Research Article

Network Communication Protocol Reverse Engineering Based on Auto-Encoder

Tianxiang Yu 1,2, Yang Xin 1,2, Yuexin Tao 1,2, Bingqing Hou 1,2, and Hongliang Zhu 1,2

1 School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
2 State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang, Guizhou 550025, China

Correspondence should be addressed to Yang Xin; yangxin@bupt.edu.cn

Received 5 November 2021; Revised 22 May 2022; Accepted 27 September 2022; Published 6 October 2022

Academic Editor: Chien Ming Chen

Copyright © 2022 Tianxiang Yu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Network communication protocol reverse engineering is useful for network security, including protocol fuzz testing, botnet command infiltration, and service script generation. Many models have been proposed to generate field boundary, field semantic, state machine, and some other format information from network trace and program execution for text-based protocol and hybrid protocols. However, how to extract format information from network trace data for binary-based protocol still remains a challenging issue. Existing network-trace-based models focus on text-based and hybrid protocols, using tokenization and some other heuristic rules, like field identification, to perform reverse engineering, which makes it hard to apply to binary-based protocol. In this paper, we propose a whole mechanism for binary-based protocol reverse engineering based on auto-encoder models and other clustering algorithms using only network trace data. After evaluation, we set some metrics and compare our model with existing other models, showing its necessity to the field of protocol reverse engineering.

1. Introduction

Network protocols define the message format and transmission order of communication packets exchange among communication entities ruled by traditional seven-layer Open System Interconnection (OSI). With the rapid development of communication network scale and depth, many corporations and institutions choose to use proprietary communication protocols in their internal network, communication devices, and business application services to protect their information security and business service security. While these proprietary network protocols help these corporations and institutions expand their ability to provide more services and form their own application system, vulnerabilities in these implementations of proprietary protocols will bring huge potential threats to the network service and sensitive user data, which are maybe caused by just a little check fault. Even general and public network protocols have undetected vulnerabilities, like the flaw in UPnP protocol which makes countless routers and the network on them vulnerable [1].

In the past decades, the cybersecurity community has put forward a myriad of methods and corresponding algorithms to detect vulnerabilities in these public or proprietary protocols under different conditions. For example, when the definition and technical details of a protocol are accessible and the implementation of protocol is also achievable, many methods in programming analysis have been proposed to detect vulnerabilities in these public protocols, like taint tracking [2], static code analysis [3–6], symbolic execution [7, 8] and white-box fuzzing [9, 10].

However, when the specification of protocols is hard to obtain, like proprietary protocols, the mining and analysis of vulnerabilities in these protocols will be hard and complex. Reverse engineering is needed in the proprietary protocol analysis. There are two main methods to solve the reverse engineering problem, network trace method [11] and tainted data method [12]. Network trace method is the first
technique wildly used in automatic protocol reverse engineering. It captures the traffic packet in the communication using protocols to be analysed and analyse the text-based or binary content in this packet. The network trace method for protocol reverse engineering (PRE) is good at identifying and locating the keywords, field delimiters, and data field in these packets. Tainted data method infers the message format of proprietary protocols from the dynamic execution of application using these protocols. In this paper, we mainly focus on the network trace method for protocol reverse analysis. After the reverse analysis of proprietary protocols, we still need to do some fuzzing with the reverse result to find the potential vulnerabilities in these protocols, as discussed in this survey [13].

Many tools have been invented to carry out the PRE task. The algorithms used in these tools are mostly unsupervised, extracting the same or similar parts in a large number of packets captured on networks to analyse and obtain the format information of messages of the target protocol. Basic multiple sequence alignment algorithm has been proposed by PI based on algorithms in bioinformatics field [14]. There are many flaws in multiple sequence alignment algorithm, like the decrease in accuracy caused by alignment of messages in different formats and the lack of consideration on the structure. So, many clustering and sorting algorithms have been proposed to improve the effectiveness and performance of the algorithm and solve the reverse engineering problem [15–17].

Although existing methods have solved the protocol network trace reverse analysis problem from different aspects, the algorithms they based on are mostly shallow. The algorithms put attention to the similarity matrix or information entropy to split the network message. That is to say, the feature extraction of these algorithms is shallow and maybe insufficient to extract enough information to reverse engineer protocol message formats precisely. And although existing analysis algorithms can be applied to different proprietary protocols, they cannot change their properties to fit one specific protocol better. Existing algorithms use a lot of heuristic methods and expertise knowledge to increase the mutual information among packet sequences and improve the performance, encountering some problems when used in binary-based protocol reverse engineering.

To supplement this research, we propose a novel deep-learning based unsupervised auto-encoder model to solve the clustering problem in PRE field. For initial clustering of PRE, we build an auto-encoder model to extract the low dimensional features of messages. Then we propose a density-based clustering algorithm. For further clustering of PRE, by introducing double-loss mechanism which consists of information keeping loss and outputs difference loss, we build another auto-encoder model to extract the mutual field information and evaluate similarity in these protocol messages. The information keeping loss makes the model keep the most information as possible. The outputs difference loss makes the model keep the mutual information among all these proprietary protocol messages which represents the immutable part in these messages. Based on the clustering result, we can use progressive sequence alignment algorithm to identify which part of the message is immutable and mutable, getting the format information of protocols.

We do our experiments on HTTP and FTP protocols. We put raw message data to the model and get the format information of protocols. By comparing our result and the protocol specification, we prove our model is efficient in reverse engineering of protocols. In order to better distinguish different models, we set some metrics for these models and compare our models with other existing models.

Our contribution is listed as follows.

(i) By introducing and building an unsupervised deep-learning model, we proposed an algorithm to solve the initial clustering problem of binary-based protocol reverse engineering.

(ii) We propose a new training loss of auto-encoder which can be used to evaluate similarity of protocol messages and perform further clustering.

(iii) As far as we know, we firstly propose a whole mechanism for binary-based protocol reverse engineering using auto-encoder and only network trace data.

2. Background and Motivation

In this section, we will explain the background information of protocol reverse engineering and unsupervised deep-learning model in the first and second part. In the final part, we will explain the usage of proprietary protocol reverse engineering, including the motivation behind this paper.

2.1. Existing Protocol Reverse Algorithms and Tools Based on Network Trace

2.1.1. Protocol Information Project. Protocol information project (referred as PI) was proposed by Marshall Beddoe in 2004, inspired by sequence alignment algorithm in bioinformatics field. PI [14] extracts the structure information of proprietary protocols from many communication packets to understand the field boundary. PI uses the Smith–Waterman [18] algorithm to generate a distance matrix among these packet sequence, then uses unweighted pair-group method with arithmetic means to do clustering analysis to sequences based on the distance matrix and finally uses the Needleman Wunsch algorithm [19] to get the boundary of protocol fields and identify whether the content of one field is invariable or not. PI can only get protocol information about field boundary and content type, needing addition manual analysis to understand the meaning of content. PI is suitable for simple protocols and works badly applying to sophisticated protocols.

2.1.2. Discoverer. Discoverer [15] was proposed by Cui in 2007 based on their previous work “RolePlayer” [20]. The algorithms used in Discoverer are mostly heuristic, designed with the help of expertise knowledge. The first step in Discoverer is tokenization and initial clustering. It dissects
the byte content in protocol packets to text and binary fields. Then it uses delimiters to divide fields to tokens in protocol packets. Then it does the initial clustering by token pattern instead of sequence pattern because there will be error in the alignment result if the length is uncertain. The second step is token semantic inference and recursive clustering. Properties of token are easy to judge, like content type and variable type. For semantics of token, Discoverer designs heuristic rules for all semantics, like length, offset, format distinguisher, and cookie. Then, Discoverer recursively clusters the result of initial clustering based on the format distinguisher. The final part of Discoverer is merging and sequence alignment. Discoverer is very effective in reverse analysis application of many protocols. However, it relies heavily on the heuristic algorithm and expertise knowledge. Also, Discoverer ignores the binary-based protocol problem, as discussed in [21].

2.1.3. Other Algorithms. Besides network-based algorithms, there are still some algorithms using both network-based and program-based methods, like Polyglot [22], AutoFormat [23], Tupni [24], and Reformat [25]. The above algorithms is used to extract information in these protocol packets to get the field boundary and field semantics of protocol. For state machine inference of protocol, there are some other algorithms, like PEXT [26] and Prospex [27].

2.2. The Usage of Properties Protocol Reverse Engineering. Protocol reverse engineering is very useful. Except basic protocol security analysis before proprietary protocols are used in large-scale business internal network or public protocols are used as standards, PRE may play a good role in many communication security scenarios, such as botnet command infiltration and honeypot script generation [28].

Aiming at the current popular botnet based on IRC protocol, protocol reverse engineering can be used to analyse the IP address, port, user name, password, and other information of IRC server in the botnet program, so as to further track the target botnet effectively, and provide conditions for in-depth analysis of the characteristics of botnet.

For command and control (C&C) protocols used by botnets, automatic protocol reverse engineering can be used to extract the format of protocol messages sent by applications implementing protocol specifications, and infer the field semantics of messages sent and received by applications [29]. The C&C protocol for botnets, like many other application layer protocols, is not documented. Automated protocol reverse engineering techniques are able to understand unrecorded protocols, for example, and they can activate botnet penetration, in which security analysts rewrite messages sent and received by botnets to contain malicious activity, and provide the botnet administrator with an illusion of successful and unhindered operation [30].

3. Methodology

In this section, we will present an overview of our protocol reverse engineering model. We divide our model into three parts, density-based clustering with abstract features, most mutual information keeping based on auto-encoder, and sequence alignment.

3.1. Overview. The message format of network protocol is complicated. Even the message format of stateless text-based protocol can be divided into two categories, the requesting message and the response message. The reverse engineering result will be devastating if we categorize messages of different formats into one class and use protocol reverse engineering algorithms.

So, the first step of protocol reverse engineering is clustering. If we already know the proprietary protocol is simple and maybe has just two formats, such as the request and the response, like HTTP protocol, we can just split the messages to two parts according to the IP address. However, most proprietary protocols are stateful and complicated. We need to cluster messages of these protocols so that the message in one class can be more similar and the sequence alignment result can be more accurate. The initial clustering methods of previous works focus on similarity matrix, like PI [14] uses UPGMA and Smith–Waterman algorithm, or little deeper features, like Discoverer [15] uses token sequences. After the initial clustering, some methods include further clustering, like Discoverer performs recursive clustering based on token semantics inferred by heuristic rules and performs format merging to the result of recursive clustering to avoid overfitting.

The second step of protocol reverse engineering is sequence alignment. Most reverse engineering methods perform sequence alignment to the result of clustering to get the final result.

Our auto-encoder-based model can also be divided to two parts, clustering and sequence alignment. In the clustering phase, our model uses density-based clustering method with low dimensional features to perform the initial clustering. We use an auto-encoder model to learn the latent space representation in these messages. By using regularization mechanism and reducing the distortion between input and output. We can consider the latent space representation as abstract features of high dimensional message data of proprietary protocols. Then we can use these abstract data to perform density-based clustering algorithm which is more suitable for our application scenario. The specification of proprietary protocol may have multiple formats and the number of formats is uncertain. So, we have to use density-based clustering algorithm with no need to specify the number of clusters. The density-based clustering algorithm can also filter some abnormal messages which may have long data field or mistakes.

Then we propose a double-loss auto-encoder model to keep the most mutual information and evaluate similarity among messages in these clusters. In order to better distinguish the two auto-encoder models, we regard previous
auto-encoder model as the first model and the auto-encoder model with double loss as the second model. We take the parameters of the first model as the initial parameters of the second model because these parameters have been proven efficient to learn the representation of messages. Then we extend the input tensor from one message to two messages. The second model gets two messages and outputs two messages. The training loss of the second model contains two parts, one is the distortion between the input and output, the other is the distortion between the outputs of two messages. After training, we can use the second model to evaluate similarity among messages and build guidance tree to perform further clustering.

In the sequence alignment phase, we perform the progressive sequence alignment algorithm [31] on each clustering result and get many protocol formats.

3.2. Density-Based Clustering with Abstract Features. In this section, we will explain the details of the first auto-encoder model and the density-based clustering algorithm, including data preprocessing and the example of result.

The raw protocol communication data is data frame captured by some network analysis software, like Wireshark.

The first part of density-based clustering is targeted protocol messages extraction. Most protocols can get the targeted protocol messages by removing the redundant headers, like HTTP. However, for protocols which have too much data to transport, message or packet fragment has been performed before transportation. To get complete messages of these protocols, we need to perform message or packet fragments recombination to the raw data at first. In some cases, we do not need to do fragments recombination even when message or packet fragments really exist since what we really care is the header of protocol message and the little difference in some header data, like identification, total length, and fragment offset, will not have much impact on the reverse engineering result or the fuzz test result. It is worth noting that now the targeted protocols we are talking about are protocols in the application layer. There is not much difference in the process of protocol messages, segments, or packets extraction, so we only talk about the protocols in the application layer, and so are the other parts of this paper. The process of targeted protocol messages extraction is shown in Figure 1.

The second part of density-based clustering is abstract feature extraction of high dimensional message data. It is worth noting that our model focuses mainly on binary-based protocols and their binary fields because for those text protocols or hybrid protocols, there have been a lot of excellent and efficient algorithms to solve the protocol messages clustering problem, like Discoverer. We may run our experiments on text protocols or hybrid protocols, but we will consider these text protocols or hybrid protocols as binary-based protocols and consider their binary message data as our input data.

Most network protocols are hybrid protocols. These hybrid protocols contain a lot of binary fields to increase its own efficiency and use some text fields to make its messages readable to some extent. These text fields are mostly encoded and decoded by American Standard Code for Information Interchange (ASCII). One letter is encoded to one Byte content in message. For convenience that we can easily handle binary fields and potential text fields, we consider one Byte content in message as one data element and use a number between 1 and 256 to represent it. We keep 0 as the zero-padding number used in the input processing of deep-learning model.

As for the abstract feature extraction auto-encoder model, the shape of the input tensor should be changed according to the max length of messages and cannot be too long because long messages must contain a large number of data fields which is not helpful to analyse the format of the message header. We recommend to discard the redundant part of the messages. In our first auto-encoder model, we set the length of input to 256. We add three fully connected layers to the encoder phase and decoder phase of the auto-encoder model and the output lengths are 128, 64, and 32, respectively. We consider the output of encoder as the abstract features of message and perform further clustering on these outputs. The structure and configurations of the first auto-encoder model are shown in Figure 2.

We use \( x \) to represent the input. \( x \) has the shape of (None, 256). We use \( h_1, h_2, h_3, f \) to represent the output of the dense layer 1, 2, and 3 and the abstract features in the encoder. We use \( W_1, W_2, W_3, W_4, b_1, b_2, b_3, b_4 \) to represent the matrix and bias parameters in the encoder layer. The formulas about computing \( h_1, h_2, h_3 \) are shown in the following equation:

\[
\begin{align*}
\hat{h}_1 &= \text{relu}(W_1x + b_1), \\
\hat{h}_2 &= \text{relu}(W_2\hat{h}_1 + b_2), \\
\hat{h}_3 &= \text{relu}(W_3\hat{h}_2 + b_3), \\
f &= \sigma(W_4\hat{h}_3 + b_4).
\end{align*}
\] (1)

We use \( h'_1, h'_2, h'_3, x' \) to represent the output of dense layer 1, 2, and 3 in the decoder. We use \( W'_1, W'_2, W'_3, W'_4, b'_1, b'_2, b'_3, b'_4 \) to represent the matrices and bias parameters in the decoder dense layers. The formulas about the training loss and computing \( h'_1, h'_2, h'_3, x' \) are shown in the following equation:

\[
\begin{align*}
\hat{h}'_1 &= \text{relu}(W'_1f + b'_1), \\
\hat{h}'_2 &= \text{relu}(W'_2\hat{h}'_1 + b'_2), \\
\hat{h}'_3 &= \text{relu}(W'_3\hat{h}'_2 + b'_3), \\
x' &= \text{relu}(W'_4\hat{h}'_3 + b'_4), \\
\text{loss} &= \text{mse}(x, x').
\end{align*}
\] (2)

It is worth noting that if we change the activation function from relu to tanh or sigmoid, performance will not change much. Readers can try different activation functions to build their own model. Also, some subtle changes in dimension of abstract features will not have much impact on performance. In the Figure 2 above, we set the dimension of abstract features to 5, but it will not hurt if you change the dimension to 2-8.
The final part of density-based clustering is performing clustering on the abstract features and dividing the protocol messages into several classes. Clustering is essential in protocol reverse engineering for the reason that directly applying sequence alignment algorithm on messages of difference formats will greatly reduce accuracy of the PRE result.

Density-based clusters are dense data regions in the data area which are separated by data region with low density. One density-based cluster consist of densely connected data points. To solve the clustering problem in the PRE field, we use density-based clustering algorithm due to the fact that this clustering algorithm is suitable for processing data with uncertain clusters and distributed in different shapes, which is also the demand in PRE field, that is, the uncertain number of formats and potential data distribution.

We propose the density-based protocol message clustering algorithm (referred as DBPMC). We take the abstract features from the first auto-encoder model as the input of DBPMC on the ground that feature extraction and low-dimensional features will improve the effectiveness of clustering algorithm. We use $F = \{f_1, f_2, \ldots, f_n\}$ to represent the input of DBPMC while $n$ representing the number of messages. We have made some modifications to the classic density-based clustering algorithm, DBSCAN [32] and OPTICS [33]. The pseudo code of DBPMC is shown in Figure 3.

Please note that after we get the clustering result, we can just ignore the next information merging step and perform sequence alignment algorithm on these clusters of protocol messages. Of course, it will need additional manual work to get the analysis result of PRE; however, when the messages are simple enough, just clustering and sequence alignment will still produce good result. The whole clustering process is shown in Figure 4.

### 3.3. Information Merging Based on Auto-Encoder

As we have known that the result of the previous step is enough to perform sequence alignment algorithm, this information merging step is still important for the reason that directly applying sequence alignment algorithm to binary message data is hard. We propose this information merging method based on auto-encoder to transform protocol messages from binary level to token level to some degree without the help of heuristic rules or expertise knowledge so that we can measure the similarity of binary messages better.

On this problem, previous works use ASCII decoder and similar ones to decode binary messages and decide the potential text fields, then use a series of rules to decide the semantics of fields. This method works poorly when binary fields occupy the majority of message, segment, packet, or frame.

Trying to solve this problem, we propose an auto-encoder model with two different training losses to keep the mutual and critical information in binary message so that similar and also different binary byte fields in two messages can be transformed to one binary byte block which can be used to measure the difference between two binary messages. When performing sequence alignment and manual analysis on these binary-based protocol messages, we can use the second auto-encoder model to compute the distance matrix of binary-based protocol messages.

One training loss is the distance between the input and output, also the basic auto-encoder training loss. The other training loss is the distance between two parts of output. Different from the first auto-encoder model, the shape of input and output data in the second auto-encoder model is changed.

For every cluster we get from the DBPMC algorithm, we build a new auto-encoder model which has the same model structure but different tensor shape with the first auto-encoder model. To reduce the computational work, we use
parameters of the first auto-encoder model as initial parameters of the second auto-encoder model for the reason that these parameters have been proven efficient to extract features from these protocol message data. We also add a hyper parameter $h_2$ as part of the second training loss. The structure and configurations of the second model are shown in Figure 5.

We want to keep mutual and critical information in these clusters of binary-based protocol messages by introducing another training loss while we do not want to force the output of every binary message to be the same because the number of clusters may not be the true number of protocol formats and we still want to split the cluster. So, we make a hypothesis that the output training loss of messages under the same format will drop more quickly than that under different formats. So, when first training, we just need to record the number of training epochs when the model converges. Then we reset the auto-encoder model, and use half epochs when training to get the final second auto-encoder model so that we can separate messages of different formats by analysing the training loss before convergence happens.

3.4. Sequence Alignment. Now we have got the clusters of protocol binary messages and an auto-encoder model which can keep the mutual and critical information of these messages. However, the clustering result may not be accurate enough and will have a bad impact on the result of sequence alignment, especially when messages contain binary part. Optionally, we can use the second auto-encoder model to do further clustering. By considering the mean-squared loss between two outputs as the similarity of two messages, we can build a phylogenetic tree of messages. After setting a distance threshold, we can split the phylogenetic tree into several parts and perform the final sequence alignment and get complete result. The sequence alignment process is shown in Figure 6.

4. Material

4.1. Data Material. We collect message data of HTTP protocol and FTP protocol using network protocol analyser software Wireshark. Although HTTP protocol and FTP protocol are all text protocols, we use their binary information before decoding as the input. We collect about 50k messages to train the model during daily use of HTTP and FTP protocol. Readers can also use some request generation tool, like POSTMAN, to simulate communication. Any network tools that can capture network communication and generate .pcap files can be used in the raw data obtain phase. About the preprocessing of these raw messages, we firstly extract the Bytes data of the .pcap file using Python module “io.BytesIO” (in our experiment, we extracted the Bytes data of the application layer of the HTTP .pcap file). Then we turn every Bytes data to a number between 1 and 256 to represent it using Python function “list.” We set 0 as the padding number. For each communication message, we only need the first 256 Bytes. And these 256 numbers form a vector as the input of deep-learning model. Since we use autoencoder as our unsupervised deep-learning model, we do not need to label each data.

4.2. Software and Hardware Material. We implement our models and algorithms in Python3.8 with “TensorFlow2.4” and “pandas” modules. We create inherited encoder and decoder

![Figure 2: The structure and configurations of the first auto-encoder model. The model consists of two parts, encoder and decoder. The configurations are normal. Further fine tuning of these configurations and model parameters is recommended according to the application scenario.](image_url)
decoder class from TensorFlow’s “keras.Model” class and build the auto-encoder model. As for data visualization, we use “Embedding Projector” published by Google.

5. Results

5.1. Overview. Firstly, we need to define the evaluation method. We evaluate our result by comparing the inferred formats with true message formats of protocol. For example, in HTTP protocol, arbitrary number of “parameter: value” tokens are allowed to exist in one message. Since we mainly focus on the binary-based protocol messages reverse engineering, we cut the message to a relatively short part, ignoring most text data field. Instead of treating each ordering of the tokens as one format, we only care about the critical processor state of protocol and treat each process or state as one format. By this way, http just has two main formats, request and response which can be divided into more formats according to the http code and request method.

To prove that low dimensional feature extraction and density-based clustering is actually needed, we firstly perform dimensionality reduction algorithm on these protocol message data. The result is shown in Figure 7.

Figure 7 shows that PCA plays a role in clustering and separates message data to two big classes. Although the result of clustering is not good enough to perform further clustering or sequence alignment, it still shows that the message data have the value of being clustered and there is necessity to perform other feature extraction algorithm on the message data.

We take these data as the input of the first auto-encoder model. We implement the auto-encoder model with functional API so that after training we can treat the encoder and its parameters as a new model easily. We input message data again and get low-dimensional features data. After that we perform DBPMC algorithm on these features data. To better show the clustering result and the necessity to use density-based algorithm, we also perform uniform manifold

Algorithm 1 density-based protocol message clustering algorithm (DBPMC)

Input: set of protocol message abstract features, represented as \( F = \{f_1, f_2, \ldots, f_n\} \)

Output: clustering result, represented as \( F_1, F_2, \ldots, F_m \)

1. Initialize \( \varepsilon \) as the neighborhood, 0 as the empty ordered list, compute \( M \) as neighborhood density threshold with the formula of \( M = h_1 * n * \varepsilon^2 \) in which \( h_1 \) is a hyper parameter.
2. Get core points \( C = \{c_1, c_2, \ldots\} \) from \( F, N_c(c) = \{ y \in F | d(y, c) < \varepsilon, |N(y)| > M \} \)
3. Compute core distances of all points \( CD = \{d_1, d_2, \ldots, d_n\} \), \( |N_{d_i}(c)| = M \)
4. Make all points as unvisited
5. For each point in \( F \)
6. Set reachable distance undefined, \( RD = \{rd_1, rd_2, \ldots, rd_n\} \)
7. For each unvisited point \( f_i \)
8. Get \( N \), then neighbors of \( f_i \)
9. Set \( f_i \) visited
10. Output \( f_i \) to \( o \), the ordered list
11. If the number of elements in \( N \) is bigger than \( M \)
12. Initialize \( S \) as an empty priority queue
13. For each unvisited point \( n_j \) in \( N \)
14. Set new reach dist as \( nrd = max (cd_{d_i}, dist(n_j, f_i)) \)
15. If reachable distance of \( n_j \) is undefined
16. \( rd_{n_j} = nrd \)
17. \( S.insert(n_j, rd_{n_j}) \)
18. Else if \( rd_{n_j} > nrd \)
19. \( rd_{n_j} = nrd \)
20. Update the priority queue \( S \)
21. While \( S \) not empty do
22. \( S.pop \) the first element \( s_1 \)
23. Set this element visited
24. Output \( s_1 \) to \( o \), the ordered list
25. If the number of neighbors of \( s_1 \) is bigger than \( M \)
26. Insert all unvisited neighbors of \( s_1 \) to \( S \), like step 13-20
27. Reorder \( RD \) according to \( o \), the ordered list, represented as \( RD' \)
28. Compute the number of clusters according to the local maximum in the \( FD' \)
29. Sort elements in \( F = \{f_1, f_2, \ldots, f_n\} \) to these clusters and get the final result

Figure 3: Pseudo code of DBPMC algorithm.
approximation and projection (UMAP) algorithm, the result is shown in Figure 8.

We adjust the parameter of DBPMC algorithm and get 15 clusters. One cluster mainly contain the request message with GET method, these messages contain almost the same “parameter: value” tokens, like Connection, Accept, User-Agent, and Host; one cluster mainly contains response messages of the continuation state, these messages contain binary data field; other clusters mainly contain messages of some response state or request method. Measuring the classification accuracy is hard, we specify the category of every cluster, randomly sample 100 messages, and compute accuracy. The classification accuracy is 56%. It is not a good result. However, considering the complicated binary data field and uncertainty in communication, the result is not bad for PRE field. And during later sequence alignment phase, we can still set the distance threshold, split trees and, get the main part of one cluster furtherly.

Then, we train the second auto-encoder model and use this model to evaluate the similarity matrix of messages in one cluster. With the similarity matrix, the phylogenetic tree of messages is built for every cluster, and the phylogenetic tree can be split to many sub-trees. These sub-trees contain messages that are different with others in one cluster. The phylogenetic tree building process is a further clustering. For each sub-tree, we perform progressive sequence alignment (PSA) algorithm to get one inferred format of protocol. For the reason that we run our experiments on only binary field or text field which is not decoded, the inferred format we get is numerical. The PSA process example is shown in Figure 9.

We summarize all inferred formats of protocol and get the final result. For example, cluster which mainly contains messages with GET method outputs protocol format of the GET method. The format contains some immutable binary
information, we consider this as a correctly inferred format. The HTTP dataset contains several formats, “GET,” “POST,” “HEAD,” “200 O/K “304 Not Modified”, “206 Partial Content,” and “Continuation”. Messages with “Continuous” state only contain data field which is only able to be decoded by receiver, so we do not consider “Continuous” as a true format. As for the remaining formats, except for “206 Partial Content”, all formats are successfully generated.

With the inferred immutable binary fields, we can know the basic structure of protocol. If a protocol mainly contains binary field which cannot be decoded to text field, we can build a dictionary for these immutable parameters so that we can get the encoding of all control information of binary-based protocol. zJ_hat is a good beginning of reverse engineering which is a complicated binary-based protocol. If the binary field of protocol messages can be decoded using ASCII, UTF, or some other standards, we can calculate the frequency and other statistical characteristics in binary field data and find the most possible message decoder that text-based protocol may use.

We also do some experiments on FTP protocol. For the reason that messages of FTP protocol are mostly simple and stable, like the encoding of “331 Password required,” “USER share” and “220 Microsoft FTP Service,” messages are easily to cluster and analyse. zJ_he clustering result of FTP low-dimensional features is shown in Figure 10.

Now we have proven that our model can solve the clustering problem of binary-based protocol based on only network-trace data. zJ_he result of principle component analysis (PCA) algorithm on the message data is shown in Figure 7. The data were divided into 15 classes.

The final result of HTTP and FTP protocol is shown in Figure 11.
format of protocol, we think it is accurate. We repeat the process several times and finally get the accurate performance of our model which is shown in Figure 12.

5.2. Comparison with Other Studies. It is hard to compare our methods with other existing methods in PRE field because for different methods, their definition of correct result of reverse engineering is different and there is no standard for PRE field. In order to better distinguish different methods, we set some metrics for these methods, including input type, output type, analyzable protocol type, and methods used by model. Output types include field boundary, field semantic, and state machine. The attribute metric is shown in Table 1.

From the above table, we have known that our model provided a solution to solve the problem “Binary-based protocol cannot be directly reverse engineered based on only network trace data” that cannot be solved by existing models. That is our main breakthrough and contribution. About the performance of the model, different researches pick different standards to prove their effectiveness, we compare the performance of our model with that of existing models. The performance matrix is shown in Table 2.

Models based on network trace are usually relatively weak. There have been many good researches about using both network trace information and other information, like program execution. In these network-trace-based models, due to the lack of information, many heuristic methods have been proposed, like tokenization and field identification, which makes it difficult to be used in binary-based protocol reverse engineering. To supplement this research, we...
propose an auto-encoder-based model which can leverage low-dimensional features and difference loss of messages to perform initial clustering, further clustering and get the result eventually. It works finely on binary field data which are binary field data in binary-based protocol or text field data that have not been decoded. The model still has a lot to improve, like the model input can include more information and some other classification methods can be included in the model to achieve a certain effect of field identification. We will talk about the future work later.
Table 2: Performance metric of protocol reverse engineering models.

<table>
<thead>
<tr>
<th>Existing protocol reverse engineering model</th>
<th>Performance of existing model</th>
<th>Performance of our model under similar standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discoverer [15]</td>
<td>More than 90% of our inferred formats correspond to exactly one true format;</td>
<td>On average, 89.5% of inferred blocks correspond to true formats of protocol. The top accuracy is 97.9 when applying to FTP.</td>
</tr>
<tr>
<td>PI [14]</td>
<td>Can be applied to byte stream that contains structure, like HTTP.</td>
<td>Binary-based protocol can be analysed, including byte stream with or without structure.</td>
</tr>
<tr>
<td>Autoformat [23]</td>
<td>Identify individual message fields with 93.4% high accuracy compared with Wireshark.</td>
<td>If we consider Wireshark can identify all fields in the message correctly. We achieve 89.5% on average and 90%+ on some specific protocol.</td>
</tr>
<tr>
<td>Netzob [17]</td>
<td>The illustration of performance is complicated. They leverage contextual information and achieve high accuracy (80% on average, sometimes 100%) as well as high conciseness (0.5–0.6).</td>
<td>We can achieve high accuracy as shown above. But we cannot achieve conciseness due to the limitation of binary-based protocol. In the experiment, only 30%–40% of real formats can be found.</td>
</tr>
</tbody>
</table>

6. Discussion

In this section, we discuss the limitations of our PRE model and the future work.

The first part of limitations is the huge computational work. We use deep unsupervised learning method to extract abstract features or compute similarity for better clustering. For one protocol to be analysed, we need to put every message sample into the first auto-encoder model for dozens of times. If we want to improve the effectiveness of the first auto-encoder model, adding layers or adding hidden units are also accepted choices, however, but it also brings additional computational work to the whole PRE model.

For the second auto-encoder model, we need to train several auto-encoder models according to the number of clusters. For every cluster, we need to put every two messages into the specific auto-encoder for training. The computational work in the second part is even bigger than that of the first auto-encoder model.

The second part of limitations is the manual work in our PRE model. Since our model only generates binary field boundary that can be used as control information dictionary of binary-based protocols or text field boundary after choosing appropriate decoder. These data may be duplicate or incorrect which needs manual work to fix it. In order to obtain complete information about protocol, additional analytical work is required. In the future work, since the low dimensional features extracted by auto-encoder have been proven effective, we may consider adding more protocol-related data, like program execution, to better use the features. We may also consider adding some classification models to the whole PRE mechanism so that we can get some field semantics for these binary field data. Now as we have got the clusters of messages which correspond to states of protocol, in the future work, we may also consider a state machine inference algorithm to obtain a complete state machine from these clusters and corresponding messages.

Besides limitations, there are also some interesting facts about the application of our PRE model. As we all know, the result of PRE is usually used in fuzz testing of communication protocols or communication devices which use these protocols. However, most researchers carry on fuzz testing when they actually know the structure of communication device or specify one particular network protocol to fuzz. For example, some fuzz testing researches focus on the NTP service of communication device, like switch and router, because researchers know that the NTP service of the device is open and they can perform reverse engineering and fuzz testing on this service to test the stability of the device and even affect communication function. However, these situations are rare under real conditions. As we have shown in above chapters, we now can get the clustering graph of communication. A bigger cluster means that protocols or devices use this part of communication more often. A smaller cluster or even zones between clusters mean that protocols or devices rarely use this part of communications, maybe because this part of communication only corresponds to just few functions. So, they may lack enough security check or flow control facing complicated communication of the rare part. This mechanism can also be used in fuzz testing of protocol and devices that we only send these rare parts of communication to the devices and protocols. We have used this method to detect some vulnerabilities when fuzzing an integrated device of switching and routing. For possible security risk, we will not publish these vulnerabilities. But readers who are interested in this method can perform further research on the application of our model.

7. Conclusions

In order to supplement network-trace-based reverse engineering field for binary-based protocols, based on auto-encoder and its low dimensional feature extraction ability, we propose a whole mechanism which contains auto-encoder models and other clustering and alignment algorithms to perform reverse engineering for binary-based protocols and text-based protocol. After the initial clustering, further clustering, and sequence alignment, we get some immutable binary fields which can be used as control information dictionary of binary-based protocol or text fields after decoding of text-based protocol. The model works finely on several protocols, dividing messages into correct clusters. For future work, the model still has a lot to improve, such as adding more information and more inference algorithm.
Data Availability
The experiment of this paper is just a basic theoretical verification. The data used to support this paper are very easy to obtain using network protocol analyser software Wire-shark. Readers can read the “Material” section of this paper for information about data.

Conflicts of Interest
The authors declare that there is no conflicts of interest regarding the publication of this paper.

Acknowledgments
This research was funded by the “National Key R&D Program of China (2020YFB1708602, 2020YFB1708600)” and the “Foundation of Guizhou Provincial Key Laboratory of Public Big Data (No. 2017BDKFFJ015, No. 2018BDKFFJJ008, No. 2018BDKFFJJ020, No. 2018BDKFFJJ021).”

References


