Automatic Reverse Engineering of Unknown Security Protocols from Network Traces

Yudan Fan*, Yuna Zhu, Lin Yuan
Department of Information Security
Zhengzhou Information Science and Technology Institute
Zhengzhou, China
e-mail: 1151463213@qq.com

***Abstract*—Previous network-based protocol reverse engineering methods have only considered plaintext format of payload, and have not been suitable for security protocols which include many ciphertext data. We propose a novel approach to reverse security protocols from network traces–named SPREA (security protocols reverse engineering approach). SPREA extracts protocol keywords sequences hierarchically using sequential pattern mining for the first time, which would provide a new idea for plaintext format parsing. On this basis, SPREA utilizes the randomness feature of ciphertext data to locate ciphertext fields based on entropy estimation. Then SPREA infers the state machine using sequential pattern mining and Prospex method. Finally we evaluate SPREA on four classical security protocols. The experimental results show that without using dynamic binary analysis, SPREA can parse true protocol format and infer state machine purely from network traces with high accuracy.**

Keywords-key managment; key aggragate; cloud storage; CPA security

I. Introduction

With the extensive use of cryptographic techniques, security protocols are widely used in many key network applications. Meanwhile, new attacks against security protocols are continuously increasing. The formal analysis approaches of security protocols are based on specific adversarial model to detect potential attacks, and can only give security analysis results ideally. When considering the dynamic factors during protocol running, the results are inaccurate. So the formal analysis can not ensure protocol security during actual running under the complicated and changing environment. Therefore, it is a key problem to analyze protocol security online and dynamically in the field of information security.

It is a prerequisite of dynamic security analysis to identify protocol, reconstruct protocol session and obtain the current running status of protocol based on detailed protocol specification. There are many private protocols with non-published specifications at present. Their security can not be analyzed using the existing protocol analysis tools, for example, Wireshark. Therefore, security protocol reverse engineering is important for dynamic security analysis of protocol.

Security protocols include much ciphertext information. Attackers can not decrypt cipertexts. They usually replay and relay ciphertexts to attack protocols. Therefore, not only plaintext formats need to be reversed, but also ciphertext features need to be extracted.

The current protocol reverse engineering methods are classified into program-based methods [1-6] and network-based methods [7-15]. The former [1-6] analyze a protocol using a program that implements the protocol as input sources, and can reverse cryptographic protocols. Prospex [3] first constructs an augmented prefix tree acceptor (APTA) T to accept all the packet sequences in the training set, and then use Exbar algorithm to find the minimal DFA that is consistent with T. The obtained minimal DFA represents protocol state machine. ReFormat[4] is a system that aims at driving encrypted message format. It monitors the program's processing of the encrypt messages, locates the buffers that contain the decrypted data, but can only handle programs where there exists a single boundary between decryption and normal protocol processing. To solve the problem, Dispatcher [5-6] extends Reformat to identify every instance of encryption, hashing, compression and obfuscation, and to identify the buffers holding the unencrypted data before encryption, but the universality and accuracy of its identity strategies need to be validated further. Program-based methods use dynamic data flow analysis to understand how a program consumes an input message, but the protocol program may not be obtained. They are hard to realize and their applications are limited. Network-based methods analyze a protocol using network traces which are conveniently collected when a protocol parsing program is unavailable. Compared with program-based methods, they have fewer limitations and would be used wider. The Protocol Informatics (PI) project [7] parses protocol message formats using multiple string alignment algorithms found in the bioinformatics field. Discoverer [8] proposes recursive clustering and type-based sequence alignment to infer message formats. In [9], the latent relationship among n-grams is discovered by grouping protocol messages with the same semantics, and then message formats are inferred by keyword based clustering and cluster sequence alignment. The above methods only extract message formats and fields, not considering protocol state machines inference. It is necessary for online security analysis of security protocols to consider the order of protocol messages and to infer protocol state machines. PEXT [10] divides protocol sessions into different conversations. Each conversation represents a protocol state, and the protocol state machine is inferred according to state transform sequences. BFS [11] extracts the relevant fields from arbitrary protocols to construct a state model. Wang et al. [12] propose P-PSM to infer a probabilistic protocol state machine. Luo et al.[13] parse massage formats based on the Apriori algorithm and infer protocol state machines. Bossert G et. Al[11-12] initiate the open-source project Netzob [14-15]. On the basis of an extension Angluin L*，Netzob inference Mealy automata of protocols using network traces. Network-based methods only consider the plaintext information of protocol, and don't apply to security protocols which include many ciphertext data.

In this paper, we propose a new reverse engineering method for security protocols, named SPREA. SPREA extracts protocol invariable fields, variable fields, ciphertext length fields and ciphertext fields, purely from network traces, and infers protocol state machines. The main contributions of this work are: 1)In the keywords sequences extraction phrase, we mine the ordered keywords sequences, using sequential pattern mining for the first time, to parse protocol invariable invariable fields, variable fields. 2)In the ciphertext fields identification phrase, on the basis of the extracted keywords sequences, we utilize byte sample entropy and entropy estimation to locate ciphertext field according to the randomness feature of ciphertext, providing a new idea for ciphertext information utilization. 3)In the state machine inference phrase, we obtain the main state transition path by sequential pattern mining, and further discover protocol state machine based on Prospex[3] to character the protocol behaviors.

II. PROBLEM STATEMENT

Definition 1: Protocol session refers to a complete interaction between protocol participants .

Definition 2: Protocol packet payload sequence refers to the order of bytes in packet payload, denoted by $p=<s_1,s_2,\ldots s_i,\ldots,s_n>$, where s_i （$1\leq i\leq n$）belongs to a hexadecimal number set ={00, 01, …, FF}. n=|p|>0. |p| is the length of p. The finite sequence set composed of the same protocol's p is referred as protocol sequence set.

Definition 3: Protocol keywords refer to the protocol constants which represent packet type and transmit the control information.

Protocol keywords include protocol name, version, control codes and so on. They occur frequently in protocol traces. According to the position of keywords, they are divided into fixed offset keywords and non-fixed offset keywords. There are one or more keywords in protocol packets for mostly network protocols [1]. The control information and user data are effectively distinguished using keywords.

In view of the fact that data mining is the most effective means for data analysis, classical association rule mining methods are used for protocol keywords extraction[13][16], but they only consider fixed offset keywords, and can not handle the non-fixed offset keywords. Sequential pattern mining extends association rules mining further to discover frequent subsequences as patterns in a sequence database, widely used in DNA sequences analysis, Web access pattern analysis, etc.
In this paper, we first extract the ordered keywords sequences based on sequential pattern mining. A Keyword belongs to an invariable field, while bytes between two keywords belong to a variable field. Then we locate ciphertext fields in variable fields and infer protocol state machines further.

III. SPREA METHODOLOGY

A. SPREA Architecture

SPREA architecture includes four phrases, as shown in Fig. 1.

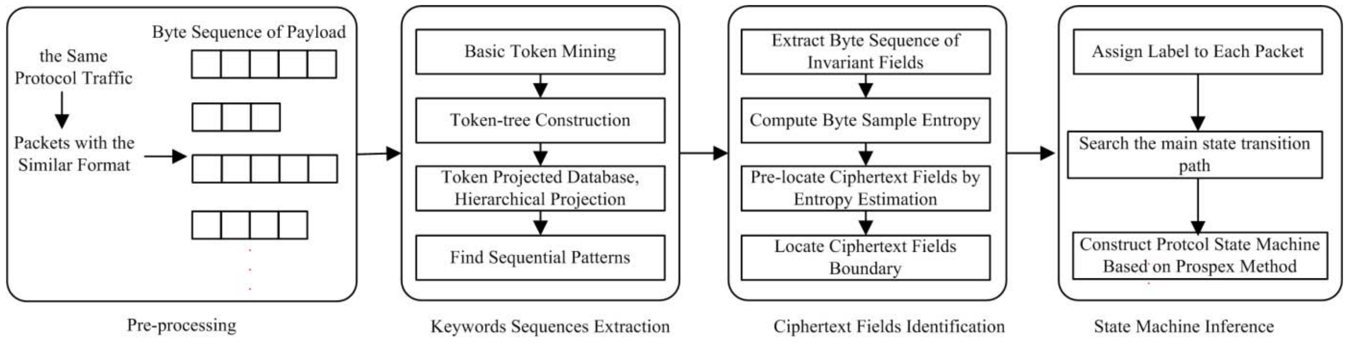

Byte Sequence of Payload
the Same Protocol Traffic
Packets with the Similar Format
Basic Token Mining
Token-tree Construction
Token Projected Database, Hierarchical Projection
Find Sequential Patterns
Extract Byte Sequence of Invariant Fields
Compute Byte Sample Entropy
Pre-locate Ciphertext Fields by Entropy Estimation
Locate Ciphertext Fields Boundary
Assign Label to Each Packet
Search the main state transition path
Construct Protcol State Machine Based on Prospex Method
Pre-processing
Keywords Sequences Extraction
Ciphertext Fields Identification
State Machine Inference

Figure 1. SPFPA architecture

1) Data pre-processing: we run the same protocol program in each host of LAN, capture packets by Wireshark, and filter the specified protocol traffic by filter rules for public datasets. Then the pure protocol traffic is clustered using statistical features, i.e. packet length, direction and offset, to obtain the same type packets, and the protocol packet payload sequences in the same cluster are extracted further.

2) Keywords sequences extraction: We extract the ordered keywords sequences based on sequential pattern mining and consequently identify protocol invariable fields and variable fields. ①We mine basic tokens of packet payload sequences set in the same cluster—1-position-token with fixed-offset and min-length-token with min_len adjacent bytes. ②We build token-tree to store the basic token information. ③We establish token projected database based on token-tree. Then we mine protocol pattern hierarchically, and further obtain the ordered keywords sequence.

3) Ciphertext fields identification: On the basis of keywords sequences extraction, we parse the variable fields between two keywords further. ①We utilize byte sample entropy and entropy estimation to pre-locate ciphertext fields according to the randomness feature of ciphertext. ② We search ciphertext length fields heuristically, and further locate ciphertext fields.

4) State machine inference: We assign label to each packet, construct the main state transition path by sequential pattern mining, and further infer protocol state machine based on Prospex method [3].

B. Keywords Sequences Extraction

We analyze the features of sequential pattern mining for security protocols, and propose a protocol sequential pattern mining method to extract the ordered keywords.

1) Features of Sequential Pattern Mining for Security Protocol: The existing sequential pattern mining methods only consider the values of sequence elements, ignoring the order of elements. They assume adjacent elements are independent of each other. But a protocol sequence pattern has its specific requirements:

- The frequent patterns with adjacent bytes or fixed position are more meaningful than others. ① Protocol keywords are composed of one or more bytes. Adjacent frequent patterns have more semantic information than others. ②A fixed-offset keywords are closely related to a position. Using sequential pattern mining directly will produce many redundant and short patterns, so we should utilize the frequent patterns with adjacent byte or fixed position to improve the efficiency of protocol sequential mining.
- We should mine protocol keywords sequences hierarchically and sequentially and need not discover all the protocol sequential patterns. The byte sequence of a protocol payload is ordered and continuous. The keywords are also in order. So we should extract keywords sequences combing with the ordered feature.

In conclusion, to meet the needs of protocol sequential pattern mining, we will design a sequential pattern mining method for security protocol to extract protocol keywords.

2) Basic Token Mining: We mine frequent bytes with fixed position for fixed-offset keywords, and frequent adjacent byte sequences which are greater than min_len for non-fixed offset keywords.

Definition 4: Support sup: Given protocol sequences set P (see definition 2), where $p \in P$, if a sequence x is a subsequence of p, then s(x)=1, else s(x)=0. The support of x in P is defined as sup(x)= $\sum_{p \in P} s(x)/|P|$, where |p| is the total sequence number of P.

Definition 5: 1-poistion-token: Given a user-specified min_sup threshold, if a byte x always appears at the fixed position, and sup(x)≥min_sup, then the byte x is referred as 1-position-token.

Definition 6: min-length-token: Given two user-specified thresholds—min_sup and min_len, if x is a adjacent byte sequence with min_len length, and sup(x)≥min_sup, then sequence x is referred as min-length-token.

We encode the raw byte sequence of a packet payload. Each byte is encoded as 5 characters. The first three are hexadecimal value of the byte offset. The fourth and fifth are hexadecimal value of the byte. Then we mine frequent 1-items in encoded protocol sequences set to find 1-position-tokens and the corresponding position information.

A byte in a min-length-token may be a 1-positon-token. To mine min-length-tokens which are disjoint with 1-position-token set, we encode the bytes except for 1-positon-tokens in a raw payload sequence, i.e., the sequence between two 1-poistion-tokens X=$b_1,b_2,\ldots,b_k$ (for the last 1-position-token, X is the byte sequence after the token) is encoded. Each element is bytes with min_len length, X'=$b_1\ldots b_{min\_len}$, $b_2\ldots b_{min\_len+1},\ldots,b_{k-min\_len+1}\ldots b_k$. Then we mine min-length-tokens in the encoded sequence set.

3) Token-tree Construction: Combining with the ordering feature of tokens, we present a new tree structure based on FP-tree[17] to store basic tokens information of protocol sequences.

Definition 7: Token-tree: ①Token-tree is composed of root, prefix sub-tree constituted by basic tokens, and basic token head table. ②Each node (except root) is represented by (token-name, count, offset, overlap, pnode), where token-name is the name of basic token related to the node; count is the number of sequences which include the token; offset is the token's position, when position is non-fixed, it is null; overlap is the number of bytes which overlap its parent node; pnode represents the parent node thread. A n-length-token with n byte length Tn=b1b2…bn (n>min_len) can be represented as connection of t basic tokens ($\left\lceil \frac{n}{\min\_len} \right\rceil \le t \le n - \min\_len + 1$). In order to include the total basic tokens for token-tree, the value of overlap is:

$$overlap = \begin{cases} 0 & \text{if a node and its parent node are disjoint and their positons are adjacent} \\ min\_len\text{-}1 & \text{if a node and its parent node are overlap with } min\_len\text{-}1 \text{ bytes and} \\ & \text{their positons are adjacent} \\ null & \text{if the positions of a node and its parent node are nonadjacent} \end{cases}$$

③Each tuple of token head table is composed of token name and Node-link (a pointer to the first node with the same token name in token-tree). Pointers of the same token are connected to a link. ④Considering the ordering of protocol byte sequence, each branch's nodes are ordered in accordance with that they appear in protocol sequences. In the prefix sub-tree of basic tokens, if a sequence's prefix is not the same as any sequence's of the existing branches in token-tree, we create a new branch from the root node.Tree with the above characteristics is referred as token-tree.

Definition 7: Tokens join: If t1 and t2 are basic tokens, $t_1 = b_1 b_2 \cdots b_l$, $t_2 = b'_1 b'_2 \cdots b'_{l'}$, and the last m bytes of t1 are the same as the first m bytes of t2 (m<l，m<l') , then the join of t1 and t2 is $b_1 b_2 \cdots b_m b'_{m+1} \cdots b'_{l'}$, denoted $t_1 \|_m t_2$.

Definition 8: Token sequence: A node of token-tree is denoted by nodek =(tk, countk , offsetk, overlapk, pnodek). For any two nodes nodei and nodej in a branch of token-tree, the path between the two nodes $node_i \rightarrow node_{i+1} \ldots \rightarrow node_j$ represents token sequence $t_i \|_{overlap_{i+1}} t_{i+1} \|_{overlap_{i+2}} \cdots \|_{overlap_j} t_j$.

Token-tree Property 1: The support of any node (except root) is not less than the support of its descendant nodes.

Proof: When multiple sequences include tokens with the same prefix, the tokens are stored by the same branch in token-tree. Therefore, if a protocol sequence includes a node of token-tree, then it must include all ancestor nodes of the node, and consequently ancestor nodes' support is greater than the node's.

Token-tree Property 2: For a branch path from root to a token node nodek , the support of the path's corresponding sequence is equal to the support of nodek.

Proof: Protocol sequences are ordered. The path also follows the order of protocol sequences. From property 1 we can see that when a branch include nodek, it must include all the ancestor nodes of nodek —node1, node2, …nodek-1. Therefore, the support of sequence corresponded to root$\rightarrow node_1 \rightarrow node_2 \ldots \rightarrow node_k$ is equal to the support of $node_k$

4) Protocol Sequential Pattern Mining: Protocol packets are parsed hierarchically from left to right. Consequently, we generate token projected database based on token-tree to mine protocol keywords sequences hierarchically.

PrefixSpan[18] need not generate candidate frequent itemsets. Its idea is that the prefix sequence of a frequent sequence is also frequent, and a sequence can be obtained by extending prefix sub-sequence through limited steps. Inspired by PrefixSpan, we construct token projected database to mine protocol sequence patterns.

Definition 10: Token projection and token projected database: Given two token sequences x1 and x2, where $x_1 = t_1 \|_{l_1} t_2 \|_{l_2} \cdots \|_{l_{n-1}} t_n$, $x_2 = t'_1 \|_{l'_1} t'_2 \|_{l'_2} \cdots \|_{l'_{m-1}} t'_m$ (m<n，l_i is the overlapped bytes between t_{i+1} and t_i，l'_i is the overlapped bytes between t'_{i+1} and t'_i), x_2 is a sub-sequence of x_1 ($x_2 \neq x_1$), if a token sequence $x_3 = t''_1 \|_{l''_1} t''_2 \|_{l''_2} \cdots \|_{l''_{m-1}} t''_m$ (m<k=n) exists, which satisfies 1) x_2 is a prefix of x_3, 2)x_3 is the longest sub-sequence of x1 which satisfies 1), then x_3 is the projection of x1 with regard to x2. In protocol sequence set, the set composed of all the x2's suffix sequences in x_3 is referred as the projection database of x2.

The projection of any protocol token node ti is the corresponding sequence of the path from ti to leaf nodes. In protocol sequence set, the projection database of ti is the set composed of sequences corresponded to all sub-tree after ti .

PrefixSpan projects for all the frequent 1-item iteratively. But protocol sequences are in order. In a branch of token-tree, the projection of a node includes the projections of its descendant nodes. We project hierarchically in accordance with the hierarchical order of token-tree, without generating projections of all the frequent tokens, as shown in fig. 2.

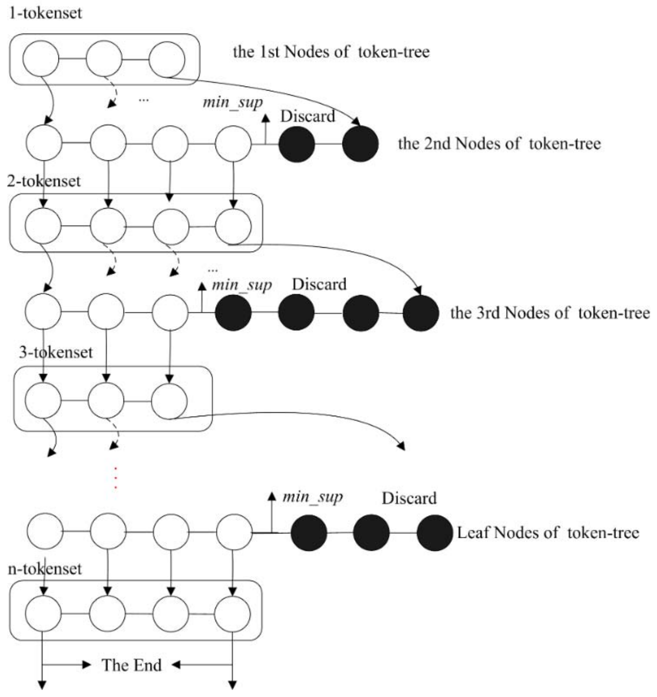


Figure 2. Sequential pattern mining

We discover 1-tokenset Token1—a set composed of the first tier token.

- We project with regard to each token in Token1. In the projection database of token1 (token1∈Token1), we search vertically according to the branches of token-tree, and compute the support of the second tier tokens (the descendant nodes of token1, denoted token2). ① If sup(token2)≥min_sup in the projection of token1, we continue to project token2 iteratively. i.e., in the projection database of token2, we search the third tier tokens vertically (the descendant nodes of token2, denoted token3).②If sup(token2) <min_sup in the projection of token1, according to property 1 in section 3.2.3, the support of token2's descendant nodes is less than min_sup, then we will not project for the branch.
- In the projection of tokeni, we compute the support of the (i+1)th tokens (the descendant nodes of tokeni, denoted tokeni+1). We do this iteratively until reach leaf nodes of each branch in token-tree, and consequently extract keywords sequences of packets in the same cluster.

We analyze the features of sequential pattern mining for security protocols, and propose a protocol sequential pattern mining method to extract the ordered keywords.

C. Ciphertext Field Identification

1) Ciphertext Field Pre-location: Protocol fields are always in bytes[8]. We take payload bytes with the same offset as a random variable, and describe byte distribution by entropy. Security protocols always use fresh random numbers which are not usually contained in other network protocols to prevent attacks. We take random number fields as generalized ciphertext fields.

Definition 9: Byte sample entropy: Given a set with the same type protocol packets $\Sigma = \{packet_i\}$ $(1 \le i \le N)$, if x_j is a byte variable with offset j in packets (the value range of x_j is 0-255), the byte sample entropy of x_j is $H(x_j) = -\sum_{k=1}^{255} f_k \log f_k$, where f_k is the frequency of x_i with value k in $\{packet_i\}$.

A ciphertext byte variable satisfies the uniform distribution approximately [19]. If a byte belongs to a ciphertext field, it takes random values in network flows. Its entropy is maximal because of its uncertainty. The value of a plaintext byte variable is in certain extent, because protocol specification has specific semantics. If a byte belongs to a plaintext field, its entropy is less than the entropy of a ciphertext byte. So we present a ciphertext field pre-location algorithm based on entropy, as shown in algorithm 1.

Algorithm 1. Ciphertext field pre-location algorithm
Input: payload byte sequence *P* of packets with the similar format, protocol keywords sequences;
Output: ciphertext distribution interval *I*= $\{I_1, I_2, \ldots, I_K\}$, the interval number *K*.

```
1: For each keyword K
2:     find the next Keyword K' and obtain the byte sequence between the two keywords;
3:     for each byte variable $B_i$
4:         E=ByteEntroyCompute($B_i$);
5:         E'=N-truncated-entroyCompute ();
6:         CompareEntroy(E,E'); // to determine whether the byte belong to ciphertext
7:     End for
8:     Record the ciphertext Byte;
9:     Merger the continuous ciphertext Byte as ciphertext interval ;
10: End for
11: Return I, K
```

- Compute byte sample entropy : On the basis of keywords extraction, we extract the byte sequence between two keywords, take bytes with the same offset (we count offset from the position of the front keyword) as random variables, and compute byte sample entropy $H(x_j) = -\sum_{k=1}^{255} f_k \log f_k$.
- Entropy estimation: The uniform distribution is the maximum entropy distribution among all continuous distributions. Its entropy is $-\sum_{k=1}^{255} \frac{1}{256} \times \log \frac{1}{256} = \log 256 = 8$ ideally. But byte sample entropy is related to the selected sample. If the sample length N is finite, especially N~256 or N<256, there are errors between the value of byte sample entropy and 8. Only when $N \to \infty$, the value of byte sample entropy is 8 approximately [20]. So we can note compare the value of byte sample entropy with 8 directly.

To identify ciphertext byte variables, we use entropy estimation to compute the entropy of uniform distribution with *N* sample numbers ($N > 16$[21]), and compare it with the value of byte sample entropy. When $p = \{p_i\}_{i \in \{0,1,\cdots,255\}}$ satisfies the uniform distribution, N-truncated entropy $H_N(p) = \frac{1}{256^N} \sum_{\substack{n_0,\cdots,n_{255} \in \mathbb{N} \\ n_0+\cdots+n_{255} \in \mathbb{N}}} \left[\binom{N}{n_0,\cdots,n_{255}} \times \left(\sum_{i=0}^{255} -\frac{n_i}{N} \log \frac{n_i}{N} \right) \right]$ [20] is an unbiased estimator, where n_i is frequency of value i, $\binom{N}{n_0,\cdots,n_{255}} = \frac{N!}{n_0! \cdots n_{255}!}$. Specifically, ① generate all possible sample of length N according to *p*; ②estimate the entropy based on Maximum Likelihood Estimation (MLE); ③take the average of the MLE estimates as $H_N(p)$.

- Pre-locate distribution interval of ciphertext: ① Compute N-truncated entropy $H_N(p)$ of the uniform distribution and its confidence interval $H_N(p) \pm 4 \times SD(\hat{H}_N^{MLE})$ by Monte-Carlo method according to [10] (When *N*>16, the confidence level is 99.9% approximately[10]), where $SD(\hat{H}_N^{MLE})$ is standard deviation of $\hat{H}_N^{MLE}$. ② Compare byte sample entropy with $H_N(p)$. If a byte entropy lies confidence interval, the byte satisfies the uniform distribution and is a ciphertext byte. ③Merger the continuous ciphertext bytes as ciphertext interval.

2) Ciphertext-field Boundary Location: The relative position with regard to keywords and the length of a ciphertext may be variable. Therefore the pre-located ciphertext interval may be not ciphertext field boundary.

A ciphertext format is as follows. ① Length field|| ciphertext data with variable length. The length of a ciphertext is related to cryptographic algorithm, plaintext length and key length. It is usually variable. Variable fields must use a length field before it, so that the protocol receiver can parse it.②Ciphertext data with fixed length. In this case, there is not a length field before ciphertext possibly.

According to the relative position of a ciphertext and a keyword, there are two cases. ①The relative position of a ciphertext is fixed. If the ciphertext length is fixed, the pre-located ciphertext interval is the ciphertext field boundary, or else, the start position of pre-located ciphertext interval is also that of the ciphertext field, but the end position of the ciphertext field is uncertain. ②The relative position of a ciphertext is variable. The pre-located ciphertext interval is a part of ciphertext field, but the boundary of ciphertext field is uncertain.

Therefore, after pre-locating ciphertext interval, we search ciphertext length fields heuristically and locate the boundary of ciphertext fields further (we mainly consider length fields represented by hexadecimal numbers).

The identify strategies of ciphertext length fields are as follows. ①Ciphertext length fields are usually represented by hexadecimal numbers of 1~2B length. The length of a protocol ciphertext is limited, usually 128b~4096b, and the corresponding decimal number is 16~512. ②A ciphertext length field is related to a ciphertext length. The length changes with the ciphertext length. ③ Ciphertext length field can not appear after its related ciphertext.

The length of pre-location ciphertext interval I_i by section 3.3.1 is denoted as l_1 (in B). The byte sequence between the start keyword and I_i 's start position is denoted as *BS*. In BS, the length between a byte and the next keyword is denoted as l_2 (in B). The ciphertext-field boundary location algorithm is shown in algorithm 2.

```
Algorithm 2: Ciphertext-field boundary location algorithm
Input: ciphertext field interval $I= \{I_1, I_2,…,I_K\}$, the interval number $K$.
Output: ciphertext field $C= \{C_1, C_2,…,C_m\}$, ciphertext field number $m$.
1: For each distribution interval $I_i$
2:    If $l_1 \geq 16$ then
3:        $n$=OneByteValue($BS$);
4:        m=TwoByteVaule($BS$);
5:        if $l_1 \leq n \leq l_2$ or $m \leq min(l_2, 512)$   then
6:              ByteRocord(); // record the byte which is a possible ciphertext length field
7:        End if
8:    End if
9:    L=CiphertextLength ($n,m$); // determine ciphertext length field
10:    If   L=Null then
11:        PlaintextLengFieldSearch($BS$);
12:        Obtain ciphertext field hierarchically;
13:    End if
14: End for
15: Return $C, m$
```

- Searching all the possible ciphertext length fields in BS. If $l_1 \geq 16$, there may exist a ciphertext field between keywords. We compute the values of each byte *n* and each two continuous bytes *m*. ① if $l_1 \leq n \leq l_2$ or $m \leq \min(l_2, 512)$, we record a length vector (the start keyword *K*, the end keyword *K'*, the relative offset *O* of byte with regard to *K*, byte value, the corresponding byte sequence); ② When *m*=*n* (for example both "00 C0" and "C0" are length fields), the length fields are overlap, and we take *m* as the preferred length field.
- •Determining if the recorded *n* and *m* are length fields further. If the intersection of the length vectors (the start keyword *K*, the end keyword *K'*, the relative offset *O* of byte) is not null, *O* is fixed, the intersection element is the correct length vector and the corresponding byte sequence is the ciphertext field; else *O* is variable, we will search plaintext length fields further to identify plaintext fields, and parse packets left-to-right hierarchically to locate the ciphertext field boundary.
- •Obtaining the ciphertext field boundary when there is not a ciphertext length field. In this case, the length of the ciphertext is fixed. We will search plaintext length fields further. If there is not a paintext length fields, the paintext length is also fixed and the relative offset of the ciphertext is fixed, consequently the pre-located ciphertext interval is the ciphertext field; else we identify plaintext fields using plaintext length fields, and parse packets left-to-right to locate the ciphertext field boundary.

D. State Machine Inference

State machine inference algorithm is shown in algorithm 3.

```
Algorithm 3. State machine inference algorithm.
Input: clusters of protocol CP, keywords sequence set K, ciphertext fields C
Output: protocol state machines.
1: TypeCluster=Clusterhandle(CP, K);
2: LabelAssign(TypeCluster);
3: represent each session as a label sequence, obtain label sequences set L
3: PrefixSpan(L)
4: GetLonegestPattern();
5: M=GetMainPath();
6 :T=APTA(L, M)
7: Exbar(T)
8: Return state machines
```

The goal of state machine inference is to reveal the order in which different types of messages should be sent and to character the protocol behaviors. This object can be achieved by three steps.

1) Assigning a label to each packet: The packets in the each previous cluster maynot be with the same type. For each cluster, we extract packets with the same type using the keywords extraction result, remove packets with other type, and further construct a new type cluster where packets have the same type. Then each packet will be assigned with a state according to the type cluster result. Each session can be represented as a label sequence, where an element of the sequence is a packet type.

2) Searching the main state transition path: We mine protocol sequence patterns which occur in all packets of TypeCluster using PrefixSpan, and then get the longest sequence pattern as the main protocol path. Consequently, we build protocol state machine of main path.

3) Constructing the protocol state machine: Similar to Prospex, we build an augmented prefix tree acceptor (APTA) to accept all packet states in the same session, and leverage Exbar algorithm to find the minimal DFA which represents protocol state machine.

IV. Experiment Evaluation

SSL protocol and SSH protocol are widely used in network. Needham-Schroeder (NS) public protocol and sof protocol are classical security protocols. We evaluate SPREA on the four classical security protocols, and the result shows SPREA can infer true protocol format and state machine effectively with high accuracy.

A. Dataset

Protocol dataset is shown in table 1, where NS program and sof program are generated by Spi2Java [23], and run in each host of LAN. Lua is an embedded scripting language supported by Wireshark. After capturing packets by Wireshark, we extract the related information of protocols by Lua and experiment using SPREA.

TABLE I. Protocol Dataset

Protocol	Flow number	Packet Number	Data Sources
SSL	4000	28468	WAN
SSH	270548	846382	InfoVisContest[1]
NS	2000	24000	LAN
sof	2000	24000	LAN

B. Parameters Setting

We select 100 complete sessions from dataset for each protocol, as the training dataset. There are two important parameters—*min_len* for min-length-token and *min_sup* for the support threshold. Identification features are mostly included in the first protocol packet[24], so we set parameters according to the first packet.

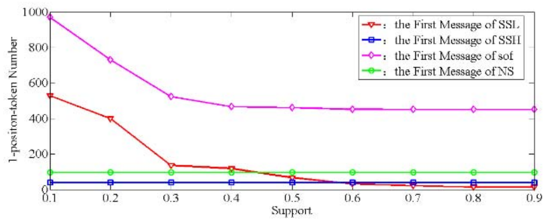


Figure 3. 1-position-token number under different min_sup

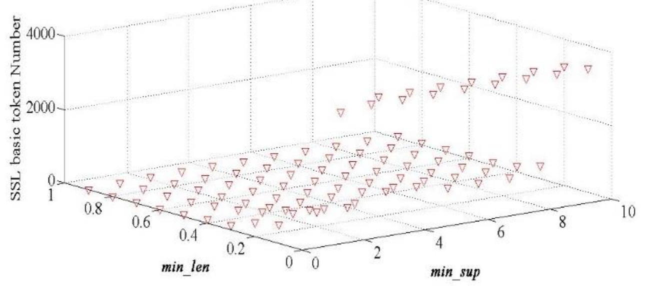
Fig. 3 shows 1-positon-token number under different min_sup. Fig. 4 shows basic token number under different min_sup and min_len. When $0.6 \le sup \le 0.9$, the mined basic token number begins to stabilize, so we set min_sup=0.6. When min_len=1, many redundant frequent items are extracted. When min_len≥2, the mined basic token number shows little change, so we set min_len=2.

C. Result

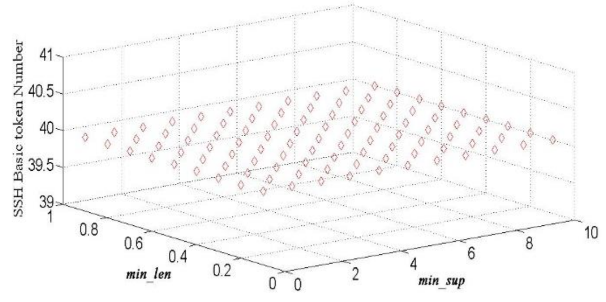
The existing network-based methods mainly use private dataset, and parse the plaintext format of protocol and it is difficult to compare the methods with ours. In order to validate SPREA, we evaluate the performance of state machines using two metrics—recall and false positive. For each protocol, we select M sessions as the training set, used for inferring protocol state machine; the others is as the testing set, used for evaluating the inferred state machine. The recall of inferred state machines is used for measuring the coverage of the inferred automata, and is defined as the

[1] http://2009.hack.lu/index.php/InfoVisContest

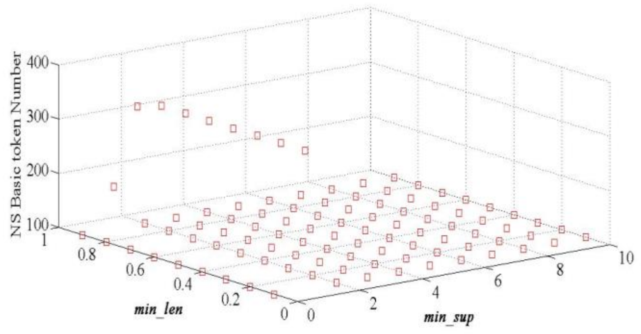
percentage of the valid sessions which are accepted by the inferred automata. The false positive of inferred state machines is used for measuring the soundness of the inferred automata, and is defined as the percentage of the invalid sessions which are accepted by the inferred automata.



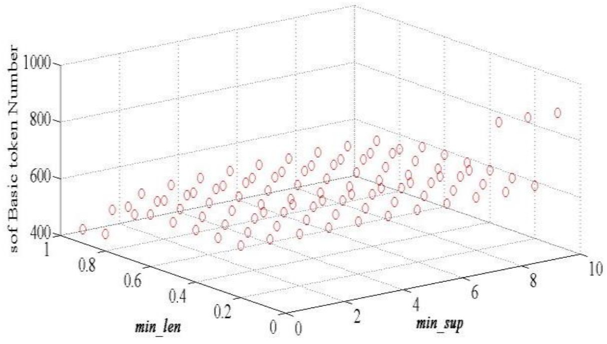
(a) SSL



(b) SSH



(c) NS



(d) sof

Figure 4. Basic token number under different *min_sup* and *min_len*

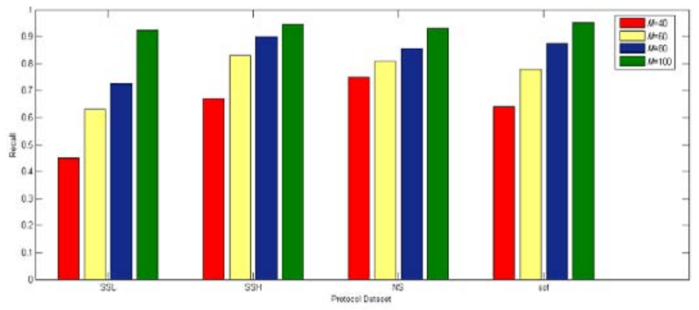
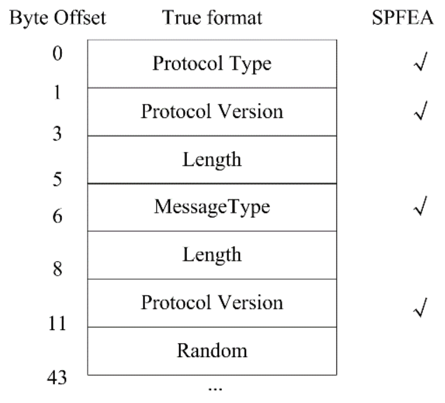
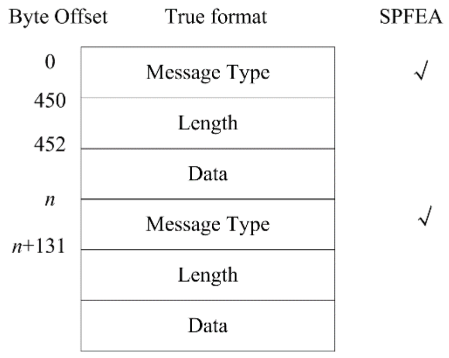


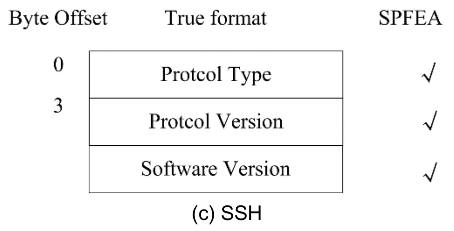
Figure 5. Recall under different training set size *M*

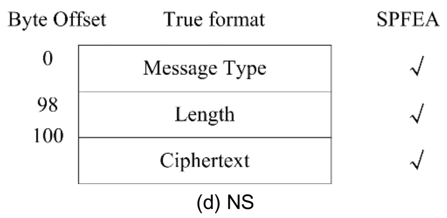


(a) SSL



(b) sof





Note: "√" represents correct parsing

Figure 6. Protocol format parsing result of the first packet.

The metrics under different training set size M is shown in fig. 5. If M is small, not enough message diversity is available to support the construction of the automata. When M≥100, the recall is above 92.3%. On the other hand, the false positive is 0 under different M. The inferred state machines are of good quality.

In order to evaluate the performance of the inferred message formats, when M=100, we compare the inferred message formats with published protocol specifications and Wireshark parsing results (for example, fig 6 is Protocol format parsing result of the first packet). It can be seen that SPREA can parse message formats for unknown security protocol effectively.

V. CONCLUSIONS

We propose SPREA for security protocol reverse engineering. SPREA extracts protocol keywords sequences based on sequential pattern mining at the first time, which would provide a new idea for plaintext format parsing. Then SPREA locates ciphertext length field and the corresponding ciphertext field, and infers protocol state machines further. We evaluate SPREA on four classical security protocols. The results show that SPREA can reverse protocol effectively purely from network traces with high accuracy.

However SPREA has limitations as follows. 1) Parameters—min_len and min_sup are selected by experimenting. Because the keywords distribution is different for different security protocols, the parameter setting may be also different. The parameter values in the paper are not the best for other protocols possibly. Though we can obtain the best parameter values by testing again, the process need manual intervention and is time-consuming. In future work, we should design an adaptive parameter setting method, which can adjust parameters adaptively for different protocols. 2) Because security protocols utilize many cryptographic techniques, we can not obtain protocol message format accurately and may not label packet types correctly. So we should introduce probability and confidence level to infer probabilistic state machine for security protocols

REFERENCES

[1] Caballero J, Yin H, Liang Zhenkai, et al. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis[C]. Proc. of the 14th ACM Conference. on Computer and Communications Security, New York: ACM, 2007. 317−329

[2] Cui Weidong, Peinado M, Chen K, et al. Tupni: automatic reverse engineering of input formats［C］. Proc of the 15th ACM Conference on Computer and Communications Security, New York: ACM, 2008: 391-402

[3] Comparetti P M, Wondracek G, Kruegel C, et al. Prospex: protocol specification extraction［C］. Proc of the 30th IEEE Symposium on Security and Privacy, Washington : IEEE, 2009: 110-125

[4] Wang Zhi, Jiang Xuxian, Cui Weidong, et al. ReFormat: automatic reverse engineering of encrypted messages［C］. Proc of European Symposium on Research in Computer Security, Berlin : Springer, 2009: 200-215

[5] Caballero J, Poosankam P, Kreibich C, et al. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering[C].Proc of the 16th ACM Conference on Computer and Communications Security, New York: ACM, 2009: 621-634

[6] Caballero J, Song D. Automatic protocol reverse-engineering: message format extraction and field semantics inference[J]. Computer Network, 2013, 57(2): 451-474

[7] Beddoe M.The Protocol information project[EB/OL]. [2004-10-05], http://www. tphi.net/ awalters/ PI.html

[8] Cui Weidong, Kannan J, Wang H J. Discoverer: Automatic protocol reverse engineering from network traces[C]. Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium. Berkeley: USENIX, 2007: 199-212

[9] Wang Y, Yun X, Shafiq M Z, et al. A semantics aware approach to automated reverse engineering unknown protocols[C]//Network Protocols (ICNP), 2012 20th IEEE International Conference on. IEEE, 2012: 1-10.

[10] Shevertalov M, Mancoridis S. A reverse engineering tool for extracting protocols of networked applications[C]//Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on. IEEE, 2007: 229-238.

[11] Trifilo A, Burschka S, Biersack E. Traffic to protocol reverse engineering[C]//Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE, 2009: 1-8.

[12] Yipeng Wang, Zhibin Zhang, Danfeng etc. Inferring Protocol State Machine from Network Traces: A Probabilistic Approach. Applied Cryptography and Network Security 2011, Volume 6715 : 1-18.

[13] Luo Jianzhen, Yu Shunzheng. Position-based automatic reverse engineering of network protocols[J]. Journal of Network and Computer Application. 2013(36):1070-1077

[14] Olivier Tétard. . Netzob. [EB/OL]. [2013-02-02], http://www.netzob.org/

[15] Bossert G, Guihéry F, Hiet G. Towards automated protocol reverse engineering using semantic information[C]. Proceedings of the 9th ACM symposium on Information, computer and communications security. New York, Honolulu, HI, 2014:51-62

[16] Yuan Zhenlong, Xue Yibo, Dong Yingfei. Harvesting Unique Characteristics in Packet Sequences for Effective Application Classification[C]. Proc. of the 1st IEEE Conference on Communications and Network Security (CNS), National Harbor: IEEE, 2013:341-349

[17] Han Jiawei, Pei Jian, Yin Yiwen. Mining frequent patterns without candidate generation[C]. Proc of ACM SIGMOD. New York: ACM, 2000:1-11

[18] Pei Jian, Han Jiawei, Mortazavi-Asl B, et al. Mining sequential patterns by pattern growth: The PrefixSpan approach[J]. IEEE Trans on Knowledge and Data Engineering, 2004, 16(10): 1-17

[19] Bonfiglio D, Mellia M , Meo M, etal. Revealing Skype Traffic: When Randomness Plays with You. SIGCOMM Comput. Commun. Rev., 37(4):37-48, 2007.

[20] Olivain J . Goubault-Larrecq J. Detecting subverted cryptographic protocols by entropy checking. Research report LSV-06-13, 2006.

[21] Paninski, L.: A coincidence-based test for uniformity given very sparsely sampled discrete data[J]. IEEE Transactions on Information Theory. 2008, 54(10), 4750-4755

[22] Pironti A, Pozza D, Sisto R. Spi2Java User Manual-Version 3.1[R]. Torino: Polytechnic University of Turin, 2008

[23] Aceto G, Dainotti A Donato W, et al. PortLoad: Taking the best of two worlds in traffic classification[C] //Proc IEEE INFOCOM. New York: IEEE Communications Society, 2010:1-5.