Automatic Reverse Engineering Method for Extracting Well-trimmed Protocol Specification

Young-Hoon Goo
Dept. of Computer and Information Science
Korea University, Sejong, Korea
+82-44-860-1378
gyh0808@korea.ac.kr

Kyu-Seok Shim
Dept. of Computer and Information Science
Korea University, Sejong, Korea
+82-44-860-1378
kusuk007@korea.ac.kr

Myung-Sup Kim
Dept. of Computer and Information Science
Korea University, Sejong, Korea
+82-44-860-1347
tmskim@korea.ac.kr

ABSTRACT
Emergence of high-speed Internet and ubiquitous environment has led to a rapid increase of applications and malicious behaviors with various functions. Many of the complex and diverse protocols that occur under these situations, are unknown protocols that are at least documented. For efficient network management and network security, protocol reverse engineering that extract the specification of the protocols is very important. While various protocol reverse engineering methods are being studied, each of methods has some limitations. In this paper, we propose the reverse engineering method for extracting well-trimmed protocol specification. The proposed method can extract intuitive field formats, message formats with semantics, flow formats, and protocol state machine of the unknown protocol. We implement our approach in a prototype system and demonstrate the validity of our approach through experimenting it over HTTP protocol.

Keywords
Protocol reverse engineering; Field Format; Message Format; Flow Format; Finite state machine; CSP Algorithm;

1. INTRODUCTION
Today's emergence of high-speed Internet has led to not only generation of massive traffic but also rapid increase of developed applications and malicious behaviors in various functions. Many of the complex and diverse protocols that occur under these situations, are unknown or proprietary protocols that are less documented. Some of these protocols include the proprietary protocols, and protocols that are used in various kind of attacks. For efficient network management and network security, protocol reverse engineering that extract the specification of the protocols is very important.

While various protocol reverse engineering methods are being studied, each of methods has some advantages and some limitations. Some prior methods are mostly manual, therefore time-consuming and error-prone. Although many of the automatic protocol reverse engineering methods are proposed to address this problem, these methods do not extract intuitive message formats. Many of previous works extract too specific message formats or too general message formats. Extracting too specific message formats means that it cannot extract intuitive protocol specifications by extracting too many message formats. Extracting too general message formats means that it cannot extract all the possible values of the field which belongs to the message format, or extracts message formats whose fields are sufficiently subdivided. Another limitation of prior methods is that it can extracts only part of the protocol specification. Some methods only extract protocol syntax, protocol semantics or protocol state machine. However, the ideal protocol reverse engineering is to extract all of these, protocol syntax, semantics, and state machine.

In this paper, we proposed the reverse engineering method for extracting well-trimmed protocol specification which can extract the protocol intuitive syntax, semantics, and finite state machine. The novelty of the protocol syntax extracted by the proposed method is that it consists of the field formats, the message formats, and the flow formats. The proposed method uses the modified sequential pattern algorithm hierarchically to extract these formats. Additionally, it uses the modified sequential pattern algorithm recursively to extract the all the values the fields format can have. The main advantage of the proposed method is that the extracted message format is a fully separated message format with fields of four types without any blank parts.

The rest of this paper is organized as follows. Section II describes the related works and the scope of the problem. Section III describes the proposed method in detail, and section IV describes experiment. Finally, Section V presents conclusive remarks and a brief look for the future research directions.

2. RELATED WORKS & PROBLEM SCOPE
Protocol reverse engineering is the process of extracting specifications of unknown protocols which is deriving specifications of application layer protocols in general. The goal of protocol reverse engineering is to extract detailed structures including syntax, semantics, FSM, and etc. related to the three main components of the protocol. Extracted specification can indicate message types the protocol has, the format in which these types of messages are organized, and the order in which these operate.

The traditional approach of protocol reverse engineering methods are mostly manual. A typical example is the Generic Application-level Protocol Analyzer (GAPA) released by Borisov et al [1]. GAPA is a framework for verifying and parsing network protocol specifications through hand-written syntax. The manual protocol reverse engineering is very tedious, time-consuming and error-prone. In high-speed network environment like today, automatic protocol reverse engineering rather than manual reverse engineering is valid to cope with the speed of emergence of new applications and various highly intelligent attacks.

Automatic protocol reverse engineering can be divided into two categories: execution trace based protocol reverse engineering and network trace based protocol reverse engineering.

Execution trace based protocol reverse engineering is the methods of analyzing execution traces logged how the program binary that

implements the protocol processes messages by using dynamic taint analysis. Since these methods analyze the execution of actual program binaries, the accuracy of the format extraction can be improved, but it is practically difficult to obtain the program binary of the unknown protocol. Besides, in general, only the received messages of the host receiving the protocol message are analyzed.

On the other hand, network trace based protocol reverse engineering methods analyze network traces captured by monitoring network packets of the protocol. Therefore, there is no need to access the program binaries, so much more convenient, and it can analyze not only received messages but also send messages by monitoring the routers connecting the external and target network. Hence, we focus on network trace based protocol reverse engineering because of practicability and convenience.

The output of protocol reverse engineering is largely composed of syntax, semantics, and protocol state machines. Ideally, for extracting a detailed protocol specification, all possible information should be inferred, including protocol syntax, semantics, and protocol state machine, and the syntax is to be clear. However, previous methods have some limitations.

First, many of previous methods only extract some part of syntax, semantics and protocol state machine [2].

Second, many of these methods extract too specific syntax. Too specific syntax is to extract too many message formats, which means it is not intuitive message format of the protocol. Therefore, that syntax can be efficient for analyzing each packet, but it is not effective in understanding intuitive structure of protocol. To address this problem, we define three levels of formats which are field format, message format, and flow format and extract them using hierarchically modified sequential pattern algorithm. We call this algorithm Hierarchical Contiguous Sequential Pattern (Hierarchical CSP) algorithm. The propose method extract not only sufficient summarized message formats with semantics but also flow formats. The meaning of each format is described in section III.

Third, many of these methods extract too general syntax. These methods are mostly use the way based on the frequency, such as entropy filter, LDA with the appearance probability. Therefore, these methods can extract only one value that is the most frequent, rather than extracting all the possible values for each fields. Another case is to extract syntax that is not sufficiently subdivided into the fields that make up the message formats. Figure 1 shows this problem by exemplifying the HTTP protocol.

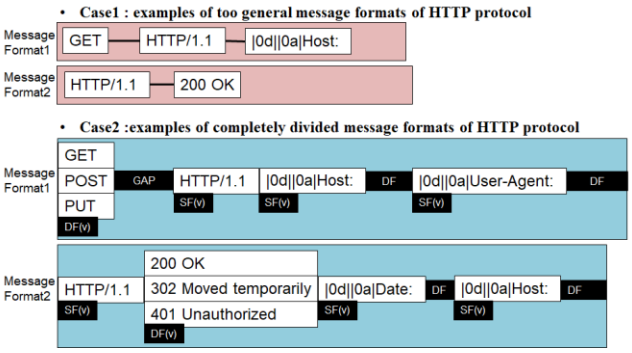

Figure 1. The example of a too general syntax as output and clear syntax as output

To address this problem, the proposed method extracts all the possible values for each field using Recursive CSP algorithm. Besides, since the method has a module for dividing the additional fields that are not extracted by the field format extraction module into four types of fields, it is possible to extract a completely divided message format without an empty portion.

3. THE PROPOSED METHOD

3.1 Terminology and the Overview of the Proposed Methodology

The proposed methodology extracts field formats, message formats, and flow formats step by step using Hierarchical CSP algorithm.

The field format consists of four types of SF(v), DF(v), DF, and GAP. "(v)" means "value", and "(v)" marked field format is the field in which values can be predicted. SF(v) denotes a field having a static value and a fixed length. DF(v), DF and GAP are fields whose values are dynamic. These dynamic fields may be fixed in length or may be variable. DF(v) is a dynamic field whose value is predictable. DF and GAP are fields whose values are too extremely dynamic to predict their values. The difference between DF and GAP is that DF is a field whose length can be predicted to some extent, but GAP is a field cannot be predicted because its length is too variable.

A SF(v) is the single contiguous substring in messages. A message format is the contiguous sequence of these field formats that appear in the same message. The message formats also includes the semantics of the fields. A flow format is a contiguous sequence of these message formats that appear in the same flow. Flow formats can help to understand the typical flow types of protocols and can be used for minimization of protocol state machines.

Figure 2 shows the overview of the proposed method for extracting well-trimmed specification. The method largely consists of four phases which are Message Assemble, Syntax Inference, Semantics Inference, and the Behavior Inference. The preprocessing phase is to collect network traces of unknown protocol in flow units which are sets of bidirectional packets with the same 5-tuple (source address, destination address, source port, destination port, L4 protocol).

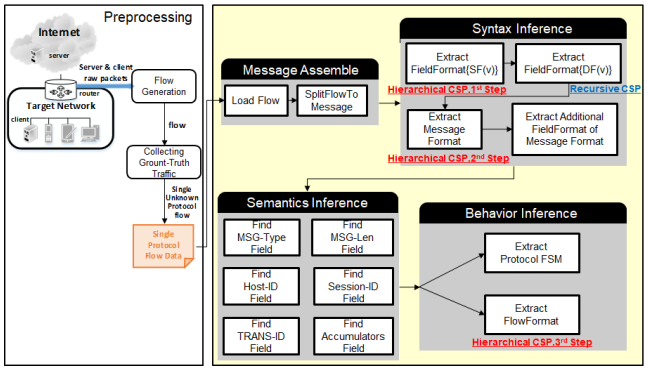

Figure 2. The overview of proposed method

In Message Assemble phase, flows of the unknown protocol is loaded first, and then the system splits flows to message units.

In Syntax Inference phase, the system extracts SF(v) using 1st step of Hierarchical CSP algorithm, and extracts all possible values for some extracted SF(v) to find DF(v) using Recursive

CSP algorithm. Then, it extracts the message formats with SF(v) and DF(v) using 2nd step of Hierarchical CSP algorithm. After extracting the message formats, the system extracts additional field formats for the blank parts of the message formats.

In Semantics Inference phase, the system find the meaning of the field formats that make up the message formats.

In Behavior Inference phase, the system extracts protocol state machine, and flow formats. The protocol state machine is extracted based on observation of input traffic data by using the extracted message format. The flow formats extracted using the 3rd step of Hierarchical CSP algorithm.

3.2 Contiguous Sequential Pattern Algorithm

For protocol reverse engineering, we develop the Contiguous Sequential Pattern (CSP) algorithm. It is a modified sequential pattern algorithm suitable for extracting protocol syntax. The original version of sequential pattern algorithm targeted purchase history data of a market to find sequential purchase patterns. However, the protocol syntax is not just a time-series subsequences of a message, but a contiguous subsequence of a message. For example, to obtain value of field format, we must extract not a sequence of discrete bytes, but byte-stream that is contiguous. Therefore, the objective of CSP algorithm is to extract contiguous sequential pattern for protocol syntax. This algorithm is based on the Apriori property that any subsequence of a frequently occurring sequence is also frequent. This method generates candidate subsequences and checks the support value of each candidate to determine frequently occurring subsequences. In addition, this algorithm improves performance by integrating modified algorithms such as AprioriAll, AprioriTID, AprioriHash, etc.

When extracting the 3 types of formats mentioned above, the system perform the Hierarchical CSP consisting of three steps and the CSP algorithm applied is exactly same and only the input transactions(sequences), length-1 item unit constituting the sequence, and support units are different. Figure 3 shows intuitively the process of Hierarchical CSP.

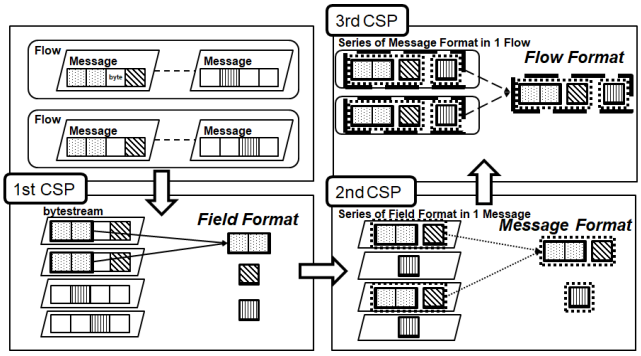

Figure 3. The process of Hierarchical CSP

The 1st step of Hierarchical CSP extracts common subbytestreams that satisfy certain frequencies as SF(v). These are contiguous characters, hex values or combination of them, in a set of message sequences. The 2nd step of Hierarchical CSP extracts a contiguous series of SF(v) which satisfy certain frequencies appearing in the same message sequences. These are skeletons of message formats. The extracted message formats are completed through the "Extract Additional Field Format of Message Format" module and the Semantics Inference phase as shown in Figure 2. The 3rd step of Hierarchical CSP extracts a contiguous series of message formats which satisfy certain frequencies appearing in the same flow. These are flow formats. Figure 4 shows the pseudo algorithm for CSP.

```
Input : SequenceSet, Min_Supp
Output : SubSequenceSet
01: foreach sequence S in SequenceSet do
02:   foreach item i in sequence do
03:     L1 ← L1 ∪ i;
04:   end
05: end
06: k ← 2;
07: while Lk-1 ≠ ϕ do
08:   foreach candidate c in Lk-1 do
09:     supp ← calSupport(c, SequenceSet);
10:     if(supp < Min_Supp) then
11:       Lk-1 ← Lk-1 − c;
12:     end
13:   end
14:   Lk-1 ← extractCandidate(Lk-1);
15:   k++;
16: end
17: SubSequenceSet ← ∪k Lk;
18: deleteSubset(SubSequenceSet);
19: return SubSequenceSet;
```

Figure 4. The pseudo algorithm of CSP

3.3 Message Assemble

As described above, in Message Assemble phase, flows of the protocol is loaded, and then each flow is split to messages.

We defined a methodology for assembling packets of each flow into message units: for messages transported over UDP it is assumed that each one packet is one message; for messages transported over TCP it is assumed that each consecutive set of packets with the same direction is one message.

3.4 Syntax Inference

The objective of Syntax Inference phase is to extract intuitive message formats that all the fields of message format are fully categorized into SF(v), DF(v), DF, and GAP as shown in Figure 1 and 2.

In the first module, "Extract Field Format{SF(v)}", the system extracts SF(v) using 1st step of Hierarchical CSP.

In the second module, "Extract Field Format{DF(v)}", the system selects SF(v) that is likely to be converted to DF(v) from among the all extracted SF(v) by checking the condition. This condition is that the support value is not 100% and the position variance is low enough (we use 200). The system then performs Recursive CSP for each selected SF(v).

The system performs the following procedure on all SF(v) satisfying the condition that can be DF(v) mentioned above. First, the system create a database that does not contain the SF(v) from the original database. The system truncates these message sequences which not contain the SF(v) based on the minimum offset and maximum depth of the SF(v). Next, it perform CSP from these message sequences. The system stores the value which has the highest support value among the output of the CSP in the value array of the SF(v). The above process is repeated until no more new values are extracted. Then, this SF(v) has a set of

values, so it is converted to DF(v). Therefore, This DF(v) has all the possible values. Figure 5 shows the process of Recursive CSP by exemplifying the method field and status code field of the HTTP protocol.

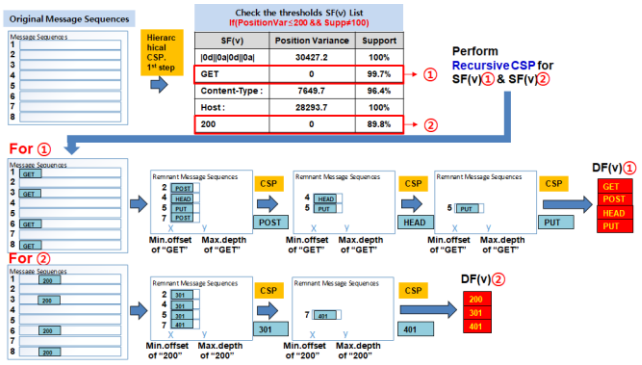

Figure 5. The process of Recursive CSP by exemplifying the HTTP protocol

In the third module, "Extract Message Format", the system extracts message formats using 2nd step of Hierarchical CSP under the condition that length-1 items are extracted SF(v) and DF(v).

In the fourth module, "Extract Additional FieldFormat of MessageFormat", it classifies all the part between the SF(v) and the DF(v) constituting each message format as new SF(v), DF(v), DF or GAP. In each extracted message format, the method to classify the part between the preceding field format and the following field format to SF(v), DF(v), DF, or GAP is as follows. Two thresholds are used in this module. The first is the 'variance of length of the part between the preceding field format and the following field format' and the second is the 'maximum length of the part'. Firstly the system only collects a set of message sequences from the original database (original message sequences) belonging to the message format to analyze. The system find the dataset corresponding to the part to be classified in the collected a set of message sequences. The system calculates the maximum length and variance of the lengths of the found dataset. If the variance is greater than the first threshold (we use 5000), it is classified as GAP, otherwise it is classified as not-GAP field. The GAP field means that length and value are very variable. Next, if the maximum length of the not-GAP field is less than the second threshold (we use 25), it is classified as DF (v), otherwise it is classified as DF. DF means that the value is extremely dynamic, but the length is somewhat fixed. If the part is classified as DF(v), the system stores all the values which are in the part. Next, If the part has only one value, it is classified as SF(v). Do this for all the part between the initial field formats of the message format. Finally, do above procedure for all message formats.

3.5 Semantics Inference

In Semantics Inference phase, the system finds fields corresponding to 6 predefined types of semantics through algorithms of each semantics type. We borrow FieldHunter [3]'s methodology to find the semantics of the fields that make up the message formats. This is because methodology of FieldHunter extracts the most specific kind of semantics among many semantics extraction methodologies.

The 6 predefined semantics types are MSG-Type, MSG-Len, Host-ID, Session-ID, Trans-ID, and Accumulators. To infer this, as in the fourth module of Syntax Inference, the system collects only the data corresponding to a specific field format in a specific message format in the whole message sequences, and determines whether each of the 6 semantics types corresponds. Thus, one field format can have multiple semantics. This process is performed for all DF(v) in all message formats.

3.5.1 MSG-Type

The field corresponding to MSG-Type is dynamic field whose value is neither too random nor constant, and this field has opposite fields to match. In other words there is a causal relationship likewise request/response. The algorithm uses entropy metric: $H(x) = -\sum p_i \log 2 p_i$ and causality metric: $I(q;r) / H(q)$ to find this field. $I(q;r)$ is mutual information which represents $H(q)+H(r)-H(q,r)$ of information theory. q means the value of the field, r means the value of opposite field which has opposite direction.

3.5.2 MSG-Len

The field corresponding to MSG-Len is dynamic field whose value means length of the message. The algorithm uses Pearson correlation coefficient to verify that the values of the fields are in a linear relationship.

3.5.3 Host-ID

The field corresponding to Host-ID is dynamic field whose value is specific to source address, such as email address, user id, and host IP address. The algorithm uses categorical metric: $R(x,y) = I(x;y)/H(x,y)$ to find this field.

3.5.4 Session-ID

The field corresponding to Session-ID is dynamic field whose value is specific to session. The algorithm uses categorical metric like the algorithm of finding Host-ID.

3.5.5 TRANS-ID

The field corresponding to TRANS-ID is dynamic field whose value is specific to transaction which is pair of values of request and response. The algorithm uses H(x) metric and verify that the value of the field is same with the value of the opposite field to find this field.

3.5.6 Accumulators

The field corresponding to Accumulator is dynamic field whose value constantly increases over time. The algorithm verifies if there is a constant increment to find this field.

3.6 Behavior Inference

In Behavior Inference phase, the system extracts protocol state machine and flow formats.

3.6.1 Extract Protocol FSM

In the proposed method, a state (node) of the finite state machine is the single extracted message format, which means a set of messages with same type. This module extracts the transitions between states by matching the extracted message formats which are states to the input traffic to extract the protocol state machine. In this process, record the number of matches for each transition to calculate the transition probability of each state. It is very useful for packet replay because it helps predict what message types will occur. The extracted finite state machine can help to confirm in what order the protocol message types operate. Each path connected from the Start state to the End state means each flow type.

3.6.2 *Extract Flow Format*

This module extracts flow formats using 3rd step of Hierarchical CSP under the condition that length-1 items are extracted message formats. The extracted flow formats represents the main flow types of the protocol, hence can help to understand the specification of the protocol. In addition, these flow formats can be used to generalize and minimize the protocol FSM.

4. Experiment and Result

In this section we evaluate the efficacy of our approach in inferring protocol specification of known protocol. We implement our approach in a prototype system in C++ code on Linux. The system takes a network capture file either in the libpcap or Netmon format as input. The system extracted 4 xml files which describes information of field format, message format, flow formats, and finite state machine and 1 png file which shows finite state machine.

Data was 4 traffic traces of HTTP protocol which collected from 4 different host. Table 2 shows quantitative information of data and summary of experimental result.

Table 1. Summary of experimental results

Traffic Info.	Flow : 359, packet : 3841, byte : 4.67MB Message Assemble result - messages : 1189(req. : 598, res : 591)
Format info.	Field Format : 49(req. : 26, res. : 23)
	Message Format : 31(req. : 17, res. : 14) - Coverage : 99.92%(1188/1189) - Correctness : 98.44%(127/129)
	Flow Format : 3
FSM info.	State : 15, transition : 54
Time	11.65s

The precondition for correct semantics and finite state machine is that the protocol syntax is correct. Therefore, the main performance evaluation of protocol reverse engineering is whether the protocol syntax represents the correct message format. To evaluate the message formats, we use two metrics which are correctness and coverage. Correctness means how many of the true formats can be analyzed by the extracted message formats. It is calculated as the number of true message format which are matched with the extracted message formats / the number of whole true message format. Coverage means how many of messages can be analyzed by the extracted message formats. It is calculated as the number of messages which are matched with extracted message formats / the number of whole messages.

Figure 6 shows summary of structure of the extracted message formats. The system compressed 1189 messages into 31 message formats, and each message format was appropriately subdivided into field formats without any empty part.

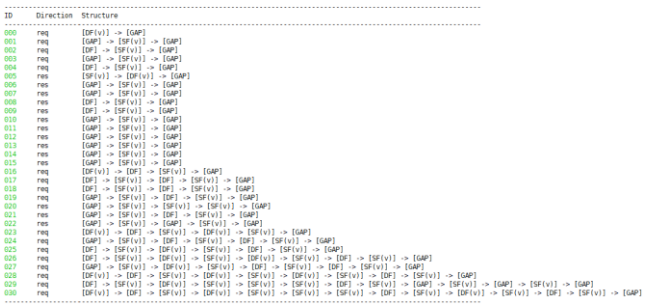

Figure 5. The Summary of Structure of the Message Formats

Figure 7 shows Message Format ID 30 as a sample, one of the message formats in Figure 6. This sample represents the HTTP request message format which is subdivided into 14 fields. The fields are in the order of Method [DF(v)] - URL [DF] - Version and Host [SF(v)] - value of Host field [DF(v)] - ... - GAP. It reflects the mandatory components: Method field, URL field, Version field and some optional HeaderName field of the HTTP request message format in a non-GAP format. It provides information about minimum offset, maximum depth, minimum length, maximum length, average length, and semantics of all field formats which make up the message formats. This information indicates whether the field format has a fixed position, fixed length, static value, and etc.

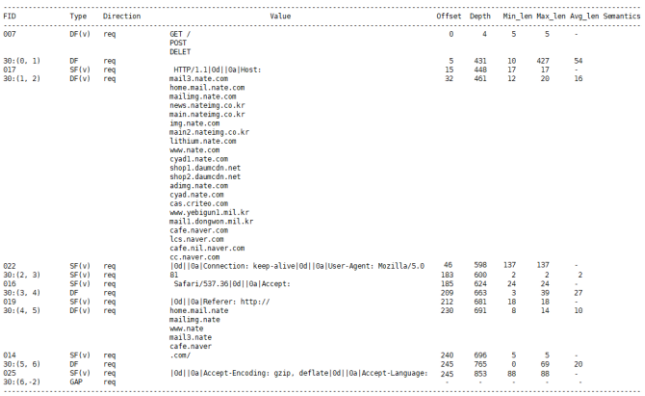

Figure 6. Sample of One of Extracted Message Format

5. Conclusion and Future work

In this paper, we propose a reverse engineering method for well-trimmed protocol specification. We have defined three types of formats which are field format, message format, and flow format to acquire a clear protocol specification and proposed a hierarchical CSP and a recursive CSP to extract such formats. The novelty of this method is that it extracts sufficiently fine-grained message formats and extracts a sufficiently compressed message types for input messages so that we can know the intuitive structure of the unknown protocol. As the future work, we plan to improve the method to be able to extract specification of protocol of all layer of OSI 7 layers.

6. ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea Government(MSIT) (No.2018-0-00539-001) and by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00513)

7. REFERENCES

[1] Nikita Borisov, David J. Brumley, Helen J. Wang, and Chuanxiong Guo. 2007. Generic application-level protocol analyzer and its language. *In Network and Distributed System Security Symposium*.

[2] John Narayan, Sandeep K. Shukla, T. Charles Clancy, A Survey of Automatic Protocol Reverse Engineering Tools, *Journal ACM Computing Survey, Vol. 48, Issue. 3, No. 40*, 2016.

[3] I. Bernudez, A. Tongaonkar, M. Iliofotou, M. Mellia, M. Munafo. Automatic Protocol Field Inference for Deeper Protocol Understanding, *IFIP Networking Conference*, 2015.