Computer Communications 194 (2022) 1–14

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Sub-messages extraction for industrial control protocol reverse engineering

Yuhuan Liu [a,b], Fengyun Zhang [a,b], Yulong Ding [a,b,*], Jie Jiang [a,b], Shuang-Hua Yang [a,c,*]

[a] Shenzhen Key Laboratory of Safety and Security for Next Generation of Industrial Internet, Southern University of Science and Technology, Shenzhen, China
[b] Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China
[c] Department of Computer Science, University of Reading, Reading, UK

ARTICLE INFO

Keywords:
Industrial Internet of Things
Industrial control protocol
Protocol reverse engineering
Protocol message format

ABSTRACT

The Industrial Internet of Things (IIoT) connects various industrial devices and processes for smart manufacturing purposes. The industrial devices and processes may employ standard or private communication protocols. Protocol Reverse Engineering (PRE) can infer the format of the unknown protocol by analyzing traffic traces. Existing work in the field mainly focuses on Internet protocol only, handling text messages. PRE for industrial control protocols is difficult and particularly designed for IIoT for real-time interconnection among industrial devices. Given the phenomenon that many consecutive sub-messages are often embedded in a lengthy message payload and have a similar format, a novel sub-messages extraction algorithm is proposed in this work by using template iteration as an intermediate step to form a full message format inference framework. An improved evaluation criterion is also proposed to evaluate the sub-messages extraction results. We carry out our algorithm on three standard industrial control protocols and two unknown protocols. Experiments show that adding our sub-messages extraction in PRE for IIoT can greatly improve the accuracy of the overall protocol format inference compared with the existing work.

1. Introduction

The IIoT focuses on information communication among physical equipment, people, and processes between different technical areas, including network connectivity and industrial control systems [1]. The equipment systems of different companies often adopt their own designed industrial control protocols (ICPs). Some are public, e.g., Modbus, CAN, DNP3 and EtherCAT, but most are private. Companies use private protocols to protect their intellectual property rights and hope that their protocols will become industry standards. Nowadays, many companies put forward their own protocol standards, such as S7 of Siemens, FINS of Omron, and MELSEC of Mitsubishi. For large companies, subsystems of IIoT are embedded with their core and innovative technical privileges, so they hope users to purchase their whole series of IIoT products to build the overall system. However, due to the different advantageous products of each company, especially the advantages of many small companies in some products, users often choose the products from various companies to establish IIoT system, which leads to the problems of product interconnection and joint security defense. In addition, the upgrading of old equipment is slow and expensive. The application layer protocol of new equipment is often unknown, which makes it impossible for users to connect the new equipment to the network. In the future, there will eventually be a unified industry standard protocol specification, e.g., OPC protocol [2] is promising, but this process is long. OPC is mainly aimed at the unified communication between different control devices in the upper layer, while the controlled devices in the lower layer may still use private protocols. Besides, many companies' latest devices are already compatible with OPC protocol, but their old devices may not be compatible. The private protocols are the immediate obstacle to overcome in the device interconnection and are also the first line of defense for system security, which can prevent some attacks against protocol vulnerabilities, but can also be bypassed by attackers.

The popularity of the IIoT increases the risk of network security problems. Most of the IIoT security analysis such as intrusion detection system (IDS) [3–5], vulnerability assessment [6,7], fuzzing [8–10] and attack detection [11,12] needs to be carried out based on known public protocols. For example, the malicious nodes spread malware through communication with legitimate nodes so that the fully parsed protocol can be used to assemble malicious attack messages and simulate industrial control network confrontation [13]; knowing the protocol field format can help set intrusion detection strategy and field backtracking [14]; and fuzz testing needs to use communication protocol to generate random messages for vulnerability scanning. These works are difficult, if not impossible, on unknown protocols. So, it is necessary and urgent to fully understand the communication protocol's format. One way to get the specifications of unknown protocols is Protocol Reverse Engineering (PRE).

* Corresponding authors.
E-mail addresses: liuyh2019@mail.sustech.edu.cn (Y. Liu), zhangfy2019@mail.sustech.edu.cn (F. Zhang), dingyl@sustech.edu.cn (Y. Ding), jiangj@sustech.edu.cn (J. Jiang), yangsh@sustech.edu.cn (S.-H. Yang).

https://doi.org/10.1016/j.comcom.2022.07.025
Received 12 January 2022; Received in revised form 21 April 2022; Accepted 14 July 2022
Available online 21 July 2022

PRE is mainly divided into two categories according to the analysis objects: execution trace [15–21] and traffic trace. For ICPs, the execution trace analysis is complex because it requires the approval of the memory and executable program of the corresponding software of the device. In contrast, traffic trace analysis is based on the protocol messages in communication, which is easy to obtain. Encrypted or compressed messages are hard to reverse, so they are not considered in this paper. There are three main objectives: clustering of message types, inference of message format and semantics, and speculation of protocol state machine [22]. This paper focuses on the inference of protocol format based on non-encrypted and non-compressed traffic trace without any modification to the protocol itself.

The algorithms and models of traditional PRE focus on Internet protocols [23]. Since the Internet protocols are mainly in textual form, most methods come from natural language processing (NLP) [24]. Other methods for binary protocols are mainly inspired by the algorithms in bioinformatics, e.g., Needleman–Wunsch (NW) algorithm [25]. Due to the need for real-time and reliability, most ICPs communicate in binary non-encrypted form [26], which is abstract and incomprehensible. So, the NLP-based parsing methods are not suitable for ICPs.

Previous work [27–29] mainly focus on inferring the protocol header's format while the random payload may affect the inferred result of the header format. The analysis of the payload may also be beneficial to the whole reverse work. For instance, the text fields often appear in the payload, so that Discoverer [30] segments message fields by identifying text and binary. The payload can be roughly divided into two types: operation type (e.g., Set, Start and End) and data transmission type. The data can be formatted or unformatted. When transmitting formatted data, the payload is lengthy and likely to be composed of multiple similar sub-messages, e.g., Modbus [31], EtherCAT [32], S7 Communication [33], DNP3 [34], C37.118 [35], and Ethernet/IP-CIP [36]. The format of these sub-messages is theoretically the same: a fixed-length sub-message header with a variable-length sub-message payload. Fig. 1 shows several messages in the S7 Communication protocol and the underlined fields are the formatted data. There may be a different number of sub-messages under the same command, e.g., m_8 and m_9. For the unformatted data, its value is random, so it is often regarded as noise, and this paper also filters it.

Different from the external comparison in previous work, i.e., comparing the difference among messages to infer the protocol format, sub-message extraction is an internal comparison, which is more effective for the inference of formatted payload. Accurate sub-message extraction results are essential for traffic trace-based PRE, e.g., it can unify the messages with the different number of sub-messages into an integrated one, which is beneficial for the message type clustering; the format of sub-messages can be inferred accurately based on the referred sub-messages, which is a significant part of the total message format; and it may provide some guidance (e.g., keywords discovery) for the semantic analysis.

In this work, we design a comprehensive process to infer the format of the whole message, which is composed of several sub-messages. To achieve the goal of accurate extraction of sub-messages, we propose a template iteration approach, including an application of autocorrelation function (ACF) to the template initialization, a novel way to apply dynamic programming (DP) algorithm for segmentation based on a fixed template, and an adapted progressive multiple sequence alignment (PMSA) algorithm for template updating. We name the process as **SEIP**: **S**ub-messages **E**xtraction for **I**ndustrial control **P**rotocol. We evaluate SEIP on three standard industrial control protocols: Modbus-TCP, S7-Communication, and Ethernet/IP-CIP and two unknown protocols. The results illustrate that our approach has a better performance than the previous algorithms. The main contributions of our work are as follows:

1. We design an effective process called SEIP to infer the whole protocol format. To the best of our knowledge, it is the first trial to parse the long message payload of industrial control protocol.
2. To solve the problem of sub-messages extraction, we propose a template iteration approach, including a template initialization method using ACF, a novel segmentation method using DP, and an adapted PMSA algorithm to generate new templates.
3. Given the characteristic of sub-message continuity, the result of sub-messages extraction can have some offsets. We propose an improved evaluation criterion to evaluate the result more accurately.
4. We evaluate SEIP on three standard industrial control protocols and two unknown protocols. Experiments show that SEIP can significantly improve the accuracy of the overall format inference process.

The rest of this paper is structured as follows: We first revisit and discuss related work in PRE in the next section. Then, we present the SEIP process in Section 3. After that, we introduce the details of the sub-messages extraction algorithm in Section 4. We introduce an improved evaluation criterion for sub-messages extraction in Section 5. We apply SEIP to several commands of three industrial control protocols and conduct extensive experiments in Section 6. Finally, we conclude our work in Section 7.

2. Related works

The pioneering work of PRE by analyzing communication messages is Protocol Informatics (PI) [37], led by Beddoe. He first applies the sequence alignment algorithm in bioinformatics to the alignment of protocol messages. Based on Beddoe's work, Bossert designs an efficient and open reverse tool called Netzob [38,39]. Sequence alignment is not only between two sequences but more often among multiple sequences. However, multiple sequence alignment (MSA) is NP-complete [40]. The common way to solve MSA is the progressive multiple sequence alignment (PMSA) algorithm [41], which employs an iterative method to match pairwise, e.g., UPGMA method [42]. PMSA is widely adopted in Bioinformatics, e.g., ClustalW [43] and T-coffee [44]. PI and Netzob also use PMSA to solve the MSA issue. The sequence alignment algorithms aim to maximize global or local match; but do not pay attention to calculating the similarity of consecutive segments, which could not extract sub-messages from a single protocol message.

ScriptGen [45] and Discoverer [30] also use the sequence alignment algorithm to infer protocol format. But unlike PI and Netzob, they propose some effective statistical methods to extract tokens and align tokens instead of bytes. It seems to be more suitable for textual protocols than binary protocols because textual protocols usually consist of words, which can be abstracted into tokens. Biprominer [46] designs a transition probability model to extract binary protocol messages, mainly based on statistical methods. ProDecoder [47] adopts the Latent Dirichlet Allocation model based on keyword identification, which supports keyword-based protocols. PRE-Bin [28] employs the Bayesian model to determine the field boundaries. Wang et al. [48] and Hei et al. [49] both adjust the Actor–Critic algorithm to make it more suitable for extracting frequent keywords. FieldHunter [50] applies statistical correlations to extract some specific fields derived from the author's investigation of standard protocols. IPART [29] uses an improved voting expert algorithm in industrial protocols, mainly focusing on the protocol header parsing. NetPlier [51] uses probabilistic inference to recognize the command keywords. The command keywords are not necessarily just one field, but can also be multiple and scattered in the message. For instance, in Fig. 1, the bold fields are the command keywords and each message has a different number and location of command keywords. ReFSM [52] uses the extended finite state machine to interpret the data flow. NEMESYS [53,54] cuts out segments from a single message through the difference of adjacent bytes. NEMETYL [55] clusters protocol types based on the result of NEMESYS.

Although there are various methods and tools to infer the protocol format, a segmentation extracted from sub-messages inference in a single message remains unsolved. This work is an intermediate step in the whole protocol format inference process without any prior knowledge that may significantly improve the overall accuracy.

No.	Time	Source	Destination	TPKT + COTP + S7 Communication	Commands
m_1	3.888	192.168.1.10	192.168.1.40	03 00 00 16 11 **e0** 00 00 00 07 00 c1 02 02 00 c2 02 02 02 c0 01 0a	CR Connect Request (0x0e)
m_2	3.892	192.168.1.40	192.168.1.10	03 00 00 16 11 **d0** 00 07 00 03 00 c1 02 02 00 c2 02 02 02 c0 01 0a	CC Connect Confirm (0x0d)
m_3	3.892	192.168.1.10	192.168.1.40	03 00 00 19 02 **f0** 80 32 **01** 00 00 02 00 00 08 00 00 **f0** 00 00 01 00 01 01 e0	DT Data (0x0f); ROSCTR: Job (1); Function: Setup communication (0xf0)
m_4	3.895	192.168.1.40	192.168.1.10	03 00 00 1b 02 **f0** 80 32 **03** 00 00 02 00 00 08 00 00 00 00 **f0** 00 00 01 00 01 00 f0	DT Data (0x0f); ROSCTR: Ack_Data (3); Function: Setup communication (0xf0)
m_5	3.895	192.168.1.10	192.168.1.40	03 00 00 07 02 **f0** 00	DT Data (0x0f)
m_6	3.910	192.168.1.10	192.168.1.40	03 00 00 1f 02 **f0** 80 32 **07** 00 00 0e 00 00 08 00 06 00 01 12 04 **11 43 02** 00 **ff** 09 00 02 30 38	DT Data (0x0f); ROSCTR: Userdata (7); Function: Request (0x11)->Block function(3)-> List blocks of type(2); Return code: Success (0xff)
m_7	3.911	192.168.1.10	192.168.1.40	03 00 00 1f 02 **f0** 80 32 **07** 00 00 10 00 00 08 00 06 00 01 12 04 **11 43 02** 00 **ff** 09 00 02 30 42	DT Data (0x0f); ROSCTR: Userdata (7); Function: Request (0x11)->Block function(3)-> List blocks of type(2); Return code: Success (0xff)
m_8	3.933	192.168.1.40	192.168.1.10	03 00 00 25 02 **f0** 80 32 **07** 00 00 0e 00 00 0c 00 08 00 01 12 08 **12 83 02** 01 00 00 00 00 **ff** 09 00 04 00 01 22 03	DT Data (0x0f); ROSCTR: Userdata (7); Function: Response (0x12)->Block function(3)-> List blocks of type(2); Return code: Success (0xff)
m_9	3.942	192.168.1.40	192.168.1.10	03 00 00 51 02 **f0** 80 32 **07** 00 00 10 00 00 0c 00 34 00 01 12 08 **12 83 02** 01 00 00 00 00 **ff** 09 00 30 00 00 22 07 00 01 22 07 00 02 22 11 00 03 22 07 00 04 22 07 00 07 22 07 00 c8 22 10 02 bd 22 10 0b b8 22 07 03 e8 22 07 03 ea 22 07 03 e9 22 07	DT Data (0x0f); ROSCTR: Userdata (7); Function: Response (0x12)->Block function(3)-> List blocks of type(2); Return code: Success (0xff)
m_{10}	4.012	192.168.1.10	192.168.1.40	03 00 00 16 11 **e0** 00 00 00 01 00 c1 02 01 00 c2 02 01 02 c0 01 09	CR Connect Request (0x0e)
m_{11}	4.016	192.168.1.40	192.168.1.10	03 00 00 16 11 **d0** 00 01 00 03 00 c0 01 09 c1 02 01 00 c2 02 01 02	CC Connect Confirm (0x0d)
m_{12}	4.016	192.168.1.10	192.168.1.40	03 00 00 19 02 **f0** 80 32 **01** 00 00 ff ff 00 08 00 00 **f0** 00 00 01 00 01 07 80	DT Data (0x0f); ROSCTR: Job (1); Function: Setup communication (0xf0)
m_{13}	4.020	192.168.1.40	192.168.1.10	03 00 00 1b 02 **f0** 80 32 **03** 00 00 ff ff 00 08 00 00 00 00 **f0** 00 00 01 00 01 00 f0	DT Data (0x0f); ROSCTR: Ack_Data (3); Function: Setup communication (0xf0)
m_{14}	4.023	192.168.1.10	192.168.1.40	03 00 00 1f 02 **f0** 80 32 **01** 00 00 00 01 00 0e 00 00 **04** 01 12 0a 10 02 00 10 00 00 83 00 00 00	DT Data (0x0f); ROSCTR: Job (1); Function: Read Var (0x04)
m_{15}	4.026	192.168.1.40	192.168.1.10	03 00 00 29 02 **f0** 80 32 **03** 00 00 00 01 00 02 00 14 00 00 **04** 01 ff 04 00 80 a9 10 00 00 00 00 01 01 00 00 00 00 00 00 00 00 00	DT Data (0x0f); ROSCTR: Ack_Data (3); Function: Read Var (0x04)
m_{16}	4.036	192.168.1.10	192.168.1.40	03 00 00 43 02 **f0** 80 32 **01** 00 00 45 00 00 32 00 00 **04** 04 12 0a 10 09 00 07 00 01 1d 00 a2 d4 12 0a 10 09 00 f2 00 8b 83 00 f2 81 12 0a 10 09 00 ce 00 4a 81 00 45 58 12 0a 10 1c 00 39 00 2a 83 00 1a 38	DT Data (0x0f); ROSCTR: Job (1); Function: Read Var (0x04)
m_{17}	4.040	192.168.1.40	192.168.1.10	03 00 00 3d 02 **f0** 80 32 **03** 00 00 45 00 00 02 00 28 00 00 **04** 04 ff 09 00 08 43 4c 57 4e 4a 73 4f 52 ff 09 00 08 6a 35 64 57 76 53 79 54 ff 09 00 02 75 69 ff 09 00 06 59 59 79 45 6c 4a	DT Data (0x0f); ROSCTR: Ack_Data (3); Function: Read Var (0x04)

Fig. 1. Example of S7 Communication protocol (with protocol Id 0x32) messages. S7 Communication protocol is based on TPKT and COTP. This example includes the connection of COTP, the setup of S7 Communication protocol and several commands that can have sub-messages structure.

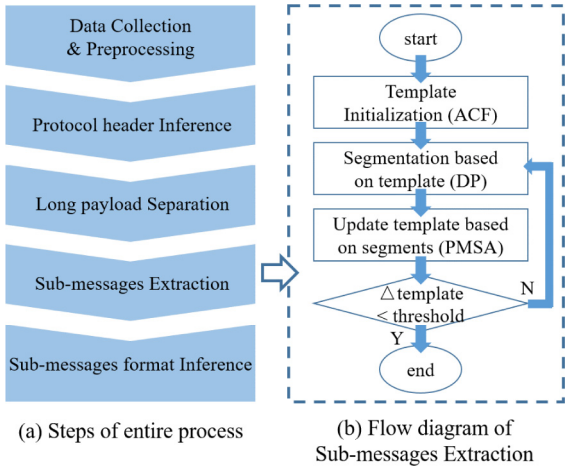


Fig. 2. Flow diagram of SEIP.

3. SEIP process

This section introduces the process of SEIP, as illustrated in Fig. 2a, including five steps: data collection and preprocessing, protocol header inference, long payload separation, sub-messages extraction, and sub-messages format inference. For the core step of sub-messages extraction, we propose a template iteration approach, as shown in Fig. 2b, including template initialization, segmentation based on the template, and update template based on segments.

3.1. Data collection and preprocessing

We collect the open-source data [56–58] of the industrial control protocol. The collected dataset command types are shown in Table 1. The Sub-messages column shows the number of command types that can be composed by several sub-messages. Modbus/TCP is a common industrial control protocol. Its operation objects mainly include coils, registers, files, etc. Among them, the operation of the files has the structure of sub-messages. S7 Communication belongs to Siemens and is based on TPKT and COTP protocol. Its protocol commands can be divided into operation commands, e.g., Setup, Start and Stop, and data transmission commands. Most data parts are formatted so that they can be recognized as sub-messages. EthernetIP/CIP belongs to ODVA and it has a special class of commands called Multiple Service Packet. Almost all commands for data operation or service have their corresponding multiple packet transmission commands. DNP3.0 is based on IEC870-5 and has a three-tier structure. In its application layer, many operations target multiple objects which can be regarded as sub-messages. C37.118 is a grid protocol and all its data frames have a part called Phasors recording several voltages, currents and phases. EtherCAT protocol is a Fieldbus protocol based on Ethernet. Its protocol format is a frame header and several datagrams. Every datagram owns a command so that it can be recognized as a sub-message.

Table 1
Collected dataset command types.

Protocol	Command Types	
	Total	Sub-messages
Modbus/TCP	10	2
S7Communication	36	11
EthernetIP/CIP	31	7
DNP3.0	25	12
C37.118	6	2
EtherCAT	8	8

We use Wireshark [59], a widely-used network protocol analyzer, to clean the communication messages, including deleting invalid, duplicated, and retransmitted packets, filtering the underlying communication protocols (e.g., TCP/IP), and extracting the payload (application layer data) for subsequent steps. In addition, the analysis results can be utilized as the ground truth of our work.

3.2. Protocol header inference

This step includes two goals: one is to separate the protocol header and payload accurately, the other is to infer the protocol header format. Due to the randomness of the data part in the payload, the entropy of the payload is larger than that of the protocol header [60]. The entropy function is $H(b_x) = \sum_i -p_i \log_2 p_i$ where p_i is the probability of value i at the same byte position b_x among different messages. For example, in Fig. 1, the first position is 0x03 in all messages, so $p_{0x03} = 1$ and $H(b_0) = 0$. The more backward the position is, the greater the probability that it is a payload. Since the length of some messages

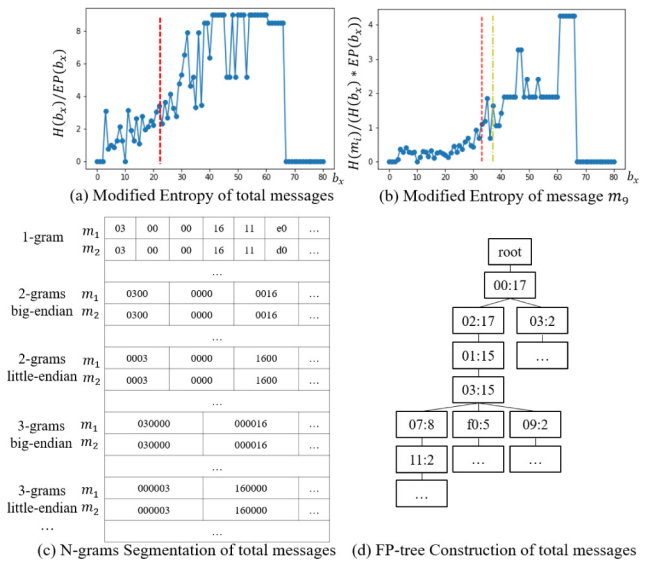


Fig. 3. Header inference and objective payload separation using the messages in Fig. 1 as examples.

cannot reach the backward positions, we set a percentage of messages that are at the position b_x:

$$EP(b_x) = \frac{Messages\ with\ position\ b_x}{Total\ messages} \tag{1}$$

The modified entropy is $H(b_x)/EP(b_x)$. Fig. 3a is an example of the modified entropy of the messages in Fig. 1. The red dashed line is the average position of the total message payloads.

For each message m_i, its modified entropy is its contribution to the total modified entropy:

$$H_{m_i}(b_x) = \frac{H(m_i)}{H(b_x) * EP(b_x)} \tag{2}$$

Fig. 3b is an example using message m_9. The red dashed line is the first sub-message starting position and the yellow dash–dot line is the second sub-message starting position. We use a threshold θ_H to separate the protocol header and payload.

Through the N-grams segmentation [61], we can get a variety of field segmentation. Fig. 3c is an example using the total messages in Fig. 1. We also consider the difference between big-endian and little-endian. Then we calculate the Pearson correlation [62] among all the fields and the message length, e.g., the Pearson value between the fourth byte (0x16 in m_1) in 1-gram and the message length is 1.0 so that it may be a length field (The ground truth is that the third bytes 0x0016 in 2-grams big-endian is the length field).

We also use the frequent item analysis and construct FP tree [63], as shown in Fig. 3d, e.g., the fifth byte (0x02) and the sixth byte (0xf0) are in the same branch of the FP tree so that they may be the associated frequent items (The ground truth is that the byte 0xf0 is the command of COTP and the byte 0x02 is the length of COTP). The command set corresponding to the message can also be found in FP tree, e.g., the command set (The bold bytes) of m_3: {0xf0, 0x01} is in the FP tree.

We distinguish the fixed fields and find the correlation between fields. Finally, we merge the adjacent fields of the same type and the fields adjacent to the zero-padding fields. The following high entropy non-aligned part of the message is the message payload for subsequent analysis.

The accurate cutting of the protocol header plays a crucial role in subsequent sub-message extraction. Errors in cutting the protocol header will lead to incorrect payload data, resulting in template initialization errors. The inaccurate separation of header and payload leads to low accuracy of the following sub-messages extraction step, but may not lead to poor results in sub-messages format inference. In the evaluation section, we evaluated and discussed the impact of incomplete separation situations.

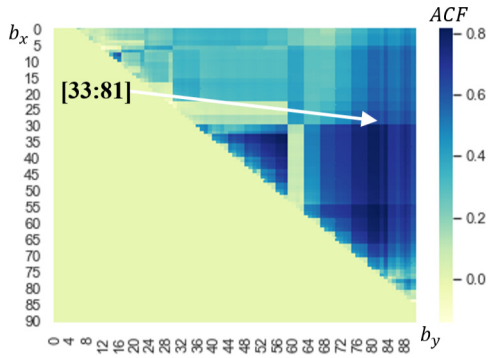


Fig. 4. ACF Heating map of extended message m_9 in Fig. 1.

3.3. Long payload separation

The subsequent work is aimed at identifying the lengthy payload with sub-message structure, we first filter out the messages with long payload. We calculate the mean and variance of the message lengths. Then, we select the message whose length is greater than the mean and the square of the difference between the length and the mean is greater than the variance. Some message payloads are lengthy without any sub-message structure, e.g., Modbus/TCP Read Multiple Registers. The length of each register is two bytes, so the payload is completely random. We set an autocorrelation function (ACF) threshold to filter these messages. Sometimes, several different sub-message formats are mixed in one message payload. This situation is hard to analyze and will also be filtered if it cannot reach the ACF threshold.

In some protocol messages, the sub-messages may just be a part of the payload, e.g., C37.118. Sometimes, the modified entropy in the previous step may fail to separate the payload. Both these two situations can be solved by setting a sliding window and calculating the ACF value of the part in the window. Fig. 4 is an example of dealing with message m_9. To illustrate that the sub-messages may be a part of the payload, we added 10 random numbers at the end of message m_9. We use the heating map to show the ACF value for sliding windows in all lengths and positions. The horizontal and vertical axes correspond to the left and right boundaries of the sliding window. The true sub-messages position is [33:81]. We can find the possible sub-messages part with maximum length by searching the maximum product of the window size and the ACF value within a confidence interval (99% of the maximum ACF value), written as below:

$$\begin{aligned}(b_x, b_y) &= \arg\max\ (b_y - b_x) * ACF(b_x, b_y) \\ &s.t.\ ACF(b_x, b_y) > 99\% * \max ACF\end{aligned} \tag{3}$$

3.4. Sub-messages extraction

For the long payload composing of several sub-messages, we propose a novel template iteration approach to extract the sub-messages. We adopt the position corresponding to the peak value in the autocorrelation function (ACF) of the payload as the first sub-message boundary and choose the first sub-message as the initial template. Then, based on the previous pairwise sequences alignment algorithm [25], we design a dynamic programming (DP) algorithm to extract each sub-message. Finally, we employ an adapted progressive multiple sequence alignment (PMSA) algorithm [41] to generate a new template. After the difference between the new and old templates is less than a threshold, the iteration stops. So, we can get the aligned sub-messages from a single message payload. The details of this step are illustrated in Section 4.

3.5. Sub-messages format inference

So far, we get a list of aligned sub-messages, including several gaps obtained internally after completion. So, the work in this part is similar to the step of protocol header format inference. When there is a gap in the alignment, the location corresponding to the gap is identified as a part of the sub-message payload. For sub-message payload parts, we merge the adjacent ones since they are numerically random, while for sub-message header parts, we need to analyze further and divide them into different fields.

To compare with the previous method and verify the effectiveness of SEIP, it is necessary to calculate the accuracy of overall inference results, including inferring protocol header format and inferring sub-message format.

4. Sub-messages extraction

This section introduces the details of the sub-messages extraction algorithm. To extract sub-messages from a single message payload after separating the protocol header and payload, we design a template iteration approach, as shown in Fig. 2b.

4.1. Template initialization

We assume that the structures of sub-messages in a single message are similar, which means that all sub-messages are composed of several fields. The length and value of the corresponding fields can be different but similar. Therefore, we can find a template to match each sub-message. For template initialization, we extract the first sub-message in the payload as the initial template.

We adopt the autocorrelation function (ACF) [64] to extract the first sub-message from a single message payload. ACF is widely used in signal internal periodicity extraction. It uses Pearson correlation [62] at signal's different time points to reveal the correlation between time points. Since the sub-messages are similar and continuous, they are periodic in general so that ACF can be applied.

Let $B = (b_1, b_2, b_3, \dots, b_n)$ represents the message's sequential bytes and $\bar{b}$ is the average value of B. So, the message autocorrelation function ACF at periodic point k is:

$$ACF_k = \frac{\sum_{i=1}^{n-k}(b_{i+k} - \bar{b})(b_i - \bar{b})}{\sum_{i=1}^{n}(b_i - \bar{b})^2} \tag{4}$$

The number of items in the numerator with a small k is more than that with a large k. When the numerator is positive, the ACF value at a small k is often larger than that at a large k because more items are accumulated. For some long sub-messages, the ACF is higher in some short periods, i.e., with a small k. So we need to set a threshold θ_{acf} as the minimum sub-message length for k. The peak value in ACF represents the highest correlation. So, we choose the corresponding position of the peak value as the boundary position of the initial template t:

$$\begin{aligned} t &= (b_1, b_2, b_3 \dots b_{k^*}) \\ k^* &= \arg\max_k \ ACF_k \quad s.t. \ \ k > \theta_{acf} \end{aligned} \tag{5}$$

Fig. 5 shows 4 examples of the ACF results, we choose the index of the maximum value as the initial template length and the shadow part in ACF is the confidence interval.

If each sub-message is of equal length, there will be a peak at the boundary of each sub-message ideally, and ACF can directly extract all sub-messages, such as message m_9 (each sub-message boundary ($k = 4, 8, 12, 16 \dots$) gets a peak in ACF), shown in Fig. 5b. However, due to the different lengths of sub-messages, the positions of these peaks may shift or even disappear. Therefore, the accuracy of ACF for the first sub-message boundary is higher than the other sub-message boundaries.

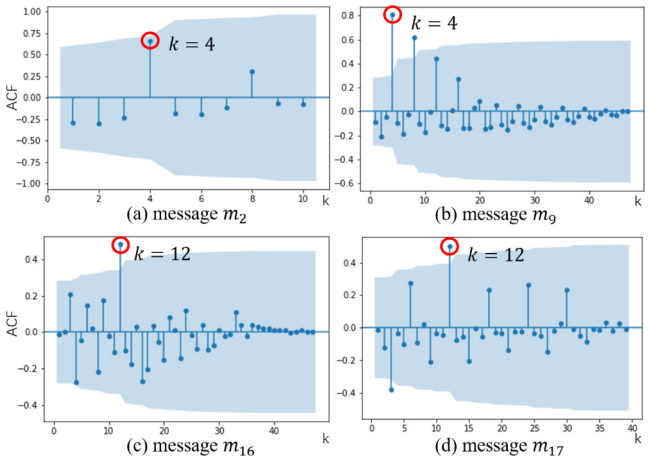


Fig. 5. The ACF results on the payload of message m_2, m_9, m_{16} and m_{17}.

The accuracy of the initial template is essential for subsequent inference because two or more sub-messages can also combine into one template, which leads to a local optimum. So the threshold value should not exceed the real first sub-message boundary. Otherwise, it is likely to get to the second or later sub-message boundary when choosing the peak value, resulting in a lengthy template. In this work, we use the extracted first sub-message as the template and design the following algorithm to extract other sub-messages.

In real applications, we use a dictionary to record all the historical sub-message templates. When analyzing the next new message, a historically similar template can replace the initial template generated by ACF.

4.2. Segmentation based on template

To extract continuous sub-messages according to a fixed template, we design a dynamic programming (DP) algorithm based on existing static alignment algorithms.

The Needleman–Wunsch (NW) [25] algorithm is a typical DP algorithm for pairwise sequences alignment. It needs to set a Penalty matrix $P_{[b_i,b_j]}$ and a gap penalty p_g in advance and update a score matrix $\mathrm{NW}(b_i, b_j)$ between the pairwise sequences using the following equation:

$$\mathrm{NW}^*(b_i, b_j) = \max \begin{cases} \mathrm{NW}(b_{i-1}, b_{j-1}) + P_{[b_i,b_j]} \\ \mathrm{NW}(b_{i-1}, b_j) + p_g \\ \mathrm{NW}(b_i, b_{j-1}) + p_g \end{cases} \tag{6}$$

The initial NW uses a Penalty matrix $P_{[b_i,b_j]} = \delta(|b_i - b_j|)$. It equals to 1 only when $b_i = b_j$ and the others are 0. The gap penalty is 0 so that an infinite number of gaps can be added in the sequence alignment. We adjust NW alignment algorithm to a fuzzy one, as shown in Fig. 6. The Penalty matrix is calculated by a function $f(|b_i - b_j|)$ (the function is user-defined and can be linear, quadratic, or others) and the gap penalty is negative to control the gap quantity. The fuzzy alignment algorithm enables similar parts of the payload to be aligned.

The NW algorithm uses gap completion making itself tolerant of high-entropy data fields, so we choose it as the basic algorithm for alignment. But instead of just searching for a single maximum matching score of the pairwise messages, we define the maximization of the cumulative matching score as the objective function as follows.

Let $S = \{s_1, s_2, s_3, \dots, s_m\}$ denotes a non-overlapping segmentation of the message's sequential bytes B. Each s_i represents a sub-message inferred in the intermediate process and consists of $|s_i|$ bytes, while $|S|$ represents the number of segments. $b_{|s_i|}$ with $1 \le i < |S|$ is the right boundary of inferred sub-message s_i. We set t as a fixed template and $|t|$ is the length of the template in bytes, so $b_{|t|}$ is the right boundary of template t. We define the cumulative matching score of S as below:

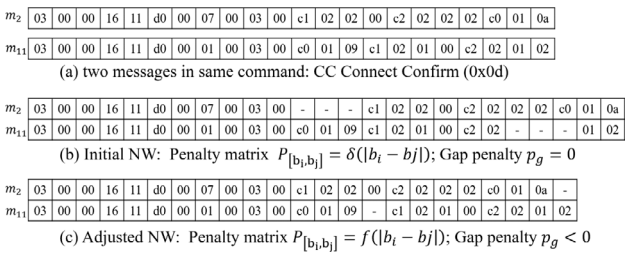


Fig. 6. An example of m_2 and m_{11} using adjusted NW alignment algorithm.

Definition 1. Cumulative matching score of segments S is

$$\mathrm{Cum}(S) = \max_{t} \sum_{s_i \in S} \mathrm{NW}(b_{|t|}, b_{|s_i|}) \tag{7}$$

$\mathrm{NW}(b_{|t|}, b_{|s_i|})$ is the value at the bottom right of the score matrix in NW algorithm for template t with segment s_i. So, we can get the best segmentation S^* of this template t as follows.

$$S^* = \arg\max_{S} \ \mathrm{Cum}(S) \tag{8}$$

We apply a DP algorithm to solve this problem. During the calculation of NW, a score matrix $\mathrm{NW}(b_i, b_j)$ is generated and updated. If we directly use the NW algorithm for the template with the message, we can find that each value in the last column represents the final score of a segment matching to the template. The segment starts from the first byte of the message and ends at the byte corresponding to the value. So we can calculate all score matrices in NW for the template with all starts of the message. The values of the last columns in the score matrices are the final scores of all segments.

Then, we choose the last columns of all matrices to form a new upper triangular matrix M. In the new matrix, once a byte is selected, all bytes before this byte in this row are selected as a segment. The value of the byte is the final score of the segment matching the template.

To calculate the maximum cumulative matching score, we update the values in each row by adding the maximum value in the previous column from top to bottom, as shown below:

$$M^*(i,j) = M(i,j) + \max M(0 : i-1, i-1) \quad i,j > 0 \tag{9}$$

Finally, we extract the sub-messages from bottom to top. We choose the maximum value in the last column of the matrix. The row with the maximum value is the last sub-message inferred. Then we can find the maximum value in the previous column and infer all sub-messages in turn.

Fig. 7 is an example using the payload of message m_2 to demonstrate the entire inference process. The message payload is (0xc1, 0x02, 0x02, 0x00, 0xc2, 0x02, 0x02, 0x02, 0xc0, 0x01, 0x0a). The initial template length is calculated by ACF, as shown in Fig. 5a, which is 4 so that the template is (0xc1, 0x02, 0x02, 0x00). The ideal match result is (0xc1, 0x02, 0x02, 0x00), (0xc2, 0x02, 0x02, 0x02), and (0xc0, 0x01, 0x0a). For the convenience of calculations, we set the NW penalty matrix ($P_{[b_i,b_j]} = f(|b_i - b_j|)$) to the linear function, written as $f(|b_i, b_j|) = -2 * |b_i - b_j|/255 + 1$. We choose $p_g = -0.1$ as the gap penalty. In Fig. 7a, we calculate the NW score matrices of all starts of the message with the template. The values in the last columns are the final scores of the segments in the message matching the template. So we use the last columns to form a new upper triangular matrix M, as shown in Fig. 7b. Then we use Eq. (9) to update the matrix. Finally, we choose the maximum value in the columns to extract the sub-messages in turn, as illustrated in Fig. 7c. The bold value corresponds to the maximum value selected in the current column. The orange squares correspond to the selected segments. Finally, we get the same segmentation as the ideal result.

In Fig. 8, we use the initial NW to compare. The final segmentation is (0xc1, 0x02, 0x02, 0x00), (0xc2, 0x02), (0x02, 0x02, 0xc0, 0x01, 0x0a). Because it is an accurate alignment algorithm, only three 0x02 in the subsequent sequence can be aligned with the initial template (0xc1, 0x02, 0x02, 0x00) so that the segmentation result is not ideal.

The Penalty matrix and gap penalty play significant roles in the Segmentation step. Fig. 9 shows an example that due to the gap penalty being too small (−0.1) it did not get the ideal segmentation until the gap penalty reached −0.3.

4.3. Update template based on segments

After obtaining the segmentation S^*, we need to update the template to maximize the cumulative alignment score of S^*, which is written as $t^* = \arg\max_t \ \mathrm{Cum}(S^*)$, where t^* is an updated new template. This issue is equivalent to finding the optimal multiple sequence alignment result of S^*. We apply the progressive multiple sequence alignment (PMSA) algorithm to solve this problem. We can rewrite t^* as follows:

$$t^* = \arg\max_{t} \sum_{b_i \in \mathrm{PMSA}(S^*)} P_{[b_i,t]} \tag{10}$$

$\mathrm{PMSA}(S^*)$ is the result of multiple sequence alignment in segments S^*. We adjust the PMSA algorithm to make it more suitable for inferring the new template from the segments. The adjusted PMSA algorithm contains three basic steps: *Alignment*, *Revert*, and *Merge*.

- *Alignment*: NW algorithm is used to do the pairwise alignment. The input is two messages, and the output is the corresponding aligned two messages with several gaps, as shown in Fig. 6.
- *Revert*: If a message in *Alignment* step is formed by the *Merge* step of several messages, it will revert to these messages and update them by adding aligned gaps in the *Alignment* step.
- *Merge*: Several aligned messages are merged into one message by calculating Eq. (10). There are many gaps in the aligned messages. If the number of gaps $|g_i|$ in a specific position i is more than a threshold θ_g, the position i will be discarded.

The gap number threshold θ_g determines the direction of template updating. If θ_g is too large, i.e., the template can tolerate more gaps, so the new template is more extended than the original, while if θ_g is small, the new template is shorter. We use the idea of agglomerative cluster [42] to determine the order of message alignment. Therefore, the whole process is to generate a distance matrix between messages and then align the messages step by step from bottom to top. The alignment process consists of the above three basic steps.

In Fig. 10, we use the outcome of the example in Fig. 8 to explain the template updating process. We first align (0xc1, 0x02, 0x02, 0x00) and (0xc2, 0x02). Since the two segments have no ancestors, the new segment is generated according to the alignment result. Here we consider that when the number of gaps in a position $|g_i|$ is larger than half quantity of the segments θ_g, the new segment discards the position. So 0x02 and 0x00 are still in the new segment. The penalty matrix $P_{[b_i,b_j]}$ is equal to $-2 * |b_i - b_j|/255 + 1$. The new segment is generated by maximizing the cumulative sum of $P_{[b_i,t]}$, as illustrated in Eq. (10). Then, after the next pairwise alignment, the ancestor segments exist, so they need to be updated. They insert the gaps at the position as the aligned segment. The New Segment 2 is generated by merging all ancestor segments. The 0xc0 and 0x0a are discarded since the quantity of the gaps in their positions are larger than the threshold. Finally, we can get a final segment, and it is the new template for iteration.

When the similarity between the new and the previous template is less than the threshold θ_t, the iteration stops, and we get the final segmentation result. The whole algorithm is shown in Algorithm 1.

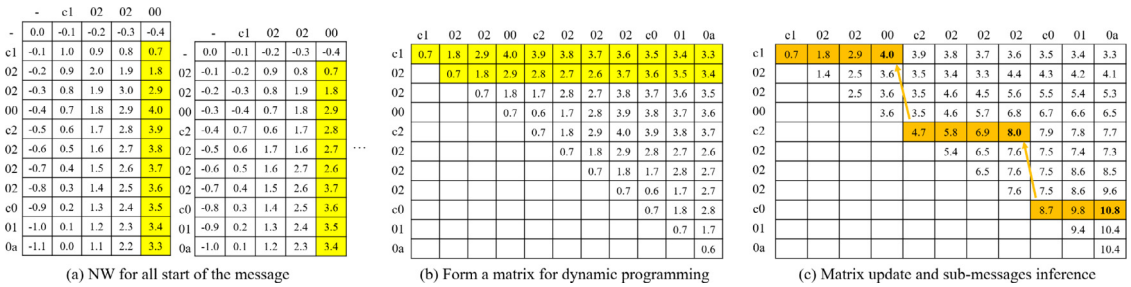


Fig. 7. An example using the payload of the message m_2 to verify the Sub-messages Extraction process based on a fixed template inferred by the Template Initialization step. The Penalty matrix $P_{[b_i,b_j]} = -2 * |b_i - b_j|/255 + 1$ and the Gap penalty $p_g = -0.1$.

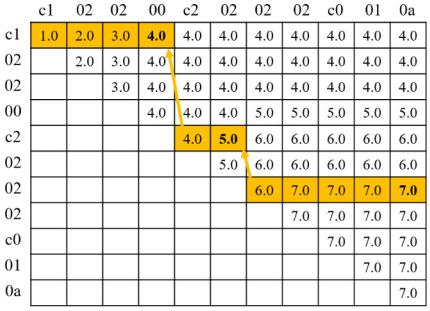


Fig. 8. Same example of Sub-messages Extraction process using initial NW. The Penalty matrix $P_{[b_i,b_j]} = \delta(|b_i - b_j|)$ and the Gap penalty $p_g = 0$.

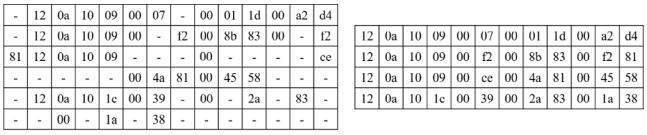


Fig. 9. The influence of different parameters in NW on DP results, taking message m_{16} as an example. The Penalty matrix $P_{[b_i,b_j]} = -2 * |b_i - b_j|/255 + 1$.

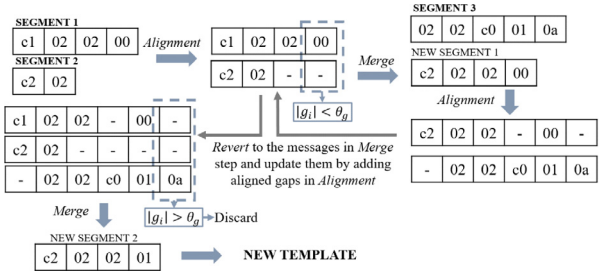


Fig. 10. An example of the Template Update process using the result of Fig. 8 and the Penalty matrix $P_{[b_i,b_j]} = -2 * |b_i - b_j|/255 + 1$.

Table 2
Compare SAMS with FMS.

Boundary list	$\sum\|\delta\|$	FMS	ω^*	$\sum\|\delta_r^*\|$	SAMS
real : [0,8,15,26,40]	–	–	–	–	–
infer 1:[0,8,16,26,40]	1	0.926	0	1	0.926
infer 2:[0,7,16,26,40]	2	0.853	0	2	0.853
infer 3:[0,9,16,27,40]	3	0.779	1	0	0.905
infer 4:[0,6,13,24,40]	6	0.368	−2	0	0.819

Algorithm 1 Sub-messages Extraction Algorithm

Require: Payload sequence B, threshold θ_{acf}, θ_g and θ_t
Ensure: Segmentation S
Initialize template t^0 =ACF(B, θ_{acf})
iteration $l = 0$
repeat
  S^l =Segmentation(B, t^l)
  t^{l+1} =TemplateUpdate(S^l)
until $\Delta t < \theta_t$
return S^l
function ACF(B, θ_{acf})
  $k^* = \arg\max_k \frac{\sum_{i=1}^{n-k}(b_{i+k}-\bar{b})(b_i-\bar{b})}{\sum_{i=1}^{n}(b_i-\bar{b})^2} \quad k > \theta_{acf}$
  return $t = (b_1, b_2, \cdots, b_{k^*})$
end function
function Segmentation(B, t)
  $S^* = \arg\max_S \sum_{s_i \in S} \mathrm{NW}(b_{|t|}, b_{|s_i|})$
  return S^*
end function
function TemplateUpdate(S, θ_g)
  $t^* = \arg\max_t \sum_{b_i \in \mathrm{PMSA}(S)} P_{[b_i,t]} \quad s.t.\ |g_i| < \theta_g$
  return t^*
end function

5. Evaluation criterion

Ref. [53] proposes an evaluation criterion for message format inference called Format Match Score (FMS). Due to the offset alignment of the sub-messages matching, we modify FMS by adding an offset alignment item called Sub-message Alignment Matching Score (SAMS).

As defined above, $|S|$ is the number of segments in segmentation S, which is also the number of inferred sub-messages. So, $|s_i|$ is the length of sub-message s_i. We set $|\hat{s}|$ as the average length of all sub-messages in S. Then, we define $|R|$ as the number of real sub-messages. $R = \{r_1, r_2, r_3 ...\}$ is the set of real sub-messages, where r_k with $1 \le k < |R|$ is a real sub-message with length of $|r_k|$ bytes. $b_{|r_k|}$ is the right boundary of real sub-message r_k. We use δ_r as the distance of a real boundary $b_{|r_k|}$ to the nearest sum of inferred boundary $b_{|s_i|}$ and a fixed offset ω^*, as shown below.

$$\begin{aligned} &\delta_r^* = \min\{|b_{|s_i|} + \omega^* - b_{|r_k|}|\} \\ &s.t.\ \ b_{|r_k|} - \frac{1}{2}|r_k| \le b_{|s_i|} + \omega^* < b_{|r_k|} + \frac{1}{2}|r_{k+1}| \end{aligned} \tag{11}$$

where the constraint means that the sum of $b_{|s_i|}$ and ω^* (the inferred result corresponding to $b_{|r_k|}$) should near to $b_{|r_k|}$.

The fixed offset ω^* is an integer calculated by searching the minimum sum of δ_r in the scope of $\frac{1}{2}|\hat{s}|$:

$$\omega^* = \arg\min_{\omega} \sum \delta_r \quad s.t. \quad -\frac{1}{2}|\hat{s}| \le \omega \le \frac{1}{2}|\hat{s}| \tag{12}$$

Table 3
Commands data composed by sub-messages (Size:1000).

Protocol	Command	$\overline{L_s}$ [1]	$\overline{N_s}$ [2]
Modbus-TCP	Read File Record (0 × 14) req	7.00	8.46
	Read File Record (0 × 14) rsp	15.00	8.46
	Write File Record (0 × 15) req/rsp	19.98	8.50
S7-Communwith ProtocolId (0 × 32)	Job (0 × 01) Read Var (0 × 04)	12.00	11.53
	Ack (0 × 03) Read Var (0 × 04)	6.89	18.78
	Req (0 × 11) Func (0 × 2_0 × 01)	12.00	11.37
	Rsp (0 × 12) Func (0 × 2_0 × 01)	8.02	11.67
Ethernet/IP-CIPwith Multi-Service Packet(0 × 0a)	Get Attribute (0 × 03) Req (0 × 0)	20.93	9.94
	Get Attribute (0 × 03) Rsp (0 × 1)	24.04	7.97
	Service (0 × 4c) Req (0 × 0)	17.97	11.52
	Service (0 × 4c) Rsp (0 × 1)	9.99	11.67

[1] $\overline{L_s}$ is the average length of sub-messages in bytes.
[2] $\overline{N_s}$ is the average number of sub-messages.

Due to the addition of the offset item, it is necessary to add an offset penalty to the coefficient. The final form of SAMS is shown below:

$$\mathbf{SAMS} = \underbrace{\exp\left(-\left|\frac{\omega^*}{|\hat{s}|}\right|\right)}_{\text{Offset penalty}} \cdot \underbrace{\exp\left(-\left(\frac{|R|-|S|}{|R|-1}\right)^2\right)}_{\text{Segmentation penalty}} \cdot \underbrace{\frac{1}{|R|-1}\sum_{1\le k<|R|}\exp\left(-\left(\frac{\delta_r^*}{2}\right)^2\right)}_{\text{Match gain}} \tag{13}$$

The Offset penalty penalizes the gain of the added offset in the Match gain, which makes the larger offset in the sequence, the lower the SAMS score. The Segmentation penalty penalizes the number difference of sub-messages, and the Match gain accumulates the matching score of sub-message boundary speculation.

Table 2 illustrates an example to explain the difference between SAMS and FMS. The first column is the real and several inferred sub-message boundaries. The second and third columns use FMS to evaluate the quality of speculation, while the fourth to sixth columns illustrate SAMS with the offsets. Due to the offset alignment in sub-message speculation, infer 3 should be better than infer 2, which cannot be reflected in FMS. However, in SAMS, when the offset term is 0, SAMS and FMS are identical, while when the offset term is not 0, SAMS can make the conjectural sequence with offset alignment get an appropriate score.

6. Evaluation

We evaluate SEIP on three industrial control protocols: Modbus-TCP, S7-Communication, and Ethernet/IP-CIP. To compare and verify the effectiveness of the proposed sub-message extraction method, we adopt the analysis results of Wireshark to generate enough similar messages by randomizing the number of sub-messages, the value of sub-message header's fields, and the value and length of sub-message payload's fields. We extract some special commands, which are composed of several sub-messages. By expanding or sampling data, we set the number of all commands' sizes to 1000, as shown in Table 3.

6.1. Parameter selection

We first verify the effect of different parameters on the results. We choose the command CIP_0a_03 as an example shown in Table 4. The left column (Template) compares different strategies and algorithms in the Template Initialization step. The intermediate columns compare different parameter combinations in the Segmentation step including the Penalty matrix functions (Penalty) and the Gap penalty (Gap). The right columns are the results of different combinations including the average per message SAMS score (SAMS), offset $\overline{|\omega^*|}$ (byte) (Offset), iterations (Iter) and time (second) (Time). SEIP performs different iterations according to different messages and each iteration needs to run several dynamic programming algorithms. We argue that PRE based on communication messages analysis can be totally offline and one-time work so that the time consumption is not critical.

The ACF-M row uses the maximum value in ACF while the ACF-C row applies the first value outside the confidence interval. The FFT row is the maximum value in fast Fourier transform. Due to the impact of short period, the first few items of ACF or FFT tend to be too large. We need to set a minimum length threshold of a sub-message. However, we can only delete the first few excessive values since we do not have any prior knowledge of the sub-message length in a command. So, we choose $\theta_{acf} = 3$ as the threshold. It means that the shortest sub-message length is 3 and the Fix row uses 3 as the initial template length.

According to the example shown in Fig. 9, it is vital to find a suitable set of a penalty matrix and a gap penalty for NW. A lower gap penalty leads to the extraction of too many segments, while a higher gap penalty results in the lengths of sub-messages converging towards the template length, which cannot tolerate the difference in length among sub-messages. So we also compare different functions in calculating penalty matrix $P_{[b_i,b_j]}$ and different values of the gap penalty. All the functions are mapped to [−1,1] and the scope of $x = |b_i - b_j|$ is [0,255]. The best function in calculating penalty matrix is $tanh'(x)$ and the best corresponding gap penalty is −0.7, which is slightly smaller than the value of $P_{[0x00,0x10]} = -0.69$. When the difference between two bytes is larger than 16, SEIP will choose to add a gap rather than align these two bytes.

In the template update step, θ_g determines the tolerance of the new template to the gaps. Due to the lack of knowledge about whether the template current state is too long or too short, we can only set θ_g as half of the current inferred number of sub-messages.

Finally, we adopt $\theta_t = 0.1$ as the template dissimilarity threshold between the new and old templates. When the dissimilarity is less than θ_t, the iteration stops, and we get the final results of sub-messages extraction: a list of aligned sub-messages.

6.2. Results of sub-message boundary inference

The SAMS results on the sub-messages boundaries of several commands are shown in Table 5. We compare the results of Netzob [38] and NEMESYS [53] in terms of SAMS. For most commands, they cannot find the sub-message structure because their algorithm goals do not focus on extracting the continuous similarity of the internal structure of a message. However, NEMESYS gets a relatively good score in command S7_0x03_0x04 Read Var Ack_Data. The sub-message of command S7_0x03_0x04 has 4 parts: Return code, Transport size, Length and Data, e.g., m_{15} and m_{17} in Fig. 1. The Return code is normally Success (0xff). The Transport size is variable, e.g., BIT (0x03), BYTE (0x04) and INTEGER (0x05). NEMESYS uses the difference between adjacent bytes to determine the field boundary. For m_{17}, it gets a bad cutting result since the Data part is totally random. But if the sub-messages are the

Table 4
Parameter selection.

Template	Penalty	Gap	SAMS	Offset	Iter	Time
ACF-M	tanh'(x)	−0.7	**0.973**	**0.07**	1.48	**59.57**
ACF-C	tanh'(x)	−0.7	0.963	0.07	3.11	81.91
FFT	tanh'(x)	−0.7	0.960	0.07	2.47	73.95
Fix	tanh'(x)	−0.7	0.949	0.09	4.77	104.44
ACF-M	tanh'(x)	−0.5	0.961	0.12	1.36	108.69
ACF-M	tanh'(x)	−0.6	0.972	0.11	1.44	60.04
ACF-M	tanh'(x)	−0.8	0.972	0.08	1.56	61.65
ACF-M	tanh'(x)	−0.9	0.969	0.09	1.44	111.94
ACF-M	linear(x)	−0.7	0.770	0.71	**0.84**	78.63
ACF-M	inverse(x)	−0.7	0.316	5.31	1.54	87.42
ACF-M	quadratic(x)	−0.7	0.851	0.36	0.96	91.36
ACF-M	tanh(x)	−0.7	0.407	4.29	1.40	73.97
ACF-M	sigmoid(x)	−0.7	0.430	3.96	1.42	74.98
ACF-M	sigmoid'(x)	−0.7	0.970	0.08	1.09	100.84

Table 5
The SAMS results of several commands.

COMMAND	SEIP			Netzob	NEMESYS
	SAMS	Iter	Offset	SAMS	SAMS
Modbus_0 × 14_req	**0.997**	0.16	0.00	0.000	0.165
Modbus_0 × 14_resp	**0.753**	4.30	1.51	0.009	0.002
Modbus_0 × 15_req	**0.883**	2.09	0.92	0.000	0.000
S7_0 × 01_0 × 04	**0.958**	3.32	0.24	0.032	0.002
S7_0 × 03_0 × 04	**0.837**	4.23	0.29	0.000	0.397
S7_0 × 11_0 × 2_0 × 01	**0.970**	3.33	0.17	0.037	0.000
S7_0 × 12_0 × 2_0 × 01	**0.743**	1.98	0.81	0.010	0.128
CIP_0 × 03_0 × 0	**0.973**	1.48	0.07	0.000	0.000
CIP_0 × 03_0 × 1	**0.885**	1.32	0.17	0.000	0.000
CIP_0 × 4c_0 × 0	**0.952**	1.79	0.08	0.000	0.000
CIP_0 × 4c_0 × 1	**0.783**	1.19	0.54	0.026	0.089

payload similar to m_{15}, NEMESYS can cut well because most bytes in the Data part are near to zero and far from the header of next sub-message (0xff).

6.3. Full message format inference

All experiments were conducted on a 20-cores CPU server (Intel(R) Xeon(R) CPU E7-4820 v4 @2.00 GHz) with 128G main memory. Table 6 shows the execution time (min) and memory (MB) consumption on the datasets. SEIP consumes more time than the available Netzob and NEMESYS. The core step of SEIP, sub-messages extraction (SubMsg Extra), leads to large time and space complexity, which could vary a lot for different protocol commands. In each iteration, the template needs to be compared with the payload. So, the time complexity is $O(\overline{L_s}^3 \cdot \overline{N_s}^2 \cdot \hat{I} \cdot N)$, where $\overline{L_s}$ is the average length of sub-messages, $\overline{N_s}$ is the average number of sub-messages, $\hat{I}$ is the number of iterations plus 1, and N is the number of messages. For well-formatted commands, e.g., Modbus_0x14_req, the iterations number is close to 0, so that the $\hat{I}$ is close to 1. In addition, using the historical sub-message templates can reduce the number of iterations. Table 6 uses historical templates for acceleration. We argue that the time and memory consumption is acceptable as PRE is an offline technique and hence one-time effort.

To compare SEIP with Netzob and NEMESYS, we infer the total message format based on the result of sub-messages extraction, including the protocol header inference, as shown in Table 7. The SAMS and FMS are equivalent when there is no offset.

In general, the score of full message inference (S-FullMsg) should be close to the score of sub-message boundary inference (S-SubMsg). However, a higher S-SubMsg may result in a lower S-FullMsg, e.g., S7_0x01_0x04 and CIP_0x4c_0x0, as shown in Table 8. The italics represent the payloads. It is caused by two different reasons. For S7 protocol, there are too many continuous fixed fields, these fields are regarded as a whole in the analysis, but the reality is that each byte or word has its meaning, which eventually leads to a low S-FullMsg. For CIP protocol, since we use gaps to distinguish the sub-message header from the payload, the accurate alignment of the sub-message payload lets part of it be mistakenly considered as the sub-message header part, which also results in a poor S-FullMsg. In contrast, a low S-SubMsg may also lead to a high S-FullMsg, e.g., Modbus_0x14_resp. The total message format inference eliminates the influence of the same offset generated in the process of sub-messages extraction so that the S-FullMsg is higher than S-SubMsg.

Netzob achieved relatively good performance in the request commands of Modbus, e.g., Modbus_0x14_req. Because in this command, the characteristics of sub-message structure are obvious, the data part is short and the length is fixed. So, it is suitable for initial NW alignment in Netzob. NEMESYS obtains better cutting results in the response commands of CIP, e.g., CIP_0x4c_0x1. The response sub-messages of these commands are mainly characterized by high-value short headers and low-value long payloads. Some zero-padding fields are located at the boundary between the header and the payload. These fields allow NEMESYS to precisely locate the boundary.

The results of header format inference and sub-message format inference are shown in Table 8. Each row shows the inference results under different n in n-grams, corresponding to Fig. 3. The final inference results can be filtered by setting some strategies, e.g., merge adjacent fields of the same type. Many fields cannot be inferred because they do not have statistical properties. The real meaning of the field cannot be obtained only by analyzing the traffic trace. Our work is only to give a possible inference result to assist the operators in protocol reverse engineering.

6.4. Impact of protocol header inference

The accuracy of protocol header cutting has an important impact on the subsequent reasoning. We study the effect of header cutting on sub-messages extraction, as shown in Table 9. We choose CIP_0x03_0x0 as an example to compare the impact of different header cutting errors. Its sub-messages average length is 20.93. We adopt header cutting errors of ±5, ±10, ±15, ±20, ±25, and ±30 bytes. The '+ ' represents more header cutting cases and the '-' represents fewer header cutting cases. We calculate the score of the sub-messages extraction (S-SubMsg) and the score of the full message inference (S-FullMsg).

The first line is the result of using ACF and actual header cutting. The following lines are the results of using different protocol header cutting errors in SEIP. According to Table 9, when the number of header cutting error bytes is close to the average length of the sub-message, the S-SubMsg will enter the recovery stage, especially for more header cutting cases. The fewer header cutting cases bring noise data to the payload and the more header cutting cases reduce sub-message segments. For ACF, noise data may destroy the internal periodicity of the payload. On the contrary, the reduction of payload causes periodic bias. So, the overall results of more header cutting cases are better than that of fewer header cutting cases. Besides, the S-FullMsg is also affected together with the S-SubMsg. But there are still some keywords aligned in the segments with errors in the sub-message structure so that the change of S-FullMsg is smoother than that of S-SubMsg.

6.5. Unknown protocols sub-messages extraction

The unknown protocols do not have the ground truth to evaluate so that we can only show the sub-messages extraction results and the inferred results. The true meaning of the fields in the unknown protocol cannot be determined. We can only infer the statistical meaning of some fields with statistical characteristics. It needs to be determined by combining additional information, e.g., the operations from operator and the display information on HMI.

S7 Communication protocol with the protocol Id (0x72) is a new series for S7-1200 PLC and HMI. The protocol specification is still private. Melsec protocol belongs to Mitsubishi company [65]. Its products

Table 6
Execution time and memory consumption.

COMMAND	Netzob		NEMESYS		SEIP					
					Header Infer		SubMsg Extra		SubMsg Infer	
	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory
Modbus_0 × 14_req	0.164	11.973	0.202	303.633	0.013	13.621	6.134	74.688	1.305	10.430
Modbus_0 × 14_resp	0.111	37.879	0.259	381.121	0.020	23.137	102.612	177.025	22.368	1.801
Modbus_0 × 15_req	0.725	48.145	0.341	513.531	0.018	19.598	203.272	180.938	45.074	3.793
S7_0 × 01_0 × 04	0.145	38.832	0.332	496.984	0.037	27.531	166.221	190.469	16.723	6.644
S7_0 × 03_0 × 04	18.238	38.211	0.277	390.648	0.028	27.617	51.362	162.813	12.260	2.633
S7_0 × 11_0 × 2_0 × 01	0.159	40.766	0.353	525.191	0.097	38.926	195.513	172.344	16.570	4.273
S7_0 × 12_0 × 2_0 × 01	0.149	35.988	0.297	422.551	0.091	37.863	68.933	164.375	10.704	1.180
CIP_0 × 03_0 × 0	35.058	86.133	0.458	614.926	0.133	62.391	241.179	189.375	33.691	1.102
CIP_0 × 03_0 × 1	973.033	77.094	0.467	627.730	0.139	60.453	187.299	186.563	42.156	2.480
CIP_0 × 4c_0 × 0	2.255	80.301	0.514	672.672	0.134	62.965	102.691	181.250	33.214	5.094
CIP_0 × 4c_0 × 1	1.528	48.520	0.335	479.098	0.131	53.660	23.593	96.876	6.148	1.488

Table 7
The FMS results of the commands.

COMMAND	FMS (SAMS)		
	SEIP	Netzob	NEMESYS
Modbus_0 × 14_req	**0.893**	0.600	0.457
Modbus_0 × 14_resp	**0.890**	0.405	0.387
Modbus_0 × 15_req	**0.875**	0.587	0.467
S7_0 × 01_0 × 04	**0.572**	0.327	0.321
S7_0 × 03_0 × 04	**0.816**	0.010	0.236
S7_0 × 11_0 × 2_0 × 01	**0.676**	0.371	0.341
S7_0 × 12_0 × 2_0 × 01	**0.785**	0.358	0.339
CIP_0 × 03_0 × 0	**0.732**	0.441	0.425
CIP_0 × 03_0 × 1	**0.667**	0.048	0.598
CIP_0 × 4c_0 × 0	**0.528**	0.144	0.379
CIP_0 × 4c_0 × 1	**0.691**	0.332	0.603

are divided into different series. Although the protocol manuals of each series are public, Wireshark does not include them so that we regard them as unknown.

We use SEIP to extract the sub-messages in them. Since we do not know the true meaning of each field, we can only display the sub-message result examples, as shown in the Tables 10, 11. The Msg num row is the total amount of the message data collected from open data [58] and the number of messages identified as having sub-messages (within an ACF threshold equals 0.6). The Template row shows the final templates inferred by SEIP and the num row is the number of messages corresponding to them. The Example row displays the examples in each template including the id, length, sliding window and its corresponding ACF value of the message example.

The extracted sub-messages can have some offsets. For instance, the third template in Table 11 appears to have an offset of one byte. Since we do not know the true meaning of the field [e1 07 0c 1a 06 1c 1b], the semantic meaning of {01, 02, 03, 04} seems to be the keywords of sub-message sequence number, and the following payloads in real data are the keywords {05, 06, ...} with other unknown meaning bytes. So, the results of SEIP can provide some guidance for the follow-up analysis.

7. Discussion

7.1. Limitations

IIoT interconnection is mainly divided into two major networks: equipment level and enterprise level for interconnection and joint security defense. For enterprise level networks, a series of safety measures can be used for protection, e.g., OPC UA, a promising unified industry standard protocol, uses encryption, signature, authentication, and other methods to complete in-depth defense layer by layer. However, for equipment level networks, non-encrypted and non-compressed communication is often used because the traditional industrial environment is a closed LAN so that it is considered safe and reliable. So, both encryption and compression affect the real-time performance and are usually not used in equipment level. However, with the development of IIoT, the original internal network needs to be integrated with the external network, which makes the original non-encrypted messages vulnerable to attack. The attacks on the Internet are diverse and difficult to be traced, e.g., buffer overflow, format string and side channel attacks. Researchers need to build an environment based on known protocols to conduct attack and defense simulation experiments, and finally provide experimental guidance for formulating defense strategies. Therefore, it is necessary to reverse analyze unknown private protocols.

For PRE by analyzing traffic trace, the most significant limitation is the requirement of non-encrypted and non-compressed messages. Fortunately, most of the existing industrial private protocols are non-encrypted and non-compressed due to their real-time requirements. The common solution is to obtain the original message information before compression and encryption, but it needs to obtain the memory content information during the execution process of the corresponding software program. Therefore, this does not belong to the category of reverse through traffic trace, but belongs to the way of reverse through the execution trace. Man in the middle attack [66] can add an agent to obtain decompressed or decrypted information when two entities interact, but it also needs to obtain the execution environment of the entity and the topology of the target network. This paper only studies the algorithm based on non-compressed and non-encrypted communication messages.

7.2. Future work

The unknown protocol specification obtained through PRE is the foundation of IIoT system security defense and interconnection. This paper indirectly contributes to the system security and device interconnection. It is an attempt to extract the internal continuous similarity (i.e. sub-message format) of the message under the goal of completely reversing the unknown protocol specification, so as to contribute to the overall protocol format inference results. The PRE work is hard and complex. The traditional way of PRE is manual, e.g., the open source project SAMBA took 12 years to successfully reverse Microsoft's SMB protocol. The PRE work in recent ten years all hope to shorten the time of manual reverse and reduce the complexity through automation. The method proposed in this paper can help researchers obtain the internal characteristics of the message and speculate the format of the message with sub message structure.

It is necessary to use more commands in different protocols to verify the SEIP to further improve our algorithm. Although clustering is not required between protocols, clustering is still needed to separate different commands in the same protocol for industrial control protocols. Future work could be focused on analyzing how to separate different forms of commands from each other. Clustering analysis is based on the similarity between individuals. Because the messages in the same type

Table 8
Comparisons between the inference of SEIP and ground truth.

	Modbus_0 × 14_resp	S7_0 × 01_0 × 04	CIP_0 × 4c_0 × 0
SEIP (Header Infer)	{(0,0):[(‘linear’)];(2,2):[(‘fix’,0)]... (0,1):[(‘linear’)];(2,3):[(‘fix’,0)]... (3,2):[(‘fix’,0)];(4,3):[(‘fix’,0)]... ... (2,5):[(‘len’,1,6),(‘cor’,1,−2,(8,8))]...}	{(0,0):[(‘fix’,3)];(1,1):[(‘fix’,0)]... (0,1):[(‘fix’,768)];(1,2):[(‘freq’,[0,1])]... (1,0):[(‘fix’,3)];(2,1):[(‘freq’,[0,256])]... ... (0,3):[(‘cor’,46.175,1,(14,17)), ...]...}	{(0,0):[(‘fix’,112)];(1,1):[(‘fix’,0)]... (0,1):[(‘fix’,28672)];(1,2):[(‘len’,1,24)]... (1,0):[(‘fix’,112)];(3,2):[(‘len’,1,24)]... ... (8,11):[(‘fix’,0)];(9,12):[(‘fix’,0)]...}[1]
GT (Header Infer)	{(0,1):Transaction Id,(‘linear’); (2,3):Protocol Id,(‘fix’,0); (4,5):Length,(‘len’,1,6); ... (8,8):Byte Count,(‘len’,1,8)}	{(0,0):Version,(‘fix’,3); (1,1):Reserved,(‘fix’,0); (2,3):Length,(‘len’,1,4); ... (18,18):Item Count }	{(1,0):Command,(‘fix’,112); (3,2):Length,(‘len’,1,24); (8,5):Session Handle; ... (54,53):Number of Service }
SEIP (SubMsg Extra)	[*05-* 0601 *23016e* - -, *-0d* 0601 - - - *0a* -, *02-* 0e02 *02007d01df*, ... *07-* 0601 *1e00a500b3*]	[120a10 04 00c2 006283 0014b0, 120a10 08 0049 00a781 007c79, 120a10 05 0001 005d1d 00c6b1, ... 120a10 1d 0028 00fa82 00ab8f]	[4c0220 62 24 6806 0071 00 *74006900730077006500*, 4c0220 61 24 6f04 0071 00 - - *65006400* - - *6500*, 4c0220 6f 24 6e05 006b 00 - - *6700* - - *76006300*, ... 4c0220 68 24 6605 0065 00 *72006d00* - - *75006e00*]
GT (SubMsg Extra)	[05 06 *0123016e*, 0d 06 *010a020e0202007d01df00fe*, 05 06 *00c40224*, ... 07 06 *011e00a500b3*]	[12 0a 10 04 00c2 0062 83 *0014b0*, 12 0a 10 08 0049 00a7 81 *007c79*, 12 0a 10 05 0001 005d 1d *00c6b1*, ... 12 0a 10 1d 0028 00fa 82 *00ab8f*]	[4c 02 2062 2468 *0600710074006900730077006500*, 4c 02 2061 246f *0400710065006400 6500*, 4c 02 206f 246e *05006b00670076006300*, ... 4c 02 2068 2466 *0500650072006d0075006e00*]
SEIP (SubMsg Infer)	{(1,1):[‘freq’, [6]];(2,2):[‘freq’, [1]]; (1,2):[‘freq’, [1537]]; (2,1):[‘freq’, [262]]}	{(0,0):[(‘fix’,18),(‘len’,1,−6)]... (0,1):[(‘fix’,4618),(‘len’,1,−4612)]... ... (0,2):[(‘fix’,1182224), ...]... }	{(0,0):[(‘fix’,76)];(1,1):[(‘fix’,2)];(2,2):[(‘fix’,32)]... (0,1):[(‘fix’,19458)];(1,2):[(‘fix’,544)]... ... (0,2):[(‘fix’,4981280)]... }
GT (SubMsg Infer)	{(0,0):Length,(‘len’,1,1); (1,1):Command,(‘fix’,6)}	{(0,0):Specification,(‘fix’,18); (1,1):Length,(‘len’,1,2); ... (8,8):Area}	{(0,0):Request&Service,(‘fix’,76); (1,1):Path Size; (2,3):Path Class; (4,5):Path Instance}

[1] (pos):[list of inferred results]. (‘fix’, v): the value of field is fixed v; (‘len’, k, b): field * k + b = MsgLen;
(‘cor’, k, b,(pos)): field * k + b = (pos); (‘freq’,[v_1, v_2, ...]): values in the list are frequent; (‘linear’): the values in field are linearly varying.

Table 9
Impact of protocol header cutting errors.

CIP_0 × 03_0 × 0	SEIP				
	S-SubMsg	Iter	Offset	Time	S-FullMsg
ACF	**0.973**	**1.48**	**0.07**	**59.57**	**0.732**
ACF with −5	0.501	2.38	4.67	**170.66**	0.714
ACF with −10	0.451	2.79	8.44	211.38	0.719
ACF with −15	0.663	**1.72**	6.07	187.16	0.743
ACF with −20	**0.700**	2.72	**3.24**	234.13	**0.758**
ACF with −25	0.594	3.86	3.34	285.67	0.742
ACF with −30	0.368	4.74	4.03	333.80	0.653
ACF with +5	0.754	1.60	4.96	148.81	0.791
ACF with +10	0.464	1.61	8.39	135.08	0.744
ACF with +15	0.497	1.95	5.75	129.11	0.745
ACF with +20	**0.757**	1.78	**2.44**	114.65	0.778
ACF with +25	0.755	**1.45**	4.32	106.54	**0.810**
ACF with +30	0.552	1.53	7.06	**103.92**	0.782

can have different numbers of sub-messages, the similarity between these messages may be lower than that with other messages, resulting in poor clustering results. The template generated by SEIP can be applied to represent the payloads with different numbers of sub-messages, so that the similarity between these messages is improved and that with other messages is reduced, e.g., in Fig. 1, the NW similarity [25] (i.e., the number of same bytes in order) between m_{15} and m_9 is 27, while that between the same type m_{15} and m_{17} is only 22. Since m_{15} has a higher similarity with m_9 rather than m_{17}, it is easier to group with m_9. So, the same type of messages m_{15} and m_{17} are clustered into different categories. SEIP can extract sub-messages and generate the corresponding template. If using the template of m_9 to represent its payload, which is only 4 bytes (i.e., [02 6b 22 09]), the similarity between m_{15} and m_9 will be reduced to 21, so that m_{15} can be grouped with m_{17}.

In Section 6.5, we can find the sub-message structure from the communication messages and infer the unknown protocol format, but the real content information of the unknown protocol format still cannot be obtained. It is also an inherent problem in the protocol reverse engineering by analyzing traffic trace. For the actual reverse of industrial control protocol, it is often possible to obtain the commands, transmission contents, display results and other information executed by the tester in the operation. The information will be combined with the reverse results of traffic trace, so as to obtain the real specification of unknown protocols. We take this part as our future work.

8. Conclusion

In this paper, a comprehensive process SEIP for inferring the format of industrial control protocol is proposed and verified with three standard protocols and two unknown protocols under the improved evaluation criterion SAMS. Aiming at the phenomenon that many messages have multiple sub-messages, SEIP proposes a sub-message extraction algorithm based on template iteration for the first time. It uses ACF to initialize the template, designs a DP algorithm to extract sub-messages, and adjusts PMSA method to update the template. Experiments show that the algorithm can effectively extract sub-messages and significantly improve the inference accuracy of the full message format.

The sub-message results obtained by SEIP can be used for subsequent PRE analysis, such as protocol type clustering and state machine generation. This is also the first attempt at payload internal analysis. Previous work did not pay attention to the analysis of payload, ignored the effect of payload on PRE, and always regarded payload as noise. We believe that the payload processing of SEIP can widely improve various protocol reverse tools. Besides, the goal of the algorithm in SEIP is to extract the internal continuity in stream data. We believe that SEIP can be used not only for format inference of different industrial control protocols but also for other situations with similar continuous sequences.

Table 10
Result examples of S7 communication protocol with protocol Id (0x72).

Msg num	Total: 483 Sub-Msg: 114					
Header Infer	{(0,0):[('fix',3)];(1,1):[('fix',0)];(2,2):[('cor',1,0,(9,9))];(3,3):[('cor',1.003,−15.261,(10,10))];(4,4):[('fix',2)];(5,5):[('fix',240)]... (0,1):[('fix',768)];(2,3):[('len',1,0)];(4,5):[('fix',752)];(5,6):[('fix',61568)];(6,7):[('fix',32882)];(8,9):[('freq', [512])]... (1,3):[('len',1,0)];(4,6):[('fix':192640)];(5,7):[('fix':15761522)];(11,13):[('freq', [3211264])];(12,14):[('freq', [5])]... (4,7):[('fix',49315954)];(11,14):[('freq', [822083589])];(12,15):[('freq', [1356])];(13,16):[('freq', [347136])]...}					
Template	[9f 5a 00 03 00 00]	[c0 40 82 3e 00 05 84 80]	[3f 82 3c 00 04 82]	[88 15 00 04 02 02 a3]	[00 81 49 02 88 39 00]	...
num	16	10	9	8	7	...
Example	id: 52 length: 79 window: [33:62] ACF: 0.797 [9f 58 00 03 00 00, 9f 59 00 03 00 00, 9f 5a 00 03 00 00, 9f 5b 00 03 00 01, 9f 5c 00 04 00 -]	id: 3 length: 137 window: [61:83] ACF: 0.647 [81 40 82 3d 00 04 84 80, c0 40 82 3e 00 04 84 80, c0 40 82 3f 00 15 - -]	id: 136 length: 215 window: [108:126] ACF: 0.665 [3a 82 3b 00 04 82, 40 82 3c 00 04 82, 40 82 3d 00 04 84]	id: 260 length: 206 window: [81:113] ACF: 0.776 [87 6a 00 03 00 00 a3, 87 6b 00 09 - 00 a3, 88 10 00 - 02 05 a3, 88 11 00 - 01 01 a3, 88 18 20 04 11 88 80]	id: 385 length: 97 window: [36:55] ACF: 0.647 [00 81 49 02 88 39 00, 00 81 49 02 88 39 01, 00 81 49 02 88 - -]	...
SubMsg Infer	{(0,0):[('fix',159)]... (0,1):[('linear')]... (0,2):[('linear')]... }	{(0,0):[('freq', [192])]... (1,2):[('fix',16514)]... (1,3):[('linear')]... (1,4):[('linear')]...}	{(0,0):[('freq', [64])]... (1,2):[('linear')]... (1,3):[('linear')]... (1,4):[('linear')]...}	{(0,0):[('freq', [135])]... (5,6):[('freq', [163])]...}	{(0,0):[('fix',0)]... (0,1):[('fix',129)]... (0,2):[('fix',33097)]... (0,3):[('fix', ...)]...}	...

Table 11
Result examples of Melsec protocol based on UDP.

Msg num	Total: 2868 Sub-Msg: 729				
Header Infer	{(1,1):[('freq', [1])];(2,2):[('freq', [0])];(3,3):[('freq', [0])];(4,4):[('freq', [0])];(5,5):[('freq', [17])];(6,6):[('freq', [17])];(8,8):[('freq', [0])]... (1,2):[('freq', [256])];(2,3):[('freq', [0])];(3,4):[('freq', [0])];(4,5):[('freq', [17])];(5,6):[('freq', [4369])];(8,9):[('freq', [0])]... (1,3):[('freq', [65536])];(2,4):[('freq', [0])];(3,5):[('freq', [17])];(4,6):[('freq', [4369])];(8,10):[('freq', [0])]... (1,4):[('freq', [16777216])];(2,5):[('freq', [17])];(3,6):[('freq', [4369])];(24,27):[('freq', [335544320])]...}				
Template	[00 56 00 26 00 45 00 51 00 4a 00 64 00 64]	[00 00 41 00 00 00 01 02]	[1b 02 e1 07 0c 1a 06 1c]	[00 00 a2 00 00 00 00 00 00 00 a2 00 00 00 00 00 ff 00 00 00 00 00 00 00]	...
num	106	77	68	45	...
Example	id: 1192 length: 113 windows: [62:111] ACF: 0.900 [00 30 00 24 00 4d 00 45 00 4c 00 50 00 52, 00 4a 00 24 00 5c - - 00 53 00 6f 00 75, 00 72 - - 00 63 00 65 00 49 00 6e 00 66, 00 6f 00 2e 00 43 00 41 00 42 - - 00 -]	id: 62 length: 285 windows: [231:277] ACF: 0.802 [00 00 30 00 00 00 00 02, 00 00 31 00 00 00 00 02, 00 00 42 00 00 00 00 02, 00 00 46 00 00 00 00 01, 00 00 4a 00 00 00 10 00, 00 00 56 00 00 00 - -]	id: 166 length: 1015 windows: [146:178] ACF: 0.748 [00 01 e1 07 0c 1a 06 1c, 1b 02 e1 07 0c 1a 06 1c, 1b 03 e1 07 0c 1a 06 1c, 1b 04 e1 07 0c 1a 06 1c]	id: 668 length: 1019 windows: [58:629] ACF: 0.949 [- 03 a2 00 00 00 00 00 00 00 a2 00 00 00 00 00 ff - 00 00 00 00 00 00 00, 00 00 a2 00 00 00 00 00 00 00 a2 00 00 00 00 00 ff - 00 00 00 00 00 00 00, ... 00 00 a2 00 00 00 00 00 00 00 a2 00 00 00 00 00 ff - 00 00 00 00 00 00 00, 00 00 a2 00 00 00 00 00 00 00 a2 00 00 00 00 00 - 00 00 2b 02 00 00 00 00]	...
SubMsg Infer	{(0,0):[('fix',0)]...}	{(0,0):[('fix',0)]... (0,1):[('fix',0)]... (3,5):[('fix',0)]...}	{(0,0):[('freq', [27])]... (1,2):[('linear')]... (1,3):[('linear')]... (1,4):[('linear')]...}	{(0,0):[('freq', [0])]... (1,2):[('freq', [162])]... (1,3):[('freq', [41472])]... (1,4):[('freq', [10616832])]...}	...

CRediT authorship contribution statement

Yuhuan Liu: Conceptualization, Methodology, Validation, Writing – original draft. **Fengyun Zhang:** Validation, Software, Data curation. **Yulong Ding:** Supervision, Formal analysis, Writing – review & editing. **Jie Jiang:** Investigation, Project administration. **Shuang-Hua Yang:** Supervision, Writing – review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by the National Natural Science Foundation of China under Grant Nos. 61873119 and 92067109, Science and Technology Planning Project of Guangdong Province under Grant Nos. 2021A0505030001, Shenzhen Science and Technology Innovation Program under Grant Nos. ZDSYS20210623092007023, and Educational Commission of Guangdong Province under Grant Nos. 2019KZDZX1018.

References

[1] S. Figueroa, J. Añorga, S. Arrizabalaga, A survey of IIoT protocols: A measure of vulnerability risk analysis based on CVSS, ACM Comput. Surv. 53 (2) (2020) 44:1–44:53, http://dx.doi.org/10.1145/3381038.

[2] OPCFoundation, OPC, https://opcfoundation.org/.

[3] H. Esquivel-Vargas, M. Caselli, A. Peter, Automatic deployment of specification-based intrusion detection in the bacnet protocol, in: Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy, Dallas, TX, USA, November 3, 2017, 2017, pp. 25–36, http://dx.doi.org/10.1145/3140241.3140244.

[4] S. Longari, M. Penco, M. Carminati, S. Zanero, CopyCAN: An error-handling protocol based intrusion detection system for controller area network, in: Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy, CPS-SPC@CCS 2019, London, UK, November 11, 2019, 2019, pp. 39–50, http://dx.doi.org/10.1145/3338499.3357362.

[5] C. Sheng, Y. Yao, Q. Fu, W. Yang, A cyber-physical model for SCADA system and its intrusion detection, Comput. Netw. 185 (2021) 107677, http://dx.doi.org/10.1016/j.comnet.2020.107677.

[6] M. Zolanvari, M.A. Teixeira, L. Gupta, K.M. Khan, R. Jain, Machine learning-based network vulnerability analysis of industrial internet of things, IEEE Internet Things J. 6 (4) (2019) 6822–6834, http://dx.doi.org/10.1109/JIOT.2019.2912022.

[7] R. Negi, P. Kumar, S. Ghosh, S.K. Shukla, A.G. Intern, Vulnerability assessment and mitigation for industrial critical infrastructures with cyber physical test bed,

in: IEEE International Conference on Industrial Cyber Physical Systems, ICPS 2019, Taipei, Taiwan, May 6-9, 2019, 2019, pp. 145–152, http://dx.doi.org/10.1109/ICPHYS.2019.8780291.

[8] W. Lv, J. Xiong, J. Shi, Y. Huang, S. Qin, A deep convolution generative adversarial networks based fuzzing framework for industry control protocols, J. Intell. Manuf. 32 (2) (2021) 441–457, http://dx.doi.org/10.1007/s10845-020-01584-z.

[9] D. Tychalas, M. Maniatakos, IFFSET: in-field fuzzing of industrial control systems using system emulation, in: 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020, 2020, pp. 662–665, http://dx.doi.org/10.23919/DATE48585.2020.9116365.

[10] H. Zhao, Z. Li, H. Wei, J. Shi, Y. Huang, SeqFuzzer: An industrial protocol fuzzing framework from a deep learning perspective, in: 12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019, 2019, pp. 59–67, http://dx.doi.org/10.1109/ICST.2019.00016.

[11] D. Huang, X. Shi, W. Zhang, False data injection attack detection for industrial control systems based on both time- and frequency-domain analysis of sensor data, IEEE Internet Things J. 8 (1) (2021) 585–595, http://dx.doi.org/10.1109/JIOT.2020.3007155.

[12] A. Khaled, S. Ouchani, Z. Tari, K. Drira, Assessing the severity of smart attacks in industrial cyber-physical systems, ACM Trans. Cyber Phys. Syst. 5 (1) (2021) 10:1–10:28, http://dx.doi.org/10.1145/3422369.

[13] H. Zhou, S. Shen, J. Liu, Malware propagation model in wireless sensor networks under attack-defense confrontation, Comput. Commun. 162 (2020) 51–58, http://dx.doi.org/10.1016/j.comcom.2020.08.009.

[14] P. Hadem, D.K. Saikia, S. Moulik, An SDN-based intrusion detection system using SVM with selective logging for IP traceback, Comput. Netw. 191 (2021) 108015, http://dx.doi.org/10.1016/j.comnet.2021.108015.

[15] J. Caballero, H. Yin, Z. Liang, D.X. Song, Polyglot: automatic extraction of protocol message format using dynamic binary analysis, in: Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007, 2007, pp. 317–329, http://dx.doi.org/10.1145/1315245.1315286.

[16] G. Wondracek, P.M. Comparetti, C. Krügel, E. Kirda, Automatic network protocol analysis, in: Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008, 2008.

[17] Z. Lin, X. Jiang, D. Xu, X. Zhang, Automatic protocol format reverse engineering through context-aware monitored execution, in: Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008, 2008.

[18] J. Caballero, P. Poosankam, C. Kreibich, D.X. Song, Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering, in: Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009, 2009, pp. 621–634, http://dx.doi.org/10.1145/1653662.1653737.

[19] W. Cui, M. Peinado, K. Chen, H.J. Wang, L. Irún-Briz, Tupni: automatic reverse engineering of input formats, in: Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008, 2008, pp. 391–402, http://dx.doi.org/10.1145/1455770.1455820.

[20] P.M. Comparetti, G. Wondracek, C. Krügel, E. Kirda, Prospex: Protocol specification extraction, in: 30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA, 2009, pp. 110–125, http://dx.doi.org/10.1109/SP.2009.14.

[21] Z. Wang, X. Jiang, W. Cui, X. Wang, M. Grace, Reformat: Automatic reverse engineering of encrypted messages, in: Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings, 2009, pp. 200–215, http://dx.doi.org/10.1007/978-3-642-04444-1_13.

[22] S. Kleber, L. Maile, F. Kargl, Survey of protocol reverse engineering algorithms: Decomposition of tools for static traffic analysis, IEEE Commun. Surv. Tutor. 21 (1) (2019) 526–561, http://dx.doi.org/10.1109/COMST.2018.2867544.

[23] J. Luo, S. Yu, Position-based automatic reverse engineering of network protocols, J. Netw. Comput. Appl. 36 (3) (2013) 1070–1077, http://dx.doi.org/10.1016/j.jnca.2013.01.013.

[24] J. Duchêne, C.L. Guernic, E. Alata, V. Nicomette, M. Kaâniche, State of the art of network protocol reverse engineering tools, J. Comput. Virol. Hacking Tech. 14 (1) (2018) 53–68, http://dx.doi.org/10.1007/s11416-016-0289-8.

[25] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453, http://dx.doi.org/10.1016/0022-2836(70)90057-4.

[26] X. Feng, Q. Li, H. Wang, L. Sun, Characterizing industrial control system devices on the internet, in: 24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016, 2016, pp. 1–10, http://dx.doi.org/10.1109/ICNP.2016.7784407.

[27] A. Trifilo, S. Burschka, E.W. Biersack, Traffic to protocol reverse engineering, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009, 2009, pp. 1–8, http://dx.doi.org/10.1109/CISDA.2009.5356565.

[28] S. Tao, H. Yu, Q. Li, Bit-oriented format extraction approach for automatic binary protocol reverse engineering, IET Commun. 10 (6) (2016) 709–716, http://dx.doi.org/10.1049/iet-com.2015.0797.

[29] X. Wang, K. Lv, B. Li, IPART: an automatic protocol reverse engineering tool based on global voting expert for industrial protocols, Int. J. Parallel Emergent Distrib. Syst. 35 (3) (2020) 376–395, http://dx.doi.org/10.1080/17445760.2019.1655740.

[30] W. Cui, J. Kannan, H.J. Wang, Discoverer: Automatic protocol reverse engineering from network traces, in: Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007, 2007.

[31] Schneider, Modbus, https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[32] Beckhoff, EtherCAT, https://ethercat.org/default.htm.

[33] Siemens, S7, https://sourceforge.net/projects/s7commwireshark/.

[34] IEEE-1815, DNP3, https://dnp.org/Portals/0/AboutUs/DNP3PrimerRevA.pdf.

[35] IEEE, C37.118, https://standards.ieee.org/ieee/C37.118.1/4902/.

[36] Rockwell, EIP, https://odva.org/wp-content/uploads/2020/06/PUB00123R1_Common-Industrial_Protocol_and_Family_of_CIP_Networks.pdf.

[37] M.A. Beddoe, Network protocol analysis using bioinformatics algorithms, 2004, Toorcon.

[38] G. Bossert, F. Guihéry, G. Hiet, Towards automated protocol reverse engineering using semantic information, in: 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014, 2014, pp. 51–62, http://dx.doi.org/10.1145/2590296.2590346.

[39] G. Bossert, netzob, 2016, https://github.com/netzob/netzob.

[40] L. Wang, T. Jiang, On the complexity of multiple sequence alignment, J. Comput. Biol. 1 (4) (1994) 337–348, http://dx.doi.org/10.1089/cmb.1994.1.337.

[41] R. Durbin, S.R. Eddy, A. Krogh, G.J. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, 1998, http://dx.doi.org/10.1017/CBO9780511790492.

[42] R.R. Sokal, A statistical method for evaluating systematic relationships, Univ. Kansas, Sci. Bull. 38 (1958) 1409–1438.

[43] J.D. Thompson, D.G. Higgins, T.J. Gibson, CLUSTAL w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, Nucleic Acids Res. 22 (22) (1994) 4673–4680.

[44] P.D. Tommaso, S. Moretti, I. Xenarios, M. Orobitg, A. Montanyola, J. Chang, J. Taly, C. Notredame, T-coffee: a web server for the multiple sequence alignment of protein and RNA sequences using structural information and homology extension, Nucl. Acids Res. 39 (Web-Server-Issue) (2011) 13–17, http://dx.doi.org/10.1093/nar/gkr245.

[45] C. Leita, K. Mermoud, M. Dacier, ScriptGen: an automated script generation tool for honeyd, in: 21st Annual Computer Security Applications Conference (ACSAC 2005), 5-9 December 2005, Tucson, AZ, USA, 2005, pp. 203–214, http://dx.doi.org/10.1109/CSAC.2005.49.

[46] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, L. Guo, Biprominer: Automatic mining of binary protocol features, in: 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2011, Gwangju, Korea, October 20-22, 2011, 2011, pp. 179–184, http://dx.doi.org/10.1109/PDCAT.2011.25.

[47] Y. Wang, X. Yun, M.Z. Shafiq, L. Wang, A.X. Liu, Z. Zhang, D. Yao, Y. Zhang, L. Guo, A semantics aware approach to automated reverse engineering unknown protocols, in: 20th IEEE International Conference on Network Protocols, ICNP 2012, Austin, TX, USA, October 30 - Nov. 2, 2012, 2012, pp. 1–10, http://dx.doi.org/10.1109/ICNP.2012.6459963.

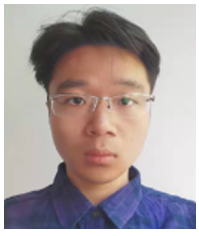
[48] Y. Wang, N. Zhang, Y. Wu, B. Su, Y. Liao, Protocol formats reverse engineering based on association rules in wireless environment, in: 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013, 2013, pp. 134–141, http://dx.doi.org/10.1109/TrustCom.2013.21.

[49] X. Hei, B. Bai, Y. Wang, L. Zhang, L. Zhu, W. Ji, Feature extraction optimization for bitstream communication protocol format reverse analysis, in: 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications / 13th IEEE International Conference on Big Data Science and Engineering, TrustCom/BigDataSE 2019, Rotorua, New Zealand, August 5-8, 2019, 2019, pp. 662–669, http://dx.doi.org/10.1109/TrustCom/BigDataSE.2019.00094.

[50] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, M.M. Munafò, Automatic protocol field inference for deeper protocol understanding, in: Proceedings of the 14th IFIP Networking Conference, Networking 2015, Toulouse, France, 20-22 May, 2015, 2015, pp. 1–9, http://dx.doi.org/10.1109/IFIPNetworking.2015.7145307.

[51] Y. Ye, Z. Zhang, F. Wang, X. Zhang, D. Xu, NetPlier: Probabilistic network protocol reverse engineering from message traces, in: 28th Annual Network and Distributed System Security Symposium, NDSS 2021, Virtually, February 21-25, 2021, 2021.

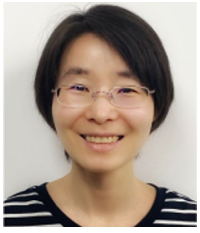
[52] Y. Lin, Y. Lai, Q.T. Bui, Y. Lai, ReFSM: Reverse engineering from protocol packet traces to test generation by extended finite state machines, J. Netw. Comput. Appl. 171 (2020) 102819, http://dx.doi.org/10.1016/j.jnca.2020.102819.
[53] S. Kleber, H. Kopp, F. Kargl, NEMESYS: network message syntax reverse engineering by analysis of the intrinsic structure of individual messages, in: 12th USENIX Workshop on Offensive Technologies, WOOT 2018, Baltimore, MD, USA, August 13-14, 2018, 2018.
[54] S. Kleber, nemesys, 2020, https://github.com/vs-uulm/nemesys.
[55] S. Kleber, R.W. van der Heijden, F. Kargl, Message type identification of binary network protocols using continuous segment similarity, in: 39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, on, Canada, July 6-9, 2020, 2020, pp. 2243–2252, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155275.
[56] J. Smith, ICS-pcap, 2015, https://github.com/automayt/ICS-pcap.
[57] G. Miru, s7-pcaps, 2015, https://github.com/gymgit/s7-pcaps.
[58] T. Yardley, ICS-security-tools, 2018, https://github.com/ITI/ICS-Security-Tools/tree/master/pcaps.
[59] G. Combs, Wireshark, https://www.wireshark.org/.
[60] S. Whalen, M. Bishop, J.P. Crutchfield, Hidden Markov models for automated protocol learning, in: Security and Privacy in Communication Networks - 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings, 2010, pp. 415–428, http://dx.doi.org/10.1007/978-3-642-16161-2_24.
[61] K. Rieck, C. Wressnegger, A. Bikadorov, Sally: a tool for embedding strings in vector spaces, J. Mach. Learn. Res. 13 (2012) 3247–3251.
[62] J. Benesty, J. Chen, Y. Huang, I. Cohen, Pearson correlation coefficient, in: Noise Reduction in Speech Processing, Springer, 2009, pp. 1–4.
[63] H. Li, B. Shuai, J. Wang, C. Tang, Protocol reverse engineering using LDA and association analysis, in: 11th International Conference on Computational Intelligence and Security, CIS 2015, Shenzhen, China, December 19-20, 2015, 2015, pp. 312–316, http://dx.doi.org/10.1109/CIS.2015.83.
[64] H. Lüke, H.D. Schotten, H. Hadinejad-Mahram, Binary and quadriphase sequences with optimal autocorrelation properties: a survey, IEEE Trans. Inf. Theory 49 (12) (2003) 3271–3282, http://dx.doi.org/10.1109/TIT.2003.820035.
[65] Mitsubish, Melsec, https://www.mitsubishielectric.com/fa/products/cnt/plc/index.html.
[66] K. Liu, M. Yang, Z. Ling, H. Yan, Y. Zhang, X. Fu, W. Zhao, On manually reverse engineering communication protocols of linux-based IoT systems, IEEE Internet Things J. 8 (8) (2021) 6815–6827, http://dx.doi.org/10.1109/JIOT.2020.3036232.



Yuhuan Liu is currently a Master student in the Department of Computer Science and Engineering at Southern University of Science and Technology (SUSTech). His research interests include protocol reverse engineering and industrial control network security. He received his B.E. degree in 2019 with the major of electronic and information engineering from Nankai University, China.

Fengyun Zhang is a Ph.D. student in the Department of Computer Science and Engineering at Southern University of Science and Technology (SUSTech). His research interests include UWB indoor location and industrial control network protocol reverse engineering. He received his B.E. degree in 2014 with the major of electronic and information engineering, and his M.E. degree in 2017 with the major of signal and information processing both from Southwest University, China respectively.

Yulong Ding received the B.A.Sc. and M.A.Sc. degrees in Chemical Engineering from Tsinghua University, China, in 2005 and 2008, respectively and the Ph.D. in Chemical Engineering from University of British Columbia, Canada, in 2012. He is currently a Research Assistant Professor with the Academy for Advanced Interdisciplinary Studies of Southern University of Science and Technology. His main interests are Industrial Internet of Things and Cyber Security.



Jie Jiang received the B.A.Sc. degrees in Automation from Chang'an University in 2007 and the M.S. degree from Xi'an Jiaotong University, China, in 2010, respectively, and the Ph.D. in Artificial Intelligence from Delft University of Technology, Netherlands, in 2015. She is currently a Research Assistant Professor with the Department of Computer Science and Engineering of Southern University of Science and Technology. Her main interests are Artificial Intelligence and Cyber Physical System.

Shuanghua Yang received the B.S. degree in instrument and automation and the M.S. degree in process control from the China University of Petroleum (Huadong), Beijing, China, in 1983 and 1986, respectively, and the Ph.D. degree in intelligent systems from Zhejiang University, Hangzhou, China, in 1991. He was awarded DSc from Loughborough University in 2014 to recognize his academic contribution to wireless monitoring research. He is currently a chair professor of computer science and executive deputy dean of graduate school with Southern University of Science and Technology (SUSTech), Shenzhen, China. Before joined SUSTech in 2016 he had spent over two decades in Loughborough University, as a professor in computer science and head of department. His current research interests include cyber–physical system safety and security, Internet of Things, wireless network-based monitoring and control. He is a Fellow of IET and a Fellow of InstMC, U.K. He is an Associate Editor of the IET Journal Cyber–Physical Systems Theory and Applications.