A Method for Extracting Static Fields in Private Protocol Using Entropy and Statistical Analysis

Min-Seob Lee
Computer Information and Science
Korea University
Sejong, Korea
chenlima2@korea.ac.kr

Young-Hoon Goo
Computer Information and Science
Korea University
Sejong, Korea
gyh0808@korea.ac.kr

Kyu-Seok Shim
Computer Information and Science
Korea University
Sejong, Korea
kusuk007@korea.ac.kr

Sung-Ho Yoon
A&B Center
LG Electronics
Seoul, Korea
sungho.sky.yoon@lge.com

Se-Hyun Ji
Computer Information and Science
Korea University
Sejong, Korea
sxzer@korea.ac.kr

Myung-Sup Kim
Computer Information and Science
Korea University
Sejong, Korea
tmskim@korea.ac.kr

***Abstract*—Modern society is turning into the environment in which high-capacity network traffic generated by the development of high-speed internet. As a result, new applications and malicious behaviors are increasing exponentially. Most protocols that occur in these network environments are private protocols. Because private protocols do not have any specifications open, it is very important to analyze the structures of the private protocol for efficient network management and security. Various protocol reverse engineering methodologies have been studied so far, but there is not standardized methodology to extract the protocol's field. Therefore, this paper proposes a methodology for clearly extracting fields of the smallest unit of protocol's structure, and conducts experiments and validates performance on the protocols that are actually being used.**

Keywords—Protocol Reverse Engineering, Private Protocol, Field

I. INTRODUCTION

Modern society is turning into the environment in which high-capacity network traffic generated by the development of high-speed internet. New applications and malicious behaviors are also increasing continuously as network traffic usage increases exponentially. Many of the various protocols that arise in this environment are private protocols. Since various malicious behaviors are being carried out using these private protocols, it is very important to analyze the structures of the private protocols for efficient network management and security.

A study named Protocol Reverse Engineering is being studied steadily to analyze the structure of the private protocol. Although various existing studies have proposed a protocol reverse engineering methodology, there is no standardized methodology that clearly extracts the fields of the protocol clearly, and each has some limitation. Protocol reverse engineering can be divided into two main ways. The first method is the passive protocol reverse engineering. This method is very time-consuming and error prone because it performs protocol reverse engineering manually. To solve this problem, the automatic protocol reverse engineering method was proposed as the second method. This method has the advantage of automatically performing protocol reverse engineering, but the previously proposed methodologies extract too many message types or do not extract fields clearly. In addition, current methodologies have the limitation that cannot extract all fields in the target protocol. An ideal protocol reverse engineering should clearly extract the protocol structure, including syntax, semantics, and FSM.

The goal of the proposed methodology is to extract the clear protocol's fields that are necessary to perform for the ideal protocol reverse engineering. The key idea of this methodology is to extract delimiters for text-based protocols only and to separate fields by defining and extracting the three types of delimiters. In addition, only if the delimiter does not exist or is based on binary-based protocols, the fields are extracted using the Apriori algorithm that is sequential pattern-mining techniques, and the location information of the fields in the message.

The composition of this paper followed by the introduction, defines related works and the problem to be solved in chapter 2 and details the methodology proposed in chapter 3. Chapter 4 describes the results of experiments on three types of text-based protocols that are in use, and finally chapter 5 describes conclusions and future works.

II. RELATED WORKS

A. Protocol Reverse Engineering

Protocol reverse engineering is the extraction of specifications for unknown or undocumented network protocols, generally meaning extracting specifications for unknown application layer protocols at the OSI 7 layer[1]. The goal of protocol reverse engineering is to extract detailed structures of protocols related to syntax, semantics, and timing, which are components of the protocol The detailed structure of a protocol means what types of messages the target protocol has, how the message types are organized, and in what order they operate.

Traditional protocol reverse engineering used passive methods. Because a person directly checks the specifications of a network protocol, the elements of the protocol can be

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2018R1D1A1B07045742) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2018-0-00539,Development of Blockchain Transaction Monitoring and Analysis Technology)

extracted correctly. However, depending on human error is, the results are variable, error-prone, and time-consuming. Therefore, it is inevitable to use automatic protocol reverse engineering methods to react to various malicious behaviors in the network environment in today's society where high-capacity network traffic occurs.

Automatic protocol reverse engineering can be generally divided into the execution trace-based analysis method and the network trace-based analysis method[2]. The execution trace-based analysis method monitors the execution of the program binaries that use the target protocol and uses the logged execution traces. This method is realistically difficult because it can be only used when a program binary using the target protocol can be obtained. By contrast, since the network trace-based analysis method use network packets, the analysis is possible even if program binaries are not accessible. Therefore, we used the network trace-based analysis method.

B. *Prior Study*

The prior studies using the network trace-based analysis method includes AutoReEngine[3], Netzob[4], Trifilo[5], Veritas[6], ReverX[7], and Pext[8]. Among these prior studies, methodologies for extracting fields from protocols are AutoReEngine, Netzob, Veritas, ReverX. In this paper, Netzob and AutoReEngine are selected as comparison methods to validate the performance of the proposed methodology. The reason why we select these methodology is that the Netzob is an open source project, and AutoReEngine is comparatively easy to implement.

Netzob is the protocol reverse engineering method that uses the top-down method. In the first step, it reassembles the target network traffic into messages. Then, it performs sequence alignment by using the Needleman-Wunch algorithm and extracts symbols. A symbol means a message type. In the final step, it uses the UPGMA algorithm to measure the similarity between each symbol and groups the symbols into clusters with a similarity greater than the one entered by the user.

AutoReEngine is the protocol reverse engineering method that uses the bottom-up method. In the first step, it reassembles the target network traffic into messages. Then, it uses the Apriori algorithm to extract strings that frequently occur in sessions into fields. In the final step, it uses field's location information in messages to extract final fields.

C. *Problem Definition*

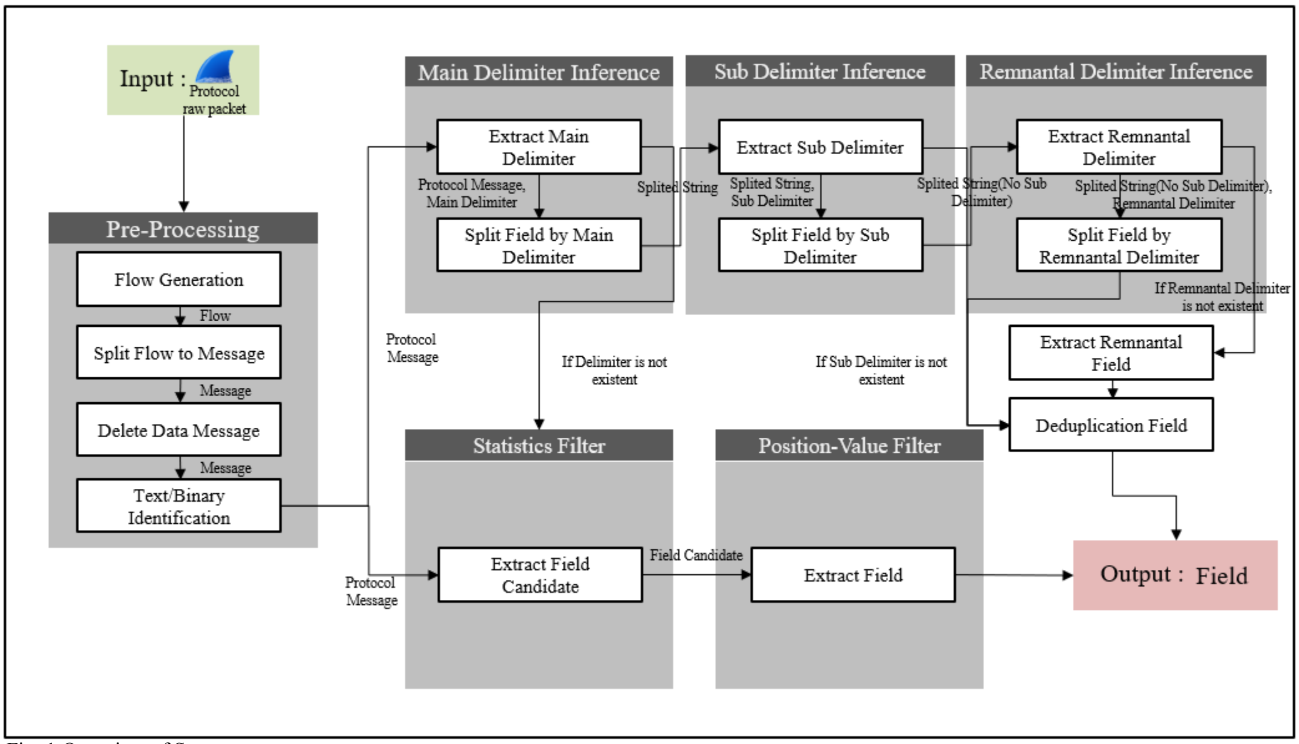
Protocol reverse engineering extracts three results : syntax, semantics, and FSM. Syntax means the format of how messages are configured in the target protocol. Semantics means the meaning of each field that configure a message type and FSM means a finite Automata that describes the order in which message types operate.

The prior studies extract field by each methodology, but they have several limitations. First, some methodologies have a limitation that divide simply fields by known delimiter. In this case, the methodology extracts data part that have the same value with delimiter into the field. In addition, if the target protocol do not have delimiter, methodology cannot extract field. Second, some methodologies extract too many message types because they apply sequence alignment by the same value. Because a message type means a distinct characteristic of the target protocol, the methodology that extracts too many message types is not an ideal protocol reverse engineering method. Therefore, we suggest the methodology that extracts delimiters clearly and divides field clearly by using delimiters. In addition, if the target protocol has no delimiters, the methodology extracts field using by statistical information and location information of fields.

III. PROPOSED METHOD

A. *Overview*

Figure 1 shows the overview of the proposed method in this paper.



Input : Protocol raw packet
Pre-Processing
Flow Generation
Flow
Split Flow to Message
Message
Delete Data Message
Message
Text/Binary Identification
Main Delimiter Inference
Extract Main Delimiter
Protocol Message, Main Delimiter
Split Field by Main Delimiter
Splited String
Sub Delimiter Inference
Extract Sub Delimiter
Splited String, Sub Delimiter
Split Field by Sub Delimiter
Splited String(No Sub Delimiter)
Remnantal Delimiter Inference
Extract Remnantal Delimiter
Splited String(No Sub Delimiter), Remnantal Delimiter
Split Field by Remnantal Delimiter
If Remnantal Delimiter is not existent
Extract Remnantal Field
Deduplication Field
Protocol Message
If Delimiter is not existent
If Sub Delimiter is not existent
Statistics Filter
Position-Value Filter
Protocol Message
Extract Field Candidate
Field Candidate
Extract Field
Output : Field

Fig. 1 Overview of System

B. Pre-Processing

In the first step, the target network traffic reassembles into flows. A flow means the message's sequence with the same 5-tuples(Source IP Address, Destination IP Address, L4 Protocol, Source Port, Destination Port). Next, this methodology splits flows into messages and deletes messages that only have data part. Finally, the methodology discriminates whether the target network protocol is text-based or binary-based. If target network protocol is text-based protocol the methodology check that target network protocol has delimiter. If target network protocol has delimiter, the methodology goes to the main delimiter inference step and goes to the statistics filter step if it doesn't have delimiter or it is binary-based protocol.

C. Main Delimiter Inference

```
Input : Protocol Message, Delimiter Candidate
Output : Main Delimiter

01: foreach Delimiter Candidates do
02:     foreach Messages do
03:         Calculate Offset;
04:         Calculate Occurrence count in each Message;
05:     end
06: end
07: foreach Delimiter Candidates do
08:     Calculate Entropy(Occurrence count);
09: end
10: foreach Delimiter Candidates do
11:     foreach Messages do
12:         Calculate Average of Offset;
13:         Calculate Variance(Average of Offset);
14:     end
15: end
16: foreach Delimiter Candidates do
17:     if(Entropy of Occurrence count < Threshold1
18:         && (Threshold2 < Variance of Offset Average< Threhold3))
19:         Extract to Main Delimiter;
20:     endif
21: end
```

Fig. 2 Pseudocode of Extract Main Delimiter

In this step, the methodology extracts the main delimiter that separates messages into each field using by two properties of main delimiter. The first property of the main delimiter is that it occurs in a similar number in all messages. Because a particular protocol consists mostly of similar types of messages, each message also has a similar number of delimiters. The second property of delimiter is that it is distributed evenly within the message. Because the role of the main delimiter is to separate fields within a message, it must be evenly distributed within the message and, if it occurs to be one-sided, it is difficult to define the main delimiter. Therefore, in this paper, the methodology extracts the main delimiter using by the two properties of the main delimiter. Figure 2 shows pseudocode of this step.

First, for predefined delimiter candidates, the methodology calculates offset and number of occurrences in each message. Next, for predefined delimiter candidates, the methodology calculates the entropy of the number of occurrences in the message. Then, for predefined delimiter candidates, the methodology calculates the variance of offset average. For a delimiter candidate to extract into the main delimiter, the entropy value must be very small and the variance value of the offset average should not too small or too large. This is because the entropy value is small means that the delimiter candidate occurs in a similar number in messages, and that the variance of the offset average is not too small or too large means that it occurs evenly in messages. Therefore, in this paper, the delimiter candidate that has small entropy value and not too small or too large variance value of the offset average extract into the main delimiter. Then, the methodology split messages into each field used by the main delimiter. If a non-printable character or a main delimiter occurs after the main delimiter, the methodology determines that the latter part is the data of the target protocol and does not extract data to field.

D. Sub Delimiter Inference

```
Input : Splited String, Delimiter Candidate
Output : Sub Delimiter

01: foreach Delimiter Candidates do
02:     foreach Splited String do
03:         Calculate Offset;
04:         Calculate Occurrence count in each Message;
05:     end
06: end
07: foreach Delimiter Candidates do
08:     Calculate Variance(Occurrence count);
09: end
10: foreach Delimiter Candidates do
11:     foreach Splited String do
12:         Calculate Average of Offset;
13:         Calculate Variance(Average of Offset);
14:     end
15: end
16: foreach Delimiter Candidates do
17:     if(Variance of Occurrence count < Threshold1
18:         && Variance of Average of Offset == min.Variance of Offset Average
19:         Extract to Sub Delimiter;
20:     endif
21: end
```

Fig. 3 Pseudocode of Extract Sub Delimiter

In this step, the methodology extracts key field in the field. Fields are generally divided into key and value or value and value. Therefore, in this step, the methodology extracts sub delimiter that separates key and value. Sub delimiter has two properties similar to the main delimiter. The first property of sub delimiter is a property that it occurs in a similar number in all split string. The split string means field that extracted by main delimiter. Because a field generally consists of key-value or value-value, sub delimiter usually occurs once in a field. Second property of sub delimiter is a property that it occurs in a similar offset in split string. Therefore, in this paper, the methodology extracts sub delimiter using by the two properties of sub delimiter. Figure 3 shows the pseudocode of this step.

First, for predefined delimiter candidates except for the main delimiter, the methodology calculates offset and number of occurrences in each split string. Next, for predefined delimiter candidates except for the main delimiter, the methodology calculates the variance of the number of occurrences in split string. Then, for predefined delimiter candidates except for the main delimiter, the methodology calculates variance of offset average. In order for a delimiter candidate except main delimiter to extract into a sub delimiter, the variance of the number of occurrences must be very small and the variance of offset average is the minimum value. Therefore, in this study, the delimiter candidate except for the main delimiter that has two very small variance values extract into sub delimiter. Then, the methodology split split strings into each key field used by the sub delimiter.

E. Remnantal Delimiter Inference

In first step, the methodology extracts remnantal delimiter in split string with no sub delimiter. The algorithm used in this step is very similar to the algorithm for extracting sub delimiter. The difference from the case of extracting the sub delimiter is that the delimiter candidate is extracted as remnantal delimiter when the variance of offset average satisfies Threshold.

```
Input : Splited String(with no Sub Delimiter), Remnantal Delimiter
Output : Field(Key)

foreach Splited String do
    split Splited String to Field Sequence
    construct Request Response Field Sequence
end
foreach Request/Response Field Sequence do
    calculate Remnantal Delimiter Count
    numbering each Field Position in Request/Response Field Sequence
end
if(All Request/Response Field Sequence's Remnantal Delimiter Count is Same)
    foreach Request/Response Field Sequence do
        apply hash function to each Field
    end
    foreach Request/Response Field Sequence do
        calculate Entropy each Position
    end
    if(Field Position's Entropy < 2)
        extract all Field in Position
    endif
else
    foreach Request/Response Field Sequence do
        foreach Field Position do
            if(Field Position <= min.Remnantal Delimiter Count)
                extract to Field
            else
                extract Remnantal Field Sequence as Field;
            endif
        end
    end
endif
```

Fig. 4 Pseudocode of Split Field by Remnantal Delimiter

In second step, the methodology extract remnantal field. Figure 4 shows pseudocode of this step. First, the methodology divides split string into request direction with response direction and calculates remnantal delimiter number for each direction. In addition, the methodology separates the fields using the remnantal delimiter and assigns indexes to the fields. If number of remnantal delimiter in each direction split string are all same, the methodology applies hash function at each field. Then, the methodology calculates entropy of the same index fields. A small entropy value means that the field values of occur in the index are not variable. Therefore, the methodology extracts the fields belonging to the index with a small entropy value as key fields. If number of remnantal delimiter in each direction split string are not same, the methodology separates the fields upto the minimum number of remnantal delimiter and extracts them into the key fields.

F. Remnantal Field Inference

In this step, the methodology extracts key field in split strings(remnantal strings) that do not have remnantal delimiter and sub delimiter. First, the methodology checks whether fields in each direction extracted by sub delimiter are the same size and field index. The field index means the index of each direction field when it is separated by a sub delimiter. If fields size and field index in each direction extracted by sub delimiter are the all same, the methodology cuts the remnantal strings into the field size as calculated above and extracts them into the key fields. If only fields size in each direction extracted by sub delimiter is the all same, the methodology doesn't extract remnantal strings as the key fields. After this step, the methodology extracts final fields by eliminating duplicated fields.

G. Statistics Filter

$$message_{supp} = \frac{number\ of\ (Req\ or\ Res)\ messages\ containing\ item}{total\ number\ of\ (Req\ or\ Res) messages}$$

$$flow_{supp} = \frac{number\ of\ flows\ containing\ item}{total\ number\ of\ flows}$$

$$Server_{supp} = \frac{number\ of\ flow\_set\ containing\ item}{total\ number\ of\ server\_set}$$

Fig. 5 Three Support Units

If the target network protocol has no delimiters or is binary-based protocol, the methodology performs this step after pre-processing step. In this step, the methodology extracts field candidates by using the Apriori algorithm used three support. Figure 5 shows the three support units. First, message support checks how frequently the field candidates occur in each direction messages. Because it applies support in each direction messages, it is possible to extract frequently occurring string in each direction messages as field candidates. Second, flow support checks how frequently the field candidates occur in flows. By using flow support, the methodology can remove strings that occur frequently in only a few flows. Finally, server support checks how frequently the field candidates occur in server set. The server set is a set of elements that make up all of the flows that are connected between one serer and the clients that communicate with the server. By applying this support throughout the target network traffic between all clients that each server communicates with, it can eliminate the frequent strings that occur only in a few server set.

H. Position-Value Filter

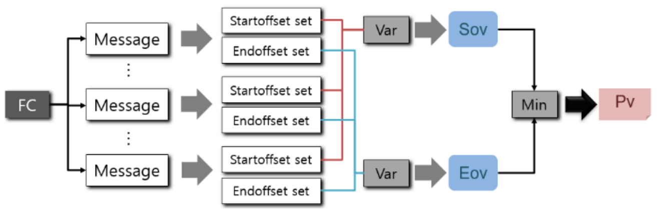


Fig. 6 Overview of Position-Value Filter

In this step, the methodology checks the location information of the field candidates extracted by prior step. Figure 6 shows the overview of Position-Value Filter. It eliminates field candidates whose location value is below a specific threshold. The small value of location information means that field candidates occur in a fixed position in the message. The field candidates have a startoffset set and endoffset set for each message it occurs. The startoffset set is a set of offsets calculated based on beginning of a message, and the endoffset set is a set of offsets calculated based on end of a message. The field candidates calculate the variance of the startoffset of all messages(Sov) and the variance of the endoffset of all messages(Eov). A Pv means the minimum value between Sov and Eov. If the Pv is very small, it means that the field candidate occurs in a fixed position within the messages. Therefore, the methodology extracts field candidates with Pv below the threshold into the final field.

IV. EXPERIMENT

In this step, we compare the results of the AutoReEngine and the proposed method for the HTTP protocol. Since the Netzob extract too many fields, we compare the AutoReEngine with the proposed method. Table 1 shows the traffic information.

TABLE 1 TRAFFIC INFORMATION

	Flow	Pkt	Message	Byte
FTP	3	62	50	12323
SMTP	6	259	224	21022
HTTP	359	3841	1189	4674813

A. Correct Answer Setting

Since there is currently no objective answer to evaluate protocol reverse engineering, we set the correct answer for the three protocols and performed the experiment.

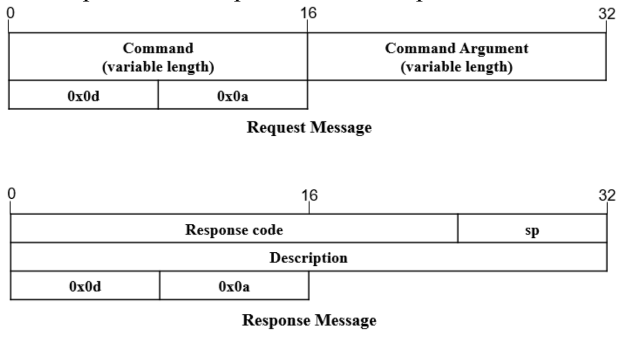


Fig. 7 Structure of the FTP

Figure 7 shows the structure of the FTP. For the FTP, we define the command and response code as the correct answer. This is because argument and description are only descriptions of commands or response codes.

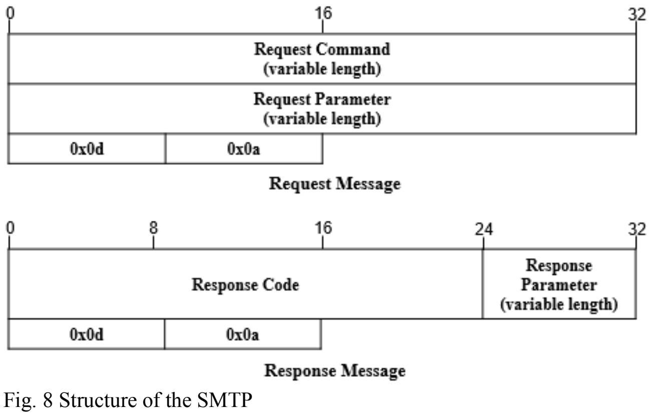

0
16
32
Request Command (variable length)
Request Parameter (variable length)
0x0d
0x0a
Request Message
0
8
16
24
32
Response Code
Response Parameter (variable length)
0x0d
0x0a
Response Message

Fig. 8 Structure of the SMTP

Figure 8 shows the structure of the SMTP. For the SMTP, we define the request command and the response code as the correct answer. This is because two parameters are only parameter of the request commands or the response codes.

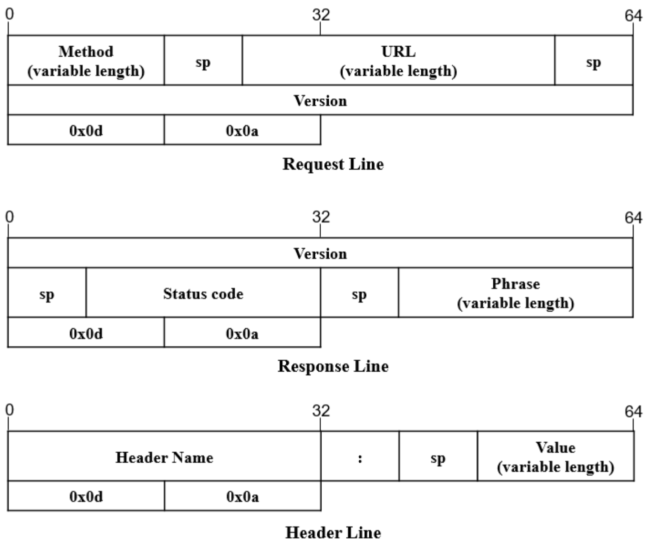


Fig. 9 Structure of the HTTP

Figure 9 shows the structure of the HTTP. For the HTTP, we define the method, version, status code, phrase, and header name as the correct answer. Because other fields are too variable or cannot represent the property of the HTTP, we exclude them from the correct answer.

B. Experiment Result

TABLE 2 EXPERIMENT RESULT

	Recall	Precision
Proposed Method(FTP)	100%	100%
Proposed Method(SMTP)	100%	100%
Propsed Method(HTTP)	100%	100%
AutoReEngine(FTP)	70%	41.17%
AutoReEngine(SMTP)	91.66%	34.78%
AutoReEngine(HTTP)	11.69%	24.53%

Table 2 shows the experiment result. While the proposed method extracted all the correct answers for three protocols, the AutoReEngine do not extract most of the correct answers for the HTTP. In addition, the AutoReEngine derived a lower performance than the proposed method for SMTP, FTP.

V. CONCLUSION

In this paper, we propose the methodology for separating fields by extracting delimiter of the target protocol. Since we cannot collect the private protocol traffic, we performed experiment on three text-based protocols that are currently used. As a result, we extracted all the key fields representing the properties of the target protocol and validated that the performance is superior to other methodologies. However, this methodology does not perform well for binary-based protocols. Therefore, we plan to develop an algorithm that can extract key fields for binary-based protocols in future work.

REFERENCES

[1] Young-Hoon Goo, Kyu-Seok Shim, Jee-Tae Park, Byeong-Min Chae, Ho-Won Moon, Myung-Sup Kim, "A Method of Protocol Reverse Engineering for Clear Protocol Specification Extraction", KNOM Review, Vol. 20, No. 2, pp. 11-23, Dec 2017

[2] Young-Hoon Goo, Baraka D. Sija, Sung-Ho Lee, Myung-Sup Kim, "Analyzing the Difference Between Network Trace-based and Execution Trace-based Protocol Reverse Engineering in Three Perspectives", Proceedings of Symposium of the Korean Institute of communications and Information Sciences, pp. 82-83, Jeju Island, Korea, June 2017.

[3] Jia-Zhen Luo, Shun-Zheng Yu "Position-based automatic reverse engineering of network protocols", Joural of Network and Computer Applications, Vol. 36, No. 3, Issue. 3, pp. 1070–1077 Feb.2013K. Elissa, "Title of paper if known," unpublished.

[4] Bossert, Georges, Frederic Guihery, and Guillaume Hiet, "Towards automated protocol reverse engineering using semantic information.", Proceedings of the 9th ACM symposium on Information, computer and communications security. ACM, pp. 51-62, Kyoto, Japan, June.2014

[5] Trifilo, A., Burschka, S., & Biersack, E. ,"Traffic to protocol reverse engineering", In Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on IEEE, pp. 1-8, Ottawa, Canada, Dec.2009

[6] Wang, Y., Zhang, Z., Yao, D. D., Qu, B., & Guo, L., "Inferring protocol state machine from network traces: a probabilistic approach", In International Conference on Applied Cryptography and Network Security, pp. 1-18,Nerja, Spain, June.2011

[7] Antunes, Joao, Nuno Neves, and Paulo Verissimo, "Reverse engineering of protocols from network traces.", Reverse Engineering (WCRE), 2011 18th Working Conference on. IEEE, pp. 169-178, Limerick, Ireland, Oct.2011

[8] Shevertalov, Maxim, and Spiros Mancoridis, "A reverse engineering tool for extracting protocols of networked applications", Reverse Engineering (WCRE), 2007 14th Working Conference on. IEEE, pp. 229-238, Vancouver, Canada, Oct.2007