Using Segment-Based Alignment to Extract Packet Structures from Network Traces

Othman Esoul*
Informatics Department
University of Leicester
Leicester, UK
*Email: *oe26@le.ac.uk*

Neil Walkinshaw†
Informatics Department
University of Leicester
Leicester, UK
Email: †nw91@le.ac.uk

***Abstract*—Many applications in security, from understanding unfamiliar protocols to fuzz-testing and guarding against potential attacks, rely on analysing network protocols. In many situations we cannot rely on access to a specification or even an implementation of the protocol, and must instead rely on raw network data "sniffed" from the network. When this is the case, one of the key challenges is to discern from the raw data the underlying packet structures – a task that is commonly carried out by using alignment algorithms to identify commonalities (e.g. field delimiters) between packets. For this, most approaches have used variants of the *Needleman Wunsch* algorithm to perform byte-wise alignment. However, they can suffer when messages are heterogeneous, or in cases where protocol fields are separated by long variable fields. In this paper, we present an alternative alignment algorithm known as segment-based alignment. We show how this technique can produce accurate results on traces from several common protocols, and how the results tend to be more intuitive than those produced by state-of-the-art techniques.**

1. Introduction

Network protocols play a fundamental role in the behaviour of distributed systems. Faults and vulnerabilities in protocol behaviour can easily lead to unexpected behaviour in the wider system and become focal points for attacks. The risk of such problems can be mitigated by activities such as fuzz-testing and intrusion detection [1], [5], [18], [22], [25], [34], [35], [39]. For such approaches, it is generally necessary to have an existing model that describes the expected behaviour of the network traffic.

In practice however, such models tend to be only readily available for generic protocols with well-established characteristics. Generating such specifications by hand can be an arduous, error-prone task, especially when the protocol in question is unfamiliar and not accompanied by detailed documentation (e.g. the implementation is provided in a third-party component). For this scenario, a host of protocol inference techniques have emerged.

A crucial step for any inference technique is to infer the packet structures from the data – to separate out the packets, and to identify within packets the various data fields and field headers. Current approaches [3], [11], [12], [24], [40], [45] tend to identify common patterns by attempting to align the sequences. Alignments can identify commonalities and variances, which can in turn be used to identify, for example, the tokens that are used to delimit packets, the key field identifiers, and the data fields.

Although there has been a substantial amount of research in the area, most of the emphasis has been placed on either stages *prior* to the alignment [3], [40] or on challenges such as the inference of state machines (once packet types have been identified) [9], [10], [23], [24], [42]. The underlying algorithm that is used to align packets to identify their structure tends to be the same for most techniques – the *Needleman-Wunsch* algorithm [31].

Although well suited for its original purpose of protein sequence alignment, Needleman-Wunsch can become problematic when applied to sequences of bytes from network packets. For one, it is highly sensitive to various parameters (such as the "gap-penalty" parameter) that, though honed through decades of use on protein sequences, are far from straightforward to identify for network packets (and in all likelihood need to be varied on a per-protocol basis). Secondly, it can produce highly inaccurate results when messages have an identical packet structure, but happen to contain variable-length data fields.

This paper proposes the use of *segment-based sequence alignment* [28] to align network packet data. Instead of aligning messages on a character-by-character basis, segment-based alignment constructs alignments in terms of entire sub-strings. The algorithm is not dependent upon parameters. Also, because it operates in terms of subsequences, it is more forgiving of slight discrepancies when comparing sequences that would confound Needleman-Wunsch. The approach has proven to be a successful replacement for Needleman-Wunsch within Bioinformatics, and in this paper we seek to show that it can provide a similar replacement with respect to protocol reverse-engineering.

This paper makes the following contributions:

- We present a technique based on segment-based alignment to reverse engineer packet structures from network traces.
- We present a novel approach to evaluate the accuracy

of the alignments. Instead of relying on the conventional analytical approach of scrutinising inferred packet structures, we use what we consider to be a more empirically valid approach. We use the inferred packet structures to synthesise new network messages, which we send to servers, and track whether or not the packet is parsed as valid or not.

- We offer a preliminary qualitative comparison of the alignments produced by our approach against those produced by the current state of the art – the *Protocol Informatics* (PI) [3] framework, which is based on Needleman-Wunsch algorithm.

The rest of this paper is structured as follows: In Section two, we give a brief overview of the process of protocol reverse engineering, sequence alignment, and provide a motivation for this work. Section three explains the concept of segment-based alignment in more detail. In Section 4, we explain our implementation, detailing how we integrate segment-based alignment, and how message patterns are extracted. In Section five, we evaluate the quality of the message patterns extracted from the traces. We conclude this paper by listing some of the findings and future work.

2. Background

We begin with a general overview of protocol reverse engineering techniques, and proceed to present a concise introduction to sequence alignment and how it has been used to infer the message structure of network protocols. We then conclude this section with a discussion of the limitations of the state of the art and the motivation behind this work.

2.1. Protocol Reverse Engineering

Network protocol specifications are the backbone of several security applications [5], [18], [22], [25], [34], [35]. Given an undocumented protocol (e.g., SMB or Skype), the goal of protocol reverse engineering is to extract a complete description of how to interact with them. This entails inferring the *message format*, which captures the structure of all messages, and the rules that govern the order in which these messages can be sent and received (often modelled as a *state machine*).

There are two common approaches for inferring protocol specifications: (1) by *reverse engineering* protocol executables (e.g., using dynamic execution analysis of the server while processing messages) [6], [7], [10], [13], [43], or (2) by *analysing network traffic* [2], [3], [4], [11], [23], [40], [42]. The choice of approach invariably depends on the context. In this paper, we concerned with the scenario in which one does not have access to the protocol executable, and thus is unable to use intrusive analysis approaches to scrutinise its behaviour. Accordingly, we focus on the latter approach of reverse-engineering protocols from network traffic.

Figure 1 illustrates the common sequence of steps that tend to be adopted by most traffic-based reverse engineering techniques to infer the message structure. Generally, the approach consists of the following steps:

Table 1. AN ALIGNMENT OF A SET OF MULTIPLE HTTP MESSAGES USING THE NEEDLEMAN-WUNSCH ALGORITHM AND THE PROGRESSIVE ALIGNMENT HEURISTIC.

(a) Non-aligned messages:

```
1 | GET / HTTP/1.1
2 | GET /indx.html HTTP/1.1
3 | GET /k.ico HTTP/1.1
4 | GET /img.png HTTP/1.1
5 | GET /st.css HTTP/1.1
```

(b) Aligned messages:

```
1 | GET /--------- HTTP/1.1
2 | GET /indx.html HTTP/1.1
3 | GET /---k.ic-o HTTP/1.1
4 | GET /i-mg.-png HTTP/1.1
5 | GET /--st.-css HTTP/1.1
  |-------------------------
  | *****          *********
  | GET /--------- HTTP/1.1
```

1: Message pre-processing:. Network data tends to be transmitted via 'layered' protocols. Protocols that sit on top of each other and accomplish different functions (as per the OSI [47] or TCP/IP [16] reference models). We are conventionally only interested in one layer of messages (i.e. the Application Layer), containing the target protocol. This means that we need to employ techniques by which to filter out contents of the network data that are not relevant to our reverse-engineering task. Typical examples of how this can be accomplished are discussed by Kim *et al.* [22].

2. Clustering: . Typically, application protocols involve multiple different types of messages, where each type has its own format. The *clustering* step serves to group messages of the same type together, so that they can be subjected to a more fine-grained analysis to extract their shared format (the alignment step that is the main subject of this paper). In order to facilitate clustering, packet data is often cleansed, e.g. by filtering out irrelevant data, or by using dimensionality-reduction techniques [26].

3. Alignment: . Sequence alignment algorithms take as input the clustered protocol messages and align them, exposing the structural aspects of field similarities, differences, and gaps. Typically, the alignment result is translated into a form of regular expression (or XML format) to represent the message format of each message type identified in trace.

2.2. Sequence Alignment in Packet Structure Inference

Sequence alignment algorithms have long been used in Bioinformatics [14], for example to identify relationships between protein sequences. An alignment can show precisely where two sequences are identical – which zones of the two sequences match each other, potentially indicating that they are related in some way. Previous protocol inference techniques have sought to adopt this intuition to infer the message structures. The rationale is that packets that are related to each other have certain similarities; they share

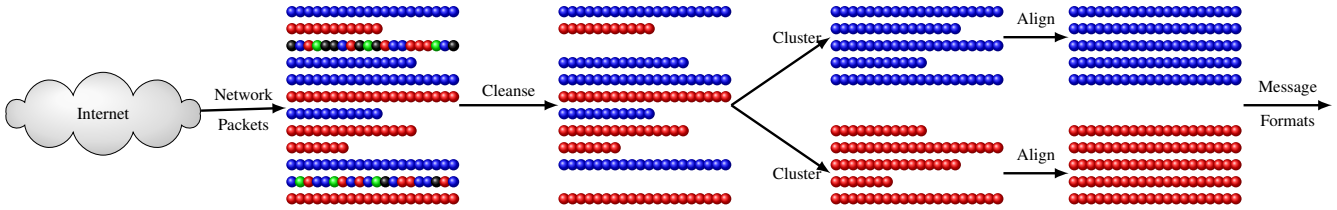


Figure 1. Common approach of inferring protocol message formats from captured network traffic.

key-words and use the same symbols to act as delimiters between different parts of a packet.

To illustrate this intuition, we can consider the set of packets in Table 1 (a), containing simple request messages from the HTTP protocol. An alignment of the messages is shown in Table 1 (b). Similar characters (bytes) are aligned to each other. Because these messages have different lengths, gaps are inserted into each message in order to align it with other messages. As a result, the alignment clearly outlines the two keywords (GET & HTTP/1.1) and the variable field in the middle (in this case the URI). The lowermost sequence at the bottom is referred to as the *consensus sequence*, which summarises the alignment.

The alignment shown in Table 1 is produced by the Needleman-Wunsch algorithm [31] (which is designed to align pairs of sequences) and *progressive alignment* technique (explained later – which supports the tractable extension of Needleman-Wunsch from pairs to multiple sequences). Typically, there are two types of algorithms; *global* (Needman Wunsch is a global algorithm) and *local* (e.g., Smith Waterman [14]). Global alignment algorithms have the goal of matching entire sequences with each other (i.e. finding a corresponding position for every element). Local alignment algorithms on the other hand merely focus on identifying regions that are strongly similar. Global alignment algorithms tend to be better suited to pairs of sequences that are broadly of a similar length and content, whereas the latter is better suited to sequences that are more diverse in nature.

Traditional global and local alignment algorithms are based on *dynamic programming* which cannot be easily extended to align more than three sequences as it becomes prohibitively expensive. For this reason, different heuristics have been developed to align multiple sequences, such as *progressive multiple sequence alignment* [14]. This approach operates by initially aligning two sequences(typically the most similar pair) using Needleman Wunsch algorithm, and then 'progressively' adding additional sequences to this fixed alignment.

2.3. Motivation

Most protocol reverse engineering techniques are based - one way or another - on the Needleman-Wunsch algorithm (and its Progressive Alignment derivatives). As discussed above, this can work when protocol messages tend to be of a similar length and content. However, this is not necessarily the case with network messages. These can contain long variable and optional fields, as is the case in HTTP protocol for example. Additionally, with Needleman-Wunsch the quality of a sequence alignment critically depends on the judicious selection of user-defined parameters (e.g. a gap penalty) which can be very difficult to choose [12], and can vary from one protocol to another. As a consequence, if the messages are heterogeneous, and the parameters fail to count for the specific characteristics of a message set or protocol, alignments can easily become highly inaccurate.

Let us consider the pairs of messages shown in Table 2. Here we show two pairs of HTTP messages of the same type, each message consists of a request-line message and optional headers. One set consists of messages that share strong similarity (the stream of bytes is similar from start to the end). The other set contains request-line messages of the same protocol that are less similar because the URI fields are of different lengths and contents, and omit some of the message headers.

We illustrate the aforementioned problems by aligning these messages with the Protocol Informatics tool [3], which is based upon Needleman-Wunsch. First it is necessary to select the Needleman-Wunsch parameters (scores for when a pair of aligned characters match and mismatch, and a penalty for any gaps that are introduced). We consider the default settings (match=1, mismatch=0, gaps=0) as used in the Protocol Informatics for protocol analysis.

The results are shown in Table 3. The alignment results show that when messages share significant similarity, the choice of the standard user-parameters can provide good alignment results. The message elements that we would expect to be aligned – '`GET /`', '`HTTP/1.1\r\nHost: www`' and the '`\r\n\r\n`' at the end of the message are all correctly aligned.

However, when the messages are dissimilar, the same algorithm with the same parameters fails to produce a suitable alignment. Although the first '`GET /`' and the final '`\r\n\r\n`' are aligned correctly, it fails to match the '`HTTP/1.1`' header. Whereas this example is necessarily small for the purpose of illustration, it is apparent the problems illustrated here can easily be exacerbated as the number of messages, their lengths, and heterogeneity increase.

Table 2. TWO SLIGHTLY DIFFERENT SETS OF HTTP GET MESSAGES.

Sets	HTTP GET Messages
Similar	`GET /myimage.jpg HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n` `GET /thisisabitlongeruri/documents/index.html HTTP/1.1\r\nHost: www.le.ac.uk\r\n\r\n`
Less Similar	`GET /myimage.jpg HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n` `GET /thisisabitlongeruricontainsmanycharacters/documents/index.html HTTP/1.1\r\n\r\n`

Table 3. ALIGNMENTS OF THE HTTP MESSAGES BY NEEDLEMAN-WUNSCH WITH THE STANDARD PARAMETERS (MATCH=1, MISMATCH=0, GAP=0).

Message type	Alignment Result
Similar	`GET /my--im---a------ge--------------------.jpg---- HTTP/1.1\r\nHost: www.bbc.com\r\nUs-er--Agent: Dillo/3-.4--\r\n\r\n` `GET /--thi-sisabitlongeruri/documents/index.---html HTTP/1.1\r\nHost: www.-------------le--.a------------c.-uk\r\n\r\n`
Less Similar	`GET /my--im---a------ge.jpg HTTP/1.1\r\nHost: www.bb---c.com\r\nU----se-------r-Ag--e---------nt: D--i--------llo-----/3-.4-\r\n\r\n` `GET /--thi-sisabitlonge--------------r--------------uric--o----n-tains-manychar-a-cters/document---s/index.html-- HTTP/-1.-1\r\n\r\n`

3. Segment-based Alignment

This paper proposes the use of *segment-based* alignment [28] to work around the intrinsic limitations imposed by global alignment algorithms such as Needleman Wunsch. The approach has been successfully adopted within Bioinformatics to improve the alignment accuracy for long and large corpora of protein sequences. In this section we will present segment-based alignment in more detail. We will start by presenting the scoring scheme that is used to assess the similarity of sequence "segments". We will then explain how this scheme is used to align sequences.

A *segment* is a contiguous sub-sequence of symbols (or in our case bytes) within a sequence. Segment-based sequence alignment operates by identifying similar pairs of segments within the sequences that it seeks to align. These matched pairs of segments are commonly referred to as a *fragment*. Each fragment is given a *weight* (score) that reflects its significance among other fragments (detailed below). The approach then seeks to find a *consistent* set of fragments from all possible sequence pairs (again detailed below), maximising the total score of these fragments. The total score of an alignment is defined as the sum of weights of the fragments involved that can be included into one single alignment without these fragments contradicting each other. The approach does not employ any kind of gap penalty which avoids the well-known difficulties concerning choosing appropriate gap penalty parameters. The rest of this section provides details on how the scores are computed, and how these feed into the alignment process.

3.1. Scoring Scheme

A scoring scheme is a function that computes a quality score for any possible alignment of a given set of sequences. This enables the use of heuristic optimisation techniques to identify *optimal alignments* [14], using the scoring function as an "objective function". For the segment-based approach the scoring scheme is based on similarities between segments (as opposed to the Needleman-Wunsch approach of only focussing on individual characters).

For its use in Bioinformatics, there is the additional complication that characters are not simply identical or different. Different pairs of characters (proteins) can share varying degrees of similarity. Consequently, to compute the scores accurately it is first necessary to provide a matrix that provides the similarity between any given pair of characters in the set of characters being considered. The scoring for a particular fragment (consisting of a pair of segments x and y) is computed first of all by summing up the similarity scores (according to the aforementioned matrix) for every pair of characters. This score is denoted $f(x, y)$.

Once the similarity score is computed, each fragment f is assigned a *weight* $w(f)$ [14]. This is computed by establishing the probability $P(f)$ of the *random occurrence* of a fragment of the same length that results in the same score. The intuition behind this is that the less likely a given collection of fragments is to occur *just by chance*, the more likely it is to be *related* so the higher its score should be. The probability of a fragment of a particular length obtaining a given score is established experimentally; before the alignment, a *probability table* is computed, whereby the probabilities for large numbers of fragments of a given length and score are computed, and are used to populate the table. Once $P(f)$ is computed, the weight can be computed as $w(f) = -logP(f)$.

3.2. Computing Alignments

The process of computing alignments of sequences amounts to finding the best combination of localised matches between sequences (fragments), whilst also ensuring that individual fragments do not contradict each other (e.g. violate the ordering of the sequence as a whole). In other words, fragments that have been chosen to be part of an alignment must be *consistent*.

For a single pair of sequences, alignment consists of finding a set of fragments that (a) do not conflict with each other, and (b) produce the highest score when their weights are summed up. These can then be chained together by a recursive chaining procedure [38].

Because direct extension of the pairwise alignment increases the computational complexity of the algorithm exponentially, a *greedy* heuristic tends to be used to align multiple sequences [28], [29]. Similarly to pairwise alignment, a consistent set of fragments with a maximum set of weights have to be selected. The multiple-sequence alignment process (for a set of N sequences) can be summarised with the following steps:

1) All $\frac{1}{2}N(N-1)$ pairwise maximum alignments are constructed, the result is the set of fragments F.
2) F is sorted according to the computed weights in a greedy fashion:
    a) Fragments are incorporated one by one into the multiple alignment starting with the fragment of maximum weight, provided they are consistent with fragments already added.
    b) Inconsistent fragments are skipped.
3) A multiple alignment is constructed from the selected (consistent) set of fragments.

4. A Segment-Based Alignment Tool to Identify Packet Structures

In this section, we present a technique that uses segment-based alignment to determine the structure of network protocol messages. Our approach follows the same generic stages as most other approaches, as shown in Figure 1. We pre-process the network messages and first apply a clustering algorithm to group them into messages that are of a similar type. We then align messages of the same type - however this time we use the segment-based alignment algorithm. These phases – pre-processing, clustering, and alignment are described below.

4.1. Data Preprocessing

The data pre-processing stage consists of two steps. First of all it is necessary to extract the relevant network traffic, and secondly it is necessary to provide an encoding that enables us to group together similar messages so that they can be clustered.

4.1.1. Traffic Classification. For this step we use a port-based method [22] to separate out relevant network traffic. This separates out messages according to their destination port-numbers, which are typically standardised for a given protocol[1]. We also use this step to make sure that there are no malformed packets, and there is no misuse of port numbers (e.g. use of non-standard port numbers for communication [25]). We also utilise this step to extract only data that belongs to the application protocol; data that belong to the *transport layer*, *network layer* and *link layer* are discarded.

4.1.2. Feature Generation & Selection. Since we intend to cluster our data, it is necessary to re-code the data in such a way that a clustering algorithm is going to be able to identify similarities and differences between messages. To enable this we reduce the 'dimensionality' of the data by identifying and selecting certain features in protocol messages that can guide the clustering algorithm. For this we use n-grams [8], [40] to tokenise protocol messages. An n-gram is a sub-sequence of n consecutive characters from a longer sequence. The set of possible n-grams are arranged in a frequency matrix. A network packet can thus be characterised as a sequence of numbers, where each number corresponds to the frequency of a given n-gram in the packet.

Messages from the same type normally have similar *n-gram frequency distributions* [40], therefore, we use the n-gram occurrences as a feature to distinguish between protocol messages. To normalise the amount of contribution of each n-gram, we apply the *Term Frequency-Inverse Term Frequency* (TF/IDF) as a weighting scheme [36]. To keep the number of dimensions (i.e. n-grams) as small as possible, we also eliminate n-grams that carry no discriminative features, that is by removing n-grams which occur very infrequently.

4.2. Message Clustering

In this step we cluster similar protocol messages into distinct clusters. We use an *agglomerative hierarchical clustering* algorithm with *complete linkage* clustering criteria [21]. This creates a hierarchy of clusters, where coarse, large clusters higher up are split up into more granular lower-level ones. Clusters are obtained by cutting the generated tree (also known as a 'dendrogram') at a given threshold T.

Agglomerative hierarchical clustering depends upon the selection of a distance measure to quantify the distances between messages. We use the *Jaccard* similarity coefficient [36]. The distance is defined as $D(a,b) = 1 - S(a,b)$, where S is the similarity of two messages represented by a and b features respectively. The chosen distance measure is a binary measure and commonly used for clustering network messages [10], [36], [42].

At this point it is important to note that, as is the case for all approaches that use clustering as a basis for finding sequences to align, the success of clustering can be highly sensitive to the choice of clustering parameters. This can include the choice of n when selecting n-grams, as well as the choice of parameters that are specific to the clustering algorithm in question. Our choice of parameters is informed by the recent empirical studies by Esoul *et al.* [15], who used these to derive suitable clustering parameters for network packets.

4.3. Segment-Based Alignment

We adopt the Dialign multiple sequence segment-based alignment technique [28] to align network packets. Specifically, our implementation is derived from the open-source implementation of *Dialign-2* [30]. As input, we provide the individual clusters of messages produced by the clustering step.

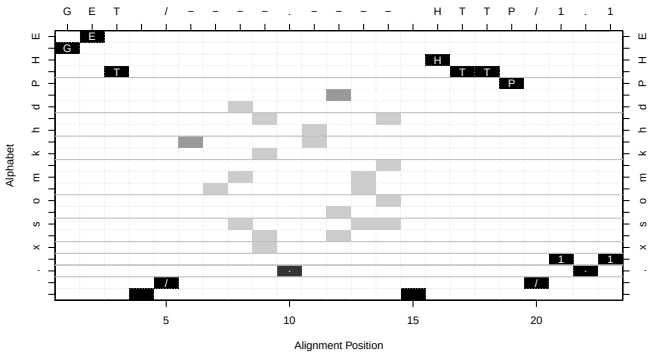
Existing implementations such as Dialign-2 tend to be heavily tailored towards the domain of Bioinformatics. For example, there are about 22 "proteinomic" amino-acids, which is why conventional implementations only consider

1. There are alternative approaches that could readily be adopted instead when the port-numbers are not fixed [22].

Table 4. A POSITION WEIGHT MATRIX GENERATED FROM THE SET OF ALIGNED HTTP MESSAGES INTRODUCED IN THE BACKGROUND (SEE TABLE 1). IT SHOWS ON THE TOP THE NUMBER OF COLUMNS (POSITIONS) AND THE ALPHABET AS THE MATRIX ROWS. TO PRESERVE SPACE, THE MATRIX ONLY SHOWS THE USED CHARACTERS OF THE ALPHABET.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
E	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
T	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	**0.4**	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	**0.4**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	**0.2**	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	**0.2**	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	**0.2**	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**1**
.	0	0	0	0	0	0	0	0	0	**0.8**	0	0	0	0	0	0	0	0	0	0	0	**1**	0
/	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0

Table 5. THE EXTRACTED MESSAGE PATTERN FROM THE CORRESPONDENT PWM WHEN THE GENERALISATION PARAMETER (T)=0.6.



22 symbols. For these, the configuration of the distance-matrix that is used to compute alignment scores tends to be tailored to amino acids, using carefully constructed distance matrices, such as the BLOSUM 62 substitution matrix [19].

In order to apply segment-based alignment to network packet sequences, it was necessary to expand the range of symbols. We adopt the convention of encoding every symbol in a packet as a hexadecimal pair. Since we are carrying out byte-wise analysis of the network stream, there are 256 possible characters that could be represented in this way. Secondly, it is necessary to replace the protein-specific similarity matrix; we provide a more straightforward identity-matrix instead. Two characters obtain a score of 1 if they are identical, and a score of 0 otherwise.

Recall that the similarity score for a fragment is then used to compute a weighting for the fragment, based on the probability that a random fragment of the same length would produce the same score. This relies on the prior computation of a probability table, which in our case had to be recomputed to suit the expanded set of characters and the new scoring tables. We computed this probability table by running 550,000 random experiments for all combinations of lengths from 1 – 40 and scores.

4.4. Pattern Extraction

In the context of network packets, aligned regions tend to correspond to one or more protocol fields. For each cluster of aligned messages, the messages are listed in aligned form (similar to the example alignments shown in Table 1(b)). This means that every aligned position can be referred to in terms of a column (e.g. column 1 refers to the first character in every sequence etc.).

To extract a message pattern, we generate a *Position Weight Matrix* (PWM), also known as *Position Specific Scoring Matrix* (PSSM). A position weight matrix (PWM) is commonly used method to represent motifs in biological sequences [46]. Typically, a PWM consists of one row for each character of the alphabet, and has one column for each position in the alignment. A PWM is constructed by counting the occurrences of each character observed at each position (each coefficient in the matrix indicates the number of times that a given character occurred at a given position), we then normalise the frequency so that the occurrence rate of each character falls within the interval [0,1], where 0 indicates that particular character in that column (position) did not occur across all messages, and 1 indicates that character occurred in every single message at that particular position. The coefficients of the matrix can now be interpreted as probabilities of a given character occurring at a given position.

As an example, Table 4 shows the PWM that corresponds to the set of sequences that was shown in Table 1. Here, the character "G" has been observed across all aligned messages in position 1, therefore, assigned weight of 1, while the character "i" observed only twice at position 6, thus given the weight 0.4.

After we generate the PWM matrix, we then select a threshold value T between 0 and 1, which we use as a basis for controlling pattern generalisation to the level we desire. A value of 1 indicates that, for an alignment, we are only prepared to consider a character to be a part of an alignment if it occurred in every message at a given position. Lower values indicate that we are allowing a degree of error - for a proportion $(1-T)$ of the aligned messages not to contain that specific character. For example, if $T = 0.6$, this indicates that we are only interested in alignments that apply to at least 60% of the sequences. The rest of the characters will be filled with gaps as illustrated in Figure 5 using the same PWM generated in the previous step. The figure shows the extracted pattern shown on the top as well as the position and weight assigned to each character.

5. Experimental Evaluation

In this section we seek to experimentally evaluate the accuracy of the packet structures that are inferred, and whether they can be of practical use. In previous works that have sought to address similar problems, (e.g. the Discoverer packet structure inference tool [11]) packet structures were

Table 6. SUMMARY OF NETWORK TRACES.

Protocol	Sample Size (Number of Messages)	Type
HTTP	4000	Text
SIP	5000	Text
TFTP	2000	Binary
SMB	5000	Binary

inferred from sets of sequences from a known protocol such as HTTP, and the message structure was compared to a reference version produced by an off-the-shelf network traffic analyser (e.g. Wireshark[2]).

Our ultimate goal is to use the inferred packet structures for the purpose of protocol testing. As a consequence, we are not just interested in whether the inferred packet patterns can be parsed. We are also interested in knowing whether the inferred packet structures can be interpreted by a server. Since our ultimate goal is to use inferred packet structures for activities such as protocol fuzz-testing, we are also interested in potentially inferring packet structures that, though not entirely valid according to some reference implementation, are still capable of eliciting responses from a server.

Accordingly, unlike the evaluation of Discoverer, we do not solely focus on the accuracy of the inferred packet structure itself. We also use our inferred message patterns to automatically synthesise messages that are sent to the server in question, and monitor its responses.

Specifically, in this part of our evaluation we seek to answer the following questions:

RQ1 Are inferred message patterns syntactically correct?
RQ2 Can inferred message patterns elicit valid server responses?

5.1. Subject Protocols

We have chosen four data samples of open (documented) network protocols, which have been obtained from an online repository of network trace files [32]. Network protocols can either be in a textual or binary format. Depending on their format, they can pose completely different challenges from a clustering and alignment perspective. Accordingly, we have selected two protocols from each family. A summary of the data samples is shown in Table 6. Their details are as follows:

HTTP & SIP:. HTTP and SIP are both text-based application-layer protocols. The messages for both protocols tend to be long and consists of several fields. Despite the fact that HTTP and SIP share similar message properties, both protocols are used for totally different purposes. The main purpose for the HTTP is to access and retrieve web documents, the SIP protocol is mainly used to manage (establish, modify, and terminate) communications sessions [37]. Furthermore, SIP is a stateful protocol, whereas HTTP is not.

2. https://www.wireshark.org/

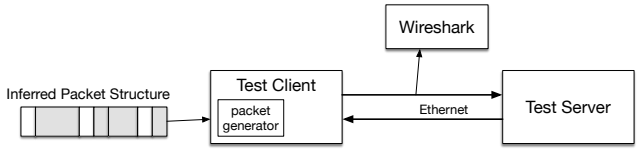


Figure 2. Evaluation methodology for the inferred request/response message patterns.

TFTP & SMB/CISF:. TFTP and SMB/CIFS are binary application-layer protocols. TFTP is a file transfer protocol with a simple message formats. TFTP is also extensively used to support remote booting of diskless devices and on some occasions for malicious purposes [12]. The SMB/CIFS is a network file sharing protocol with complex message formats. SMB/CIFS protocol consists of several flavours known as *dialects* where each dialect consists of a set of extra messages that define a particular version [20], [27]. SMB is a proprietary protocol. The equivalent open-source implementation of SMB is widely known as the *Common Internet File System* (CIFS) [20].

Another important difference between this evaluations and others is that, for various reasons (often performance related in the case of Needleman-Wunsch based approaches) evaluations of messages truncate messages, and only consider the front portion (which is often assumed to contain the most interesting fields) [40], [41]. In our case we do not truncate the messages, and always use the entire packet.

5.2. Experimental Set-up

Our experimental setup is illustrated in Figure 2. For each protocol we infer a set of protocol structures (one for each inferred packet type). These are then provided to a simple *test client*. The test client synthesises a network packet from the inferred message structure and sends the packet to the *test server* – an off-the-shelf server implementation for the protocol in question. The test server accordingly sends responses, which we analyse to determine whether the server considered the messages to be valid or not. Furthermore, we monitor all of the messages that are sent to the server with a network analyser (Wireshark in our case) to determine whether the messages are syntactically valid. The details of the individual components are elaborated below:

A Test Client:. A program that can package and send message patterns. For this task we have developed a packet generator that is able to synthesise and send different message patterns for the four selected protocols. The message synthesis is carried out in a naïve way: The final alignment text (c.f. the bottom line in Table 1, or the message pattern along the top of Table 5, including any '-'s that fill gaps, is sent verbatim.

For SMB this required a degree of tailoring; the SMB server can communicate with SMB clients through either raw TCP transport (port 445) or through NetBIOS Session (port 139). We have implemented SMB packet generator to send packets using the traditional NetBIOS session because

the samples of traces we obtained from the online repository also used this type of transport. Accordingly, before we send SMB packets to the SMB server, it is first necessary to establish a NetBIOS session [20].

A Test Server:. An implementation of the target protocol. For each protocol, we have installed a suitable off-the-shelf server. For the HTTP protocol, we have set up *Lighthttpd* server (version 1.4.33) which is mature in development and with specific server responses. For SIP we have installed *Asterisk* (Version 11.7.0). We have set up Asterisk to listen on the same port and transport protocol observed in the original captured traces (UDP transport, port 5060). For TFTP we installed *tftpd server* (netkit-0.17). Finally, we chose *Samaba Server* (version 4.3.3) as the dedicated server to test SMB extracted message patterns.

A Network Analyser:. We have chosen Wireshark (more accurately, the command line version TShark) to automatically dissect and log data (will be explained head the type of data we are interested to log) of the generated patterns.

5.3. Methodology

Both research questions revolve around packets that we generate (automatically) from inferred message patterns. For every cluster (message type), we infer a set of message patterns that are slightly different from each other (explained below), and send them over the network. It is important to note that the number of packets that were synthesised varied for each protocol, depending on the number of message clusters that were generated (i.e. the number of message-types that were inferred). Whereas the clustering resulted in two clusters for HTTP and TFTP, it yielded five clusters for SIP and eight clusters for SMB.

RQ1: Are inferred message patterns syntactically correct?. To answer RQ1, we use Wireshark to automatically sniff the packets that we have generated and sent across the network. We employ a feature integrated into Wireshark known as *Expert Information* [44]. This feature provides more information on captured network packets and whether or not they are valid (according to their target protocol). The feature is provided to assist users and experts to understand the behaviour of protocol packets.

If network packets are erroneous, Expert Information consists of a specific *severity level* of the error, and a *group* to which these errors belong. In our experiment, packets are flagged as *invalid* when their expert information belong to the groups *protocol* (violate protocol specifications), *undecoded* (data could not be decoded for some unknown reason), and *malformed* as *syntactically invalid packets*. We also, consider *unrecognised packets* by Wireshark (as the intended application protocol) as syntactically invalid packets. Thus, the answer to RQ1 is obtained by logging the expert information for all sent packets (both inferred request & response packets), and counting the number of valid and invalid packets.

To assess the sensitivity to the threshold T, the experiment was repeated, increasing T from 0 to 1 in increments of 0.1.

RQ2: Can inferred message patterns elicit valid server responses?. To answer RQ2 we monitor the response codes to the synthesised request messages from the test servers. Depending on the protocols, their response is either a valid response, or indicates that the server has failed to properly parse the message. If there is no response at all, this is counted as an invalid response.

For HTTP and SIP the status-code element is a three-digit number indicating the result of the attempt to understand and satisfy the request. The first digit of the status code defines the class of the server response. For the HTTP protocol and SIP protocols, a "400 error" code signifies a bad request – i.e. a poorly formed message, so we count this as invalid. Other responses indicate that the message has been parsed correctly. These may (for both protocols) include a 402 error (URI / URL not found) – we count this as valid, because it is a response to a packet that probably has a valid structure, albeit with a nonsensical URL.

For TFTP protocol, we have observed three types of responses: 01 (File not found), 02 (Access violation), and 05 (Illegal TFTP operation). Server errors with codes 01 and 02 are not caused by invalid packet structures, and are thus treated as valid. The 05 error code is received due to malformed packets, and are thus considered invalid.

For Samba responses, the only responses we observed were either because a request was pointing towards a missing file, or because there was an invalid combination of parameters (even though the parameters had been correctly parsed). These were counted as valid responses. There were however many requests to the server that did not receive any response at all; these were all treated as invalid responses.

As above, the experiment was repeated for different values of the alignment threshold T, increasing from 0 to 1 in increments of 0.1.

5.4. Results

RQ1: Are inferred message patterns syntactically correct?. The plots in Figure 3 contain the proportion of valid vs. invalid packets sent for each protocol. For the HTTP and SIP protocols the results indicate that we were able to generate syntactically valid message patterns for most values of T. However, lower values of T tend to produce more synthetically valid packets than higher values. For HTTP all packets are valid for $T < 0.5$, and for SIP over half of the packets are valid for $T < 0.8$.

For the binary TFTP and SMB/CISF protocols, the validity of the packets is much more sensitive to the T parameter, and fails to yield any valid packets for $T > 0.3$ in both cases.

Conclusion: For all protocols, if we choose a suitable threshold value T, which tends to be $T \leq 0.3$ for all protocols, the inferred protocol structures are sufficiently accurate to enable the synthesis of syntactically valid messages.

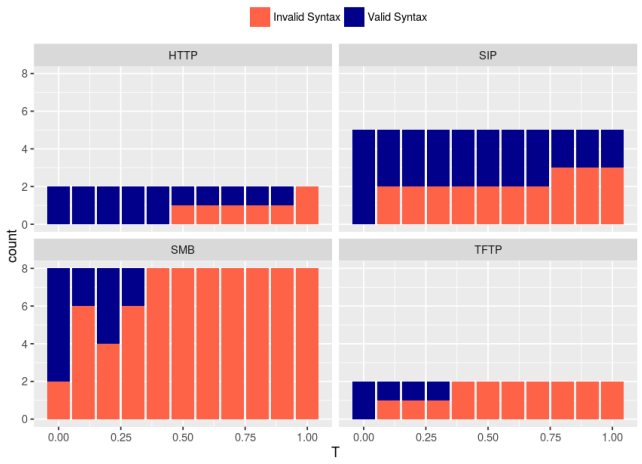


Figure 3. Number of syntactically valid/invalid message patterns - as indicated by Wireshark - in relation to the choice of the generalisation threshold (T).

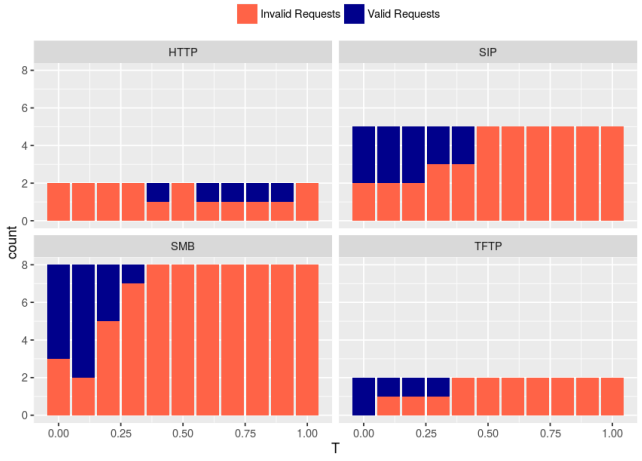


Figure 4. Number of valid/invalid message requests - returned by protocol servers - in relation to the choice of the generalisation threshold (T).

RQ2: Can inferred message patterns elicit valid server replies?. The plots in Figure 4 show the indications from the server responses as to whether the synthesised messages were valid or not. In contrast to the results for RQ1, synthesised packets for HTTP generated with low values of T failed to elicit valid server responses. Instead, this only occurred for T values in the range from 0.40 to 0.80.

For SIP, the successful responses were again restricted to lower threshold values (for $T \geq 0.5$ all packets were treated as invalid). As with HTTP, even though Wireshark categorised several of the generated packets to be valid, they elicited an invalid response from the server.

With TFTP the responses from the server exactly matched the categorisations by Wireshark. Again, low values of T tended to lead to synthetic packets that elicited a valid response.

With SMB, the responses from Samba pretty much matched the syntactic validity assessments by Wireshark. Again, lower values of T tended to yield more valid packets. However, as was the case with HTTP, a lower value of T did not guarantee the best response. For SMB, $T = 0.1$ led to a majority of synthesised packets being parsed as valid by the server.

Conclusion: For all protocols, if we choose a suitable value for T, it is possible to synthesise packet structures from inferred protocol structures where at least half of the packets will be correctly parsed by the server. The selection of T is not as clear cut as with RQ1; values that led to packets that were deemed to be syntactically correct do not necessarily lead to packets that are recognised by the server in question. Finding a suitable value of T may therefore require a degree of experimentation. Since lower values of T tended to produce the best results (at least for SIP, SMB and TFTP), it would make sense to start off with lower values and work upwards.

5.5. Discussion

5.5.1. Selecting a suitable Threshold. Our experiments indicate that the inferred packet structures can be accurate enough to synthesise valid packets. However, the validity does depend on the choice of a threshold parameter T. Choosing a high T means that an inferred packet will only contain those aligned symbols that occur in the majority of messages. If we choose a low value of T, we allow the alignment to encompass characters that occurred in a smaller proportion of the messages.

Although a low value of T has tended to produce the best results, it is generally a bad idea to set T too low (e.g. 0). If T is too low, the final alignment will end up containing many characters that are irrelevant to the packet structure - i.e. are not delimiters or keywords but instead belong to field data. However, these can still lead to combinations of characters that can lead to messages that are invalid, which is what appears to have happened with the HTTP traces, which contained various optional headers and variable-length fields.

Accordingly, when it comes to selecting a suitable value of T, it is important to accommodate the similarity of the data. In general it makes sense to start from a low value of T, e.g. $T < 0.3$. However, if the data is highly heterogeneous, then it is probably worth considering a higher value to avoid too many false-positives.

5.5.2. Threats to Validity. With respect to external and internal validity, we consider the following threats:

- We have only applied it to four protocols. It is of course possible that there are different protocols for which the performance of the approach would be different.
- For each protocol, we downloaded samples of traffic from a repository. However, since this data was not collected in an active testing environment, it is generally not the case that every feasible behaviour of the protocols in question was covered in the traces.

- With respect to internal validity, our server-responses were from specific server implementations. It is possible that other implementations of the same protocol could have provided different responses.

We did take care to mitigate these threats where possible. We selected protocols that were broadly diverse (all are widely used, spanning both text and binary families). The network data is intended to represent 'typical' network usage. It was not biased by the authors and is not (at least in its descriptions) associated with a specific purpose. For the network servers, all have been used in previous studies and considered to be accurate implementations of the original protocol specifications. As will be discussed in the Future Work section, we intend to address these threats further with a larger empirical study.

6. Qualitative Evaluation

This section provides a brief qualitative assessment of the alignments generated, and compares these to equivalent alignments generated by the Needleman-Wunsch based Protocol Informatics tool [3] (which is also to the best of our knowledge the only publicly available implementation of a protocol message alignment tool).

To illustrate the difference in the quality of alignments, we have generated a small set of similar HTTP request messages consisting of ten *GET* messages, shown in Table 7. Each message is composed of a *request-line* and various *header fields*. We could not use genuine trace messages because we need to keep them sufficiently short to fit into this paper. The aim of this comparison is to show how our segment-based alignment fares against the state-of-the art, and to explain some of the phenomena that were discussed in the previous experiments.

In our comparison, we will focus on protocol fields that are correctly aligned by both tools. The protocol fields of interest here are keywords and delimiters, since these fields are hard-coded in protocol implementations and considered a prerequisite for correct parsing [17]. We consider the set of protocol keywords in these messages to be:

```
GET, HTTP/1.1, Host, User-Agent,
Connection, close
```

We consider the set of delimiters to be:

```
/, [space], :, \r\n
```

6.1. Results

The alignment produced by Protocol Informatics (using the default user parameters) is shown in Table 8. The corresponding segment-based alignment is shown in Table 9. The tables are best viewed in colour, where the keywords and delimiters are highlighted in red and blue respectively.

Both approaches successfully align the `GET` followed by the space and the '` /`', which begins all messages. They also successfully align the '`\r\n\r\n`' that is used to finish each message.

However, for the keywords and delimiters that happen in between, the PI approach makes several misalignments, whereas they are all correctly aligned by the segment-based algorithm. The PI algorithm misaligns the `HTTP/1.1` segments for messages 4 and 8. This is because it misaligns the space before the keyword `HTTP/1.1`, forcing the `HTTP/1.1` to the end of the message. These are correctly aligned by the segment-based algorithm. The PI approach also fails to correctly align any of the `User-Agent` keywords, which are again correctly aligned by our approach.

One apparent reason for the improved quality of the segment-based alignments versus PI is that the latter focuses solely on the alignments of individual characters, whereas the former incentivises alignments of longer segments. This is why the segment-based alignments tend to successfully align the `User-Agent` keyword, whereas the same keyword disintegrates in the PI alignment.

7. Conclusions and Future Work

We have shown how a segment-based alignment algorithm can be used to align network packets for the purpose of detecting packet structures. This is a departure from typical approaches, which have been based on variants of the Needleman-Wunsch algorithm, which is prone to inaccuracy when applied to network data.

Specifically, we have:

- Shown how segment-based alignment can be used to generate accurate alignments.
- Developed a proof of concept implementation.
- Experimentally shown that the packet structures identified by our alignments can be used to automatically synthesise new messages that can be recognised by a server.
- Show that the messages tend to be syntactically valid, albeit for a suitable choice of threshold parameter.

For our future work we intend to improve our segment-based alignment implementation in several directions, both with respect to the immediate inference technique, as well as applications that it can feed into.

With respect to the alignment itself, we intend to run a larger experiment, taking into account a broader range of network protocols, and factoring in other variables, such as the volume of data used for an alignment. We also intend to further study the role of T, and to come up with more concrete heuristics as to how to choose a suitable value. We intend to examine the fine-grained structure of the identified message patterns, using more detailed pattern inference algorithms such as the *Sequitur* algorithm [33].

Ultimately, our goal is to use the inferred packet structures as a basis for effective automated fuzz-testing of network protocols. Therefore, alongside our work on improving the alignment activities, we are seeking to construct an automated network protocol fuzzer that can, thanks to technologies such as this one, devise useful test cases for protocols just by observing their passive interactions. For

Table 7. TEN SYNTHESISED HTTP MESSAGES.

No	Message
01	`GET / HTTP/1.1\r\nHost: www.google.co.uk\r\nUser-Agent: Dillo/3.4\r\n\r\n`
02	`GET /myuri.html HTTP/1.1\r\nHost: www.abc.cba.net\r\n\r\n`
03	`GET /myveryverylonguril.html HTTP/1.1\r\nUser-Agent: Firefox\r\n\r\n`
04	`GET /verylongutiueyryhwpirrytbeyuujw.html HTTP/1.1\r\n\r\n`
05	`GET /shorteruri.png HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n`
06	`GET /abitlongeruri.html HTTP/1.1\r\nHost: www.abc.cba.net\r\n\r\n`
07	`GET /veryextsj.jpg HTTP/1.1\r\nUser-Agent: Firefox\r\n\r\n`
08	`GET /lklkjlkjlkwefrweriuw bjsdfcbjhjsdfkSDFJSGDFJHNBIUWEIWYUERuiyrweiu.html HTTP/1.1\r\n\r\n`
09	`GET / HTTP/1.1\r\nHost: www.le.ac.uk\r\nUser-Agent: wget/1.2\r\n\r\n`
10	`GET /thisislonglonglonglongrilieuiueiuelk.html HTTP/1.1\r\nHost: www.google.co.uk\r\nConnection: close\r\n\r\n`

this we specifically intend to develop more sophisticated packet synthesis methods, to enable the generation of large volumes of diverse network traffic that can be used to expose and highlight vulnerabilities in networks.

References

[1] F. Aarts, H. Kuppens, J. Tretmans, F. Vaandrager, and S. Verwer. Improving active mealy machine learning for protocol conformance testing. *Machine learning*, 96(1-2):189–224, 2014.

[2] J. a. Antunes and N. Neves. Automatically complementing protocol specifications from network traces. In *Proceedings of the 13th European Workshop on Dependable Computing*, EWDC '11, pages 87–92, New York, NY, USA, 2011. ACM.

[3] M. A. Beddoe. Network protocol analysis using bioinformatics algorithms. http://www.4tphi.net/ awalters/PI/PI.html, 2004.

[4] G. Bossert, F. Guihéry, and G. Hiet. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 51–62, New York, NY, USA, 2014. ACM.

[5] J. Caballero, M. G. Kang, S. Venkataraman, D. Song, P. Poosankam, and A. Blum. Fig: Automatic fingerprint generation. In *In 14th Annual Network and Distributed System Security Conference (NDSS*, 2007.

[6] J. Caballero and D. Song. Automatic protocol reverse-engineering: Message format extraction and field semantics inference. *Comput. Netw.*, 57(2):451–474, Feb. 2013.

[7] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 317–329, New York, NY, USA, 2007. ACM.

[8] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[9] C. Y. Cho, D. Babi ć, E. C. R. Shin, and D. Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 426–439, New York, NY, USA, 2010. ACM.

[10] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 110–125, Washington, DC, USA, 2009. IEEE Computer Society.

[11] W. Cui, J. Kannan, and H. J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 14:1–14:14, Berkeley, CA, USA, 2007. USENIX Association.

[12] W. Cui, V. Paxson, N. Weaver, and R. Katz. Protocol independent adaptive replay of application dialog. Feburary 2006.

[13] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz. Tupni: automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 391–402, New York, NY, USA, 2008. ACM.

[14] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[15] O. Esoul and N. Walkinshaw. Finding clustering configurations to accurately infer packet structures from network data. Technical report, Arxiv, 2016.

[16] K. R. Fall and W. R. Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[18] M. S. A. Greene and P. Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 2007.

[19] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[20] C. Hertel. *Implementing CIFS: The Common Internet File System*. Prentice Hall Professional Technical Reference, 2003.

[21] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[22] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 11:1–11:12, New York, NY, USA, 2008. ACM.

[23] T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, AISec '12, pages 37–48, New York, NY, USA, 2012. ACM.

[24] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 203–214, Washington, DC, USA, 2005. IEEE Computer Society.

[25] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 313–326, New York, NY, USA, 2006. ACM.

[26] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[27] Microsoft. Microsoft smb protocol and cifs protocol overview. https://msdn.microsoft.com/en-gb/library/windows/desktop/aa365233(v=vs.85).aspx, 2016.

Table 8. ALIGNMENT RESULTS PRODUCED BY THE PROTOCOL INFORMATICS PROJECT USING THE DEFAULT USER PARAMETERS (MATCH=1,MISMATCH=0,GAP=0).

No	Alignment
01	GET /---------- HTTP/1.1\r\nHost: www.google.--co.uk\r\n--User-Age-----nt-: Di-llo/3.4\r\n\r\n
02	GET /---------m-yuri---.html HTTP/1.1\r\nHost: www.--abc.cb-a.-------ne--t-------\r\n\r\n
03	GET /---------myveryverylonguril.html HTTP/1.1---------\r\n--User-Age-----nt-: Firefo---x\r\n\r\n
04	GET /---------ve-rylongutiu-----eyryh---------wpir-----rytbeyuujw.html --HTTP/1.1\r\n\r\n
05	GET /---------shorteruri.png HTTP/1.1\r\nHost: www.--bbc.--co--m\r\n--User-Age-----nt-: Di-llo/3.4\r\n\r\n
06	GET /--abi---------tlong---e-ruri---.html HTTP/1.1\r\nHost: www.--abc.cb-a.-------ne--t-------\r\n\r\n
07	GET /---------very-e-----xtsj.jpg HTTP/1.1---------\r\n--User-Age-----nt-: Firefo---x\r\n\r\n
08	GET /lklkjlkjlkwefr---weriuw bjsdfcbjhjsdfkSDFJSGDFJHNBIUWEIWY---------UE--Rui-------yrwe-iu--.html --HTTP/1.1\r\n\r\n
09	GET /---------HTTP/1.1\r\nHost: www.---le.ac-.uk\r\n--User-Age-----nt-: -w-get/1.2\r\n\r\n
10	GET /thisislong--l---------onglon----glongrilieuiueuiuelk.html HTTP/1.1\r\nHost: www.google.c--o.uk\r\nConne-ctio-----n--: ---clo--se\r\n\r\n

Table 9. ALIGNMENT RESULTS PRODUCED BY OUR SEGMENT-BASED ALIGNMENT TOOL.

No	Alignment
01	GET /---------HTTP/1.1\r\nHost: www.google.co-.uk\r\nUser-Agent: Dil-------lo/3.4\r\n\r\n
02	GET /my---------u---------ri---------.html HTTP/1.1\r\nHost: www.abc--.cba.-------n-----et-------\r\n\r\n
03	GET /myveryv---e-rylong---------u---------ril---------.html HTTP/1.1\r\n-----------User-Agent: Firef-------ox--\r\n\r\n
04	GET /----v---e-rylong---------u---------tieyryhwpirryteyuujw.html HTTP/1.1---------\r\n\r\n
05	GET /sho---------rt---------er-u---------ri---------.png HTTP/1.1\r\nHost: www.bbc---.co-m-\r\nUser-Agent: Dil-------lo/3.4\r\n\r\n
06	GET /ab--i---------tlong---------er-u---------ri---------.html HTTP/1.1\r\nHost: www.abc---.cba.-------n-----et-------\r\n\r\n
07	GET /-----v---e-ry---------ex-t---------sj---------.jpg HTTP/1.1\r\n-----------User-Agent: Firef-------ox--\r\n\r\n
08	GET /lklkjlkjlkwefrw---------eriuw bjsdfcbjhjsdfkSDFJSGDFJHNBIUWEIWYUERuiyrweiu---------.html HTTP/1.1---------\r\n\r\n
09	GET /---------HTTP/1.1\r\nHost: www.---le.ac-.uk\r\nUser-Agent: wg-et-------/1.2\r\n\r\n
10	GET /thisi---------slonglonglonglonglrilieuiu---------eiuelk---------.html HTTP/1.1\r\nHost: www.google.co-.uk\r\nCon-----n-----ection: close--\r\n\r\n

[28] B. Morgenstern. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. In *Proc. Natl. Acad. Sci. USA*, pages 12098–12103, 1996.

[29] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, Mar. 1999.

[30] B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999.

[31] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[32] Netresec. Network forensics and network security monitoring. http://www.netresec.com/, 2016.

[33] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Int. Res.*, 7(1):67–82, Sept. 1997.

[34] R. Pang, V. Paxson, R. Sommer, and L. Peterson. binpac: a yacc for writing application protocol parsers. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 289–300, New York, NY, USA, 2006. ACM.

[35] V. Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, Dec. 1999.

[36] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *J. Mach. Learn. Res.*, 9:23–48, June 2008.

[37] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol, 2002.

[38] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: An improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6(1):66+, 2005.

[39] N. Walkinshaw, K. Bogdanov, J. Derrick, and J. Paris. Increasing functional coverage by inductive testing: a case study. In *Testing Software and Systems*, pages 126–141. 2010.

[40] Y. Wang, X. chun Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Z. 0002, and L. Guo. A semantics aware approach to automated reverse engineering unknown protocols. In *ICNP*, pages 1–10. IEEE, 2012.

[41] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo. Biprominer: Automatic mining of binary protocol features. In *Proceedings of the 2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '11, pages 179–184, Washington, DC, USA, 2011. IEEE Computer Society.

[42] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo. Inferring protocol state machine from network traces: a probabilistic approach. In *Proceedings of the 9th international conference on Applied cryptography and network security*, ACNS'11, pages 1–18, Berlin, Heidelberg, 2011. Springer-Verlag.

[43] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. Reformat: automatic reverse engineering of encrypted messages. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 200–215, Berlin, Heidelberg, 2009. Springer-Verlag.

[44] Wireshark. Advanced topics: Expert information. https://www.wireshark.org/docs/wsug_html_chunked/ChAdvExpert.html, 2016.

[45] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda. Automatic network protocol analysis. In *Network and Distributed Systems Security Symposium (NDSS)*, 2 2008.

[46] X. Zhang. *Position Weight Matrices*, pages 1721–1722. Springer New York, New York, NY, 2013.

[47] H. Zimmermann. Osi reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.