Copyright ⓒ 2020 KSII

Two-Pathway Model for Enhancement of Protocol Reverse Engineering

Young-Hoon Goo[1], Kyu-Seok Shim[1], Ui-Jun Baek[2], and Myung-Sup Kim[2*]
[1] Advanced KREONET Center, Korea Institute of Science and Technology Information
Daejeon, Korea
[e-mail: {gyh0808, kusuk007}@kisti.re.kr]
[2] Department of Computer and Information Science, Korea University
Sejong, Korea
[e-mail: {pb1069, tmskim}@korea.ac.kr]
*Corresponding author: Myung-Sup Kim

Received June 24, 2020; revised August 17, 2020; accepted September 14, 2020; published November 30, 2020

Abstract

With the continuous emergence of new applications and cyberattacks and their frequent updates, the need for automatic protocol reverse engineering is gaining recognition. Although several methods for automatic protocol reverse engineering have been proposed, each method still faces major limitations in extracting clear specifications and in its universal application. In order to overcome such limitations, we propose an automatic protocol reverse engineering method using a two-pathway model based on a contiguous sequential pattern (CSP) algorithm. By using this model, the method can infer both command-oriented protocols and non-command-oriented protocols clearly and in detail. The proposed method infers all the key elements of the protocol, which are syntax, semantics, and finite state machine (FSM), and extracts clear syntax by defining fine-grained field types and three types of format: field format, message format, and flow format. We evaluated the efficacy of the proposed method over two non-command-oriented protocols and three command-oriented protocols: the former are HTTP and DNS, and the latter are FTP, SMTP, and POP3. The experimental results show that this method can reverse engineer with high coverage and correctness rates, more than 98.5% and 99.1% respectively, and be general for both command-oriented and non-command-oriented protocols.

Keywords: Protocol reverse engineering, network security, two-pathway model, command-oriented protocols, contiguous sequential pattern algorithm

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

http://doi.org/10.3837/tiis.2020.11.004 **ISSN : 1976-7277**

1. Introduction

Globally, IP traffic is growing rapidly at a compound annual growth rate (CAGR) of 26.5 percent, and the advancement of Internet-of-Things (IoT) technology will result in a CAGR of 46 percent for mobile traffic in 2022 [1]. The 5G environment is faster, and also supports commercial services. However, as the number of vulnerable mobile and IoT devices increases in the faster network, the number of new applications and attack vectors will also increase. Many of the protocols in this environment are proprietary protocols that are developed and used by specific vendors, or protocols with limited or no specifications, such as botnet's command and control (C&C) protocols. Meanwhile, according to a sample of malicious codes for IoT devices collected by Kaspersky Lab from 2016 to 2018, the number of malware variants that attacked IoT devices in the first half of 2018 exceeded 120,000, more than triple the number of IoT malicious codes found throughout 2017 [2]. Malware on most IoT devices aims to create botnets to facilitate distributed denial-of-service (DDoS) attacks. An example of this is the DDoS attacks distributed via variants of the Mirai botnets after the Mirai source code was released in October 2016. Accordingly, being able to analyze C&C protocols that control malicious code in order to defend against cyberattacks is one of the reasons why structural analysis technology for unknown protocols is needed.

SAMBA is a project that reverse engineered Microsoft's Server Message Block (SMB) protocol to allow file and printer sharing between Windows and other systems [3]. Protocol reverse engineering, the task of extracting the specification of unknown protocols as in the SAMBA example, can be leveraged for heterogeneous interoperability and is essential for efficient network management and security issues. For instance, protocol reverse engineering can be helpful for firewalls and intrusion detection systems to detect and block previously unknown attacks. It can also be used for penetration testing, can be used in a smart fuzzing operation to identify network vulnerabilities, and can provide useful information as part of deep packet inspection (DPI) to analyze malware protocols [4]. However, in order to provide this information, an in-depth understanding of the protocol is first required, and the underlying technology is protocol reverse engineering [5].

In general, in order to infer the specification of an unknown protocol, an inspector such as a network administrator or a security expert manually analyzes the network traces or execution traces of the target protocol. However, this primitive method requires difficult and time-consuming processes. Therefore, the quality of the results may vary depending on the proficiency of the inspector. In addition, this manual and laborious process faces difficulties with the continual appearance of new applications and their frequent updates.

Owing to these circumstances, several automatic reverse engineering methods have been proposed [6-28]. However, there is no method that perfectly extracts the specification of a protocol, and each still has major limitations. First, most existing methods infer only some of the key elements of a protocol: syntax, semantics, and timing. Ideal protocol reverse engineering should reflect all these three key elements. Second, some existing methods extract protocol syntax that is not sufficiently compressed; they extract too many message formats, making it difficult for network administrators to catch the structure of the protocol at a glance. Third, some existing methods extract a syntax that is too generic. The message format they extract consists simply of static fields and gaps; some researchers refer to this gap as a dynamic field because it has variable values, but such a simple classification of fields is insufficient for understanding detailed protocol syntax. Lastly, many previous methods are specialized only for non-command-oriented protocols when inferring syntax.

To address these limitations, we propose an automatic protocol reverse engineering method using a two-pathway model based on network trace. By using the two-pathway model, the proposed method can universally infer the detailed specifications of both command-oriented protocols and non-command-oriented protocols. For a prompt extraction with low overhead, we modify a sequential pattern mining algorithm that finds common characters or hex values in the traffic payload. We refer to this modified algorithm as the contiguous sequential pattern (CSP) algorithm [29]. This algorithm can find protocol patterns very quickly with low memory requirements, because it eliminates candidate patterns at an early stage if they do not satisfy a minimum support threshold. As the length of the pattern increases, the computational overhead dramatically decreases. In addition, we define three types of formats which are field, message, and flow format, and four types of field formats to give a clear and detailed syntax.

The remainder of this manuscript is organized as follows: In Section 2, we present related work. We describe a two-pathway model using a CSP algorithm in Section 3. In Section 4, we evaluate the superiority of the proposed method using five protocols. Finally, we present the conclusion in Section 5.

2. Related Work

In this section, we introduce existing automatic protocol reverse engineering methods and describe their remaining limitations.

2.1 Previous Automatic Protocol Reverse Engineering Method

Several methods have been proposed to address the need for automatic protocol reverse engineering. The most common way to classify protocol reverse engineering methods is to divide them into network trace-based analysis and execution trace-based analysis, depending on whether the network trace is used as the input or the execution trace.

Execution trace-based analysis monitors a binary program that implements the protocol, and analyzes the execution traces that log how the binary program processes messages such as execution commands, memory usage, system calls, and access to specific file systems. Existing methods that use an execution trace include Polyglot [6], Tupni [7], Prospex [8], and Dispatcher [9]. Because these methods analyze the execution of an actual binary program, the accuracy of inferred results may be improved, but virtually, obtaining the binary program of an unknown protocol is not possible. For example, malicious botnet's C&C servers are likely to exist in an external network and are hidden for the success of continuous attacks. Another disadvantage is that usually, only received messages are analyzed, because these methods typically observe and analyze the program binaries of the client while processing input messages. However, in order to analyze the structure of a protocol perfectly, the sent messages must also be analyzed.

However, network trace-based analysis monitors the network packets of the target protocol and analyzes each captured network trace as input. Thus, it is possible to perform network trace-based analysis even in an environment where the host running the program binaries is inaccessible, so it is more practical than execution trace-based analysis. Network trace-based analysis scheme can analyze sent and received messages by capturing the traffic generated from the router connecting the target network with the external network. Hence, our proposed method uses network trace-based analysis, which is advantageous in terms of practicality and convenience.

Existing methods that use network trace include PIP [10], ScriptGen [11], RolePlayer [12], and AutoReEngine [13]. PIP [10] was one of the earliest automatic protocol reverse engineering methods that used network traces. This method finds protocol fields using the sequence alignment algorithm, which is used to find similarity between biological sequences such as amino acid and DNA sequences. The sequence alignment algorithm greatly influenced many of the other existing methods. ScriptGen [11] and RolePlayer [12] also use the sequence alignment algorithm, but the purpose of these methods is packet-replay to trick attackers by making them appear to be under attack. AutoReEngine [13], Wang *et al*. [14], and Ji *et al*. [15] used frequent pattern mining to extract message formats, but the target network environments were different: AutoReEngine [13] focused on application layer protocols built on Ethernet networks, Wang *et al*. [14] focused on protocols built on wireless networks, and Ji *et al*. [15] targeted unmanned aerial vehicle (UAV) control protocols.

2.2 Remaining Limitations

A protocol is a set of communication rules between two devices. The key elements of a protocol are syntax, semantics, and timing. In order to obtain abundant information about an unknown protocol, a protocol reverse engineering method must be able to extract these three key elements. However, most existing methods extract only some of these three key elements. **Table 1** shows the outputs of previous methods. For abundant specifications, we designed our proposed method to extract all three of these key elements.

Table 1. Existing methods from the viewpoint of output

Method	Issue	Output		
		syntax	semantics	FSM
Execution trace-based analysis				
Polyglot [6]	2003	✓		
Tupni [7]	2008	✓		
AutoFormat [16]	2008	✓		
ReFormat [17]	2009	✓		
Prospex [8]	2009			✓
Dispatcher [9]	2013	✓	✓(weak)	
Network trace-based analysis				
PIP [10]	2004	✓		
ScriptGen [11]	2005			✓
RolePlayer [12]	2006	✓	✓(weak)	
Discoverer [18]	2007	✓	✓(weak)	
Pext [19]	2007			✓
Trifilo *et al*. [20]	2009			✓
Biprominer [21]	2011	✓		
ReverX [22]	2011	✓		✓
Veritas [23]	2011	✓		✓
AutoReEngine [13]	2013	✓		✓
Wang *et al*. [14]	2013	✓		
Netzob [24]	2014	✓	✓(weak)	✓(manual)
FieldHunter [25]	2015	✓	✓(strong)	
WASp [26]	2016	✓	✓(weak)	
Ji *et al*. [15]	2017	✓		
Ladi *et al*. [27]	2018	✓		
READ [28]	2019	✓		

The second limitation is that the extracted syntax is not sufficiently compressed, which means that too many message formats are extracted. Thus, it is not easy for a network administrator to intuitively catch the structure of an unknown protocol. In addition, since many message formats are clustered in a top-down clustering method, fields are clustered into one field format only within each message format, so that fields that are actually the same but belong to different message formats are represented differently. This type of syntax can be useful for a detailed analysis of each individual packet in the protocol, but it is not effective for intuitively understanding the structure of the protocol. **Fig. 1** shows part of the analysis of the HTTP protocol using one of the existing methods. Although Group 1, Group 2, and Group 3 are actually of the same HTTP Request message type, and Group 4 and Group 5 are of the same HTTP Response message type, they are clustered as different groups. The red underline indicates a field that has a static value, and the blue underline indicates a field that has a dynamic value. As shown by Group 1 and Group 2 of **Fig. 1**, although the two groups have some of the actually same types of fields, the fields were determined differently to be static or dynamic field formats only within each group. In the case of Netzob [24], with 1000 HTTP protocol packets as input, 324 message formats were extracted when the similarity threshold was set to 50%, and 225 message formats were extracted when the threshold was set to 25%. To address this problem, our proposed method extracts three levels of formats, which are the field format, message format, and flow format, using the bottom-up clustering method.

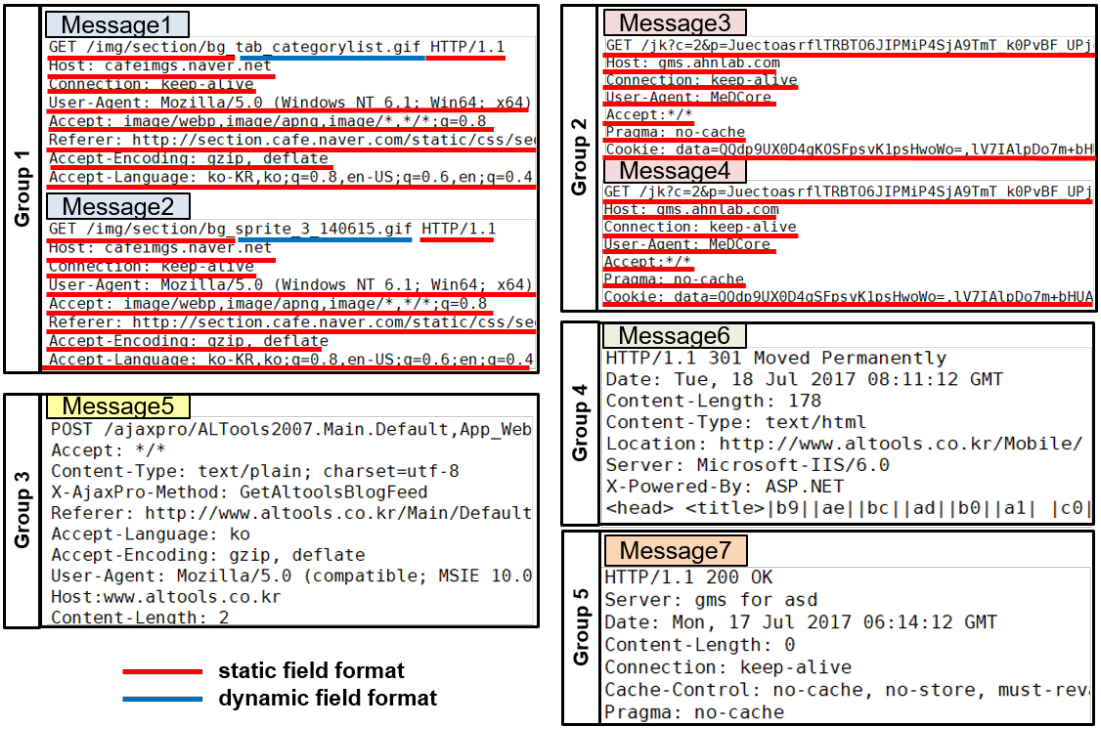


Fig. 1. Example of the problem caused by insufficiently compressed message formats

Third, many of the existing methods extract a syntax that is too generic. When inferring syntax, they use methods based on frequency, such as the Shannon theorem, Zipf's law, the support of association rule mining, or the probability of occurrence of LDA or Markov

chains, and they extract the most frequent values as static fields. Therefore, the extracted message formats are simply composed of static fields and gaps. Some methods call the gap between static fields a dynamic field, but this field does not have much meaning to discern. Therefore, the message format they extracted is only a sequence of protocol keywords. To represent a detailed syntax, we define the four types of field format and extract message formats consisting of these four types of field format, as shown in **Fig. 2**.

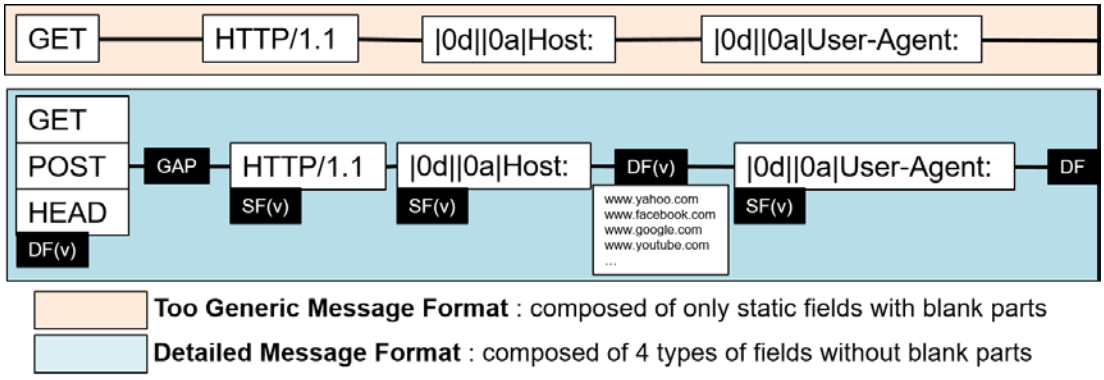


Fig. 2. Example of a message format that is too generic, and a detailed message format

The fourth limitation is that many of existing methods can only infer syntax of non-command-oriented protocols; Protocols are divided into two categories: One is non-command-oriented protocol, and the other is command-oriented protocol. Non-command-oriented protocol literally refers to a basic protocol, not a command-oriented protocol. Namely, for basic protocols, protocol keyword indicates the protocol function, so the keywords appear in the input traffic reflect what protocol functions users generally use. Therefore, generally, the keyword appears more frequently than the noise, and the randomness of the values that the field corresponding keywords can have is low. By contrast, command-oriented protocol means the following.

Command-oriented protocols are protocols with a keyword frequency that follows an almost uniform distribution, and its form is "*Keyword – Arguments*." Generally, when users use a binary program implementing these protocols, they use various commands, each at a low frequency. For example, when using an application implementing File Transfer Protocol (FTP), they first log into the server. Then, they enter the "*PWD*" command to see the working directory, the "*LIST*" command to see the list of elements in the current directory, the "*BIN*" command to change to binary mode, the "*GET*" command to download data, and the "*PUT*" command to upload data. Also, using Simple Mail Transfer Protocol (SMTP), which is another example of a command-oriented protocol, is similar; users use various functions evenly, such as checking a mailbox, sending, deleting, and opening an e-mail.

As mentioned above, because many methods use frequency-based analysis, it is not easy to infer command-oriented protocols in detail. Although there are some methods that infer command-oriented protocols such as FTP, SMTP, and POP3, but the syntax they extract is not detailed because they mainly focus on enabling packet-replay or establishing FSM. For example, they cannot extract a login keyword because the keyword occurs only once in a flow. To infer both command-oriented and non-command-oriented protocols in detail, the proposed method uses a two-pathway model.

3. Two-Pathway Model: Automatic Protocol Reverse Engineering Method

In this section, we describe our method for automatically extracting specification. First, we define the key terms used in this paper as follows. There are four types of field format that our method extracts: SF(v), DF(v), DF, and GAP. (v) stands for having values. SF(v) is a static field that has only one value, and its length is fixed. DF(v), DF, and GAP are dynamic fields that have multiple values, so their lengths may be fixed or variable. The difference between these three field formats is the predictability of their value and length. DF(v) is a dynamic field with a value and length that are both predictable because the number of values the field can have is somewhat limited. DF is a dynamic field with a length that is predictable due to low randomness, but its value is non-predictable. GAP is a dynamic field with a value and length that are both non-predictable. Unlike other existing methods, the proposed method extracts flow formats as well as field and message formats. A flow format represents the main flow type of the protocol, and consists of a sequence of message formats. It can also be used to generalize the FSM.

The system architecture of the proposed method consists of four phases, as shown in **Fig. 3**. There are two pathways in the system architecture. Path 1 is for inferring non-command-oriented protocols, and path 2 is for inferring command-oriented protocols: path 1 and path 2 are represented by the red line and the blue line in **Fig. 3**, respectively. In **Fig. 3**, for a given module, the outgoing arrows towards formats or FSM represent the output of the module. For given formats, the outgoing arrows towards a module represent the input of the module.

Generally, there is exclusiveness between non-command-oriented protocols and command-oriented protocols: if a protocol has many field formats with low occurrence, then that protocol has few field formats with high occurrence. However, we use the two pathways in parallel, factoring in the case in which a protocol has both kinds of field formats; for example, if a non-command-oriented protocol includes a step transmitting a specific message for login, the frequency of the keywords related to this step is once in a flow. In addition, to reduce the processing time and memory requirement, some modules have a condition on each path that automatically determines whether to stop a process.

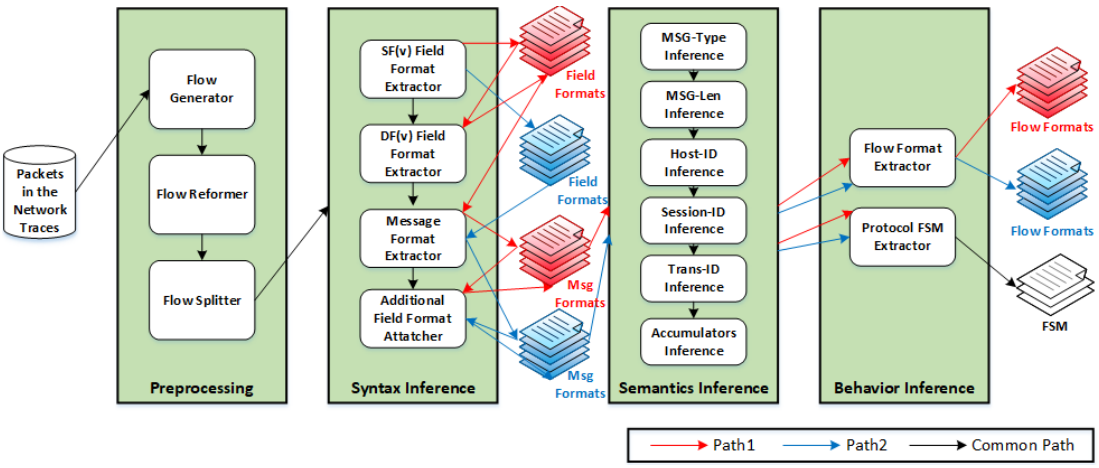


Fig. 3. System architecture of the proposed method

3.1 Preprocessing

In the preprocessing phase, after receiving network traces of the target protocol, the system generates flows. A flow refers to a bidirectional set of packets that have the same 5 tuple: the layer-4 protocol, source IP address/port, and destination IP address/port. Then, the system straightens each flow out by removing packets having no payload, and abnormal packets, such as out-of-order packets and retransmission packets. After reforming the flows correctly, the system reassembles the packets into message units in each flow. If the layer-4 protocol is TCP, the system sets the message unit to a set of consecutive packets with the same direction because TCP is a stream-based protocol, otherwise it sets the message unit to one packet.

3.2 Two-Pathway Model for Extracting Protocol Syntax

The proposed method extracts field formats, message formats, and flow formats by using the CSP algorithm hierarchically, as shown in **Fig. 4**. In this subsection, we describe how to extract these three types of format, and how to apply the format extracting method in the two-pathway model.

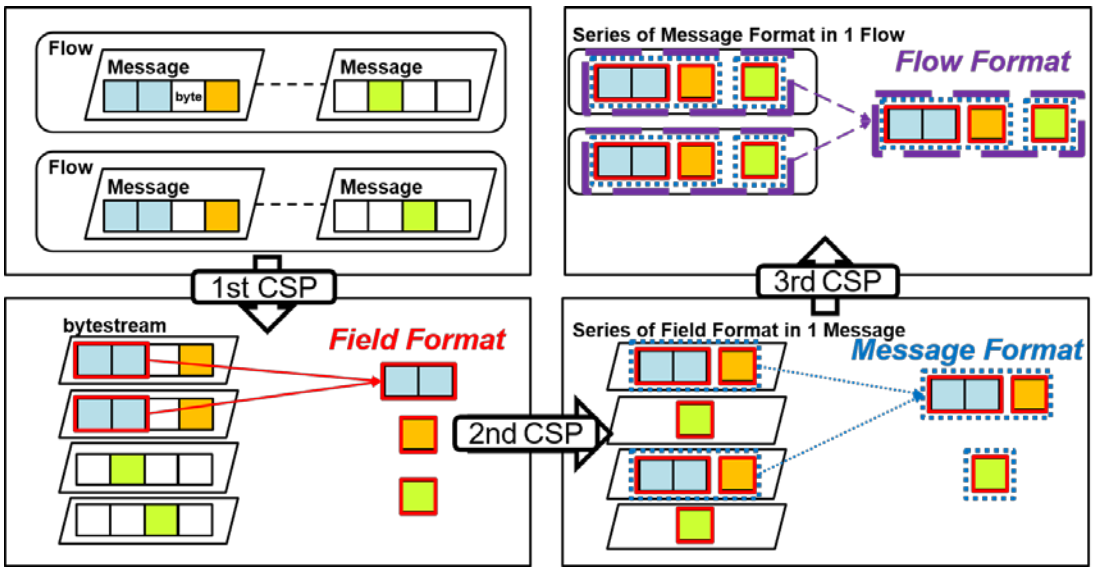


Fig. 4. Concept of extracting protocol syntax using the contiguous sequential algorithm hierarchically

The CSP algorithm is a data mining technique. The goal of this algorithm is to find frequent patterns with items that are consecutively adjacent in a large database. In this algorithm, the threshold of support value is very important. The support value is defined as the ratio of sequences having subsequence to total sequences. The algorithm generates all possible candidate subsequences and calculates the support value for each candidate. Then, it extracts subsequences that have a support value higher than the user-defined threshold.

Fig. 5 depicts the basic CSP algorithm. (1) It extracts unique length-1 items from all the input sequences and stores them in the length-1 candidate set, L_1. (2) From the L_1, it calculates the support value for each candidate, and stores the subsequences that satisfy the user-defined threshold into the frequent subsequence set. (3) It generates length-2 candidates by using the length-1 frequent subsequences. (4) It iterates (2) and (3) by increasing the

length k, until new candidates or frequent subsequences cannot be extracted. (5) Finally, it checks the relation of inclusion between frequent subsequences; if a relation is found, the included subsequences are deleted.

Input: SequenceSet, supp_threshold

Output: FrequentSubSequenceSet

```
01: for Sequence in the SequenceSet do
02:   for item a in the sequence do
03:     L1 ← L1 ∪ a;    //extract length-1 item
04:   end for
05: end for
06: k ← 2;
07: repeat
08:   for candidate c in the Lk-1 do
09:     support ← calSupport(c, SequenceSet); //calculate support
10:     if(support < supp_threshold) then //delete candidate
11:       Lk-1 ← Lk-1 – c;
12:     end if
13:   end for
14:   Lk ← genCandidate(Lk-1); //extract length-k candidate
15:   FrequentSubSequenceSet ← ∪kLk;
16:   k++;
17: until (Lk-1 ≠ ∅)
18: deleteSubset(FrequentSubSequenceSet);
19: return FrequentSubSequenceSet
```

L_1: set of length-1 candidate; L_k: set of length-k candidate;

Fig. 5. Contiguous sequential pattern algorithm

When extracting protocol syntax using the CSP algorithm, length-1 item unit and support unit vary depending on the type of format and the path. Two kinds of support unit (message-based unit and flow-based unit) are used in our method, as shown in Formula 1 and 2; A sequence of $Supp_M$ refers to one message, and A sequence of $Supp_F$ refers to one flow.

$$Supp_M = \frac{n(messages\ having\ target\ subsequences)}{n(total\ messages)} \tag{1}$$

$$Supp_F = \frac{n(flows\ having\ target\ subsequences)}{n(total\ flows)} \tag{2}$$

There are three thresholds for path 1: T_1, T_2, and T_3, and two thresholds for path 2: T_4 and T_5. By running the CSP algorithm with these user-defined thresholds, the system extracts field formats, message formats, and flow formats for each path. T_1 and T_2 are thresholds for $Supp_M$ to extract field formats and message formats of non-command-oriented protocol, respectively. T_3 is a threshold for $Supp_F$ to extract flow formats of non-command-oriented protocol. According to the definition of the CSP algorithm, subsequences that satisfy these thresholds are extracted into formats. T_4 and T_5 are thresholds for $Supp_F$ and $Supp_M$, respectively, and are used to extract the formats of the command-oriented protocol.

The set of field formats, message formats, and flow formats to be extracted for path 1 are defined as follows:

$$FieldFormatSet^{P1} = \{k|k = \langle\overline{b_0 b_1 b_2 \cdots b_n}\rangle, b_i \epsilon\ (0x00, \cdots, 0xFF), k.Supp_M \geq T_1\} \quad (3)$$

$$MsgFormatSet^{P1} = \{m|m = \langle\overline{k_0 k_1 k_2 \cdots k_n}\rangle, k_i \epsilon\ FieldFormatSet^{P1}, m.Supp_M \geq T_2\} \quad (4)$$

$$FlowFormatSet^{P1} = \{f|f = \langle\overline{m_0 m_1 m_2 \cdots m_n}\rangle, m_i \epsilon\ MsgFormatSet^{P1}, f.Supp_F \geq T_3\} \quad (5)$$

where b refers to one byte, k refers to one field format, m refers to one message format, and f refers to one flow format.

In path 1, for inferring non-command-oriented protocols, the system extracts field formats of the type SF(v) by running the CSP algorithm after setting the length-1 item unit to one byte. When extracting message formats, it sets the length-1 item unit to one field format and runs the CSP algorithm. When extracting flow formats, it sets the length-1 item unit to one message format and runs the CSP algorithm. In our experiences, generally, the best thresholds for T_1, T_2, and T_3 are 65%, 50%, and 50%, respectively.

The sets of each type of format to be extracted for path 2 are defined as follows:

$$FieldFormatSet^{P2} = \begin{Bmatrix} k|k = \langle\overline{b_0 b_1 b_2 \cdots b_n}\rangle, b_i \epsilon\ (0x00, \cdots, 0xFF), \\ k.Supp_F \geq T_4\ and\ k.Supp_M \leq T_5 \end{Bmatrix} \quad (6)$$

$$MsgFormatSet^{P2} = \begin{Bmatrix} m|m = \langle\overline{k_0 k_1 k_2 \cdots k_n}\rangle, k_i \epsilon\ FieldFormatSet^{P2}, \\ m.Supp_F \geq T_4\ and\ m.Supp_M \leq T_5 \end{Bmatrix} \quad (7)$$

$$FlowFormatSet^{P2} = \{f|f = \langle\overline{m_0 m_1 m_2 \cdots m_n}\rangle, m_i \epsilon\ MsgFormatSet^{P2}, f.Supp_F \geq T_4\} \quad (8)$$

Path 2 is for command-oriented protocols. The keywords of these protocols appear a very small number of times. However, these keywords are characterized as appearing at least once in a flow, such as a login or a logout process. Therefore, in order to extract field formats and message formats in path 2, the system has to use two support units. When extracting field formats or message formats, the system extracts the subsequences that appear with very high frequency for all flows, and simultaneously with very low frequency for all messages. Theoretically, it is correct to set T_4 to 100% and T_5 to the ratio of the total flows to the total messages, but we set T_4 to 80% and T_5 to 15% considering the case where some packets are collected from the middle of the actual flow. When extracting flow formats, the system uses only one support unit: $Supp_F$. Because it uses the extracted message formats for path 2 as length-1 items to generate the candidates of the flow formats, there is no need to worry about extracting too many flow formats.

The condition to execute the modules for path 2 is that T_5, i.e., the ratio of the total flows to the total messages, is lower than T_1. This regulation is intended to prevent the same formats from being extracted from path 1 and path 2.

3.3 Extracting Dynamic Field Formats

3.3.1 DF(v) Field Format Extractor

To extract DF(v), the system runs the CSP algorithm recursively. The DF(v) field format extractor module selects the SF(v)s that meet certain conditions, and changes their type to DF(v). The conditions are as follows:

- Condition 1: The support value is not 100%.
- Condition 2: The position variance is small.
- Condition 3: The difference between maximum depth and minimum offset is small.

Condition 1 means that the number of values this field format has is not one. Conditions 2 and 3 mean that the position of this field is somewhat fixed, and we use 200 and 40 as the thresholds for Conditions 2 and 3. The system extracts all the values that these field formats can have by running the following process for each SF(v) and satisfying the above three conditions, as shown in **Fig. 6**:

First, the system extracts only message sequences that do not have the SF(v) from all the message sequences, and truncates them based on the minimum offset and maximum depth of the SF(v). After creating a new database, i.e., new message sequences, it runs the CSP algorithm for these new message sequences. Among the outputs of the CSP algorithm, it stores the result that has the highest support value as another value of this field format. The above process is repeated until no new value is extracted. Finally, the system changes the type of this field format to DF(v).

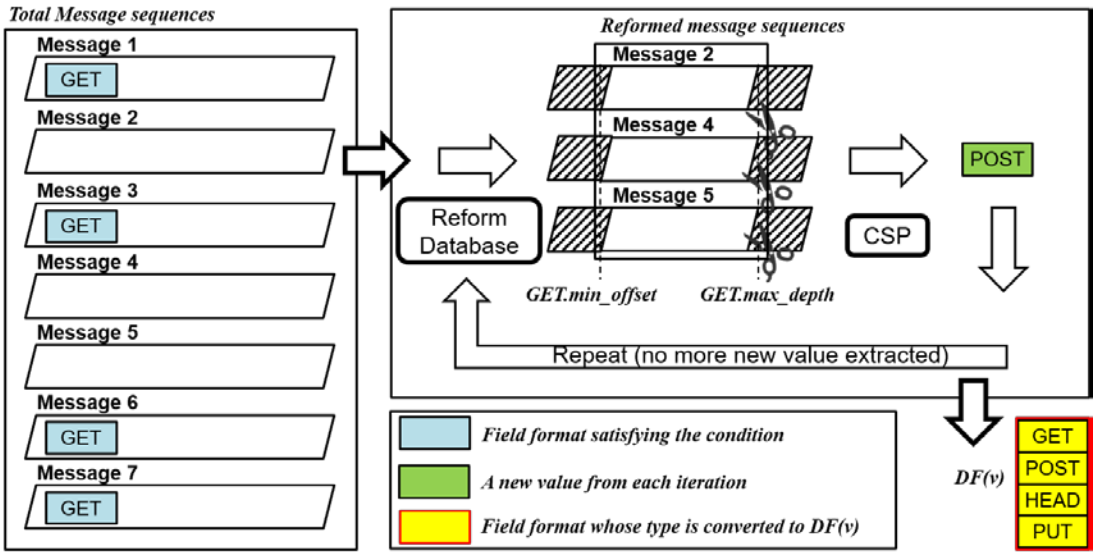


Fig. 6. Process of DF(v) field format extractor

The DF(v) field format extractor module only works with path 1. If running this module in path 2, multiple identical DF(v)s are generated because the field formats of path 2 have a very small support value. Therefore, in the case of path 2, dynamic field formats are extracted only through the additional field format attacher module, described in 3.3.2.

After extracting the DF(v) through this module, the system extracts the message formats by running the CSP algorithm using the extracted SF(v) and DF(v) as length-1 items, as mentioned in 3.2. Path 2 does not go through this module, so only SF (v)s are used as length-1 items to extract the message format.

3.3.2 Additional Field Format Attacher

After extracting message formats, the system fills in the blanks between the field formats that make up the message format with the additional field formats. **Fig. 7** shows the process of extracting the additional field formats for each message format.

First, the system collects only message sequences that have the target message format. To determine the type of each blank in the message format, it collects the length list of the data corresponding to the blank in the message sequences, and calculates the length variance and the maximum length of the data. As shown in the sequence diagram of **Fig. 7**, the system determines the type of blank. If the length variance is higher than 5000, then the blank type is determined to be a GAP. If not, it stores the maximum, minimum, and average length for the blank because the randomness of the length is not too high. After that, it checks whether the maximum length is higher than 20 or not. If the maximum length is higher than 20, then the blank type is determined to be DF, otherwise, it stores the data as the values for the blank. If the number of stored value is only one, then the type of blank is determined to be SF(v). Otherwise, the type of blank is determined to DF(v). The above process is performed for all blanks in the message format, and also for all message formats. As a result, message formats can be obtained which are composed in detail in four types of field formats of SF(v), DF(v), DF, and GAP.

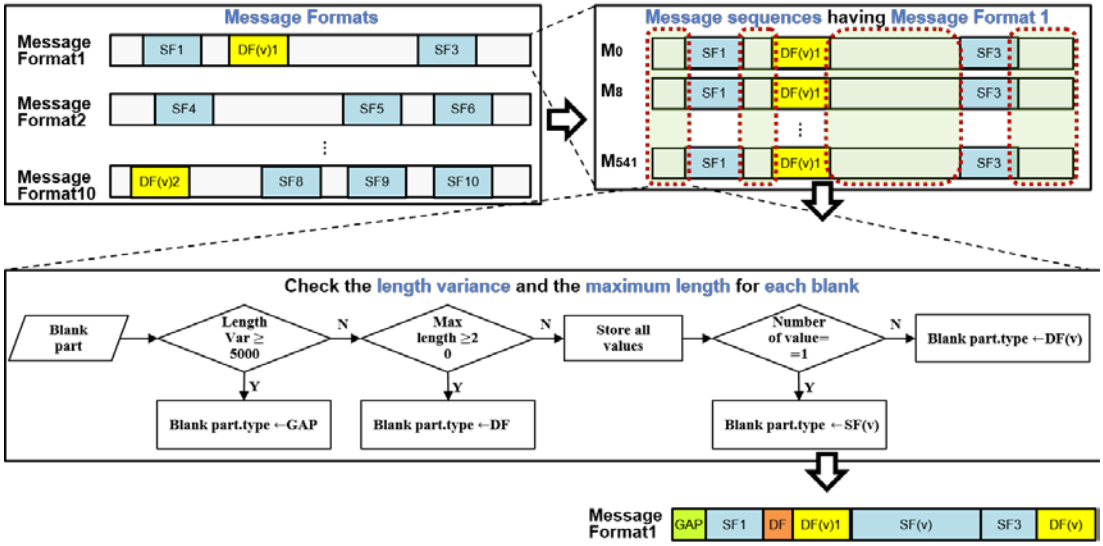


Fig. 7. Process of additional field format attacher

3.4 Semantics Inference

In the semantic inference phase, our method uses FieldHunter's [25] semantic inferencing method to ascertain the semantics of the field formats that make up each message format. This is because FieldHunter's method extracts the most specific and varied types of semantics of the previous methods, as shown in **Table 1**.

The system traverses field formats for each message format, and heuristically checks whether each field format corresponds to six predefined semantic types through each algorithm: MSG-Type, MSG-Len, Host-ID, Session-ID, Trans-ID, and Accumulators.

1) MSG-Type

The algorithm determines whether the field format corresponds to MSG-Type, and whether the following two conditions are met with the entropy and causality metrics.

• Condition 1: The number of values for the field format is not one, but is not too large.

• Condition 2: The field format has a causal relationship with the data in the opposite direction.

To check Condition 1, it uses the entropy metric: $\mathrm{H}(x) = -\sum_{i=1}^{n} p_i \log_2 p_i$ refers to the values of the field format. If the entropy is large, it means that the values of the field format are very randomly distributed, and if the entropy is zero, it means that there is only one possible value for the field format. Therefore, the system checks that the entropy is not zero and less than 0.2. To check Condition 2, it uses the causality metric: $\mathrm{I}(q;r)/\mathrm{H}(q)$. q refers to the values of the field format, and r refers to the values in the opposite direction. The system checks that the causality metric is higher than 0.8.

2) MSG-Len

The system uses the Pearson Correlation metric: $\frac{\mathrm{Cov}(X,Y)}{\sigma(X)\cdot\sigma(Y)}$ for the value of the field and the length of the message to check whether they have a strong positive linear relationship. X refers to the values of the field format, and Y refers to the lengths of the message sequences. A refers to the gradients of the linear function for X and Y, and B refers to the bias of the linear function for X and Y. The system determines that the field format corresponds to MSG-Len if the Pearson correlation metric is greater than 0.7, and the match rate between A and B is greater than 0.9.

3) Host-ID

The system uses the categorical metric $\mathrm{R}(x,y) = \mathrm{I}(x;y)/\mathrm{H}(x,y)$. x refers to the values of the field format, and y refers to source IP addresses. The system determines that the field format corresponds to Host-ID if the categorical metric is greater than 0.9.

4) Session-ID

Like the algorithm for Host-ID, the system uses the categorical metric to check whether the field format corresponds to Session-ID. The only difference is that y of the categorical metric refers to the 5 tuple information.

5) Trans-ID

A transaction is a request and response pair, so the system checks the match rate between the values of the field format and the values in the opposite direction. Also, the system uses the entropy metric to check that the values for the field format are highly random. The system determines that the field format corresponds to Trans-ID if the entropy is greater than 4, and the match rate is greater than 0.8.

6) Accumulators

The system determines that the field format corresponds to Accumulators if the values of the field format are constantly increasing over time.

3.5. Extracting Protocol FSM

In this module, the system creates protocol FSM by matching the completed message formats with all of the messages in input network traces. First, it sorts the flows and messages in the input network traces in chronological order. Next, traversing all the messages, it identifies which message format corresponds to each message, and set the

message formats as nodes of FSM. Then, it represents each occurrence sequence of identified nodes as an edge of FSM and counts the number of times each sequence is founded to calculate the transition probability. As a result, the protocol FSM is created where each state, i.e., node, represents a message format, and each transition, i.e., edge, represents the change from one message format to another. Each transition has a transition probability that can be useful for packet-replay. The FSM we extract contains all the messages of the input network trace, and we can use the extracted flow format to generalize the FSM.

4. Experiment and Results

This section discusses the experimental results of the proposed automatic protocol reverse engineering method. Our evaluation is performed on a machine running CentOS 7, with a quad-core of Intel(R) Core(TM) i7-4770K 3.50GHz CPU, and 32GB of RAM. A prototype implementation of our proposed method written in C++ was used, and a graph visualization tool used to extract png file showing FSM was Graphviz [30]. The prototype system consists of 27 classes composed of 32,295 lines of code.

As mentioned in Section 3, this method uses two pathways for inferring both non-command-oriented protocols and command-oriented protocols. Thus, we collected two application protocols, HTTP and DNS, for path 1, and three application protocols, FTP, SMTP, and POP3, for path 2. The system automatically determined which path to select, as mentioned in Section 3. **Table 2** shows the traffic information of these five protocols.

Table 2. Traffic information of five protocols for experiments

Protocols	Flows	Packets	Bytes (K)	Messages
HTTP	342	13353	17741.5	1922 (req.: 961 / res.:961)
DNS	2254	4878	797.5	4878 (req.: 2481 / res.: 2397)
FTP	535	37117	4351.0	35573 (req.: 18052 / res.: 17521)
SMTP	330	6621	572.0	6284 (req.: 3307 / res.: 2977)
POP3	8	167	40.0	136 (req.: 72 / res.: 64)

In order to evaluate the performance of our method, we measure its coverage and correctness. Coverage is the ratio of the number of messages matched with the extracted message formats to the number of total messages, as shown in Formula 9. It indicates how many messages can be analyzed in the extracted message formats. Correctness is the ratio of the number of true message formats matched with the extracted message formats to the number of total true message formats, as shown in Formula 10. It indicates how many true message formats can be analyzed in the extracted message formats.

$$Coverage = \frac{n(messages\ matched\ with\ extracted\ message\ formats)}{n(total\ messages)} \tag{9}$$

$$Correctness = \frac{n(true\ message\ formats\ matched\ with\ extracted\ message\ formats)}{n(total\ true\ message\ formats)} \tag{10}$$

Table 3 shows a summary of the experimental results. In all protocols, our method extracted protocol specifications within 1 min. Because our algorithm eliminated unsatisfactory candidate formats at an early stage and considered only satisfactory candidates when making an item length longer, the extraction execution time was reduced. The coverage and correctness values for all protocols were almost 100%.

Table 3. Summary of the experimental results

Protocols	Format Info.			FSM Info.		Proc. Time (s)	Performance	
	Field Format	Msg Format	Flow Format	State	Transition		Coverage	Correctness
HTTP	31 (req.: 21 res.: 10)	40 (req.: 22 res.: 18)	3	16	38	17.5	100%	100%
DNS	14 (req.: 3 res.: 11)	14 (req.: 3 Res.: 11)	3	8	28	48.2	100%	100%
FTP	7 (req.: 5 res.: 2)	5 (req.: 3 res.: 2)	11	7	12	27.7	98.5%	99.1%
SMTP	22 (req.: 13 res.: 9)	21 (req.: 12 res.: 9)	36	22	24	11.9	100%	100%
POP3	38 (req.: 31 res.: 7)	27 (req.: 20 res.: 7)	16	19	28	0.2	100%	100%

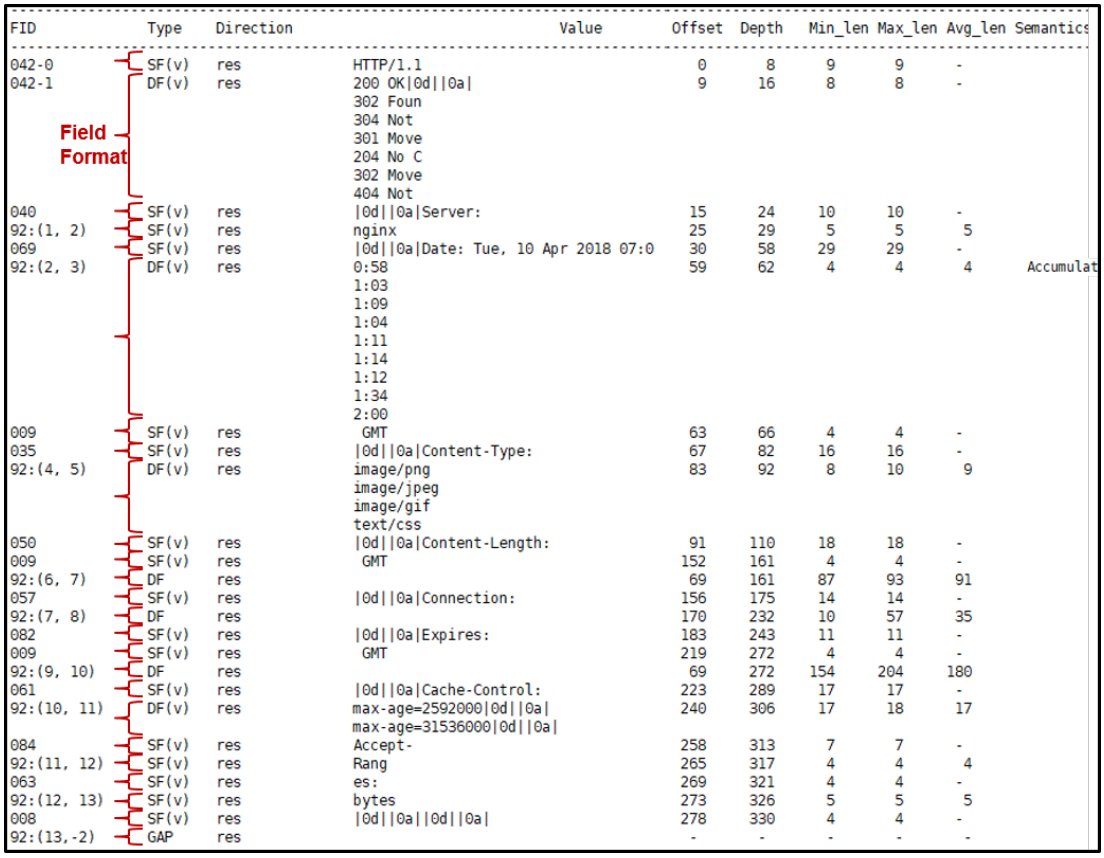
The reason that the coverage and correctness values for FTP protocols were not 100% is that there were some abnormal flows; in these flows, some packets were not captured due to the speed limit of the capture device. For sessions that lasted for a long time when capturing a large amount of traffic, there may have been missed packets due to the limitations of the capture device because new sessions continued to occur. These abnormal flows resulted in incorrect message assembly.

ID	Direction	NumberOfFieldFormat	Support	Coverage	Structure
000	req	3	0.711	0.00%	[GAP] -> [SF(v)] -> [GAP]
001	req	3	0.920	0.00%	[GAP] -> [SF(v)] -> [GAP]
002	req	2	0.997	1.40%	[DF(v)] -> [GAP]
003	req	3	0.740	0.00%	[GAP] -> [SF(v)] -> [GAP]
004	req	3	0.920	0.00%	[GAP] -> [SF(v)] -> [GAP]
005	req	3	0.920	0.00%	[GAP] -> [SF(v)] -> [GAP]
006	req	3	0.775	0.00%	[GAP] -> [SF(v)] -> [GAP]
007	req	3	0.655	0.00%	[GAP] -> [SF(v)] -> [GAP]
008	req	3	0.783	0.00%	[GAP] -> [SF(v)] -> [GAP]
009	req	3	1.000	0.16%	[DF] -> [SF(v)] -> [GAP]
010	req	3	0.920	0.00%	[GAP] -> [SF(v)] -> [GAP]
011	req	3	0.903	0.00%	[GAP] -> [SF(v)] -> [GAP]
012	req	3	1.000	0.00%	[GAP] -> [SF(v)] -> [GAP]
013	req	3	0.909	0.00%	[GAP] -> [SF(v)] -> [GAP]
014	req	3	0.920	0.00%	[GAP] -> [SF(v)] -> [GAP]
015	req	3	1.000	0.00%	[GAP] -> [SF(v)] -> [DF]
016	res	3	0.682	0.10%	[DF] -> [SF(v)] -> [GAP]
017	res	3	0.702	0.00%	[DF] -> [SF(v)] -> [GAP]
018	res	3	0.721	1.20%	[SF(v)] -> [DF(v)] -> [GAP]
019	res	3	0.651	0.26%	[DF] -> [SF(v)] -> [GAP]
020	res	3	0.970	12.02%	[DF] -> [SF(v)] -> [GAP]
021	res	4	0.707	0.00%	[DF] -> [SF(v)] -> [DF(v)] -> [GAP]
022	res	3	0.966	0.88%	[GAP] -> [SF(v)] -> [GAP]
023	res	3	0.894	0.00%	[DF] -> [SF(v)] -> [GAP]
024	res	3	0.672	0.57%	[DF] -> [SF(v)] -> [GAP]
025	res	3	1.000	0.00%	[DF] -> [SF(v)] -> [GAP]
026	req	4	0.969	48.44%	[DF(v)] -> [DF] -> [SF(v)] -> [GAP]
027	req	5	0.651	0.00%	[GAP] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [GAP]
028	res	5	0.697	0.83%	[SF(v)] -> [DF(v)] -> [DF(v)] -> [SF(v)] -> [GAP]
029	res	5	0.589	0.00%	[DF] -> [SF(v)] -> [DF] -> [SF(v)] -> [GAP]
030	res	6	0.683	0.05%	[DF] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [GAP]
031	res	6	0.579	0.00%	[DF] -> [SF(v)] -> [DF(v)] -> [DF] -> [SF(v)] -> [GAP]
032	res	5	0.535	0.00%	[DF] -> [SF(v)] -> [DF] -> [SF(v)] -> [GAP]
033	req	7	1.000	0.00%	[GAP] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [GAP]
034	req	7	0.696	0.00%	[GAP] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [GAP]
035	res	8	0.681	5.20%	[SF(v)] -> [DF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [GAP]
036	res	8	0.578	0.05%	[DF] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [DF] -> [SF(v)] -> [GAP]
037	req	8	0.508	0.00%	[GAP] -> [SF(v)] -> [SF(v)] -> [SF(v)] -> [SF(v)] -> [SF(v)] -> [SF(v)] -> [GAP]
038	req	9	0.877	0.00%	[GAP] -> [SF(v)] -> [DF] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [GAP]
039	res	25	0.576	28.82%	[SF(v)] -> [DF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [DF(v)] -> [SF(v)] -> [SI

Fig. 8. Intuitive structure of the message formats for HTTP protocol

Fig. 8 shows the structure of the message formats for the HTTP protocol. As shown in **Fig. 8**, the proposed method extracts a sufficiently compressed number of message formats, and they have an intuitive structure composed of four types of field formats.

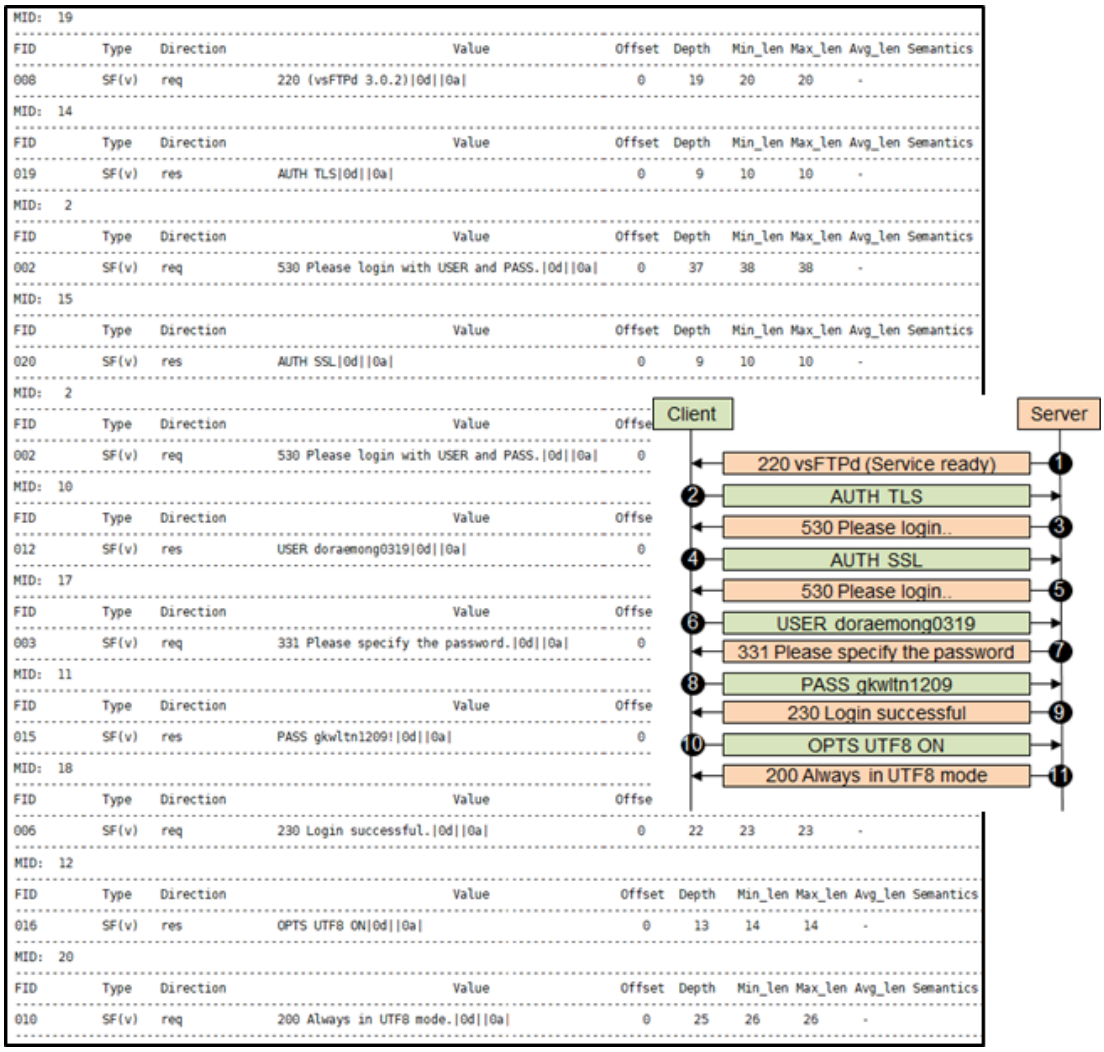
Fig. 9 is one of the extracted message formats of the HTTP protocol. It shows that the proposed method extracts the fully separated message formats without blanks, and each field formats that make up the message format have detailed information, including direction, value, length, position, and semantics. This message format indicates the response message format for the HTTP protocol, and covers all parts of the true message format: from the header line to the double pair of carriage returns and line feed. Also, our method properly determined the semantics of the Date field as Accumulators.



FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
042-0	SF(v)	res	HTTP/1.1	0	8	9	9	-	
042-1	DF(v)	res	200 OK|0d||0a| 302 Foun 304 Not 301 Move 204 No C 302 Move 404 Not	9	16	8	8	-	
040	SF(v)	res	|0d||0a|Server:	15	24	10	10	-	
92:(1, 2)	SF(v)	res	nginx	25	29	5	5	5	
069	SF(v)	res	|0d||0a|Date: Tue, 10 Apr 2018 07:0	30	58	29	29	-	
92:(2, 3)	DF(v)	res	0:58 1:03 1:09 1:04 1:11 1:14 1:12 1:34 2:00	59	62	4	4	4	Accumulat
009	SF(v)	res	GMT	63	66	4	4	-	
035	SF(v)	res	|0d||0a|Content-Type:	67	82	16	16	-	
92:(4, 5)	DF(v)	res	image/png image/jpeg image/gif text/css	83	92	8	10	9	
050	SF(v)	res	|0d||0a|Content-Length:	91	110	18	18	-	
009	SF(v)	res	GMT	152	161	4	4	-	
92:(6, 7)	DF	res		69	161	87	93	91	
057	SF(v)	res	|0d||0a|Connection:	156	175	14	14	-	
92:(7, 8)	DF	res		170	232	10	57	35	
082	SF(v)	res	|0d||0a|Expires:	183	243	11	11	-	
009	SF(v)	res	GMT	219	272	4	4	-	
92:(9, 10)	DF	res		69	272	154	204	180	
061	SF(v)	res	|0d||0a|Cache-Control:	223	289	17	17	-	
92:(10, 11)	DF(v)	res	max-age=2592000|0d||0a| max-age=31536000|0d||0a|	240	306	17	18	17	
084	SF(v)	res	Accept-	258	313	7	7	-	
92:(11, 12)	SF(v)	res	Rang	265	317	4	4	4	
063	SF(v)	res	es:	269	321	4	4	-	
92:(12, 13)	SF(v)	res	bytes	273	326	5	5	5	
008	SF(v)	res	|0d||0a||0d||0a|	278	330	4	4	-	
92:(13,-2)	GAP	res		-	-	-	-	-	

Fig. 9. Sample message format for HTTP protocol

Fig. 10 is one of the extracted flow formats for the FTP protocol, and it perfectly reflects the login process of the FTP protocol. The flow format is a main flow type, and one of the main paths of the FSM.



MID: 19

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
008	SF(v)	req	220 (vsFTPd 3.0.2)|0d||0a|	0	19	20	20	-	

MID: 14

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
019	SF(v)	res	AUTH TLS|0d||0a|	0	9	10	10	-	

MID: 2

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
002	SF(v)	req	530 Please login with USER and PASS.|0d||0a|	0	37	38	38	-	

MID: 15

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
020	SF(v)	res	AUTH SSL|0d||0a|	0	9	10	10	-	

MID: 2

FID	Type	Direction	Value	Offse
002	SF(v)	req	530 Please login with USER and PASS.|0d||0a|	0

MID: 10

FID	Type	Direction	Value	Offse
012	SF(v)	res	USER doraemong0319|0d||0a|	0

MID: 17

FID	Type	Direction	Value	Offse
003	SF(v)	req	331 Please specify the password.|0d||0a|	0

MID: 11

FID	Type	Direction	Value	Offse
015	SF(v)	res	PASS gkwltn1209!|0d||0a|	0

MID: 18

FID	Type	Direction	Value	Offse
006	SF(v)	req	230 Login successful.|0d||0a|	0	22	23	23	-	

MID: 12

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
016	SF(v)	res	OPTS UTF8 ON|0d||0a|	0	13	14	14	-	

MID: 20

FID	Type	Direction	Value	Offset	Depth	Min_len	Max_len	Avg_len	Semantics
010	SF(v)	req	200 Always in UTF8 mode.|0d||0a|	0	25	26	26	-	

Fig. 10. Sample flow format for FTP protocol

Fig. 11 is the extracted FSM of the FTP protocol, and it shows the order in which messages are transferred. Each path from the initial state to the final state refers to a flow type.

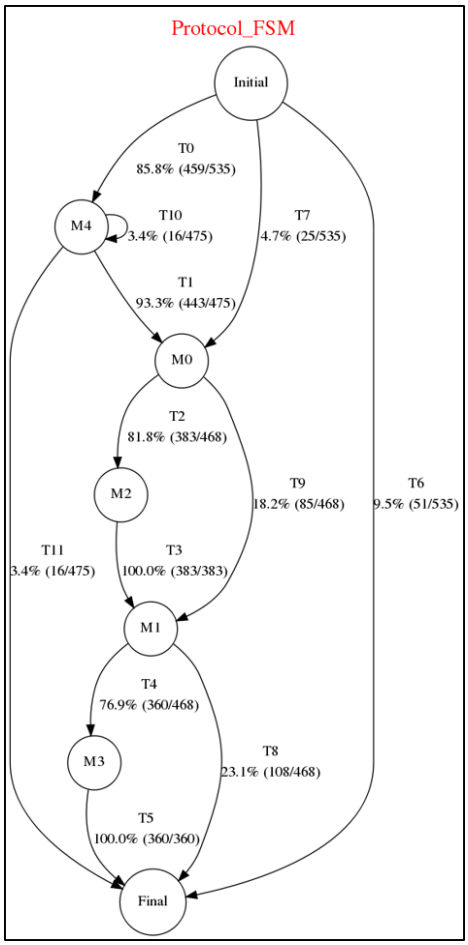


Fig. 11. Extracted FSM of FTP protocol

5. Conclusions and Future Work

In this paper, we proposed a novel method for automatic protocol reverse engineering using a two-pathway model. The proposed method can apply to both command-oriented protocols and non-command-oriented protocols. By defining three types of format and four types of field formats, the proposed method extracts intuitive and detailed protocol specifications using the CSP algorithm hierarchically. Moreover, the proposed method extracts not only syntax but also semantics, and the FSM of the target protocol.

We performed experiments with five known protocols to validate the superiority of the proposed method. In all the protocols, the inferred specification was extracted in less than 1 min. Furthermore, the proposed method exhibited good performance in terms of coverage and correctness.

In future work, we plan to develop a hybrid method that uses both execution traces and network traces to solve the problem of inferring the encrypted protocols. Further, we intend to expand the range of protocol layers to allow for more general use.

References

[1] Cisco VNI, "Cisco Visual Networking Index: Forecast and Methodology, 2016-2017," Cisco, CA, USA, White Paper C11-481360-01, June. 2017.

[2] M. Kuzin, Y. Shmelev, and V. Kuskov, "New trends in the world of IoT threats," *Kaspersky Lab*, Sep. 2018.

[3] A. Tridgell, "How Samba was written," August 2003 [Online]. Available: http://samba.org/ftp/tridge/misc/french_cafe.txt

[4] B. D. Sija, Y.-H. Goo, K.-S. Shim, H. Hasanova, and M.-S. Kim, "A Survey of Automatic Protocol Reverse Engineering Approaches, Methods, and Tools on the Inputs and Outputs View," *Security and Communication Networks*, vol. 2018, Article ID 8370341, pp. 1-17, February 2018. Article (CrossRef Link)

[5] J. Cai, J.-Z. Luo, and F. Lei, "Analyzing Network Protocols of Application Layer Using Hidden Semi-Markov Model," *Mathematical Problems in Engineering*, vol. 2016, Article ID 9161723, pp. 1-14, March 2016. Article (CrossRef Link)

[6] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis," in *Proc. of 14th ACM Conf. on Computer and Communications Security (CCS)*, pp. 317–329, October 2007. Article (CrossRef Link)

[7] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: Automatic Reverse Engineering of Input Formats," in *Proc. of 15th ACM Conf. on Computer and Communications Security (CCS)*, Ocober 2008. Article (CrossRef Link)

[8] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol Specification Extraction," in *Proc. of 30th IEEE Symposium on Security and Privacy*, pp. 110-125, May 2009. Article (CrossRef Link)

[9] J. Caballero and D. Song, "Automatic Protocol Reverse-Engineering: Message Format Extraction and Field Semantics Inference," *Computer Networks*, vol. 57, no. 2, pp. 451-474, 2013. Article (CrossRef Link)

[10] M. A. Beddoe, "The Protocol Informatics Project," 2004 [Online]. Available: http://www.4tphi.net/~awalters/PI/PI.html

[11] C. Leita, K. Mermoud, and M. Dacier, "ScriptGen: an Automated Script Generation Tool for Honeyd," in *Proc. of 21st Annual Computer Security Applications Conf.*, December 2005. Article (CrossRef Link)

[12] W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz, "Protocol-Independent Adaptive Replay of Application Dialog," in *Proc. 13th Symposium on Network and Distributed System Security (NDSS)*, February 2006. Article (CrossRef Link)

[13] J.-Z. Luo and S.-Z. Yu, "Position-based Automatic Reverse Engineering of Network Protocols," *J. Network and Computer Applications*, vol. 36, no. 3, pp. 1070-1077, May 2013. Article (CrossRef Link)

[14] Y. Wang, N. Zhang, Y.-M. Wu, B.-B. Su, and Y.-J. Liao, "Protocol Formats Reverse Engineering Based on Association Rules in Wireless Environment," in *Proc. of 12th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communication*, pp. 134-141, July 2013. Article (CrossRef Link)

[15] R. Ji, H. Li, and C. Tang, "Extracting Keywords of UAVs Wireless Communication Protocols Based on Association Rules Learning," in *Proc. of 12th IEEE Int. Conf. on Computational Intelligence and Security*, pp. 310-313, December 2016. Article (CrossRef Link)

[16] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic Protocol Reverse Engineering through Context-Aware Monitored Execution," in *Proc. of Network and Distributed System Security Symposium (NDSS)*, February 2008. Article (CrossRef Link)

[17] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "ReFormat: Automatic Reverse Engineering of Encrypted Messages," in *Proc. of Computer Security – ESORICS 2009*, in *Proc. 14th European Symposium on Research in Computer Security*, pp. 200-215, September 2009. Article (CrossRef Link)

[18] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic Protocol Reverse Engineering from Network Traces," in *Proc. of 16th USENIX Security Symposium*, pp. 1-14, August 2007. Article (CrossRef Link)
[19] M. Shevertalov and S. Mancoridis, "A Reverse Engineering Tool for Extracting Protocols of Networked Applications," in *Proc. of 14th Working Conf. on Reverse Engineering (WCRE)*, pp. 229-238, October 2007. Article (CrossRef Link)
[20] A. Trifilo, S. Burschka, and E. Biersack, "Traffic to protocol reverse engineering," in *Proc. of IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1-8, July 2009. Article (CrossRef Link)
[21] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo, "Biprominer: automatic mining of binary protocol features," in *Proc. of 12th Int. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 179-184, October 2011. Article (CrossRef Link)
[22] J. Antunes, N. Neves, and P. Verissimo, "Reverse Engineering of Protocols from Network Traces," in *Proc. of 18th Working Conf. on Reverse Engineering (WCRE)*, pp.169-178, October 2011. Article (CrossRef Link)
[23] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, "Inferring Protocol State Machine from Network Traces: a Probabilistic Approach," in *Proc. of 9th Applied Cryptography and Network Security Int. Conf. (ACNS)*, pp. 1-18, 2011. Article (CrossRef Link)
[24] G. Bossert, F. Guihery, and G. Hiet, "Towards Automated Protocol Reverse Engineering using Semantic Information," in *Proc. of 9th ACM Symposium on Information, Computer and Communications Security*, pp. 51-62, June 2014. Article (CrossRef Link)
[25] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo, "Automatic Protocol Field Inference for Deeper Protocol Understanding," in *Proc. of 14th IFIP Networking Conf.*, pp. 1-9, May 2015. Article (CrossRef Link)
[26] K. Choi, Y. Son, J. Noh, H. Shin, J. Choi, and Y. Kim, "Dissecting Customized Protocols: Automatic Analysis for Customized Protocols based on IEEE 802.15.4," in *Proc. of 9th ACM Conf. on Security and Privacy in Wireless and Mobile Networks*, pp. 183-193, July 2016. Article (CrossRef Link)
[27] G. Ladi, L. Buttyan, and T. Holczer, "Message format and field semantics inference for binary protocols using recorded network traffic," in *Proc. of 26th Int. Conf. Software, Telecommunication Computer Networks (SoftCom)*, September 13-15, 2018. Article (CrossRef Link)
[28] M. Marchetti and D. Stabili, "READ: Reverse Engineering of Automotive Data Frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083-1097, September 2018. Article (CrossRef Link)
[29] B. D. Sija, K. S. Shim and M. S. Kim, "Automatic Payload Signature Generation for Accurate Identification of Internet Applications and Application Services," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 4, pp. 1572-1593, April 2018. Article (CrossRef Link)
[30] Graphviz – Graph Visualization Software. Available: https://graphviz.org/

Young-Hoon Goo was born in Cheonan, South Korea, in 1991. He received the B.S. and Ph.D. degrees (integrated program) in computer and information science from Korea University, South Korea, in 2016 and 2020 respectively. Since 2020, he has been a postdoctoral researcher with Korea Institute of Science and Technology Information (KISTI), South Korea. His research interests include Internet traffic classification, Internet security, network management, and wireless communication.

Kyu-Seok Shim was born in Seoul, South Korea, in 1989. He received his B.S., M.S., and Ph.D. degrees in computer and information science from Korea University, South Korea, in 2014, 2016, and 2020, respectively. Since 2020, he has been a postdoctoral researcher with Korea Institute of Science and Technology Information (KISTI), South Korea. His research interests include Internet traffic classification, network management and quantum cryptography.

Ui-Jun Back was born in Seoul, South Korea, in 1993. He received his B.S. degree computer and information science from Korea University, South Korea, in 2018, where he is currently pursuing the Ph.D. degree (integrated program). His research interests include blockchain transaction monitoring, network management and Internet security.

Myung-Sup Kim was born in Gyeongju, South Korea, in 1972. He received his B.S., M.S., and Ph.D. degrees in computer science and engineering from POSTECH, South Korea, in 1998, 2000, and 2004, respectively. From September 2004 to August 2006, he was a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of Toronto, Canada. He joined Korea University, Korea, in 2006, where he is working currently as an Full Professor with the Department of Computer Convergence Software. His research interests include Internet traffic monitoring and analysis, service and network management, the future Internet, and Internet security.