Received: 18 April 2020 | Revised: 22 July 2020 | Accepted: 14 September 2020

DOI: 10.1002/nem.2140

RESEARCH ARTICLE

WILEY

A message keyword extraction approach by accurate identification of field boundaries

Young-Hoon Goo[1] | **Kyu-Seok Shim**[1] | **Min-Seob Lee**[2] | **Myung-Sup Kim**[3]

[1]Advanced KREONET Center, Korea Institute of Science and Technology Information, Daejeon, South Korea

[2]DPX Development Team, Ahnlab, Seongnam, South Korea

[3]Department of Computer and Information Science, Korea University, Sejong, South Korea

Correspondence
Myung-Sup Kim, Department of Computer and Information Science, Korea University, Sejong, South Korea.
Email: tmskim@korea.ac.kr

Funding information
Institute of Information and Communications Technology Planning and Evaluation, Grant/Award Number: 2018-0-00539; Korea Institute of Science and Technology Information (KISTI); Development of Blockchain Transaction Monitoring and Analysis Technology; Korea Government (MSIT); Institute for Information & communication Technology Planning & Evaluation (IITP); Development of SaaS SW Management Platform based on 5Channel Discovery technology for IT Cost Saving); Korea Evaluation Institute of Industrial Technology (KEIT), Grant/Award Number: 20008902; Ministry of Trade, Industry & Energy (MOTIE, Korea); Industrial Strategic Technology Development Program—Advanced Technology Center+(ATC+)

Summary

With the recent exponential increase in internet speeds, the traditional network environment is evolving into a high-capacity network environment. Network traffic usage is also increasing exponentially, as are new malicious behaviors and related applications. Most of these applications and malicious behaviors use unknown protocols for which the structure is inaccessible; hence, protocol reverse engineering is receiving increasing attention in the field of network management. Various approaches have been proposed, but they still suffer from misidentification of field boundaries. To understand message structures properly, it is important to identify accurately the boundaries of the fields constituting the protocol message; accurate keyword extraction based on this approach leads to the correct inference of message types, semantics, and state machine. In this study, we propose a message keyword extraction method using accurate identification of field boundaries from delimiter inference and statistical analysis. Through the identification of field boundaries, messages can be subdivided into fields. We evaluate the efficacy of the proposed method by applying it to several textual and binary protocols. The proposed method showed better results than did other previous studies for both textual and binary protocols.

1 | INTRODUCTION

With the trend toward development of the Internet of Things (IoT) and pervasive computing technology, the traditional network environment is evolving into a high-capacity high-speed network environment. Global internet traffic is also growing exponentially, as are malicious intentions and development of new applications.[1] Unfortunately, many applications and malicious interventions use unknown protocols for which specifications are not documented. In addition, various IoT sensor networks use customized protocols to reduce expenditure on licenses and energy usage and size. Industrial control systems use proprietary protocols for operating technology (OT).[2] The constantly evolving and

increasingly sophisticated cyber-attacks also use proprietary protocols to avoid detection. Thus, there is a growing need for protocol reverse engineering to achieve efficient network operation and management in areas such as malware analysis, network vulnerability assessment, and construction of effective detection and prevention mechanisms.[3] In particular, protocol reverse engineering has become increasingly important over the last decade, in light of the proliferation of IoT devices, intensifying global cyber-conflicts and numerous security incidents.

Protocol reverse engineering is the process of inferring an unknown protocol structure. The general process consists of three phases: syntax inference, semantics inference, and state machine inference, which represents the order in which message types are transmitted. For semantics and state machine inference to be successful, syntax inference must be performed correctly, and accurate keyword extraction is crucial for accomplishing correct syntax inference.[4–9] In this paper, we use the term "keyword" to refer to a value that one field can have; accurate keyword extraction refers to the process of extracting values that exactly one field can have, not noise such as a combination of values from two or more fields or a portion of the value of one field. Various approaches have been proposed, but they still suffer from a misidentification of field boundaries that leads to incorrect extraction of keywords. To determine the structure of the message, which is subdivided into fields accurately, we must first identify the field boundaries properly.

In this study, we propose a message keyword extraction method using accurate identification of field boundaries. The proposed method first infers three types of delimiters, which are the basic delimiter, the key delimiter, and the singleton delimiter step by step. In the case of plaintext protocols, the proposed method extracts keywords using these delimiters. In the case of binary protocols, the proposed method extracts keywords using frequent pattern mining with positional analysis.

The rest of this paper is organized as follows. Section 2 describes related work and the definition of the problem. Section 3 presents the details of the proposed method. In Section 4, we evaluate the proposed method through experiments involving HTTP/1.1, FTP, SMTP, DNS, and NetBIOS/SMB protocols. Finally, Section 5 presents conclusions and suggestions for future work.

2 | RELATED WORK AND PROBLEM SCOPE

2.1 | Protocol reverse engineering

Protocol reverse engineering is a method that extracts the structure of the target unknown protocol. It is generally conducted on application layer protocols.[10] In the early 1990s, when the internet became prevalent, interoperability and software compliance was the main purpose, but, over the past decade, its purposes and target protocol layers have become more diversified.[11] The aim of protocol reverse engineering is to extract the syntax, semantics, and timing corresponding to the fundamental components of the target unknown protocol.

Prior protocol reverse engineering methods were conducted manually so that all elements of the protocol structure could be extracted clearly. However, these manual methods require considerable time, and results can vary depending on the analyst's ability.[12] Therefore, these methods are not compatible with the rapid increase in the number of new applications and the vast amount of traffic in today's network environment.[13–15]

To address these problems, several automatic protocol reverse engineering methods have been studied; these are divided into two categories: execution trace analysis and network trace analysis.[16] Execution trace analysis must build an execution monitoring system that logs how the program processes messages of the target unknown protocol. This analysis is only made possible by acquiring a program that uses the target unknown protocol. However, access to the program of an unknown protocol is rarely available due to concealment and obfuscation efforts.[17–20] Conversely, network trace analysis only requires network packets and does not need a program implementing the target protocols.[21–24] Thus, we have used network trace analysis in view of the practicality of this type of analysis.

2.2 | Related work

The previous methods using network trace analysis include AutoReEngine,[25] Netzob,[26] Trifilo,[27] Veritas,[28] ReverX,[29] and Pext.[30] Among these, Veritas and Pext focus on inferring a state machine; hence, we compare our method with AutoReEngine and Netzob, which focus on both a state machine and protocol syntax. The reasons for choosing these methods are as follows: Netzob is one of the most promising methods and is an open-source project that can be used

easily; AutoReEngine uses frequent pattern mining considering position, as does the proposed method so that they can be compared easily and intuitively.

Netzob is a top-down method. It regards a packet as a message and measures the similarity between each pair of messages with the UPGMA algorithm. Then, it groups the messages into message type clusters if the similarity between two messages exceeds the threshold entered by a user. Lastly, for each cluster, meaning message format, the method runs the Needleman-Wunch algorithm and performs sequence alignment to divide the message format into fields.

AutoReEngine is a bottom-up method. It first reconstructs the target unknown protocol traffic into messages. It then uses the Apriori algorithm to extract byte-streams that frequently occur in sessions and determines final fields among the extracted byte-streams using positional analysis. Subsequently, it runs the Apriori algorithm again to extract message formats using fields as input.

2.3 | Problem definition

Ideal protocol reverse engineering involves extracting all of the three components of protocol, which are syntax, semantics, and timing. Syntax represents the message types in the target protocol and their structures, including field sequence, boundaries, size, and value. Semantics represents the meaning of each field, making up the message type. Timing is expressed in a state machine that represents the order of message types. The correct inference of all these components requires extracting keywords in messages accurately for message type clustering.

The previous methods have limited ability to infer message structures. First, some methods assume that the delimiter is already known, and they divide messages into fields by using the specific delimiter. These methods cannot extract keywords in a case in which the target protocol has no defined delimiters. Second, because many methods use only frequency analysis to determine the keywords, message structures with incorrect field boundaries are extracted. Third, the previous methods using a top-down approach first cluster messages into message types without keyword extraction and then subdivide each message type into fields only within each cluster. Thus, the keywords are expressed differently in different clusters; hence, it is difficult for an analyst to understand the correct message structures. The most powerful feature for clustering messages is keywords. However, because these methods do not use keywords as a feature, they extract too many message types; this leads to a poor understanding of protocol structures.

Hence, we focus on accurate keyword extraction, and, for this purpose, we perform analyses designed to identify the field boundaries.

3 | THE PROPOSED METHOD

In this section, we first provide an overview of the proposed method and a definition of the terms used; we then describe the proposed method in detail.

3.1 | Overview and definition of terms

As described earlier, the main purpose of the proposed method is to identify field boundaries and extract keywords to ascertain message structures. Figure 1 displays an overview of the proposed method. The proposed method first performs preprocessing and determines whether the target protocol is a plaintext protocol or a binary protocol. If the protocol is a plaintext protocol, the method determines whether there are predefined delimiter candidates in the target protocol and then extracts keywords using the inferred delimiters. If the protocol is a binary protocol or there are no inferred delimiters, the method extracts keywords using frequent pattern mining with positional analysis.

To infer the delimiters of the target protocol, we first define three types of delimiters: the basic delimiter, the key delimiter, and the singleton delimiter, as shown in Figure 2.

1. Basic delimiter: A message is composed of a series of fields, and each field may have a nested structure that can be divided into several subfields. A basic delimiter is a one byte or two byte separator that can divide a message into the highest-level fields. We call the highest-level field an initial field.

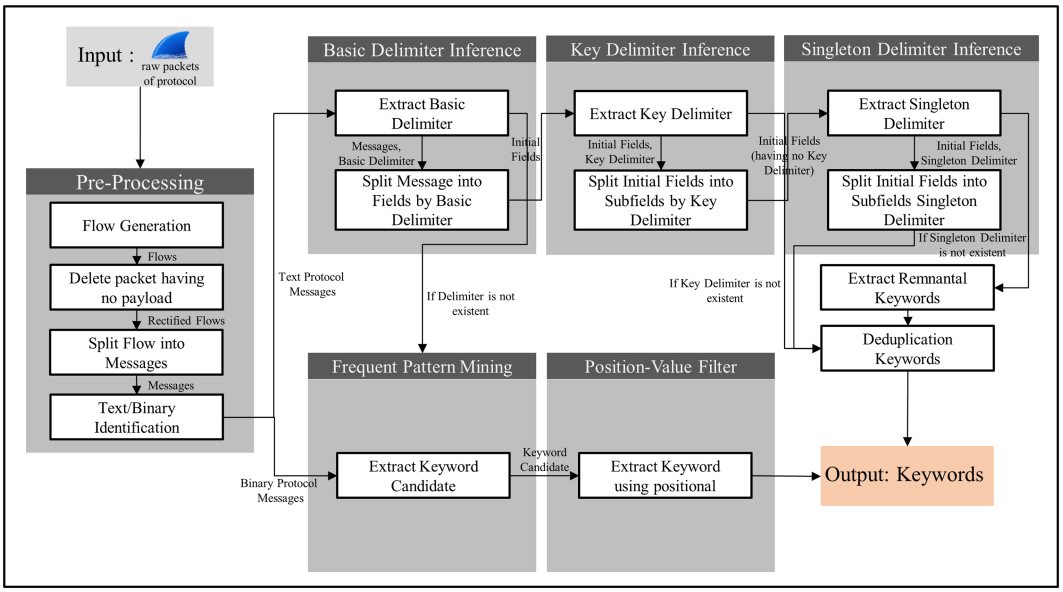

FIGURE 1 Overview of proposed method

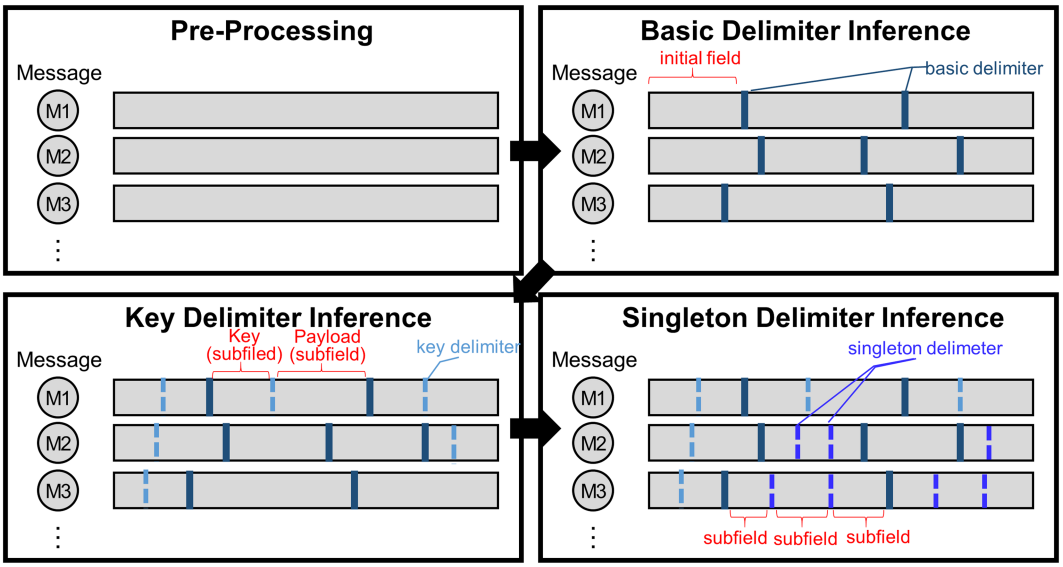

FIGURE 2 Concept of three types of delimiters

2. Key delimiter: One initial field may contain a key indicating the characteristics of the field, and a value. A key delimiter is a separator that can divide the initial field into a key and value.
3. Singleton delimiter: One initial field can be divided into several subfields that contain different values. A singleton delimiter is a separator for accomplishing this.

3.2 | Preprocessing

In the preprocessing step, the proposed method reforms the raw packets of the target unknown protocol into flows; a flow is a sequence of packets that have the same five tuples. The method then reorders the packets to resolve the retransmission and out-or-order problem[31] and excludes packets that have no payload of the transport layer. Next, it

reconstructs each flow into message sequences by following a heuristic method. A message is a protocol data unit (PDU). For a transport control protocol flow, the proposed method assembles consecutive chunks having the same direction into one message. For a user datagram protocol flow, it treats the data following the transport layer header in one packet as one message. Finally, it checks whether the target protocol is a plaintext or binary protocol. If at least 60% of the messages contain 80% of bytes as ASCII printable characters (alphanumerics, symbols, and signs, which correspond to 0x 21 to 0x7e of the ASCII code), the target protocol is identified as a plaintext protocol; if not, the protocol is identified as a binary protocol.

Subsequently, if the protocol is a plaintext protocol, it is sent to the basic delimiter inference step; if not, it is sent to the frequent pattern mining step.

3.3 | Basic delimiter inference

In this step, the proposed method first determines whether or not the target protocol has a basic delimiter. If it has a basic delimiter, it is sent to the next module in this step; if not, it is sent to the frequent pattern mining step.

We have five predefined delimiter candidates: "0x0d0a" (carriage return line feed), "0x20" (space), "0x3b" (semicolon), "0x3d" (equal sign), and "0x3a20" (colon space). The user can add more delimiter candidates.

Figure 3 presents the pseudocode of the basic delimiter inference step. First, the proposed method determines whether each delimiter candidate has two properties:

1. The number of times a delimiter appears in each message is similar for all messages in one direction. Message formats for one protocol can be broadly divided into request and response formats; hence, messages are formatted similarly in each direction.
2. The positions at which delimiters appear in each message are similar for all messages.

To apply these two criteria for each delimiter candidate, the proposed method proceeds as follows: While traversing all messages in each direction, it counts the number of times the delimiter candidate appears for each message and calculates the average of all the offsets, which are positions of the delimiter candidate, for each message. Next, it calculates the variance for offset averages and the entropy of occurrence counts. If both the entropy and the variance are low enough, the delimiter candidate is extracted as the basic delimiter. Based on our experience, we set the thresholds for the entropy and the variance to 3.2 and 2.5, respectively.

Finally, the proposed method extracts initial fields using the basic delimiter.

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i), \text{where } x_i = \text{occurence counts of the delimiter candidate for message } i. \quad (1)$$

Equation 1 provides a definition for entropy; a discrete random variable X is for occurrence counts of the delimiter candidate in each message. This represents uncertainty in the information; hence, if all the occurrence counts of each

```
Input: Messages, Delimiter Candidates
Output: Basic Delimiter, Initial Fields
01: foreach Delimiter Candidates DC_i do
02:     foreach Messages M_i do
03:         Create M_i.list_offset_of_DC_i;
04:         Calculate M_i.offset_Avrg(M_i.list_Offset_of_DC_i);
05:         Calculate M_i.occr_cnt_of_DC_i;
06:     end
07:     Calculate DC_i.variance(list of M_i.offset_Avrg);
08:     Calculate DC_i.entropy(list of M_i.occr_cnt_of_DC_i);
09:     if(DC_i.entropy < Threshold1 && (DC_i.variance < Threhold2))
10:         Extract DC_i as Basic Delimiter;
11:         return;
12:     endif
13: end
```

FIGURE 3 Pseudocode of basic delimiter inference

message are the same, the entropy is zero. In this step, a small entropy means the delimiter candidate satisfies the first property. A small variance of offset average means the delimiter candidate satisfies the second property.

3.4 | Key delimiter inference

In the key delimiter inference step, the proposed method infers the key delimiter and then, by using the key delimiter, it extracts all the keys as keywords.

Figure 4 presents the pseudocode of the key delimiter inference step. A highest-level field, which we call an initial field, generally consists of key and value. Therefore, one property of the key delimiter, which separates key and value, is that the delimiter appears for each initial field only once. The other property is that the positions in which the delimiter appears for all initial fields are similar.

To extract the key delimiter, the proposed method performs the following for each delimiter candidate; in traversing all initial fields, it counts the number of times the delimiter candidate appears for each initial field and stores the offset of the delimiter candidate. If the occurrence count of the delimiter candidate is not 1, the delimiter candidate is not the key delimiter. Otherwise, it calculates the variance of offsets. If the variance is low enough and represents the minimum value among all of the delimiter candidates, the delimiter candidate is extracted as the key delimiter. Based on our experience, we set the threshold for the variance to 1.5. We can then extract all of the keys as keywords using the key delimiter.

3.5 | Singleton delimiter inference

In the singleton delimiter inference step, the proposed method distinguishes the singleton delimiter that can separate several different values from the initial fields that do not contain the key delimiter. Subsequently, among subfields separated by the singleton delimiter, subfields are extracted that can have small kinds of values as keywords because a keyword is related to meta-information of the protocol.

Figure 5 presents the pseudocode of the singleton delimiter inference step. The singleton delimiter also has two properties that are similar to those of the basic delimiter. One is that the number of times the delimiter appears in each initial field is similar for all initial fields. The other is that the positions in which delimiters appear in each initial field are similar for all initial fields. To extract the singleton delimiter, the proposed method performs the following for each delimiter candidate; first, in traversing all initial fields that do not have the key delimiter, it calculates the number of times the delimiter candidate appears in each of the initial fields and the average of all the offsets of the delimiter

```
Input: Initial Fields, Delimiter Candidates
Output: Key Delimiter, Keywords
01: foreach Delimiter Candidates DC_i do
02:     foreach Initial Fields F_i do
03:         Calculate F_i.offset_of_DC_i;
04:         Calculate F_i.occr_cnt_of_DC_i;
05:     end
06:     if(All of F_i.occr_cnt_of_DC_i != 1)
07:         continue;
08:     endif
09:     Calculate DC_i.variance(F_i.offset_of_DC_i);
10:     if(Minimum variance > DC_i.variance)
11:         Minimum variance ← DC_i.variance;
12:     endif
13: end
14: foreach Delimiter Candidates DC_i do
15:     if(DC_i.variance < Threhold1 && DC_i.variance == Minimum variance)
16:         Extract DC_i as Key Delimiter;
17:         break;
18:     endif
19: end
20: foreach Initial Fields F_i do
21:     Split F_i into Key and Payload using Key Delimiter;
22:     Extract Key as Keyword;
23: end
```

FIGURE 4 Pseudocode of key delimiter inference

FIGURE 5 Pseudocode of singleton delimiter inference

```
Input: Initial Fields(having no Key Delimiter), Delimiter Candidates
Output: Singleton Delimiter
01: foreach Delimiter Candidates DC_i do
02:     foreach Initial Fields F_i do
03:         Create F_i.list_offset_of_DC_i;
04:         Calculate F_i.offset_Avrg(F_i.list_Offset_of_DC_i);
05:         Calculate F_i.occr_cnt_of_DC_i;
06:     end
07:     Calculate DC_i.variance(list of F_i.offset_Avrg);
08:     Calculate DC_i.entropy(list of F_i.occr_cnt_of_DC_i);
09:     if(DC_i.entropy < Threshold1 && (DC_i.variance < Threhold2))
10:         Extract DC_i as a Singleton Delimiter;
11:         return;
12:     endif
13: end
```

candidate for each initial field. Next, it calculates the variance of offset averages and the entropy of occurrence counts. If both the entropy and the variance are low enough, the delimiter candidate is extracted as the singleton delimiter. Based on our experience, we set the thresholds for the entropy and the variance at 0.4 and 1.5, respectively.

After extracting the singleton delimiter, the proposed method divides the initial fields into several subfields and assigns an index to each subfield indicating the position in the initial field. Then, to extract keywords, the proposed method performs the following: for each initial field, that is, subfield sequence, it counts the number of singleton delimiters in the initial field. Traversing all initial fields whose position is the same in a message with the same direction, (1) if the numbers of singleton delimiters of all initial fields are the same, it converts each subfield to a specific value using a hash function. This hash function is used to assign each different subfield with a unique value. The method calculates the entropy of the hash values of subfields with the same index. If the entropy is low enough, the proposed method extracts subfields with the same index as keywords. This means the randomness of values for this subfield is low. We use 2.0 for the threshold of the entropy, based on our empirical evidence. (2) If the numbers of singleton delimiters of all initial fields whose position is the same in a message with the same direction differ, the method extracts the subfields for which the index is lower than the minimum number of singleton delimiters between the initial fields. Figure 6 presents the pseudocode for the keyword extraction in this step.

FIGURE 6 Pseudocode of extracting keyword using singleton delimiter

```
Input: Initial Fields(having no Delimiter), Singleton Delimiter
Output: Keywords
01: foreach Initial Fields F_i do
02:   Split F_i into Subfield Sequence using Singleton Delimiter;
03:   Construct Subfield Sequence in opposite direction using Singleton Delimiter;
04: end
05: foreach Request/Response Subfield Sequence do
06:   Count Singleton Delimiter;
07:   Index all of subfileds in Request/Response Subfield Sequence;
08: end
09: if(All Request/Response Subfield Sequences's Singleton Delimiter Count is Same)
10:   foreach Request/Response Subfield Sequences do
11:     Hashing each subfield;
12:   end
13:   foreach Request/Response Subfield Sequences do
14:     Calculate Entropy of hash key for each index(Subfield Position);
15:     if(SubField Position's Entropy < 2)
16:       Extract all subfields with the same Position as keywords;
17:     endif
18:   end
19: else
20:   foreach Request/Response Subfield Sequence do
21:     foreach Subfield Position SP_i do
22:       if(i <= min.Singleton Delimiter Count)
23:         Extract subfields with the same Position as keywords;
24:       endif
25:     end
26:   end
27: endif
```

3.6 | Remnantal field extraction

In this step, the proposed method extracts additional keywords in the remnantal fields. A remnantal field refers to an initial field that does not have the singleton delimiter and the key delimiter.

First, the proposed method traverses all initial fields whose position is the same in a message with the same direction; it then extracts all lengths of keys of the initial fields with the same position. If all lengths of keys are the same, the method performs the following: traversing all remnantal fields for which the position is the same in a message with the same direction, it divides each of the remnantal fields with the same position into two parts based on the length of the key. If all the first parts are ASCII printable characters and all the second parts begin with a non-ASCII printable character, each first part is extracted as a keyword. After this step, the proposed method deletes duplicate keywords.

3.7 | Frequent pattern mining with positional analysis

If the target unknown protocol does not have any delimiters or is a binary protocol, the proposed method performs this step after the preprocessing step. In this step, we extract keyword candidates using the AprioriAll algorithm.[32] In this algorithm, the threshold, minimum support, is very important. Support is the ratio of the sequences having target subsequence to the total number of sequences. The AprioriAll algorithm extracts frequent subsequences whose support is greater than the user-defined minimum support threshold. In our method, the support is defined as shown in Equation 2. We use the minimum support threshold of 0.6.

$$\text{Support} = \frac{\text{Number of messages having the target bytestream}}{\text{Total number of messages}}. \tag{2}$$

After extracting keyword candidates using the AprioriAll algorithm, the proposed method performs the position-value filter step. In general, keywords appear in a somewhat fixed position within the message. Thus, in this step, the proposed method extracts keyword candidates satisfying this property. Figure 7 presents the concept of the position-value filter. A startoffset refers to the position of the keyword candidate relative to the beginning of the message. An endoffset refers to the position of the keyword candidate relative to the end of the message. First, the proposed method traverses all keyword candidates and finds all messages having the keyword candidate. Next, it traverses all messages having the keyword candidates and calculates the variance of the startoffsets (Sov) and the variance of the endoffsets (Eov). Then, PV for the keyword candidate is calculated as the smaller of the values of Sov and Eov. If the PV of the keyword candidate is low enough, this means that the keyword candidate appears in a nearly fixed position. The proposed method extracts as keywords the keyword candidates for which the PV is low enough. We use 0.05 for the threshold for PV, based on our experience.

4 | EXPERIMENT AND RESULTS

In this section, we describe two evaluation metrics and then describe the experimental results resulting from the application of our method to HTTP/1.1, FTP, SMTP, DNS, and NetBIOS/SMB protocols.

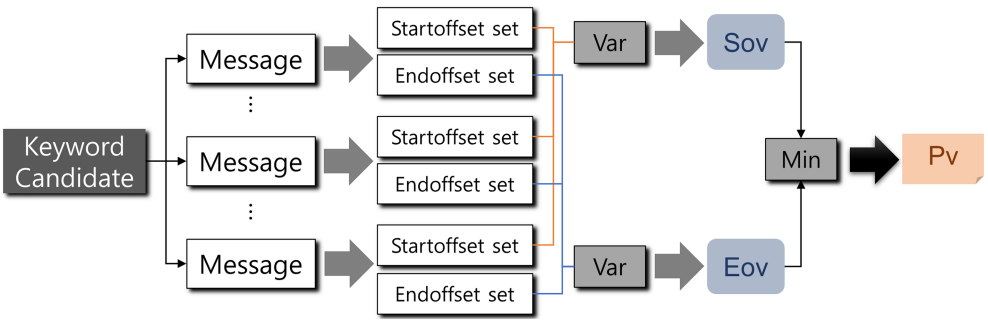

FIGURE 7 Concept of position-value filter

4.1 | Data description and evaluation metrics

To verify the proposed method, we collected traffic for five protocols from several hosts in the campus network of Korea University: HTTP/1.1, FTP, and SMTP for plaintext protocols and DNS and NetBIOS/SMB for binary protocols. Table 1 provides quantitative information on flow, packet, byte, and message-unit for input traffic in each protocol. The sizes of the input traffic shown in Table 1 are for results after the preprocessing step.

We evaluate the proposed method using the two completeness units, keyword-unit, and message-unit. Completeness refers to the number of ground-truth keywords extracted. Completeness_F is a keyword-unit metric, and it represents the number of ground-truth keywords reflected in the extracted keyword, as shown in Equation 3. Completeness_M is a message-unit metric, and it represents the number of messages from which all keywords are extracted, as shown in Equation 4.

$$\text{Completeness}_K = \frac{\begin{array}{c}\text{Number of ground}-\text{truth keywords}\\ \text{matched with the extracted keywords}\end{array}}{\text{Total number of ground}-\text{truth keywords}}, \tag{3}$$

$$\text{Completeness}_M = \frac{\begin{array}{c}\text{Number of messages}\\ \text{from which all ground}-\text{truth keywords}\\ \text{are extracted as the extracted keywords}\end{array}}{\text{Total number of messages}}. \tag{4}$$

4.1.1 | Ground-truth description

Many previous studies evaluate their methods with their own performance evaluation methods and ground-truth. It is not easy to compare the performances of different methods because of inconsistencies in the intrinsic ways evaluations are applied. Our criteria for the definition of ground-truth keywords are based on the following grounds; the ground-truth keywords are generated from input traffic. The keyword indicates the function of the protocol; hence, the keywords found in the input traffic reflect the protocol functions generally used. Therefore, the keyword appears frequently in input messages, and the randomness of the values possible for the field corresponding to the keyword is low.

For the HTTP/1.1 protocol, we set *Method* and *Version* fields in the request line of messages, *Status code* and *Phrase* fields in the response line, and *Header Name* fields in the header line to the ground-truth keyword. For the SMTP and the FTP protocols, we set *Request Command* and *Response code* fields as the ground-truth keyword. For the DNS protocol, we set *Flag*, *Number of question records*, *Number of answer records*, *Number of authoritative records*, *Number of additional records fields*, *Query type*, *Query class*, *Domain type*, and *Domain class* fields as the ground-truth keywords. For the NetBIOS/SMB protocol, we set *0xFFSMB*, *Command*, *NT Status*, and *Flags* fields as the ground-truth keywords.

TABLE 1 Traffic information

	Flows	Packets	Messages (By Our Preprocessing Step)	Bytes, K
HTTP/1.1	575	7,232	2,048	8,763
SMTP	333	6,683	6,337	569
FTP	732	39,536	37,117	4,547
DNS	2,170	4,349	4,349	723
NetBIOS/SMB	12	31,053	19,372	4,969

TABLE 2 Experimental results

Protocol	HTTP/1.1	SMTP	FTP	DNS	NetBIOS/SMB
Proposed method					
$Completeness_K$	100.00% (89/89)	100.00% (24/24)	100.00% (30/30)	81.82% (18/22)	81.25% (13/16)
$Completeness_M$	100.00% (2,034/2,034)	100.00% (6,337/6,337)	100.00% (37,117/37,117)	78.29% (3,405/4,349)	91.44% (17,713/19,372)
AutoReEngine					
$Completeness_K$	38.20% (34/89)	87.50% (21/24)	63.33% (19/30)	45.45% (10/22)	18.75% (3/16)
$Completeness_M$	10.52% (214/2,034)	77.81% (4,931/6,337)	38.63% (14,339/37,117)	25.96% (1,129/4,349)	0.00%(0/19372)
Netzob					
$Completeness_K$	80.90% (72/89)	87.50% (21/24)	90.00% (27/30)	81.82% (18/22)	75.00% (12/16)
$Completeness_M$	70.99% (5,134/7,232)	79.60% (5,320/6,683)	83.57% (33,041/39,536)	69.51% (3,023/4,349)	51.39 (15,958/31,053)

4.2 | Results

Table 2 shows the experimental results. The thresholds used in each method were set as the defaults, as follows. For the proposed method, the minimum support value was 0.5, and the other thresholds were set as described earlier. For AutoReEngine, the session support rate threshold is 0.5, and the site-specific session-set support rate threshold is 0.5. For Netzob, the similarity threshold is 0.5. The proposed method showed better results than did AutoReEngine and Netzob, in terms of Completeness_F and Completeness_M for five protocols. For the proposed method, both the metrics are 100% for plaintext protocols. However, the evaluation result for binary protocols is slightly lower than that for plain text protocols. Because a binary protocol generally does not use a delimiter and contains more dense data, perfect identification of all field boundaries is very difficult; this represents a challenging problem. However, keyword inference for the binary protocol of the proposed method is significant in that it considers the position of keywords for more precise keyword extraction. When Completeness_F is lowered, Completeness_M also tends to decrease significantly, as shown in Table 2. This indicates that correct keyword extraction is important for the inference of message format.

Conversely, AutoReEngine shows lower performance than does the proposed method. AutoReEngine only uses frequency-based analysis without identifying whether the protocol is plaintext-based or binary-based, so it cannot extract the fields that do not frequently occur in messages or sessions.

Netzob does not perform the message assemble step (in preprocessing) that converts packets into PDU, and it assumes one packet is one message. In addition, as mentioned in Section 2, Netzob uses a top-down method. That is, without extracting the keyword, the messages are clustered first and then separated into field units within each message cluster. The fields that make up the Netzob message type are a series of static field and GAP field pairs; a static field

TABLE 3 Extracted keywords

	Sample Extracted Keywords
Proposed method	GET
	POST
	HTTP/1.1
	Host
	Content-Type
	X-Frame-Options
	Vary
	Keep-Alive
	...
AutoReEngine	e|0d||0a|User-Agent: Mozilla/5.0 (Windows NT
	GET/
	3b|Win64|3b|x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36|0d||0a|Accept:
	8|0d||0a|Referer:
	HTTP/1.1|0d||0a|Host:
	/|3b|q=0.8|0d||0a|
	|0a|Expire
	GMT|0d||0a|C
	...
Netzob	GET/img/section/bg HTTP/1.1|0d|0a|Host: cafeimg.naver.net|0d||0a|Connection: keep-alive
	Accept:*/*|0d|0a|Pragma: no-cache|0d|0a|Cookie: data==00dp9UX0D4gSFpsvKlpsHwoWo-, 1V7IalpDo7m_bHU
	GMT|0d||0a|Connection: close|0d||0a||0d||0a||3c|!DOCTYPE html|3e||0a||3c|html xml: lang=|22|ko|22||3e||0a||3c|head|3e||0a||3c|meta name=|22|
	HTTP/1.1200 OK|0d||0a|Server: nginx|0d||0a|Date: Tue, 10 Apr 20
	GMT|0d||0a|Location: http://mail1.dongwon.mil.kr
	|0d||0a|Content-Type: text/html|3b|charset=UTF-8|0d||0a|Connection: Keep-Alive|0d||0a|Content-Length:

means a field that has only one value, and it is an aligned sequence of Netzob. We assumed that the static fields of Netzob are the keywords. Because Netzob uses top-down clustering, the same ground-truth keywords are extracted in different forms within each message cluster.

Table 3 shows the extracted keywords for the proposed method, AutoReEngine, and Netzob by comparing them for the HTTP/1.1 protocol. This shows that the keyword extracted by AutoReEngine and Netzob has a noise because of the misidentification of field boundaries.

5 | CONCLUSION

In this study, we proposed a novel protocol reverse engineering method for accurately extracting keywords by considering field boundary identification. Accurate keyword extraction plays a vital role in inferring the structure of message types and the subsequent analysis, including semantics and state machine. We defined three types of delimiters, which are the basic delimiter, the key delimiter, and the singleton delimiter. The proposed method extracts keywords by inferring delimiters for plaintext protocols. For binary protocols, the method uses frequent pattern mining with positional analysis to extract keywords elaborately. From experiments on five protocols and by comparing our method with prior work, we demonstrated the superiority of the proposed methodology. We plan to design a hybrid method using both network trace analysis and execution trace analysis. In addition, we plan to perform research that can identify field boundaries for the binary protocol more precisely.

ACKNOWLEDGEMENTS

This work was partly supported by the Industrial Strategic Technology Development Program—Advanced Technology Center+(ATC+) grant funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and the Korea Evaluation Institute of Industrial Technology (KEIT) (No. 20008902, Development of SaaS SW Management Platform based on 5Channel Discovery technology for IT Cost Saving), Institute for Information & Communication Technology Planning & Evaluation (IITP) grant funded by Korea Government (MSIT) (No.2018-0-00539, Development of Blockchain Transaction Monitoring and Analysis Technology), and Korea Institute of Science and Technology Information (KISTI).

ORCID

Young-Hoon Goo https://orcid.org/0000-0002-3013-7011
Kyu-Seok Shim https://orcid.org/0000-0002-3317-7000
Min-Seob Lee https://orcid.org/0000-0003-4854-9521
Myung-Sup Kim https://orcid.org/0000-0002-3809-2057

REFERENCES

1. Cisco.com. Visual networking index complete forecast highlights. 2020. https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2022_Forecast_Highlights.pdf. Published 2018. Accessed September 16, 2020.
2. Chang Y, Choi S, Yun JH, Kim S. One step more: automatic ICS protocol field analysis. In: *International Conference on Critical Information Infrastructures Security*. Springer; 2017:241-252.
3. Duchene J, Guernic CL, Alata E, Nicomette V, Kaaniche M. State of the art of network protocol reverse engineering tools. *J Computer Virol Hacking Techniq*. 2018;14(1):53-68.
4. Jiang J, Versteeg S, Han J, Hossain MA, Schneider J-G. A positional keyword-based approach to inferring fine-grained message formats. *Future Gener Comput Syst*. 2020;102:369-381.
5. Jiang D, Li C, Ma L, Ji X, Chen Y, Li B. ABInfer: a novel field boundaries inference approach for protocol reverse engineering. 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). May 2020:19–23.
6. Jiang J, Versteeg S, Han J, et al. P-gram: positional N-gram for the clustering of machine-generated messages. *IEEE Access*. 2019;7:88504-88516.
7. Sun F, Wang S, Zhang C, Zhang H. Unsupervised field segmentation of unknown protocol messages. *Comput Commun*. 2019;146:121-130.
8. Kleber S, Kopp H, Kargl F. NEMESYS: network message syntax reverse engineering by analysis of the intrinsic structure of individual messages. *In Proc. 12th USENIX Conference on Offensive Technologies*. August 2018:1–13.
9. Hossain MA, Versteeg S, Han J, Kabir MA, Jiang J, Schneider J-G. Mining accurate message formats for service API. *In Proc. 25th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2018:266–276.

10. Goo YH, Shim KS, Park JT, Chae BM, Moon HW, Kim MS. A method of protocol reverse engineering for clear protocol specification extraction. *KNOM Rev*. 2017;20(2):11-23.
11. Narayan J, Shukla SK, Clancy TC. A survey of automatic protocol reverse engineering tools. *J ACM Computer Surveys*. 2015;48(3):1-26.
12. Borisov N, Nrumley DJ, Wang HJ, Guo C. Generic application-level protocol analyzer and its language. *Network and Distributed System Security Symposium (NDSS)*. 2007:1–13.
13. Goo YH, Shim KS, Chae BM, Kim MS. Framework for precise protocol reverse engineering based on network traces. *NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium*. 2018:1–4.
14. Goo YH, Shim KS, Lee MS, Kim MS. Protocol specification extraction based on contiguous sequential pattern algorithm. *IEEE Access*. 2019;7:36057-36074.
15. Sija BD, Goo YH, Shim KS, Hasanova H, Kim MS. A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view. *Security Commun Netw*. 2018;2018(8370341):1-17.
16. Goo Y-H, Sija BD, Lee S-H, Kim M-S. Analyzing the difference between network trace-based and execution trace-based protocol reverse engineering in three perspectives. *In Proc. Symposium of the Korean Institute of communications and Information Sciences*. 2017:82–83.
17. Caballero J, Yin H, Liang Z, Song D. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. *In* Proc. 14th ACM Conference on Computer and Communications Security (CCS). October 2007:317–329.
18. Cui W, Peinado M, Chen K, Wang HJ, Irun-Briz L. Tupni: automatic reverse engineering of input formats. *In* Proc. 15th ACM Conference on Computer and Communications Security (CCS). October 2008:391–402.
19. Comparetti PM, Wondracek G, Kruegel C, Kirda E. Prospex: protocol specification extraction. *In Proc. 30th IEEE Symposium on Security and Privacy*. 2009:110–125.
20. Caballero J, Song D. Automatic protocol reverse-engineering: message format extraction and field semantics inference. *Comput Netw*. 2013;57(2):451-474.
21. Beddoe M. The protocol informatics project. http://www.4tphi.net/~awalters/PI/PI.html. Published 2004. Accessed September 16, 2020.
22. Shevertalov M, Mancoridis S. A reverse engineering tool for extracting protocols of networked applications. *In Proc. 14th Working Conference on Reverse Engineering (WCRE'07)*. June 2007:229–238.
23. Wang Y, Zhang Z, Yao DD, Qu B, Guo L. Inferring protocol state machine from network traces. *In Proc. 9th Applied Cryptography and Network Security International Conference (ACNS'11)*. 2011:1–18
24. Bermudez I, Tongaonkar A, Iliofotou M, Mellia M, Munafo MM. Automatic protocol field inference for deeper protocol understanding. *In Proc. 14th IFIP Networking Conference*. May 2015:1–9.
25. Luo J-Z, Yu S-Z. Position-based automatic reverse engineering of network protocols. *J Netw Comput Appl*. 2013;36(3):1070-1077.
26. Bossert G, Guihery F, Hiet G. Towards automated protocol reverse engineering using semantic information. *In Proc. 9th ACM symposium on Information, computer and communications security*. June 2014:51–62.
27. Trifilo A, Burschka S, Biersack E. Traffic to protocol reverse engineering. *In IEEE Symposium on Computational Intelligence for Security and Defense Applications*. December 2009:1–8.
28. Wang Y, Zhang Z, Yao DD, Qu B, Guo L. Inferring protocol state machine from network traces: a probabilistic approach. *In Proc. 9th Applied Cryptography and Network Security International Conference (ACNS)* June 2011;6715:1–18.
29. Antunes J, Neves N, Verissimo P. Reverse engineering of protocols from network traces. *In Proc. 18th Working Conference on Reverse Engineering (WCRE)*. October 2011:169–178.
30. Shevertalov M, Mancoridis S. A reverse engineering tool for extracting protocol of networked applications. *In Proc. 18th Working Conference on Reverse Engineering (WCRE)*, October 2007:229–238.
31. Lee S-K, Ahn H-M, Kim M-S. Packet out-of-order and retransmission in statistics-based traffic analysis. *In Proc. 16th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. September 2014:1–6.
32. Agrawal R, Srikant R. Mining sequential pattern. *In Proc. 11th International Conference on Data Engineering*. 1995:3–14.

AUTHOR BIOGRAPHIES

Young-Hoon Goo is a postdoctoral researcher in Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea. He received his B.S. and Ph.D. degree (integrated program) in the Department of Computer and Information Science, Korea University, Korea, in 2016 and 2020 respectively. His research interests include Internet traffic classification, network management, and protocol reverse engineering.

Kyu-Seok Shim is a postdoctoral researcher in Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea. He received his B.S., M.S., and Ph.D. degree in the Department of Computer and Information Science, Korea University, Korea, in 2014, 2016, and 2020, respectively. His research interests include Internet traffic classification, network management, and protocol reverse engineering.

Min-Seob Lee is a junior developer in Ahnlab, Seongnam, Korea. He received the B.S. and M.S. degree in the Department of Computer and Information Science, Korea University, Korea, in 2018 and 2020, respectively. He joined Ahnlab in 2020. His research interests include Internet traffic monitoring and analysis.

Myung-Sup Kim is a Full Professor in the Department of Computer Convergence Software, Korea University, Korea. He received his B.S., M.S., and Ph.D. degree in Computer Science and Engineering from POSTECH, Korea, in 1998, 2000, and 2004, respectively. From September 2004 to August 2006 he was a postdoctoral fellow in the Department of Electrical and Computer Engineering, University of Toronto, Canada. He joined Korea University in September 2006. His research interests include Internet traffic monitoring and analysis, service and network management, and Internet security.

How to cite this article: Goo Y-H, Shim K-S, Lee M-S, Kim M-S. A message keyword extraction approach by accurate identification of field boundaries. *Int J Network Mgmt*. 2021;31:e2140. https://doi.org/10.1002/nem.2140