we define a score threshold K and select the top-K 3-grams with higher score, the remaining 3-grams are discarded.

Gram Redundancy. Considering two 3-grams in the gram set, if one with lower score is a substring of another, we will remove it. If one with higher score is a substring of another, we will retain both.

Gram Irrelevance. Irrelevant payload data in protocol traces may generate irrelevant 3-grams, such as date field like "2015/04/15", message ending field like "\x0d\x0a" and message padding field like "\x00\x00\x00". These strings often occur in protocol messages but have no relevance to the protocol, and hence they should be removed.

4 Experimental Results

We evaluate our approach on two kinds of protocols, including textual and binary protocols. ProParser takes network traces of specific protocol as the system input and automatically outputs protocol signatures. In the remainder of this section, we first describe our data sets, then show our evaluation methodology and metrics. Finally, we present the experimental results including parameter tuning, method performance and efficiency. Comparative experimental results are also presented in this section.

4.1 Datasets

Our dataset consists of two well-known protocols, namely SMTP and DNS. We collect the traces from a backbone router of a major ISP on the Internet. The details of the network traces are shown in Table 1. The ground truth of a network trace means its generating application. In order to build the ground truth, we use both the port number and the DPI information to filter network traces. The traces are all raw packets with complete payload semantics. We separate the above mentioned datasets into two parts, one for training and the other for testing. The testing dataset consists of both positive and negative samples. For example, the SMTP set consists of abundant SMTP traces and the same amount of non-SMTP traces, including DNS, HTTP, IMAP and other protocols.

Table 1. Summary of The Traces

Protocol	Size(B)	Packets	Flows	Collection Time
SMTP	673M	1.54M	179K	Aug 2014
DNS	438M	1.33M	145K	Sep 2014

4.2 Evaluation Methodology and Metrics

In this section, we present our evaluation methodology first. We split datasets into two parts, one for training and the other for testing, the training set contains 90% of the dataset traces and the remaining 10% traces contribute to testing set. The amount of each protocol in the dataset is limited because of the high computation complexity of keyword inference and message clustering, while we believe that the number is large enough for extracting good protocol specification. In the training process, we rerun keyword inference several times to adjust the appropriate parameters and perform the message clustering in multiple servers simultaneously to reduce the time consumption. In testing process, we repeat the experiment several times and calculate the average values of metrics to eliminate the variation of different runs.

Table 2. The Confusion Metrics of Trace Prediction

	Actual DNS	Actual not DNS	Total
Predicted DNS	True Positive(TP)	False Positive(FP)	Predicted Positive
Predicted Not	False Negative(FN)	True Negative(TN)	Predicted Negative
Total	Actual Positive	Actual Negative	

To measure the correctness and effectiveness of ProParser, we put forward our evaluation metrics. Given a prediction of packet of targeted protocol, all possible situations are listed in the above confusion table. Table 2 reports the confusion metrics of DNS prediction. There are four possible outcomes of a prediction for a two-class case shown in the table. TP means when the packet is actually positive and is predicted as positive sample correctly. TN means when the packet is actually negative and is predicted as negative sample correctly. FP means when the packet is actually negative but is predicted as positive sample incorrectly. FN means when the packet is actually positive but is predicted as negative sample incorrectly. Based on the above four fundamental measurements, we introduce three evaluation metrics as follows.

$$recall = \frac{TP}{TP + FN}. \tag{11}$$

$$precision = \frac{TP}{TP + FP}. \tag{12}$$

We combine recall and precession into F-Measure to take advantage of their own strengths.

Table 3. Values of Tunable Parameters

Parameter Name	Parameter Value
Iteration in sIB	2000
Rerun Times in sIB	5
Cluster number in sIB	$1.5 * K$
top-k for pruning in Ranking	100
length range$[a, b]$ of 3-grams	$[3, 10]$

$$F - Measure = 2 * \frac{precision * recall}{precision + recall}. \tag{13}$$

4.3 Experimental Results

In this section, we first present the procedure of parameter tuning, and then show the performance and efficiency of our approach. We also exhibit comparative experiments.

1) *Parameter Tuning:* There are several parameters in each module of ProParser. Next, we would like to talk about how to select the optimal parameters in each module. Notice that the parameter tuning is performed only on the training data set, and it is unnecessary for the test data set. We discuss some parameters in details and list others in Table 3 due to space limitation.

Iteration Count L. Gibbs sampling algorithm, used in keyword inference, is an iterative algorithm which is directly relative to the correlation of the n-grams. Thus it is vital to select a proper iteration count L to ensure that the algorithm is convergent. By varying L from 1000 to 10000 and changing P of 40%, 60%, and 80% for DNS and SMTP protocols, respectively, the corresponding perplexity are drawn in Fig. 3. We observe that the complexity values converge at 4000 iterations for DNS and 6000 iterations for SMTP.

Keywords Number K. Keyword number K is another predefined parameter in keyword inference. In this module, markov random field model outputs K keywords with their corresponding probabilities. The K keywords in each message is regarded as K attributes of message clustering. To choose the proper K, we range K from 10 to 180 with a step length of 10 and change P as 40%, 60%, and 80%, respectively. Fig. 4 reports that the perplexity value drops substantially at first and increases gradually later under each P of DNS and SMTP. Thus we record the functional minimum value as the appropriate K for each P of DNS and SMTP.

Strength Parameter Δ. In this part we display the tuning of hyper parameters α, β as well as the strength factor Δ in markov random field model. We fix the proper L and K for each P of each protocol, and then we vary $\alpha = \{0.1, 0.5, 0.9\}$, $\beta = \{0.001, 0.005, 0.01, 0.05, 0.1\}$, $\Delta = \{0.01, 0.05, 0.1, 0.5, 0.9\}$ and $P = \{40\%, 60\%, 80\%\}$ to compute the precision and recall for ProParser. Due to space limitations we will omit the selection of α and β. Next we emphasize the tuning of Δ and the corresponding recall and precision results. Fig. 5

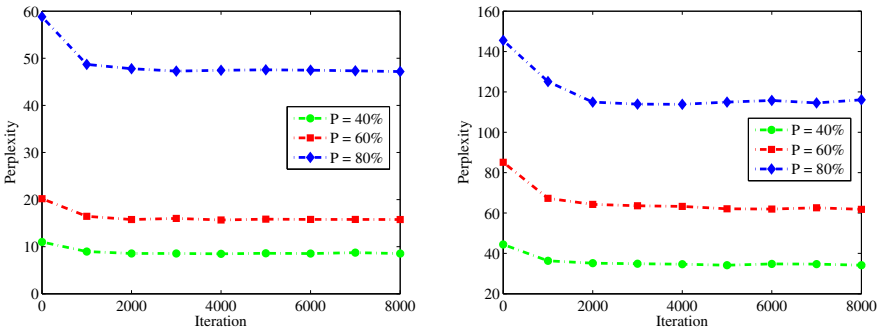

Fig. 4. Selection of Iteration for DNS and SMTP Protocols.

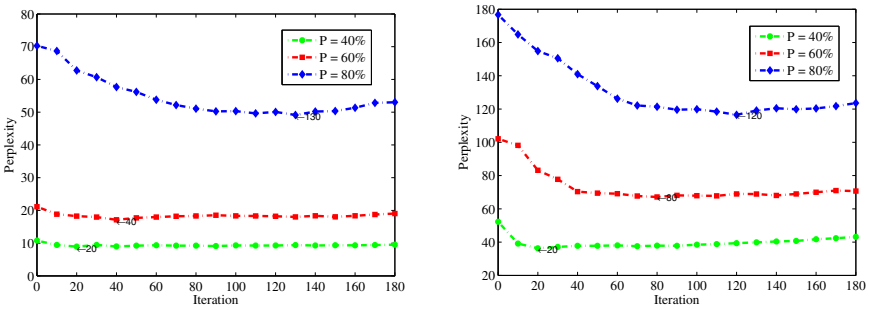

Fig. 5. Selection of the optimal number of keywords for DNS and SMTP.

shows the precision and recall for DNS by varying Δ and P values. The optimal parameter values for SMTP are $\alpha = 0.1$, $\beta = 0.005$, $\Delta = 1.0$ and $P = 0.8$, and the corresponding precision and recall are 98% and 99%. Fig. 6 shows the precision and recall for SMTP by varying Δ and P values. The optimal parameter values for SMTP are $\alpha = 0.1$, $\beta = 0.01$, $\Delta = 1.0$ and $P = 0.8$, and the corresponding precision and recall are 99% and 97%.

2) *Performance Results:* As shown in the parameter tuning, ProParser achieves a decent recall and precision for both textual and binary protocols. We also implement ProWord which is an unsupervised protocol signature extraction approach. Fig. 7 presents the precision and recall for ProParser, with comparison to the results of ProWord. It is obvious that ProParser can significantly enhance the recall without decreasing the precision of ProWord. Additionally, ProWord uses manual inspection for keywords pruning while ProParser is totally automated. Furthermore, ProWord claims that it is more concise and compact with top-K signatures. In oder to hold the decent accuracy, the K of ProWord is 100 while the volume of ProParser is approximately 250. Note that the two

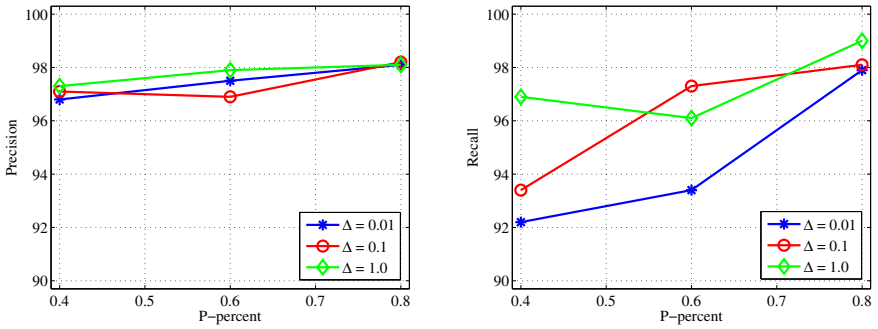

Fig. 6. Precision and Recall of ProParser for DNS.

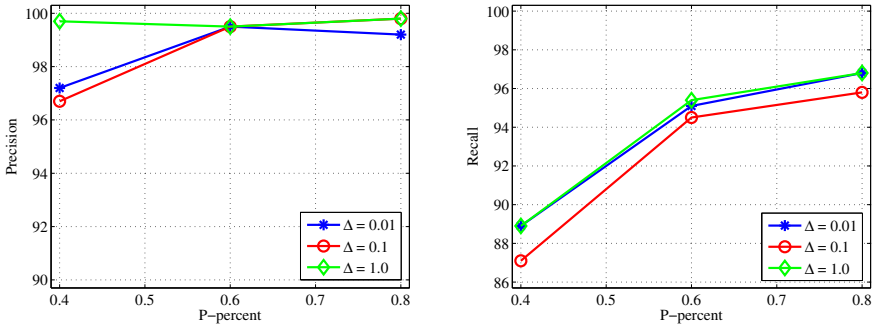

Fig. 7. Precision and Recall of ProParser for SMTP.

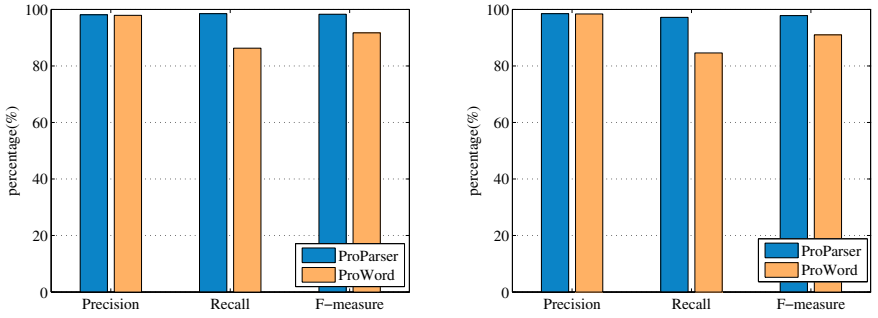

Fig. 8. Comparison of ProParser with ProWord for DNS and SMTP Protocols.

tools both run offline for signature generation, and thus it is not necessary to consider the training latency. The feature matching latencies of the two tools are approximately the same.

5 Conclusion

In this paper, we propose ProParser, a network trace-based approach for automated protocol signature inference. Our method builds on markov random field model to discover sematic relationship and spatial structure of protocol messages, which promotes the effect of message clustering. It also relies on heuristic ranking rules to find the invariant field among protocol messages. We evaluate our protocol signature inference system on real-world network traces including both textual and binary protocols. We also compare ProParser with the state-of-the-art tool, ProWord, and find that our approach performs more accurately and effectively in practice.

References

1. Cui, W., Kannan, J., Wang, H.J.: Discoverer: automatic protocol reverse engineering from network traces. In: Proceedings of the 16th USENIX Security Symposium, pp. 1–14 (2007)
2. Haffner, P., Sen, S., Spatscheck, O., Wang, D.: ACAS: automated construction of application signatures. In: Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data, pp. 197–202 (2005)
3. Wang, Y., et al.: A semantics aware approach to automated reverse engineering unknown protocols. In: Proceedings of the 20th IEEE International Conference on Network Protocol (ICNP), pp. 1–10 (2012)
4. Slonim, N., Tishby, N.: Agglomerative information bottleneck. In: Proceedings of the 12th Neural Information Processing Systems (NIPS), pp. 617–623 (1999)
5. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, pp. 391–404 (2010)
6. Slonim, N., Friedman, N., Tishby, N.: Unsupervised document classification using sequential information maximization. In: Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 129–136 (2002)
7. Griffiths, T.L., Steyvers, M.: Finding scientific topics. Proceedings of the National Academy of Sciences of the United States of America **101**, 5228–5235 (2004)
8. Finamore, A., Mellia, M., Meo, M., Rossi, D.: Kiss: Stochastic packet inspection classifier for udp traffic. IEEE/ACM Transactions on Networking, 1505–1515 (2010)
9. Wang, Y., et al.: Biprominer: automatic mining of binary protocol features (PDCAT). In: Proceedings of the 12th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 179–184 (2011)
10. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 317–329 (2007)
11. Lim, J., Reps, T., Liblit, B.: Extracting output formats from executables. In: Proceedings of the 13th Working Conference on Reverse Engineering, pp. 167–178 (2006)

12. Cui, W., Peinado, M., Chen, K., Wang, H.J., Irun-Briz, L.: Tupni: automatic reverse engineering of input formats. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 391–402 (2008)
13. Kannan, J., Jung, J., Paxson, V., Koksal, C.E.: Semi-automated discovery of applicattion session signatures. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC), pp. 119–132 (2006)
14. Ma, J., Levchenko, K., Kreibich, C., Savage, S., Voelker, G.M.: Unexpected means of protocol inference. In: Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, pp. 313–326 (2006)
15. Holger, D., Anja, F., Michael, M., Vern, P., Robin, S.: Dynamic application-layer protocol analysis for network intrusion detection. In: Proceedings of the 15th Conference on USENIX Security Symposium, pp. 257–272 (2006)
16. Yun, X., Wang, Y., Zhang, Y., Zhou, Y.: A Semantics-Aware Approach to the Automated Network Protocol Identification. IEEE/ACM Transactions on Networking **24**(1), 1–13 (2015)
17. Zhang, J., Xiang, Y., Zhou, W., Wang, Y.: Unsupervised traffic classification using flow statistical properties and IP packet payload. Journal of Computer and System Sciences **79**(5), 573–585 (2013)
18. Xie, G., Iliofotou, M., Keralapura, R., Faloutsos, M., Nucci, A.: Subflow: towards practical flow-level traffic classification. In: Proceedings of the 31th Annual International Conference on Computer Communications, pp. 2541–2545 (2012)
19. Cho, C.Y., Babic, D., Shin, R., Song, D.: Inference and analysis of formal models of botnet command and control protocols. In: Proceedings of the 17th ACM Conference on Computer and Communication Security, pp. 426–439 (2010)
20. Wang, Y., Zhang, Z., Yao, D.D., Qu, B., Guo, L.: Inferring protocol state machine from network traces: a probabilistic approach. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 1–18. Springer, Heidelberg (2011)
21. Zhang, Z., Zhang, Z., Lee, P.P.C., Liu, Y., Xie, G.: ProWord: an unsupervised approach to protocol feature word extraction. In: Proceedings of the 33th Annual International Conference on Computer Communications, pp. 1393–1401 (2014)
22. Krueger, T., Krämer, N., Rieck, K.: ASAP: automatic semantics-aware analysis of network payloads. In: Dimitrakakis, C., Gkoulalas-Divanis, A., Mitrokotsa, A., Verykios, V.S., Saygin, Y. (eds.) PSDML 2010. LNCS, vol. 6549, pp. 50–63. Springer, Heidelberg (2010)
23. Fang, H., Tao, T., Zhai, C.: A formal study of information retrieval heuristics. In: Proceedings of ACM SIGIR, pp. 49–56 (2004)
24. Azzopardi, L., Girolami, M., van Risjbergen, K.: Investigating the relationship between language model perplexity and ir precision-recall measures. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, pp. 369–370 (2003)
25. Wang, Y., et al.: Using entropy to classify traffic more deeply. In: Proceedings of the 6th International Conference on Networking, Architecture and Storage (NAS), pp. 45–52 (2011)
26. Zhang, Z., Zhang, Z., Lee, P.P.C., Liu, Y., Xie, G.: Toward Unsupervised Protocol Feature Word Extraction. IEEE Journal on Selected Areas in Communications **32**(10), 1894–1906 (2014)
27. Wang, Y., Yun, X., Zhang, Y.: Rethinking robust and accurate application protocol identification: a nonparametric approach. In: Proceedings of the 23rd IEEE International Conference on Network Protocol (ICNP), pp. 1–11 (2015)