Framework for Precise Protocol Reverse Engineering Based on Network Traces

Young-Hoon Goo[1], Kyu-Seok Shim[1], Byeong-Min Chae[2] and Myung-Sup Kim[1]
Dept. of Computer and Information Science, Korea University, Sejong, Korea[1]
Hanwha Systems, Korea[2]
{gyh0808, kusuk007, tmskim}@korea.ac.kr[1], byeongmin.chae@hanwha.com[2]

***Abstract*— Emergence of high-speed Internet and ubiquitous environment is generating massive traffic, and it 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 or proprietary 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 have been studied, there is no single standardized method to extract protocol specification completely yet, and each of methods has some limitations. In this paper, we propose the framework for precise protocol reverse engineering based on network traces. The proposed framework can extract highly elaborative and intuitive message formats, flow formats, and protocol state machine of the unknown protocol. We demonstrate the validity of our framework through an example of HTTP protocol.**

Keywords—Protocol Reverse Engineering; Field Format; Message Format; Flow Format; State Machine; CSP Algorithm;

I. 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 Skype protocol, software update protocols used by antivirus tools, the SCADA protocols, and protocols that are used in various kind of attacks [1]. 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 have been studied, there is no single standardized method to extract protocol specification completely yet, and 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 extract too specific message formats or too general message formats. Extracting too specific message formats means that it cannot extract precise protocol specifications by extracting too many message formats. Extracting too general message formats means that it is not extract all the possible values of the field which belongs to the message format, but extract only one most frequent value of the field which belongs to the message format. Another limitation of prior methods is that it can extract 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 framework for protocol reverse engineering which can extract the protocol syntax, semantics, and finite state machine. The novelty of the protocol syntax extracted by the proposed framework is that it consists of the field formats, the message formats, and the flow formats. The proposed framework use the modified sequential pattern algorithm hierarchically to extract these formats. Additionally, the proposed framework use the modified sequential pattern algorithm recursively to extract the all the values the fields format can have. The meaning of each format is described in section III.A

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 framework in detail. Finally, Section IV presents conclusive remarks and a brief look for the future research directions.

II. RELATED WORKS & PROBLEM SCOPE

This section describes the related works and the scope of the problem. 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 [2]. GAPA is a framework for verifying and parsing network protocol specifications through hand-written syntax. The manual protocol reverse engineering is time-consuming and error-prone. In high-speed network environment like today, automatic protocol 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.

This work was supported by Agency for Defense Development and HANWHA SYSTEMS under the contract (UE161105ED)

978-1-5386-3416-5/18/$31.00 ©2018 IEEE

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 precise. However, previous methods have some limitations.

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

Second, many of these methods extract too specific syntax. Too specific syntax is to extract too many message formats, which means it is not correct message format of the protocol. Therefore, that syntax can be efficient for analyzing each packet, but it is not effective in understanding protocol specifications. 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 based on integration of AprioriAll, AprioriTID, AprioriHash, etc. We call this algorithm Hierarchical Contiguous Sequential Pattern (Hierarchical CSP) algorithm. The proposed framework extract not only message formats 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 of each field that is the most frequent, rather than extracting all the possible values for each fields. Figure 1 shows this problem by exemplifying the HTTP protocol.

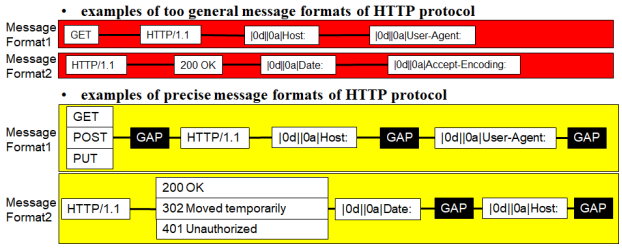

Fig 1. Example of too general message formats and precise message formats

To address this problem, the proposed framework has a module that can extract all the possible values for each field using Recursive CSP algorithm. It is very important to extract the precise field format because if the extracted field formats is not correct, then the message formats composed of these field formats are also incorrect.

III. The Proposed Framework

A. *Terminology and the overview of the proposed framework*

The proposed framework 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 somewhat predictable in length, but GAP is a field whose length cannot be predicted because its length is too variable.

A SF(v) of field format is the single contiguous substring in messages. A message format is the contiguous sequence of 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 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 framework for precise protocol reverse engineering. The framework largely consists of four phases which are Message Assemble, Syntax Inference, Semantics Inference, and the Behavior Inference. The framework's preprocessor is to collect network traces of unknown protocol in flow units which are sets of bidirectional packets with the same 5-tuple(source and destination address, source and destination port, L4 protocol).

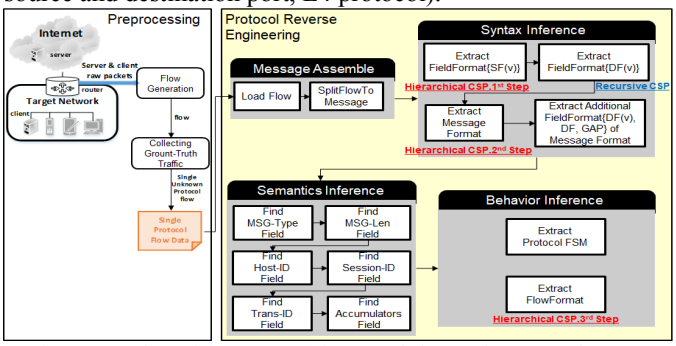

Fig 2. Overview of the proposed framework for precise protocol reverse engineering

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) of field formats using 1st step of Hierarchical CSP algorithm, and extracts all possible values for some extracted SF(v) and

combine them into 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 attaches additional dynamic fields which are DF(v), DF, and GAP to the extracted message formats.

In Semantics Inference phase, the system finds fields corresponding to six predefined semantic types through algorithms of each semantic type.

In Behavior Inference phase, the system extracts two outputs which are the protocol state machine, and the 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.

B. *Contiguous Sequential Pattern Algorithm*

For precise protocol reverse engineering, we developed 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 subsequence 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 characters, but a string (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 sequences and checks the support value of each candidate to determine frequently occurring sequences. In addition, this algorithm improves performance by integrating modified algorithms such as AprioriAll, AprioriTID, AprioriHash, etc.

When extracting the three 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 sequences, composed length-1 item, and support units are different. Figure 3 shows intuitively the process of Hierarchical CSP.

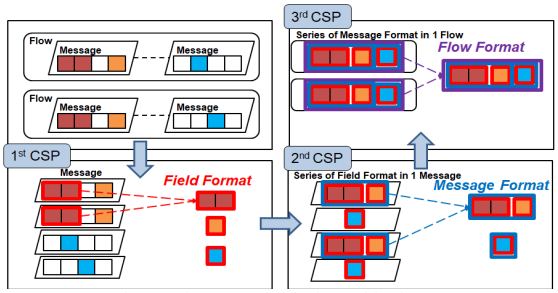

Fig 3. The process of Hierarchical CSP

The 1st step of Hierarchical CSP extracts common substrings that satisfy certain frequencies as SF(v) of field formats. 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) of field formats 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{DF(v), DF, GAP} 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.

C. *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.

D. *Syntax Inference*

The objective of Syntax Inference phase is to extract precise message formats that all the fields of message format are fully categorized into SF(v), DF(v), DF, and GAP.

In the first module, “Extract Field Format{SF(v)}”, it extracts SF(v) of field formats using 1st step of Hierarchical CSP.

In the second module, “Extract Field Format{DF(v)}”, it selects SF(v) that is likely to be converted to DF(v) by checking that the support is not 100% and the position variance is sufficiently low (we use 200). It then performs each Recursive CSP for each selected SF(v).

The process of Recursive CSP is as follows. First, create a database that does not contain the SF(v) from the original database. The created database is truncated based on the minimum offset and maximum depth of the SF(v). Next, it perform CSP from the truncated database. Then, 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 value array, so it is converted to DF(v). Therefore, This DF(v) has all the possible values. Figure 4 shows the process of Recursive CSP by exemplifying the method field and status code field of the HTTP protocol.

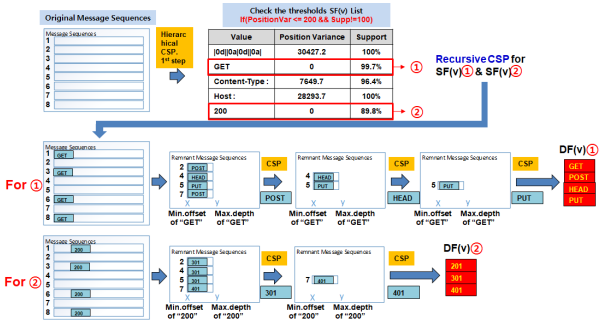

Fig 4. The process of Recursive CSP by exemplifying the HTTP protocol

In the third module, “Extract Message Format”, it 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 Field Format {DF(v), DF, GAP} of Message Format", it classifies all the part between the extracted SF(v) and the DF(v) constituting each message format as 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 DF(v), DF, 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 collect a set of message sequences from the original database(original message sequences) that correspond to the message format. In the collected set of message sequences, find the dataset corresponding to the part to be classified. 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. Do this for all the part between the field formats of the message format. Finally, do above procedure for all message formats.

E. Semantics Inference

In Semantics Inference phase, the system finds fields corresponding to six predefined semantic types through algorithms of each semantic 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 system retrieves the data corresponding to the field format of the message format from the original database and collects attributes for semantic inferring of the field format. This process only targets DF (v) which have the dynamic values of the field format of the message format.

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: $H(x) = -\sum_i^n p_i \boldsymbol{log}_2 p_i$ and causality metric : I(q;r) / H(q) to find this field. q means the value of the field, r means the value of opposite field.

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) 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.

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.

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.

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.

F. Behavior Inference

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

1) Extract Protocol FSM: In the proposed framework, a state(node) of the Protocol FSM 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. The transition probability of each state is calculated through the number of times the transition is matched. It is very useful for packet replay because it helps predict what messages will occur. The extracted FSM helps to confirm in what order the protocol message types operate. Each path connected from the Start state to the End state means each flow types of protocol.

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.

IV. Conclusion and Future Work

In this paper, we propose a framework for precise protocol reverse engineering. We have defined three types of formats which are field format, message format, and flow format to acquire a precise protocol specification and proposed a hierarchical CSP and a recursive CSP to extract such formats. The novelty of this framework is explained with the HTTP protocol as an example. As the future work, we plan to build a system that runs in real network environment based on the proposed framework. We will also improve the proposed framework and apply it to reverse engineer various protocols.

References

[1] 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.

[2] 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.

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