A Study on the Method to Extract Clear Fields From the Private Protocol

1st Min-Seob Lee
dept. of Computer and Information Science
Korea University
Sejong, Korea
chenlima2@korea.ac.kr

2nd Kyu-Seok Shim
dept. of Computer and Information Science
Korea University
Sejong, Korea
kusuk007@korea.ac.kr

3rd Young-Hoon Goo
dept. of Computer and Information Science
Korea University
Sejong, Korea
gyh0808@korea.ac.kr

4th Myung-Sup Kim
dept. of Computer and Information Science
Korea University
Sejong, Korea
tmskim@korea.ac.kr

***Abstract*—The computer network environment has been growing steadily in recent years, and as a result there has been continuous high volume of network traffic. Several applications and malicious behavior are emerging as networks are being used in many areas. As a result, applications and malicious behavior using private protocols continue to grow, and most of their private protocols are unknown or not documented. Analyzing the structure of private protocols to respond to malicious behavior or to construct an efficient network environment is an essential study in modern society. The method by which protocols are analyzed is called Protocol Reverse Engineering, and its importance has already been proven to date. While various studies address Protocol Reverse Engineering, there is no standardized way to distinguish or extract the fields that make up the protocol. Therefore, this paper proposes and validates how fields are extracted from private protocols using the Apriori Algorithm, one of the sequential pattern mining techniques.**

Keywords—protocol reverse engineering, private protocol, syntax, field

I. Introduction

Computer network environments have been growing steadily in recent years, resulting in a steady stream of large network traffic, and the increasing number of applications and malicious behavior using the network. In most cases, protocols that occur in these network environments are private protocols that are unknown or not documented. In order to clearly understand the structure of a non-public protocol, the study of Protocol Reverse Engineering has been studied steadily from the past to the present. Protocol Reverse Engineering can be used in the field of network security, for example cyberattacks. A steady stream of large and small cyberattacks around the world continue to evolve in various forms, using private protocols. To respond to these cyberattacks, Protocol Reverse Engineering is essential. Protocol Reverse Engineering can also be used in network management. It can be used in many areas, as identifying the status of network use and adjusting bandwidth to specific protocols for efficient use of limited network resources.

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 by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea Government(MSIT) (No.2018-0-00539-001,Development of Blockchain Transaction Monitoring and Analysis Technology)

Various prior studies suggest different methodologies for Protocol Reverse Engineering. But there are some limitations. Traditional Protocol Reverse Engineering is time-consuming and error-prone because it is mostly done manually. To resolve this problem, automatic Protocol Reverse Engineering methodologies have been proposed, but limitations exist. Some of the suggested methodologies do not extract fields clearly, some of them simply separate the fields with known delimiters(space, tab, etc,..). The goal of Protocol Reverse Engineering is to extract the clear specifications of the target protocol, and it is also difficult to clearly extract the field, the smallest unit of a protocol, or to do so later. Therefore, this paper proposes a clear method for extracting fields from Protocol Reverse Engineering and conducts experiments on HTTP protocols. Following the introduction, the organization of this paper defines the relevant research and problem and details on the methodology proposed in Chapter 3. Chapter 4 concludes with analysis of experimental results, conclusions and future studies in Chapter 5.

II. Related Work

A. Protocol Reverse Engineering Components

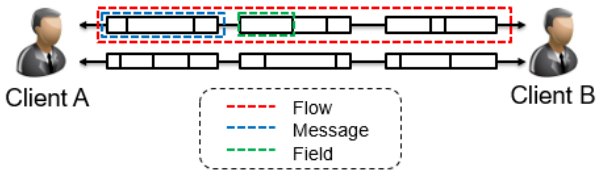

Figure 1. Protocol Components

Before describing Protocol Reverse Engineering, define the terms used when analyzing the structure of the protocol. First, flow is a set of packets used as inputs to Protocol Reverse Engineering. These packets are all of the same 5-tuple(Source IP Address, Destination IP Address, Source Port, Destination Port, L4 Protocol) packet and consist of a sequence of messages. Messages define one TCP segment as a message for TCP flow and one packet as a message for UDP flow. The message consists of a sequence of fields, which are the smallest units that have meaning when

analyzing a protocol's structure. For the HTTP protocol, such things as GET, User-Agent, HTTP/1.1 Host, are defined as fields.

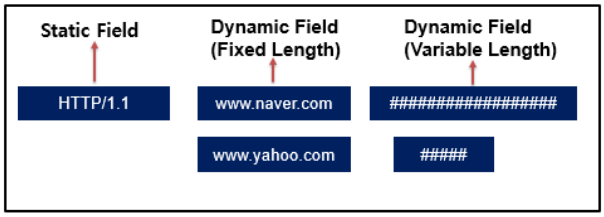

Figure 2. Type of field

There are three types of fields: Static Field, Dynamic Field with fixed length, and Dynamic Field with variable length. This paper clearly describes how to extract static fields with fixed values from a private protocol.

B. Protocol Reverse Engineering

Protocol Reverse Engineering is the study of identifying the explicit structure of a private protocol rather than the protocol in which the specifications are opened. It is usually aimed at deriving three components of unknown application layer protocols from OSI layer 7.[1] Three elements of a protocol are syntax, semantics, and timing. The goal of Protocol Reverse Engineering is to provide a clear understanding of what types of messages are in the target protocol, how they are organized and in what order they communicate.

Most traditional Protocol Reverse Engineering has been done passively by people. Passive Protocol Reverse Engineering can take a very long time because instead of being able to accurately identify all the components of a protocol, the results can vary depending on who analyzes and are performed manually. For this reason, it is not appropriate to analyze large amounts of private protocol traffic in the current network environment. In order to overcome these limitations, automated Protocol Reverse Engineering methodologies began to be designed.

Automatic Protocol Reverse Engineering is typically divided into analysis methods based on network traces and analysis based on execution traces. The biggest difference between the two methods[2] is the trace used for input. The Execution Trace based analysis uses the execution traces logged by monitoring the execution of program binaries that use the target protocol. This method is not suitable for analyzing private protocols realistically because it can only be used if program binaries implementing the target protocol are acquired. Network Trace based analysis uses the network traffic captured by the target protocol as an input. This method has the advantage of capturing traffic and performing analysis even if the program binaries implementing the target protocol are not accessible, so it is considered suitable for analyzing a private protocol. Therefore, this paper uses Network Trace based analysis methods.

C. Prior Study

A prior study using Network Trace based analysis methods included [3]AutoReEngine, [4]Netzob, [5]Trifilo, [6]Veritas, [7]ReverX, and [8]Pext. Among the preceding studies mentions above, methodologies for extracting fields from protocols are ReverX, Veritas, AutoReEngine, and Netzob. Out of these four methodologies, Netzob, which is open source, and AutoReEngine, which is the basis for the methodology proposed in this paper, are selected for testing.

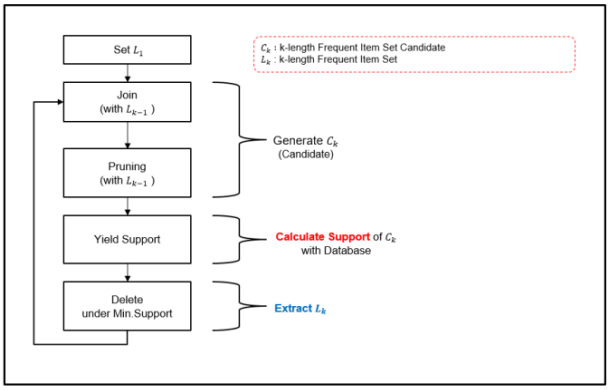

Figure 3. Apriori Algorithm

AutoReEngine receives protocol network traffic as an input. AutoReEngine consists of four main steps : Data Pre-Processing, Protocol Keyword Extraction, Message Format Extraction, and State Machine Inference. The key algorithm for AutoReEngine is the Apriori algorithm. The Apriori algorithm is an algorithm that extracts frequently occurring items as one of sequential pattern mining techniques. AutoReEngine uses this algorithm to extract commonly occurring strings in the target protocol. The values for the position variance of these strings are then obtained and the strings that occur at a fixed location are selected as the final field with the value of the position variance below the specified threshold.

Netzob is a top-down Protocol Reverse Engineering method. First, protocol network traffic is received as an input, reassembled in message units, and the Needleman-Wunch algorithms is used to conduct the sequence alignment and extract a Symbol as a result. Extract the Strings commonly seen in each Symbol into a static field(Data Variable) and the fields with different values into a dynamic field(Alternative Variable). Then use the UPGMA algorithm to measure the similarities between each Symbol and cluster Symbol with similarities beyond the user-defined degree of similarity.

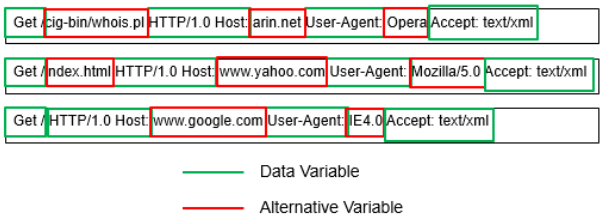

Figure 4. Field in Netzob

D. Define Problem

There are three main output of Protocol Reverse Engineering : Syntax, Semantics, and FSM. Syntax refers to the format of how messages are configured in the target protocol, while Semantics refers to the meaning of each field that constitutes a message type. FSM is a finite Automata that describes the order in which message types

communicate. This paper proposes a method for clearly extracting fields from Syntax during the output of Protocol Reverse Engineering.

In prior studies each field is extracted in a different way, with several limitations. First, when extracting fields, separate the fields by simply known delimiters. This method works well if the fields in the target protocol are separated by delimiters, but if not, the analysis is likely to be poor. For a private protocol, it is not appropriate to distinguish fields by delimiters because you do not know how they are organized. Second, we extract frequently occurring strings form the target protocol into the field. Since location information on where a field occurs in a message is not used but relies on statistical methods to extract fields, simply the frequently occurring strings are extracted into the field. A string that occurs only once when connecting and disconnecting between the server and the client may need to be extracted into a field, with limitations that cannot be extracted in this way. Therefore, this study proposes to extract fields that occur at a fixed location frequently occurring in the target protocol using the appropriate location and statistical information of the field.

III. The Proposed Method

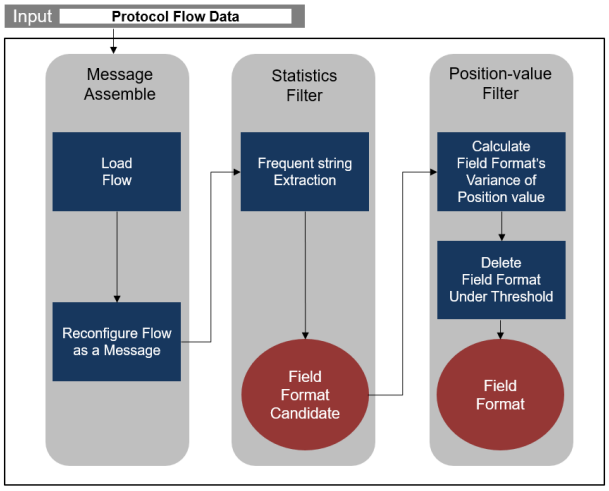

Figure 5. The Overview of Proposed Method

In the Message Assemble phase, data pre-processing is performed by receiving the target protocol network traffic as an input.

In the Statistics Filter phase, three support units are applied to the Apriori algorithm to extract commonly occurring strings. Although the basic Apriori algorithm uses one support unit, the proposed methodology uses three support units to eliminate as much noise as possible. This phase extract field candidates, which are frequently generated in all three units.

In the Position-Value Filter phase, only field candidates that occur at a fixed location using the location information of the field candidates extracted from the previous step are filtered. Only field candidates who meet the threshold specified by the user are selected as final field.

A. Statistics Filter

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

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

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

Figure 6. Three Support Unit

At the Statistics Filter step, the Apriori algorithm (Fig.3) applies three supports(Fig.6) that perform different roles to extract frequently occurring strings (field candidates).

First of all, Message Support works to separate the direction (Request, Response) of a message to examine how often field candidates occurs in a Request Message or Response Message. Because the smallest unit of support to which it can be applied is applied in more detail, support plays a key role in extracting fields from this methodology.

Flow Support examines the frequency of field candidates in the overall flow. With Flow Support, you can remove strings that occur frequently in only a few flows, but only one time in the flow, but are not extracted as fields (ex. Login Keyword). Therefore, in this methodology, the strings that appear in each flow are exceptionally extracted into fields.

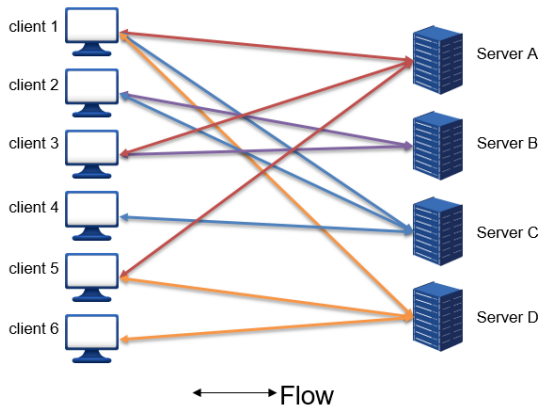

Figure 7. Flow Set

Flow_set is a set of elements that make up all of the flows that are connected between one server and the clients that communicate with that server. Flow_set Support as opposed to Message Support, is the largest unit to which support can be applied. By applying Support throughout the traffic between all clients that each server communicates with, it can filter the frequent strings that occur only with a few servers.

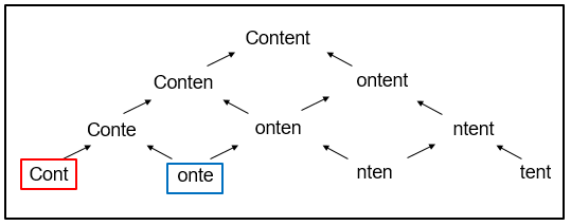

Figure 8. Delete Inclusion Relation

In this methodology, unlike the typical Apriori algorithm, we have added a portion of the inclusion relationship at each level. For example, suppose an item with a length of 4 has a keyword "Cont" and a keyword with "onte". Fig 8 shows a combination of these two keywords to create a 5-length keyword with "Conte". If all three keywords satisfy all three support units, all three keywords will appear as filed candidates. However, the keyword "Cont" does not have to be extracted as a field because it is unconditionally extended only to the keyword "Conte". Thus, this methodology eliminates field candidates that do not need to be extracted into the field by means of an inter-relational algorithm. For example, "Cont", as mentioned above, must be removed because it only extends to the keyword "Conte" as described above.

However, the keyword "onte" can be extended to "Conte" or "onten". Therefore, remove "onte" if the combined number of messages with "onte" and all 5-length keywords with which "onte" can be expanded are the same. In such a case, the "onte" should be deleted because the number of cases where the "onte" keyword can be extended is considered. Fig9 shows that pseudocode of this process.

```
Delete Inclusion Relationship

item1 = length k-1 item
item2 = length k-1 item
item3 = length k-1 item
Lk_item1 = length k item that Combine item1 with item2
Lk_item2 = length k item that Combine item2 with item3

if(# of message that appear item2 ==
   # of message that appear Lk_item1 + # of message that appear Lk_item2)
    Delete item2;
```

Figure 9. Delete Inclusion Relationship Algorithm

In the Statistics Filter step, the first step uses all characters as inputs to the Apriori algorithm to extract a set of frequently k-length strings (field candidates). And when there are two field candidates with a fully inclusive relationship among field candidates, a short candidate field is removed.

B. Position-Value Filter

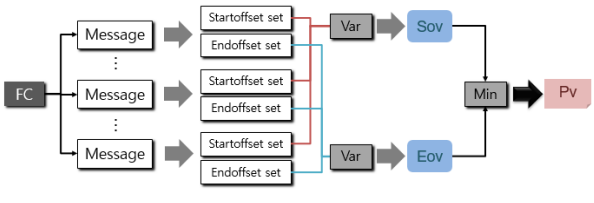

Figure 10. Detailed Structure of Position-Value Filter

In the Position-Value Filter Step, check the position information of the field candidates extracted through the Statistics Filter step. Filtering is performed based on how the calculated position information occurs below a certain threshold, that is, at a fixed position. The field candidate will appear in multiple messages in the traffic data received as shown in Figure 10. At this time, field candidates have a Startoffset set, an Endoffset set, for each message they appear. The Startoffset set is the set of position values calculated from the beginning of the message, and the Endoffset set is the set of position values calculated form the end of the messages. Field candidates calculate the variance of all of the Startoffset and Endoffset of all messages they appear. Sov and Eov, which are shown in Figure 10, are each. Define the minimum value as pv between Sov and Eov. Pv is an indicator of how certain field candidates are appearing in messages. If Pv is very small, it means that the field candidate will appear in a fixed position within the message. Therefore, it is determined that any field candidate with Pv having a value below the threshold entered will appear in a fixed location and will be selected as the final field.

IV. Experiment

A. Evaluation Metric

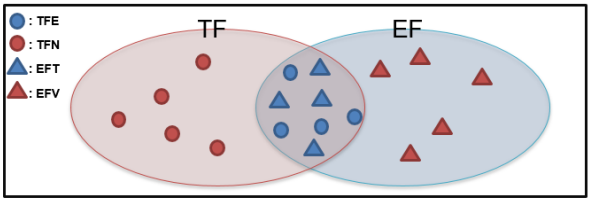

Figure 11. Evaluation Metric

In this chapter, two evaluation metrics are defined to give a visible presentation of experimental results for the HTTP protocol. The correct answer sheets that need to be extracted as fields for HTTP protocol only are defined by the Method, Version, Header Name, Status Code, and Phrase and the evaluation metric are obtained.

True Field means one field of each correct answer to be extracted with the above mentioned fields. For example, individual values such as "HTTP/1.1", "GET", "User-Agent" are called True Field.

True Field Format(TF) is a result of clustering of the same True Fields as one. Cluster all True Fields with "HTTP/1.1" values and define them as one field format. True Field Format is classified into two elements. The first element is TFE , which is included in EF. The second element is TFN, Which is not included in EF.

Extracted Field Format(EF) means a field format extracted from Automatic Protocol Reverse Engineering methodology. Since different methodologies produce different results, they have the greatest impact on the generation of evaluation metrics. Extracted Field Format is classified into two elements. The first element is EFT, which includes the True Field Format as defined above. The second element is EFV, which means values for the True Field Format are not included.

Conciseness is an evaluation metric of the EF from each methodology. Indicates the percentage of field extracted from each methodology that contain the correct answer and is a key evaluation metric for assessing the performance of each methodology.

Correctness is an evaluation metric for True Field extracted from each methodology. Indicates the percentage of True Field included in the EF extracted from any of the True Fields in the HTTP protocol traffic used as input. That is the frequency with EFT out of the total number of True Fields. Even though the Conciseness is very high, it is hard to say that a low Correctness is a good performance. Ideally, both of these evaluation metrics will have high values.

B. Experiment Result

In this chapter, the experimental results of fields extracted with the same HTTP protocol traffic as inputs for each methodology are compared. Before comparing the results of the experiments, Netzob is excluded from the results of the experiments.

Netzob's Field
HTTP/1.1|20|200|20|OK|0d||0a|Server:|20|nginx|0d||0a|Date:|20|Tue,|20|10|20|Apr|20|2018|20|07:01:11|20|GMT|0d||0a|Content-Type:|20|image/gif|0d||0a|Content-Length:|20|
|20|GMT|0d||0a|Connection:|20|keep-alive|0d||0a|Keep-Alive:|20|timeout=5|0d||0a|ETag:|20||22|55502c9
|22||0d||0a|Expires:|20|Thu,|20|10|20|May|20|2018|20|07:01:11|20|GMT|0d||0a|Cache-Control:|20|max-age=2592000|0d||0a|Accept-Ranges:|20|bytes|0d||0a||0d||0a|GIF89a|0b||00|

Figure 12. Netzob's Field

This is because Netzob has too many True Fields and is too long. The field should be the smallest unit to have meaning in Protocol Reverse Engineering, and it is difficult to define is as the field in this paper because it contains at least three True Fields. Therefore, this chapter compares the results of an experiment with AutoReEngine, which was developed by the authors of this paper, and with the suggested methodology.

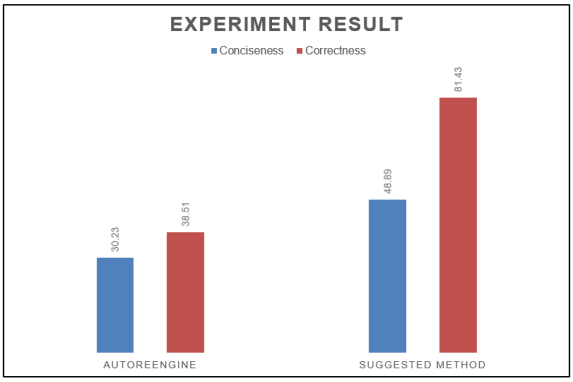

Figure 13. Experiment Result

AutoReEngine	Suggested Method
e|0d||0a|User-Agent: Mozilla/5.0 (Windows NT 10.0|3b| W	|0d||0a|Content-Type:
e|0d||0a|User-Agent: Mozilla/5.0 (Windows NT 10.0|3b| Win64|3b| x640.518	GET /
GET /	8|0d||0a|Referer: http://
8|0d||0a|Referer: http	36|0d||0a|Accept:
8|0d||0a|Referer: http://	Safari/537.36|0d||0a|Accept:
6|0d||0a|Accept:	HTTP/1.1|0d||0a|Host:
36|0d||0a|Accept:	HTTP/1.1 200 OK|0d||0a|
Safari/537.36|0d||0a|Accept:	|0d||0a|Referer: http://
HTTP/1.1|0d||0a|	|0d||0a|Host:
HTTP/1.1|0d||0a|Host:	|0d||0a|Cookie:
HTTP/1.1	|0d||0a|Expires:
HTTP/1.1 20	|0d||0a|Accept:
HTTP/1.1 200 OK|0d||0a|	|0d||0a|Accept-Language: ko-KR
	|0d||0a|Server:
	|0d||0a|Date: Tue, 10 Apr 2018 07:0
	|0d||0a|Content-Length:
	|0d||0a|Connection:
	|0d||0a|Cache-Control:

Figure 14. Detail Experiment Result

For AutoReEngine, the fields are judged to be better extracted than Netzob. One or two True Fields are properly configured. However, because AutoReEngine does not have a module to remove the inclusion relationship in the extraction process, the threshold is that all noise fields, which are completely included in a particular field, are extracted. In addition, the fields that need to be extracted are not extracted, and these problems are judged to be that the message, the smallest unit to which support can be applied, is not considered. As with AutoReEngine, for the methodology proposed in this paper, a single field is properly constructed into one or two True Fields. Unlike AutoReEngine, we improved performance by removing noise fields through the inclusion relationship removal module. It is also believed that more specific fields were extracted by extracting fields that frequently appear in all three support units.

V. Conclusion

In this paper, we proposed and verified clearly how fields are extracted from Protocol Reverse Engineering. The proposed method explicitly extracts fields from entered network traffic, taking advantage of all possible statistical and positional information of the field. But there is still more to be done. Due to nature of the Apriori algorithm, this methodology depends on the value of Minimum Support. Therefore, in future studies, we plan to conduct a study to find the optimum Minimum Support value to maximize the two evaluation metrics we presented.

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”, Jounal 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