IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 18, 2019, accepted September 3, 2019, date of publication September 10, 2019,
date of current version September 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940512

A Security Analysis Method of Security Protocol Implementation Based on Unpurified Security Protocol Trace a nd Security Protocol Implementation Ontology

XUDONG HE[1], JIABING LIU[1], CHIN-TSER HUANG[2], DEJUN WANG[1], AND BO MENG[1]
[1]School of Computer Science, South-Central University for Nationalities, Wuhan 430074, China
[2]Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA

Corresponding author: Bo Meng (mengscuec@gmail.com)

This work was supported in part by the Fundamental Research Funds for the Central Universities, South-Central University for Nationalities under Grant CZZ19003 and Grant QSZ17007, and in part by the Natural Science Foundation of Hubei Province under Grant 2018ADC150.

ABSTRACT The security analysis of Security Protocol Implementations(SPI) is an important part of cybersecurity. However, with the strength of property protection and the widely used applications of code obfuscation technology, the previous security analysis method based on SPI is hard to carry out. Therefore, under the condition that SPI is not available, this paper analyzes the security of the SPI using the unpurified security protocol traces and security protocol implementation ontology. First, we construct the implementation ontology to describes the attributes of the ontology terms. Second, the format analysis method is presented based on unpurified flow. Third, the mapping method is proposed to build the mapping between the security protocol trace and the implementation ontology. Fourth, a is presented to analyze the security of SPI. Finally, FSIA software is designed and implemented according to the method we proposed to analyze the login module of a university information system, the result shows that there is a risk of Ticket leakage in the login module. Compared to the previous method,our proposed method can deal with unpurified network traces and find the vulnerabilities of network and system.

INDEX TERMS Security protocol implementation, network trace, security protocol implementation ontology, format analysis, semantic analysis.

I. INTRODUCTION

Security Protocol Implementations (SPI) have been an important part of cybersecurity [1]. Security protocols provide reliable and secure services to entities in a variety of network communications. Nowadays, there are a lot of network-based attacks that target the vulnerabilities of specific implementations. As such, only verifying the abstract specification of security protocols is not enough to guarantee cybersecurity. Therefore, analyzing the Security of Security Protocol Implementations (SSPI) [2], [3] has attracted the interests of researchers and experts from both academia and industry.

In general, the researches on the SSPI are based on one of two assumptions: with SPI and without SPI. Table 1 lists the Comparison of security analysis methods. Under the condition that SPI is available, researchers mainly use program verification methods [4] and model extraction methods [3] to analyze the SSPI. The program verification method applies the type system and logic proof, which depends on code comments and code assertions. The model extraction method extracts the formal model and then verifies the abstract specification of the security protocol implementation. Program verification methods and model extraction methods depend on obtaining and understanding SPI. Currently, obtaining the SPI [5] is impractical because, with the strengthening of intellectual property protection, it is difficult to obtain SPI. In addition, owing to the widespread application of code obfuscation technology, understanding and analyzing obfuscated codes is becoming a challenge for researchers and experts in academic and industrial worlds. Under the

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

TABLE 1. The comparison of the security analysis methods.

Security analysis methods	precondition	result
Program verification methods[4]	SPI	Security attribute analysis
Model extraction methods[3]	SPI	Security attribute analysis
Dynamic taint methods[6-8]	Program processing trace	Protocol specification
Network trace methods[15-27]	Purified network trace	Protocol specification
Traffic classification methods[9-14]	Unpurified network trace	Type of traffic
Our Method	Unpurified network trace Protocol implementation ontology	Type of traffic Protocol specification Security analysis

condition that SPI is not available, researchers use dynamic taint [6]–[8] and network trace to analyze the SSPI. Dynamic taint analysis technology analyzes the records of the instruction sequence during the program processing and then infers the protocol format and state machine of security protocols. Dynamic taint analysis technology is not suitable for the scenarios in which multiple security applications are running concurrently under different kernel execution environments. On the other hand, the network trace analysis method is suitable for most scenarios and applications. The network trace analysis method mainly analyzes the flow or packet by statistical methods. Network traces can be captured at various network nodes on different platforms, which is quite flexible. Different types of traffic can be classified by traffic classification methods [9]–[14], and then protocol reverses engineering methods [15]–[27] can be used to analyze the protocol format and state machine of security protocols.

The above works mainly focus on purified network trace and protocol specifications inference and pay little attention to the significance of unpurified security protocol traces to analyze SSPI. In the distributed networks, a lot of hosts provide different kinds of security services and produce many unpurified security protocols trace. Hence just focusing on the purified network trace is not appropriate and enough. Apart from that, most works use network-trace-based method to analyze the specification of network protocols, and seldom to analyze SSPI. So, with the unpurified security protocol traces and security protocol implementation ontology, this paper combines the traffic classification method and network-trace-based approach to analyze SSPI by verifying the consistency of security protocol trace and security protocol implementation. The main contributions of this paper are as follows:

(1) We propose the Security Protocol Implementation Ontology Framework (SPIOF) and build a Security Protocol Implementation Ontology (SPIO), which describes the attributes of the ontology terms.

(2) We present a security protocol format analysis method called Format Analysis Method based on the Unpurified Network Traces (FAMUNT), which takes the unpurified network trace sets as inputs and generates the protocol format. The FAMUNT method consists of trace segmentation, invariable field (IF) fitting, IF classification, trace clustering, and format inference.

(3) We propose a mapping method called Security Protocol Traces to Security Protocol Implementation Ontology (SPT2SPIO), which combines the greedy algorithm and the token weight to establish the mapping from the trace to the implementation ontology.

(4) We present a security analysis method called the Security Analysis Method of Security Protocol Implementations (SAMSPI), which applies the mapping analysis method SPT2SIO to analyze the consistency of the mapping and uses the non-ontology token analysis method to detect whether there exists message leakage in the non-ontology token or not.

(5) We develop FSIA software based on SAMSPI. The input of FSIA is an unpurified protocol trace and SPIO, and the output is SSPI analysis result.

(6) We analyze the Central Authentication Service (CAS) [28] protocol SPI of the login module of a university information system with FSIA. The result indicates that there is a risk of Ticket leakage in the CAS protocol SPI, which may cause the data exfiltration [13].

The rest of the paper is organized as follows. Section II discusses the related works of protocol reverse engineering methods. Section III presents the SPIOF, FAMUNT, SPT2SIO and SAMSPI methods. Section IV develops FSIA software. Section V evaluates the SAMSPI method. Section VI presents the conclusion and future works.

II. RELATED WORKS

Without SPI, many works apply network traces or program execution trace to infer protocol formats, protocol semantic, and FSM(finite state machine). While those works seldom pay attention to protocol security.

Tammo *et al.* [15] proposed the ASAP method, which extracts the relevant fields from the network payload and maps them to the vector space and then identifies the basic direction of the vector by matrix factorization. The n-grams method is used to infer the format and semantics of the message. Wang *et al.* [16]–[18] proposed the Biprominer, the Veritas, and the Prodecoder. The Biprominer first uses the machine learning method to obtain the message pattern, then marks the message pattern in the network trace, and finally uses the probability transition model to obtain the probability description of the protocol format. The Veritas adopts the protocol format by the Kolmogorov-Smirnov method and then infers the state machine by clustering format information. The Prodecoder infers the potential rational between n-grams by clustering the same semantic message and then infers the security protocol format based on clustering and sequential pattern mining. Luo and Yu [19] proposed the Autoreengine,

which uses the improved Apriori algorithm to extract frequently occurring strings from the payload, and then, it selects security protocol keywords from the frequent string set by analysis of variance based on the position characteristics. After that, it finds the keyword sequence set and infer the field of the message format. Finally, the security protocol state machine is obtained through the most frequent communication mode. Zhang *et al.* [20] proposed the Proword, it uses the improved Voting Experts algorithm to extract candidate words through the entropy information, then, proposes a ranking algorithm to establish the candidate word order structure. The highest order word is used as the security protocol feature words. Bermudez *et al.* [21] proposed the Field Hunter, which extracts fields and infers field types by the statistical correlation between different messages and metadata. These methods make two hypotheses: (1) separators are 1 or 2 characters at prefix byte of fields, (2) separators are the non-alphabet and non-number sequence. Luo *et al.* [22] proposed an application layer network protocol message modeling method based on left-to-right inhomogeneous cascaded hidden Markov model, which describes the evolution rule between states. The method infers protocol keywords by selecting lengths with maximum likelihood probability and then recovers the message format. Tao *et al.* [23] proposed the PRE-Bin, which uses hierarchical clustering to identify the number of clusters by silhouette coefficient, then applies the modified multiple sequence alignment algorithm to analyze binary field features, and finally uses the Bayesian decision model to infer the bit-oriented field boundaries. Zhang *et al.* [24] proposed an online protocol format extraction method, which divides the network traffic into sub-flows and introduces an error decision mechanism to divide the sub-flows to ensure the correctness of the format extraction. Marchetti and Stabili [25] developed READ to analyze traffic traces based on automotive data frames. It extracts the signal boundaries by analyzing the general CAN bus traffic trace and identifies and marks different types of signals encoded in the payload of data frames. READ improves Signal extraction and classification efficiency. Luo *et al.* [26] employed the Latent Dirichlet Allocation model to characterize messages with types, and then type distribution is used to measure the similarity of messages and promote the correctness of the message cluster. Goo *et al.* [27] proposed a method that can infer the protocol format, semantic and finite state machine. It extracts field formats, message formats, and flow formats as protocol syntax by using a contiguous sequential pattern algorithm hierarchically, infers the semantic by using predefined datatype, and infers the finite state machine by using protocol format.

III. SOLUTION

The framework of the Security Analysis Method of Security Protocol Implementations (SAMSPI), which is based on unpurified security protocol trace, is shown in Fig. 1. Firstly, the SPIO is constructed according to the SPIOF and the Security Protocol Implementation Specification (SPIS).

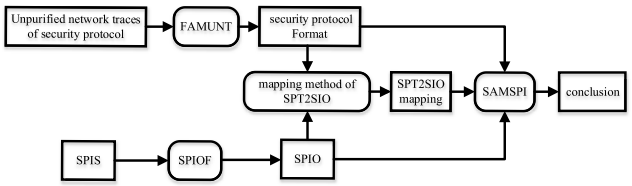


FIGURE 1. The framework of SAMSPI.

Secondly, the proposed FAMUNT method is used to purify the target network trace and produce the security protocol format. Thirdly, the SSPI is analyzed by the SAMSPI method which combines the mapping analysis method and the non-ontology Token analysis method. The mapping analysis method analyzes the correctness of the SPT2SIO mapping, and the non-ontology token analysis method detects whether there is a message leakage in the non-ontology token.

A. SECURITY PROTOCOL IMPLEMENTATION ONTOLOGY FRAMEWORK (SPIOF)

Due to the huge gap between the protocol specification and the protocol trace, the semantics of each keyword in the trace cannot be directly identified and marked using the protocol specification. The ontology can model the relationship among concepts and the relationship between concepts and instances in a general or special field. Therefore, we use the ontology to identify the semantics of each keyword. In the beginning, we construct the SPIOF, then build the SPIO, and finally, the semantics of each keyword in the trace is marked through the SPT2SIO mapping.

The SPIOF has three desirable characteristics: (1) it connects the instance and ontology through the mapping method, and measures the mapping result of the instance to the ontology; (2) the ontology provides the part of prior knowledge of the protocol semantics; (3) the ontology offers a way of message presentation, which represents the semantics of each keyword in the message and the relation between the keywords. Therefore, it is useful and beneficial to establish a SPIOF. The seven-step method is used to improve SPIOF, and then generate the SPIO.

The ontology framework is a tree structure that consists of a concept set. The SPIO is expressed by the three triples $O := \{C, H, R\}$, where C is the concept set, H is the hierarchical relationship of the concept, and R is the conceptual relationship. The SPIO can be customized according to different security protocols.

SPIOF in Fig. 2 is composed of the Flow, Msg, Token and rationales. Flow refers to the entire data stream generated in one session, which is composed of instances of Msg. Msg is generated in the process from the start to the end in one transmission. The token is a field contained in a Msg. Each Msg includes two fields of Msg Num and Token. The token is composed of Keywords, Separator, Variable Field (VF), Token Num, Token Length and Token Length Offset. Keywords are the label of a Token. The separator is the chars

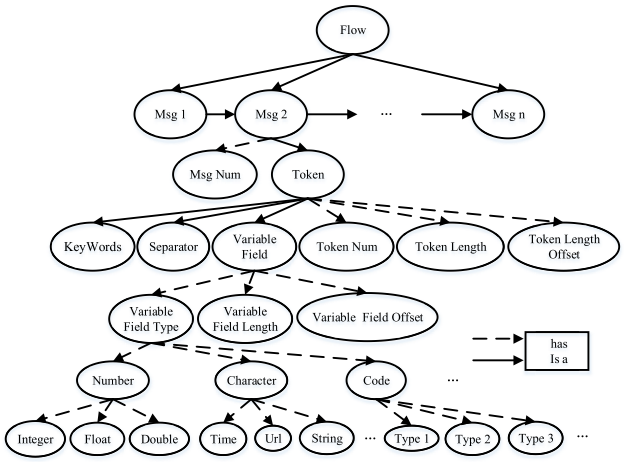


FIGURE 2. The structure of SPIOF.

between keywords and data. VF is the value of Keywords. Token Num is the sequence number of Token. Token Length is the length of the Token. Token Length Offset is the offset of the Token Length to the average Token Length. VF is composed of Variable Field Type, Variable Field Length, and Variable Field Offset. Variable Field Length is the length of the VF. Variable Field Offset is the offset of the Variable Field Length to the average Variable Field Length. Variable Field Type is the type of VF and it can be specified based on actual data types in the SPIO. For example, Variable Field Type can be Number type, Character type, and Code type. Number type can be integer, Float, and Double. Character type includes Time, Url and some String, etc. Code type is a specific type defined in the protocol.

B. FORMAT ANALYSIS METHOD BASED ON THE UNPURIFIED NETWORK TRACES (FAMUNT)

The framework of the FAMUNT method, as shown in Fig. 3 takes the unpurified trace sets as inputs and generates the format of protocol trace. FAMUNT consists of five methods: trace segmentation, IF fitting, IF classification, trace clustering, and format inference.

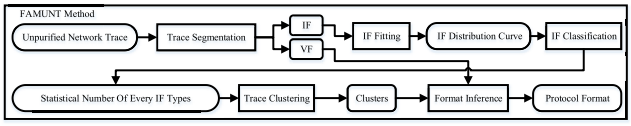


FIGURE 3. The framework of FAMUNT method.

The first step, trace segmentation method, borrows the idea of the Needleman Wunch algorithm, the trace is divided into VF and IF by comparing the result that two of traces which are the almost same length and use threshold to select the best result. In the second step, IF fitting method takes the IF distance as input, which is the distance from the IF to the first byte of the trace, and then, produces the IF distribution curve. First, it inputs the distance value, applies IF weighting method to generate the weight of the IF length, and then eliminate the illegal IF, finally it uses the B-spline method to construct the IF distribution curve based on B-spline. In the third step, IF classification method, it takes the IF distribution curve as input and generates the statistical number of every IF types. Newton CG method [29], [30] is introduced to calculates the curve extremum value. Then, we use the extremum value to split the trace into segments. After that based on the Levenshtein distance [31], it counts the statistics number of every IF type in every segment. In the fourth step, trace clustering method, it accepts the statistics number of every IF types in every segment as input and produces the trace clusters, in which it selects the largest number of IFs in every segment, and then uses K-Means method to generate the trace clusters. In the fifth step, the format inference method takes the trace clusters as input and generates the format of protocol trace. First, the trace cluster is used as input. After that, the statistics number of every IF types in every cluster is obtained. At last, it uses the Separator inference method introduced by us to infer the Separator based on the assumption that delimiter often occurs at the beginning and end of the IF and uses VF region generated by the curve to select the legal Separator. According to the separator, the trace is divided into the form of keywords, separator, and VF. Finally, the format of the security protocol trace is obtained.

1) TRACE SEGMENTATION METHOD

The trace segmentation method borrows the idea of the Needleman Wunch algorithm to divide the trace into VF and IF. We assumed that the security protocol traces with almost the same length also have similar states, which means that the distributions of keywords are relatively similar. First, unpurified network traces are sorted in reverse order by the length of the trace. Second, the Needleman Wunch algorithm is used to compare the two traces and then segment the trace into IF and VF. Third, we count the number of IF whose length is longer than 3; if satisfying this threshold, the matching is considered successful, otherwise select the next trace and go back to the second step. Finally, we count the IF distance. The trace segmentation method is shown in Fig. 4.

FIGURE 4. Trace segmentation method.

2) IF FITTING METHOD

The input of the IF fitting method is IF distance produced by the trace segmentation method. Because of inherent defects of the Needleman Wunch algorithm, an error matching is generated which may have a bad influence on the IF fitting. In the beginning, IF weighting method gives a weight value to every IF that is closely related to IF length and decreases mismatching by setting IF weight. Finally, the method uses B-spline to fit the IF weight and then produces the fitting. IF fitting method is shown in Fig. 5.

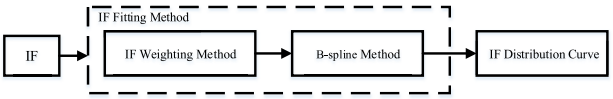


FIGURE 5. IF fitting method.

a: IF WEIGHTING METHOD

The IF weighting method assigns weights based on the length of the IF. The longer the length of the IF is, the greater the probability of a keyword belonging to the IF will be. For example, if there are four consecutive characters in two network traces, which means the length of the IF is 32 bytes, the weight is set as 3. As another example, if the length of the IF is 1 character, it can be deduced that there does not exist any keyword, because IF consists of Field delimiter [21], Key value and Keywords and thus the length of a normal IF is longer than 2 characters. In this case, the weight is set to 0. The IF weighting method is shown in Fig. 6.

$$w = \begin{cases} 0, & \text{if } 1 \le len \le 16 \\ 2, & \text{if } 17 \le len \le 24 \\ 3, & \text{if } 25 \le len \end{cases}$$

FIGURE 6. IF weighting method.

b: B-SPLINE METHOD

The B-spline method produces a function from a lot of discrete points, which not only models approximately the nonlinear relationship among the original discrete point sets and eliminates error data and interference, but also compensates for the missing data and predicts the future trends. The input of the B-spline method is the distance from the IF to the first byte of the trace and the weight W generated by the IF weighting method, and then the B-spline fitting method generates the fitting curve which represents the regular pattern of IF distribution. The B-spline equation is can be stated as below:

$$P(u) = \sum_{i=0}^{n} d_i N_{i,k}(u) \tag{1}$$

Here $d_i (i = 0, 1, ..., n)$ is the control point and $N_{i,k}(u)(i = 0, 1, ..., n)$ is the basic function of k-th order.

3) IF CLASSIFICATION METHOD

IF Classification method takes the IF fitting curve as input and uses the Newton CG method to generate the curve extremum. At last, it applies the IF statistical method proposed by us to obtain the statistics number of every IF types at each interval. The framework of the IF fitting method is shown in Fig. 7. Fig. 8 is the IF statistical method in IF Classification method.

a: NEWTON CG METHOD

Newton CG method can quickly solve the Newton equation. It takes the fitting curve and derivative function as input and calculates the extreme value of the curve.

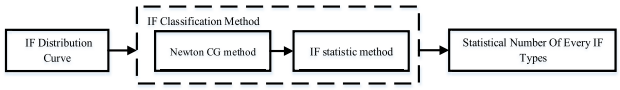


FIGURE 7. The framework of IF classification method.

Algorithm 3 IFClassificationMethod

Input: $Curve$, $UNTS[args3]$ – $UnpurifiedNetworkTraceSet$

Output: $IF[args1]$, $IFNum[args1]$

```
MaxNum[args1], MinNum[args1] ← NewtonCGAlgorithm(Curve, DerivativeCurve)
i ← 0
while !fefo < MaxNum > do
    X1[i] ← (MaxNum[i] + MinNum[i])/2
    X2[i] ← MaxNum[i]
    i ← i + 1
end while
PickIF[args1][args2] ← SelectPeakField(X1[args1], X2[args1], UNTS[args3])
PickIF[args1][args2] ← QuickSortDecend(PickIF[args1][args2])
function StrClassification(String1[arg], String2)
    flag ← 0
    i ← 0
    while !fefo < String1 > do
        if ELevenshteinDistance(String1[i], String2) > args then
            flag ← 1
            i ← i + 1
            Break
        end if
    end while
    if flag == 0 then
        String1[i + 1] ← String2
    end if
end function
i ← 0
j ← 0
while !fefo < Field > do
    while !fefo < Field[i] > do
        CallStrClassificationField[i][p], PickIF[i][j]
        j ← j + 1
    end while
    i ← i + 1
end while
i ← 0
while !fefo < Field > do
    IFNum[i] ← Field[i].Length
    i ← i + 1
end while
```

FIGURE 8. IF statistical method.

b: IF STATISTIC METHOD

IF statistic method applies interval (a, b), where $a = (x_{min} + x_{max}) \div 2$, $b = x_{max}$, x_{min} and x_{max} denoting the minimum and maximum points of the curve respectively. IF statistic method takes the IF in each interval as input and outputs the number of each type of IF in each interval according to the Levenshtein distance. IF is processed as a string. It uses the fast sorting method to sort the IF length in descending order and then calculates Levenshtein distance ratio from start to end. Finally, it classifies these IFs into different types with the corresponding threshold and counts the statistics number of every IF type at each interval.

4) TRACE CLUSTERING METHOD

The trace clustering method takes the statistics number of every IF type in each interval as input, selects the largest number of IFs in each interval and marks all the network traces according to the selected IF. Then, it converts network

trace to vectors and applies the K-Means method to generate the clusters.

5) FORMAT INFERENCE METHOD

The framework of the format inference method in Fig. 9 takes every trace cluster generated by the trace clustering method as input, and then uses trace segmentation method, IF fitting method, and IF Classification method to produce the number of the IF types and IF distribution curve. After that, the separator inference method infers the separator by comparing the heads and tails of IF and lifter IF by the VF region in the curve. At last, the IF is divided into separator and keywords, and the protocol format is obtained.

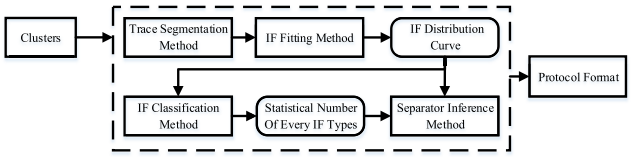


FIGURE 9. The framework of the format inference method.

The idea of the separator inference method is that the adjacent IFs may have similar separators. Separators often appear at the beginning and end of the IF and are shorter in length. The method first obtains the first 2 characters and the last 2 characters of the adjacent IF, and then compares the characters and counts the same ones. After that, it filters the separators by the VF region and generates the protocol format by combing with the keywords.

The use of the separator filter method is based on the following three conditions:

(1) The curve obtained by the IF fitting method in each trace cluster can effectively identify the VF region. VF region is distributed in the place where the curve is falling.

(2) The separator cannot appear in VF because VF is a variable field that consists of data, and the presence of a separator in VF can cause ambiguity.

(3) The first character of the delimiter either cannot appear in the VF region or appear with a low distance variance.

For condition (3), if the first character of the delimiter does not appear in the VF region, this character is regarded as a separator. Otherwise, we use distance variance to estimate. Distance variance gives information about the regular pattern of distance. If the first character of the delimiter appears in the VF region, we calculate the distance variance and use a threshold to identify. If the distance variance is lower than the threshold, we regard the character as a separator. Otherwise, we delete this character. The distance variance is calculated by the distance from the character to the IF which belongs to the same token. For example, in Fig. 10, "t = LT-" is an IF, "T-" and "t =" is Unfiltered separator. The $d'_{2,1}$, $d'_{2,2}$, and $d'_{3,1}$ is the distance of character "T." The distance variance of character "T" is d', which calculated by the $d'_{2,1}$, $d'_{2,2}$, and $d'_{3,1}$ using the variance calculation. d' is bigger than the threshold after computed the distance

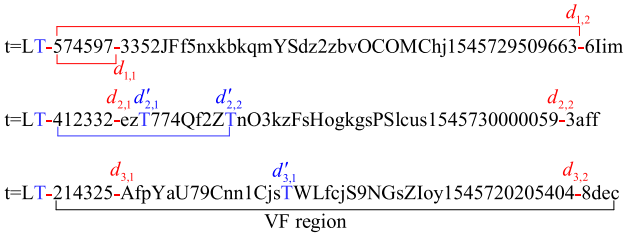


FIGURE 10. An example of separator liftering.

variance value. Therefore, "T" is the random character in the VF region. we remove the "T" from unfiltered separator "T-" to obtain the "-." In the same way, we lifter character "t." Then, we calculate the d by the distance $d_{1,2}$, $d_{1,1}, \ldots, d_{3,2}$, which is distance variance of "−," and it lower than the threshold, because "−" has a regular pattern of distribution. So, we treat "−" as a separator. As for "=," it not appears in the VF region, we treat "=" as a separator too.

C. MAPPING METHOD OF SECURITY PROTOCOL TRACES TO SECURITY PROTOCOL IMPLEMENTATION ONTOLOGY (SPT2SPIO)

We assume the security protocol implementation is not standard. So, it is very difficult to construct the relation between the trace and the implementation ontology. The captured security protocol trace is a Flow that consists of Msg and Token. The mapping method SPT2SIO in Fig. 11. combines the greedy algorithm and weight and establishes the mapping from the trace to the implementation ontology.

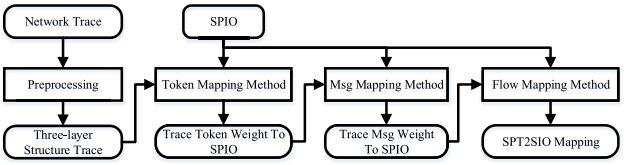


FIGURE 11. The framework of MAPPING method of SPT2SIO.

The mapping method of SPT2SIO includes four steps. The first step is preprocessing, which selects a representative security protocol trace and uses the protocol format generated by FAMUNT method to analyze selected trace into the three-layer structure which consists of Flow, Msg, and Token. The second step is Token mapping, which calculates the Token weight between the trace and the ontology through the weight of IF and the weight of IF type. The third step is Msg mapping, which takes Token weight as input and produces the Msg weights from the trace to the ontology based on the greedy algorithm. The last step is Flow mapping, which obtains the Msg weights and constructs the best mapping between trace Flow and the ontology Flow based on the greedy algorithm. Each step is explained in more detail as follows.

1) PREPROCESSING

The goal of the preprocessing method is to eliminate the replay messages and missing messages of the HTTP protocol payload. The HTTP protocol is a connectionless protocol. It ends the connection after completing a message response. Then, the method eliminates the replayed message by calculating the similarity of the message using the Levenshtein distance and selects a representative security protocol trace using cluster center discussed in section 3.2.4. Finally, it uses the protocol format generated by FAMUNT method to analyze selected trace into the three-layer hierarchy. The schematic diagram of the preprocessing is shown in Fig. 12.

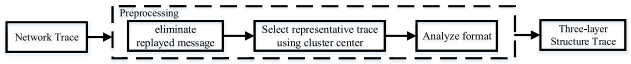


FIGURE 12. Schematic diagram of preprocessing.

2) TOKEN MAPPING

The token mapping method calculates the token weight between the trace and the ontology. The token mapping method is shown in Fig. 13. First, the key weighting method calculates the key weight from the trace to the ontology based on the Levenshtein distance ratio; second, the VF type weighting method computes the VF type weight; finally, the token weighing method uses the key weight and the VF type weight to calculate the token weight from the trace token to the ontology token.

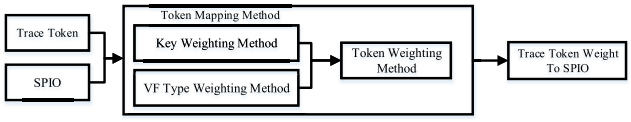


FIGURE 13. Token mapping method.

Although Token consists of the Key, separator, and VF, only Key and VF are used in the process of Token mapping. The key weight is generated by the key weighting method, which calculates the weight from trace key to ontology key with Levenshtein distance ratio. The Levenshtein distance ratio describes the similarity between the trace key and the ontology key. The VF type is produced by the VF type weighting method which calculates the weight by VF type relation and preset VF type weight. It uses regular expression [32] to describe the VF type. The key weight and the VF type weight are accepted as the input of the token weighing method to compute the weight from the trace token to the ontology token.

Separator plays an important role in separating key and VF in the security protocol. The type and length of separator have no contribution when analyzing the SSPI. Therefore, the weight of separator is not a key factor to establish SPT2SIO mapping. The key weighting method may be confused by the code obfuscation technique. Therefore, the key weight is closely related to the level of code obfuscation. However, the property of the VF type is not affected by code obfuscation, so the VF type weight is believable.

a: KEY WEIGHTING METHOD

English abbreviation is a key factor in analyzing the implementation of the protocol. For example, in the text protocol, most programmers often replace "password" with "PW" or "PWD." Therefore, the Levenshtein distance ratio is suitable for calculating the similarity between trace Key and the implementation of the ontology Key. The weighting algorithm of Key is shown as below:

$$\mathrm{W}\left(\mathrm{Key}_1, \mathrm{Key}_2\right) = 1 - \mathrm{LRatio}\left(\mathrm{Key}_1, \mathrm{Key}_2\right) \tag{2}$$

$\mathrm{W}(\mathrm{Key}_1, \mathrm{Key}_2)$ is the weight between trace Key_1 and implementation ontology Key_2. $\mathrm{LRatio}(\mathrm{Key}_1, \mathrm{Key}_2)$ is the Levenshtein distance ratio between the trace Key_1 and the ontology Key_2.

b: VF TYPE WEIGHTING METHOD

The VF type weighting method in Fig. 14 outputs the VF type weight according to the VF type relation and preset VF type weight. It uses regular expressions to describe the VF type. In the beginning, it defines the regular expressions by the implementation ontology and then uses the regular expressions to verify the trace VF. If the output is false, the trace VF is not the same type, and if the output is true, use minimum type relation to identify the minimum VF type and give the weight to the VF. Finally, it obtains the score of the VF.

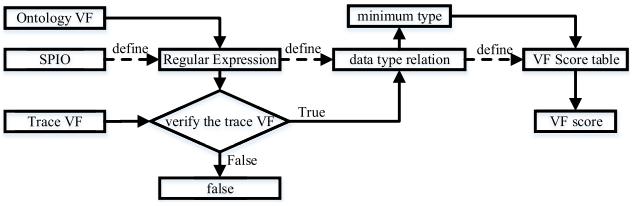


FIGURE 14. VF type weighting method.

In the VF type weighting method, we describe the data length and type by a regular expression. The data types are roughly divided into three categories: Number, Character, and Code. The Number type contains Float, Double and Int which consists of decimal and hexadecimal. The Code type is a special type defined by the implementation of the security protocol. Table 2 lists the four common data types and their regular expressions.

Because of the mapping from multiple regular expressions to one VF is many-to-one, the minimum type is introduced to address the problem. The data type relation diagram is shown in Fig. 15, in which the hexadecimal number contains the decimal number, and the URL and Time have a special format identifier. The VF type is defined by the minimum type. Finally, it gets the VF type weight by searching the Minimum type in Table 3. For example, the minimum type is

TABLE 2. Data type and regular expressions.

Data	Data Type	Regular expressions
13:12:51	Time	/^ ([0-1][0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9]) $/
Http://www.scuec.com	URL	/^http:\/\/[A-Za-z0-9]+\.[A-Za-z0-9]+[\/=\?%\-&_~` @[\]\':+!]*([^<>\"\"])*$/
F4a8c357bd	Hexadecimal	/^[0-9a-fa-F]{10}$/
123456789	Decimal	/^ [0-9] $/

FIGURE 15. Data type relation diagram.

TABLE 3. The weight of regular expressions.

Minimum type	SPIO Data Type	
	Decimal	Hexadecimal
Decimal	1	0.5
Hexadecimal	0	1

the decimal number, which matches the string "123456789." Even though this string also matches the hexadecimal number, but the weight of the str and Decimal is 1 and the weight of the str and Hexadecimal is 0.5. String "13:12:51" directly matches the regular expression of time type because of the time format.

Token weighing method

The token weighing method is used to calculate the weight from the trace token to the ontology token, which is shown in Equation (3).

$$\mathrm{W}(\mathrm{T}_1, \mathrm{T}_2) = \sqrt{\mathrm{W}(\mathrm{Key}_1, \mathrm{Key}_2)^2 + \mathrm{W}(\mathrm{VF}_1, \mathrm{VF}_2)^2} \quad (3)$$

It has two inputs. One is the Key weight calculated by the Key weighting method. The other is the VF weight computed by the VF type weighting method. The detailed algorithm of the Token mapping method is shown in Fig. 16. For example, the trace token is "PWD: abc123" and the ontology regular expression is "password: /^[0-9a-fa-F]{10}$/." The weight of PWD to password is 0.375 and the weight of "abc123" is 1. So, the weight of the trace token to the ontology token is 1.068 by equation (3). An example of the Token mapping method is shown in Fig. 17.

3) MSG MAPPING

The framework of the Msg mapping method is shown in Fig. 18, which calculates the weights of two Msg. It uses the greedy algorithm to select the weight of the two sets containing trace Token and the ontology Token. After that, it accepts matched Token weight as input and then calculates

Algorithm 4 Token mapping method
Input: TTK- Trace Token Key, OTK - Ontology Token Key
Output: $TokenWeight$- Weight of TTK to OTK
function TOKENWEIGHING(TTK, OTK) - Calculate the Euclidean distance of TTK to OTK
  $D1 \leftarrow$ KEYWEIGHTING(TTK, OTK)
  $D2 \leftarrow$ VFTYPEWEIGHTING(TTK, OTK)
  $TokenWeight \leftarrow$ EQUATION3($D1, D2$)
  return $TokenWeight$
end function

function KEYWEIGHTING(TTK, OTK) - Calculate the Levenshtein distance of TTK to OTK
  $result \leftarrow 0$
  while $!fefo < OTK >$ **do**
    $ChildNode \leftarrow$ PUTCHILDNODE(OTK)
    $Distance \leftarrow$ LEVENSHTEINRATIO($ChildNode, TTK$)
    if $result < Distance$ **then**
      $result \leftarrow Distance$
    end if
  end while
  return $result$
end function

function VFTYPEWEIGHTING(TTK, OTK) - Assign the weight of TTK and OTK
  $Weight \leftarrow$ DATATYPEMATCHINGTABLE(TTK, OTK)
  return $Weight$
end function

FIGURE 16. Token mapping method.

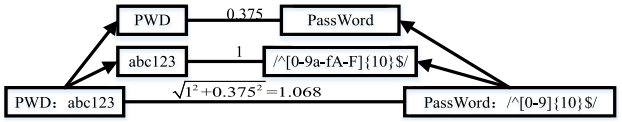


FIGURE 17. An example of the Token mapping method.

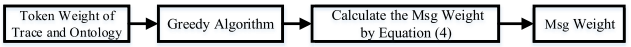


FIGURE 18. The framework of Msg mapping method.

Algorithm 5 Msg mapping method
Input: $TMsg$ - Trace Msg, $OMsg$ -Ontology Msg
Output: $MsgWeight$ - The Weight of TMsg to OMsg
function MSGWEIGHTING($TMsg, OMsg$)
  while $!fefo < TMsg >$ **do**
    $TTK \leftarrow$ PUTTOKEN($TMsg$)
    while $!fefo < OMsg >$ **do**
      $OTK \leftarrow$ PUTTOKEN($OMsg$)
      $WT_itoO_j \leftarrow$ TOKENWEIGHTING(TTK, OTK) - WT_itoO_j is the Weight of i-th TTK to j-th OTK
    end while
  end while
  while $!fefo < WT_itoO_j >$ **do**
    $MinWT_{i1}toO_{j1} \leftarrow$ GREEDYSELECT(WT_itoO_j) - Greedy Select the minimum TokenWeight
    $MsgW \leftarrow WT_itoO_j$
    DELETE($WT_itoO_j, MinWT_{i1}toO_{j1}$) - Delete the i1-th row and j1-th column from WT_itoO_j
  end while
  $MsgWeight \leftarrow$ CALCULATEMSGWEIGHT($MsgW$) - Using the equation 4
  return $MsgWeight$
end function

FIGURE 19. Msg mapping method.

the weight between two Msg by Equation (4). The Msg mapping method is shown in Fig. 19.

In the beginning, the Msg mapping method calculates the weight from the trace Token to the ontology Token based on the token mapping method and applies the greedy algorithm to select and remove a maximum weight from the trace and the ontology until all token mapping is completed.

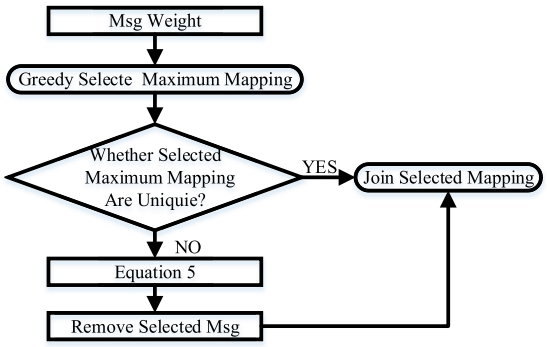


FIGURE 20. The process of Flow mapping method.

Algorithm 6 Flow mapping method
Input: $TFlow$ -Trace Flow, $OFlow$ -Ontology Flow
Output: $Mapping$
1: **function** FLOWMATCHING($TFlow, OFlow$)
2: **while** $!fefo < TFlow >$ **do**
3: $Tmsg_i \leftarrow$ PUTMSG($TFlow$)
4: **while** $!fefo < OFlow >$ **do**
5: $Omsg_j \leftarrow$ PUTMSG($OFlow$)
6: $MWT_iO_j \leftarrow$ MSGMAPPINGMETHOD($Tmsg_i, Omsg_j$) - Calculate the weight of $Tmsg_i$ to $Omsg_j$
7: $T_iO_j \leftarrow$ GREEDYSELECTMAX(MWT_iO_j)
8: **if** UNIQUE(T_iO_j) == 0 **then** - if the Max mapping T_iO_j is not unique
9: $T_iO_j \leftarrow$ EQUATION5(T_iO_j)
10: **end if**
11: $MWT_iO_j \leftarrow$ DELETE(MWT_iO_j, T_iO_j) - Delete T_iO_j from MWT_iO_j
12: $Mapping \leftarrow T_iO_j$
13: **end while**
14: **end while**
15: **return** $Mapping$
16: **end function**

FIGURE 21. Flow mapping method.

Then, it uses Equation (4) to calculate the Msg weight between trace and the ontology. The $W(Msg_t, Msg_o)$ is the Msg weight between trace Msg_t and the ontology Msg_o. $Num_o$is the total number of Tokens in the ontology, num_t is the number of successfully matched trace Tokens, and $\sum_{i=1}^{N} W(Token_t, Token_o)$ represents the total weight of the trace.

$$W(Msg_t, Msg_o) = \frac{num_t}{num_o} \sum_{i=1}^{N} W(Token_t, Token_o) \quad (4)$$

4) FLOW MAPPING

The flow mapping method aims to find an optimal mapping between the trace Msg and the ontology Msg based on the greedy algorithm. Fig. 20 shows the steps of the Flow mapping method. First, the Msg weight from each trace Msg to the ontology Msg is calculated by the Msg mapping method. Second, according to the greedy algorithm, the maximum mapping from the trace Msg to the ontology Msg is selected and removed; otherwise, if there are multiple maximum matches, it uses Equation 5, shown at the bottom of this page, to select the minimum mapping. Third, it iteratively performs the second steps until all Msg matches are completed. The algorithm of the flow mapping method is shown in Fig. 21.

Equation 5 has two inputs. One is trace Msg_t, and the other is ontology Msg_o. S is a mapping set of Msg_t and Msg_o. In the second method which takes the set S as input, Equation 5 selects the minimum mapping of Msg_t and Msg_o. The Num_T and Num_o are the total number of traces Msg and the ontology Msg respectively, t and o are the sequence number of Msg in the trace and ontology.

D. SECURITY ANALYSIS METHOD OF SECURITY PROTOCOL IMPLEMENTATIONS (SAMSPI)

SAMSPI method accepts a security protocol trace, a SPIO, and a SPT2SIO mapping as input to analyze security protocol implementation. SAMSPI method consists of the mapping analysis method and the non-ontology token analysis method. It applies the mapping analysis method to check the correctness of the SPT2SIO mapping. If the mapping is correct, it is considered that the security protocol implementations are correct. Then, we use the non-ontology token analysis method to detect whether there exists any message leakage in the non-ontology token.

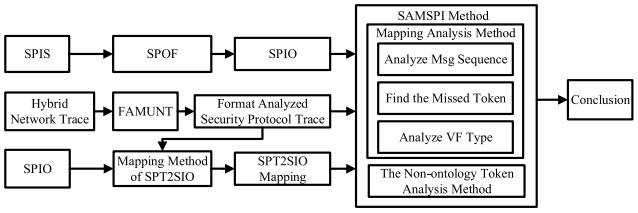


FIGURE 22. The framework of SAMSPI method.

The SAMSPI consists of four methods as depicted in Fig. 22. The first method constructs the SPIO based on the SPIOF. The second method gets the analyzed network trace by the FAMUNT method. The third method obtains SPT2SIO mapping through the mapping method of SPT2SIO. The fourth method is the SAMSPI method, which accepts a format analyzed security protocol trace, a SPIO, and a SPT2SIO mapping as input and generates the conclusion. The mapping analysis method and the non-ontology token analysis method can be found in Fig. 23.

The mapping analysis method takes the obtained SPT2SIO mapping and the format analyzed network trace as input. Protocol implementation security is verified by analyzing the consistency of the network trace VF type and the implementation ontology VF type. The details of the methods are described as follows:

$$W(Msg_t, Msg_o) = \left\{ \begin{array}{l} (Msg_t, Msg_o) \left| \left| \frac{t}{Num_T} - \frac{o}{Num_O} \right| \leq \left| \frac{t'}{Num_T} - \frac{o'}{Num_O} \right|, \right. \\ \left. \exists (Msg_t, Msg_o) \in S, \forall (Msg_{t'}, Msg_{o'}) \in S \right. \end{array} \right\} \quad (5)$$

Algorithm 1 Mapping analysis method

Input: $TMsg_iToken_j$ - Trace i-th Msg and j-th Token, *Mapping* , *SPIONotorgy*

Output: *ErrorMsg*,*ErrorToken*,*ErrorTokenType*

```
function Msg Sequence Analysis(Mapping)
    MapMsg_pToken_jq ← PutMsg(Mapping)
    i1 ← 0
    while i1 < MsgNum do
        p1,p2 ← MapMsg_i1,p1Token_j1,q1,MapMsg_i1+1,p2Token_j2,q2
        if p2 ≤ p1 then
            ErrorMsg ← MapMsg_i1,p1Token_j1,q1
        end if
        i1 ← i1+1
    end while
    return ErrorMsg
end function
function Token Sequence Analysis(Mapping,SPIO)
    MapMsg_pToken_jq ← PutMsg(Mapping)
    OMsg_pToken_q1 ← PutMsg(SPIO)
    p ← 0
    while p < MsgNum do
        if q1 ≠ q then
            ErrorToken ← p
        end if
        p ← p+1
    end while
    return ErrorToken
end function
function VF Type Analysis(TMsg_iToken_j,Mapping,SPIO)
    MapMsg_pToken_jq ← PutMsg(Mapping)
    while !feof < MapMsg_i,pToken_j,q > do
        REMsg_pToken_q ← Regular expression(SPIO)
        if Legal(REMsg_pToken_q,TMsg_iToken_j) == NULL then
            ErrorTokenType ← TMsg_iToken_j
        end if
    end while
    return ErrorTokenType
end function
function Non-ontology Token Analysis(TMsg_iToken_j,Mapping,SPIO)
    NOT ← SearchNOT(TMsg_iToken_j,SPIO) - search the Non-ontology Token(NOT) in Trace
    NOTError ← SearchNOT(NOT,TMsg_iToken_j) - search the same token in trace and NOT
    return NOTError
end function
```

FIGURE 23. Mapping analysis method.

1) Msg sequence analysis

The SPT2SIO mapping is used to analyze the Msg sequence of the security protocol traces. In Fig. 23, $TMsg_i$ $Token_j$is the trace T, j is the token sequence number and i is the Msg sequence number. The $TMsg_i$ $Token_j$ is marked as $MapMsg_{i,p}$ $Token_{j,q}$, where p is the token sequence number of the ontology and q is the Msg sequence number of the ontology. When p is increased one by one, if i is also increased, it returns True, otherwise returns False.

2) Token sequence analysis

If the Token is missing in the mapping, it may cause security risks. For the mapping $MapMsg_{i,p}$ $Token_{j,q}$, when q is increased one by one, if j exists, it returns True, otherwise returns False.

3) VF type analysis

According to the mapping $MapMsg_{i,p}$ $Token_{j,q}$, find the regular expression $Form_{p,q}$. If the $TMsg_i$ $Token_j$ matches $Form_{p,q}$, the $TMsg_{i,p}$ $Token_{j,q}$ is correct, otherwise, it is wrong.

The non-ontology Token analysis method checks the information leakage by mapping the values of the VF type in the non-ontology token.

IV. FSIA

In order to put the SAMSPI method into practice, we develop a security analysis software FSIA which consists of the FA (Format Analyzing, FA) module, the SA (Semantic Analyzing, SA) module and the ISA (Implementation Security Analysis, ISA) module. The input of the FSIA is the unpurified protocol trace and the security protocol ontology, and the output is the conclusion on the security of the SPI. The FA module is designed and implemented according to the FAMUNT method; the SA module is developed based on the mapping method of SPT2SIO; the ISA module is implemented with the SAMSPI method.

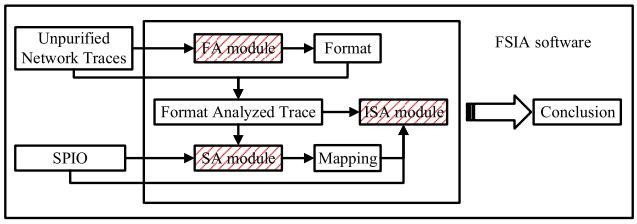


FIGURE 24. FSIA software.

The FSIA software is shown in Fig. 24. The FA module takes unpurified network trace as inputs and generates the protocol format. The SA module takes the format analyzed trace and the SPIO as input and produces the SPT2SIO mapping. The ISA module takes the SPIO, the format analyzed trace and SPT2SIO mapping as input and then generates conclusion of the security protocol implementation.

V. EVALUATION

Here we use FSIA software to analyze the security protocol implementation in the login module of a university information system. The login module supports two ways to login: user/password and WeChat QR code. The development documents show that the login module adopts the widely used CAS protocol [28], [33], [34] which consists of CAS-SSO versions and CAS-OAUTH versions. The FAMUNT takes unpurified network traces as input and outputs a protocol format of two traces, which are called red trace and green trace respectively. The mapping method of SPT2SIO takes the red and green network traces as input, and then builds the regular expression to describe the data type, define the data type relation and score, and matches the red and green network traces with the two implementation ontologies. The results show that the green network trace is CAS-OAUTH and the red is CAS-SSO. At the same time, this method establishes SPT2SIO mapping. The SAMSIP method verifies the SPT2SIO mappings, and the result shows that the application server authorization ticket in the CAS protocol has a risk of leakage, which exposes the hidden security risks existing in the login module of a University.

A. PREPROCESS

The login module of a university information system has two ways to login, either by password or by WeChat QR code. According to the development documents, the login platform adopts the widely used CAS-SSO and CAS-OAUTH protocols.

In the data preprocessing, we use the proxy function in the Burp Suite to collect 186 login traces. Since the login trace is sensitive data, we need to get the consent of the participants to collect data, so the number of login traces set is small. Fig. 25 shows the processed security protocol trace.

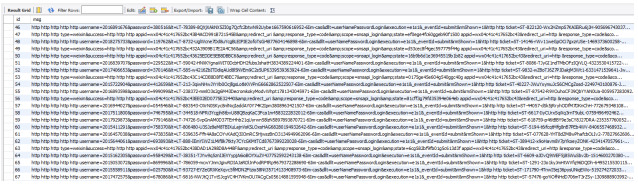


FIGURE 25. Preprocessed data.

The preprocess is as follows:

(1) Apply Filter function in the Burp suite to filter out the HTTP stream with JavaScript, XML, CSS, etc. to obtain the traces related to the security protocol.

(2) The fields that are not related to the security protocol, such as Host, User-Agent, and Accept, are removed according to the HTTP protocol specification.

(3) The fields related to the session establishment such as amp.locale, iplanetdirectorypro, and user_id, etc. are removed.

B. FORMAT ANALYSIS

In the format analysis, it takes the preprocessed HTTP trace as input. First, the trace segmentation method generates the segmented protocol trace. Second, the IF fitting method outputs the IF distribution fitting curve. Third, the IF classification method filters out the IF at each interval and marks the trace with IF and converts it into a vector. Fourth, it sends the vector to the trace clustering method and produces the red trace and the green trace by the K-means method. Fifth, the format inference method takes the red trace and green trace as input to produce the protocol format.

1) TRACE SEGMENTATION

Fig. 26 shows the execution result of Trace segmentation, where the ''−'' indicates there is no match to the character. Due to the flaws of the Needleman Wunch algorithm, the segmented original trace contains lots of mismatches. The parameters in the Needleman Wunch method are set to mismatch $= -2$, gap $= 1$, and match $= 2$.

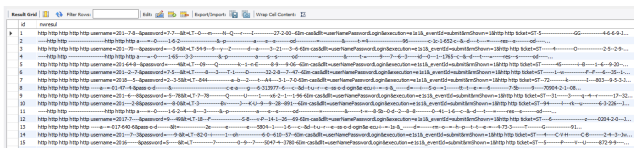


FIGURE 26. Result of trace segmentation.

2) IF FITTING

The IF fitting method uses IF weight and the B-spline to generate the B-Spline function, which is the green line in Fig. 27, where the horizontal axis is the IF distance value which is the distance from the IF to the first byte of the trace, the vertical axis is the sum of the IF weights, and the red points represent the IF type. The upper the red point is, the larger the maximum value of the spline function is, and the bigger the probability

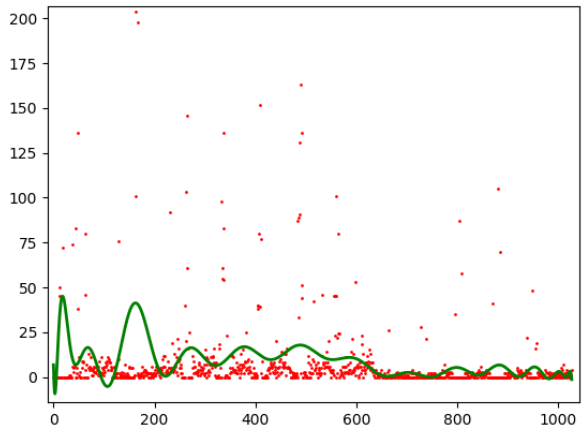
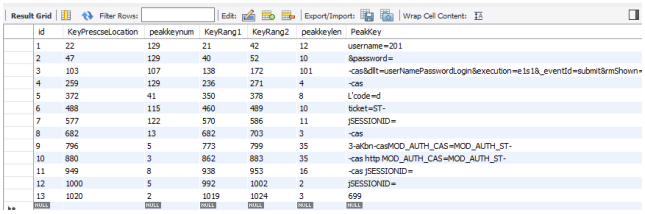


FIGURE 27. B-Spline function.

of Key is. In the B-spline fitting, a 30-th ordered spline function was used.

3) IF CLASSIFICATION AND TRACE CLUSTERING

The character classification method applies the string classification to process the IF table at the peak of the B-spline in Fig. 28. The character classification module selects the IF of the 13 peaks of the B-spline.



id	KeyPrescseLocation	peakkeynum	KeyRang1	KeyRang2	peakkeylen	PeakKey
1	22	129	21	42	12	username=201
2	47	129	40	52	10	&password=
3	103	107	138	172	101	-casdsdlt=userNamePasswordLoginSexecution=e1s1&_eventId=submit&rmShown=
4	259	129	236	271	4	-cas
5	372	41	350	378	8	L'code=d
6	488	115	460	489	10	ticket=ST-
7	577	122	570	586	11	JSESSIONID=
8	682	13	682	703	3	-cas
9	796	5	773	799	35	3-aKbn-casMOD_AUTH_CAS=MOD_AUTH_ST-
10	880	3	862	883	35	-cas http MOD_AUTH_CAS=MOD_AUTH_ST-
11	949	8	938	953	16	-cas JSESSIONID=
12	1000	5	992	1002	2	jSESSIONID=
13	1020	2	1019	1024	3	699

FIGURE 28. IF in every peak of the B-spline.

The trace clustering method uses K-Means method to generate the red and green clusters shown in Fig. 29. The clustering visualization graph is shown in Fig. 30. There are 155 red traces and 31 green traces.

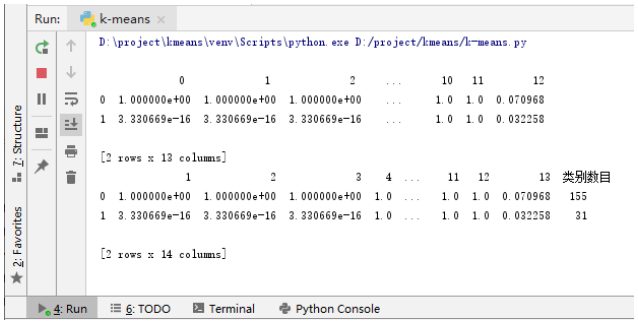


FIGURE 29. Clustering result.

4) FORMAT ANALYSIS

The format analysis method takes the red traces and green traces as input respectively and then obtains IFs. IFs in the

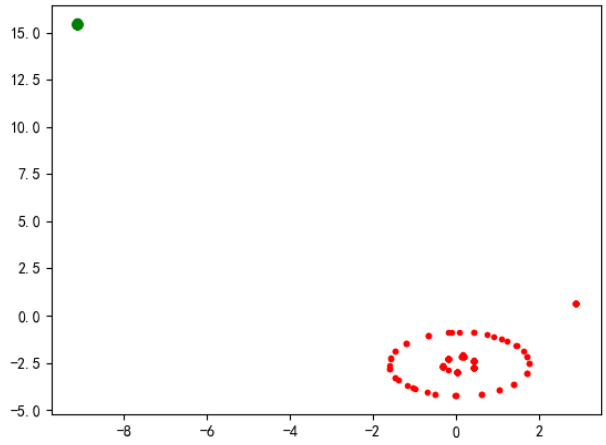
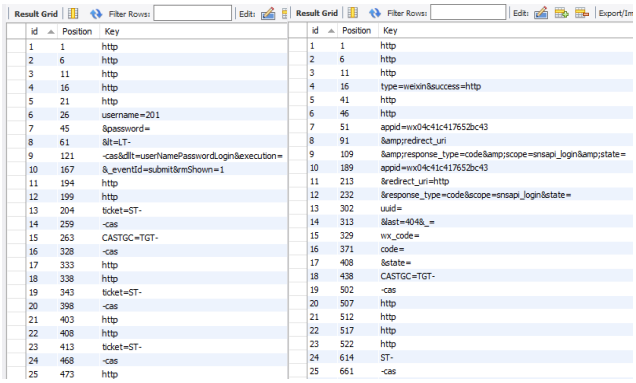


FIGURE 30. The clustering visualization graph.



id	Position	Key
1	1	http
2	6	http
3	11	http
4	16	http
5	21	http
6	26	username=201
7	45	&password=
8	61	&lt=LT-
9	121	-cas&dllt=userNamePasswordLogin&execution=
10	167	&_eventId=submit&rmShown=1
11	194	http
12	199	http
13	204	ticket=ST-
14	259	-cas
15	263	CASTGC=TGT-
16	328	-cas
17	333	http
18	338	http
19	343	ticket=ST-
20	398	-cas
21	403	http
22	408	http
23	413	ticket=ST-
24	468	-cas
25	473	http

id	Position	Key
1	1	http
2	6	http
3	11	http
4	16	type=weixin&success=http
5	41	http
6	46	http
7	51	appid=wx04c41c417652bc43
8	91	&redirect_uri
9	109	&response_type=code&scope=snsapi_login&state=
10	189	appid=wx04c41c417652bc43
11	213	&redirect_uri=http
12	232	&response_type=code&scope=snsapi_login&state=
13	302	uuid=
14	313	&last=404&_=
15	329	wx_code=
16	371	code=
17	408	&state=
18	438	CASTGC=TGT-
19	502	-cas
20	507	http
21	512	http
22	517	http
23	522	http
24	614	ST-
25	661	-cas

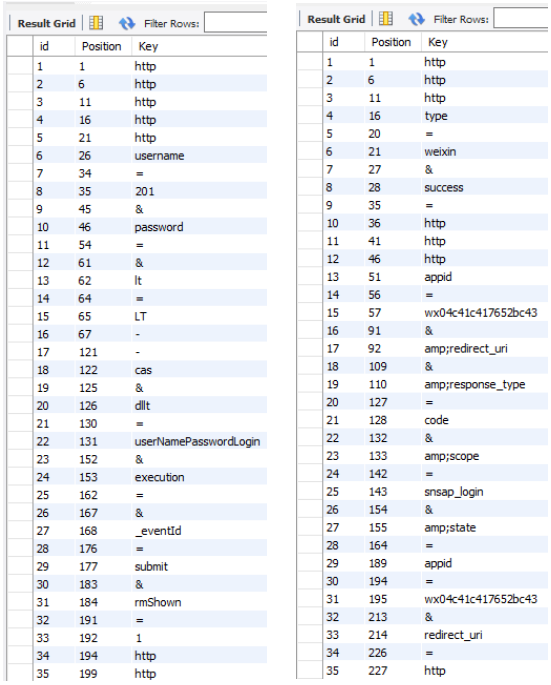
FIGURE 31. The IFs generated by red traces and green traces.

left half of Fig. 31 is generated with the red traces, while IFs in the right half are generated with the green traces.

TABLE 4. IF Frequency of red trace.

Separator	Frequency
-	0.323
T-	0.161
-c	0.129
As	0.117
&	0.097
Ti	0.097
=	0.065

The protocol format inference method takes the IFs as input and produces the protocol format. For the red trace, separator inference method takes the IF table and the curve generated by the red trace as input and then count the Statistical frequency of IF. After setting a threshold, 7 candidate separators are selected. The candidate separators are shown in Table 4. According to the condition proposed in separator inference method, VF region is selected, and then condition 3 is iteratively used to filter the 3 separators "T-," "as," "ti" because the first character of this separator is in the VF region. Finally, 4 more separators "−," "-c," "&," "=" are selected, and the protocol format is obtained. Fig. 32 shows the partial



id	Position	Key
1	1	http
2	6	http
3	11	http
4	16	http
5	21	http
6	26	username
7	34	=
8	35	201
9	45	&
10	46	password
11	54	=
12	61	&
13	62	lt
14	64	=
15	65	LT
16	67	-
17	121	-
18	122	cas
19	125	&
20	126	dllt
21	130	=
22	131	userNamePasswordLogin
23	152	&
24	153	execution
25	162	=
26	167	&
27	168	_eventId
28	176	=
29	177	submit
30	183	&
31	184	rmShown
32	191	=
33	192	1
34	194	http
35	199	http

id	Position	Key
1	1	http
2	6	http
3	11	http
4	16	type
5	20	=
6	21	weixin
7	27	&
8	28	success
9	35	=
10	36	http
11	41	http
12	46	http
13	51	appid
14	56	=
15	57	wx04c41c417652bc43
16	91	&
17	92	amp;redirect_uri
18	109	&
19	110	amp;response_type
20	127	=
21	128	code
22	132	&
23	133	amp;scope
24	142	=
25	143	snsap_login
26	154	&
27	155	amp;state
28	164	=
29	189	appid
30	194	=
31	195	wx04c41c417652bc43
32	213	&
33	214	redirect_uri
34	226	=
35	227	http

FIGURE 32. Part of protocol format.

protocol format. The red protocol format based on IFs generated with red traces is as shown on the left and another is the green trace.

C. SEMANTIC ANALYSIS

The input of the Mapping method of SPT2SIO is the format analyzed red and green traces. In the beginning, based on the development documents and CAS official implementation specification, the method establishes the CAS-SSO and CAS-OAUTH protocol implementation ontology. Then, it matches the red and green traces with CAS-SSO and CAS-OAUTH implementation ontology. The result shows that the green network trace is an instance of the CAS-OAUTH ontology, and the red network trace is an instance of the CAS-SSO. Finally, the mapping between the security protocol trace and the SPIO are constructed respectively.

1) SPIO CONSTRUCTION

According to the development documents and CAS official implementation specification, we establish the SPIO of the CAS-SSO and CAS-OAUTH. The implementation message structure of the CAS-SSO version is shown in Fig. 33. The CAS-SSO protocol version includes the client, the authentication server, and the application server. The communication between the authentication server and the application server is hard to monitor, specifically the messages Nos. 6, 9, 10, 13, so we don't construct the SPIO of these messages. Meanwhile, message No. 4 is sent to the login interface and is not related to security, so this message is discarded. The messages of the client and the application server can be captured, so the

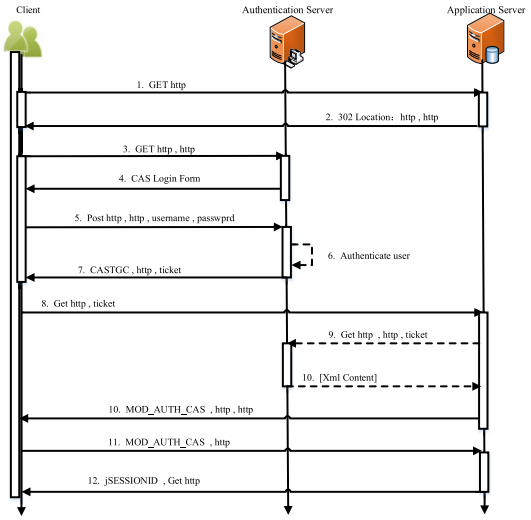


FIGURE 33. CAS-SSO message structure.

TABLE 5. CAS-SSO SPIO.

Msg	Token	Key	Msg	Token	Key
Msg1	Token1	Http	Msg4	Token1	Http
Msg1	Token2	Http	Msg4	Token2	Ticket
Msg1	Token3	Http	Msg4	Token3	MOD_AUTH _CAS
Msg2	Token1	Http	Msg4	Token4	Http
Msg2	Token2	Http	Msg4	Token5	Http
Msg3	Token1	Http	Msg5	Token1	MOD_AUTH _CAS
Msg3	Token2	Http	Msg5	Token2	Http
Msg3	Token3	Username	Msg5	Token3	Http
Msg3	Token4	Password	Msg5	Token4	Jsessionid
Msg3	Token5	CASTGC	Msg5	Token5	Http
Msg3	Token6	Http			
Msg3	Token7	Ticket			

messages Nos. 1-5, 7-8, 11-12 are analyzed. The implementation ontology of the CAS-SSO protocol message Nos. 1-5, 7-8, 11-12 can be found in Table 5.

Fig. 34 shows the protocol implementation message structure for the CAS-OAUTH WeChat version. CAS-OAUTH mainly consists of the client, the WeChat authentication server, CAS authentication server, and application server. The implementation message structure omits the messages of the CAS authentication server to the WeChat authentication server and the authentication messages of the mobile phone and the WeChat authentication server. We only preserve the messages from the client, WeChat authentication server and application server, because in the CAS-OAUTH WeChat version, it is difficult to capture the packets from the mobile phone to the WeChat authentication server and the packets from the application server to the WeChat authentication server. Therefore, we don't construct the SPIO of these messages. The implementation ontology messages that the client

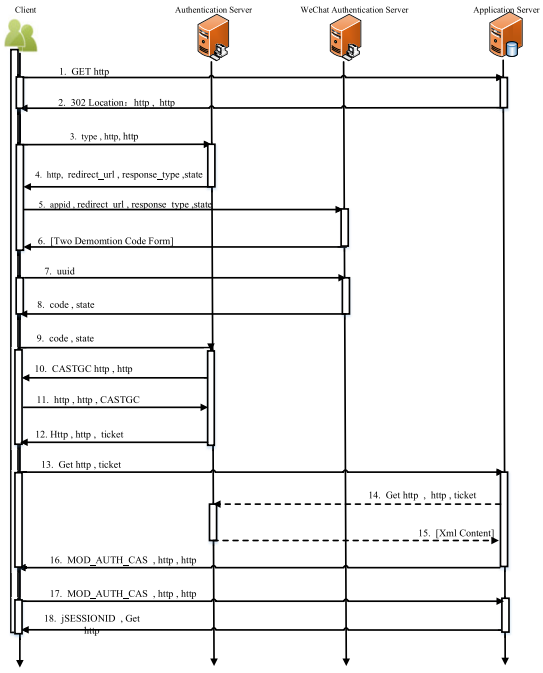


FIGURE 34. CAS-OAUTH message structure.

communicates with the WeChat authentication server and application server can be found in Table 6.

TABLE 6. CAS-OAUTH SPIO.

Msg	Token	Key	Msg	Token	Key
Msg1	Token1	Http	Msg5	Token3	CASTGC
Msg1	Token2	Http	Msg5	Token4	Http
Msg1	Token3	Http	Msg5	Token5	Http
Msg2	Token1	Type	Msg6	Token1	CASTGC
Msg2	Token2	Http	Msg6	Token2	Http
Msg2	Token3	Http	Msg6	Token3	Http
Msg2	Token4	Http	Msg6	Token4	Http
Msg2	Token5	Http	Msg6	Token5	Http
Msg2	Token6	Response type	Msg6	Token6	Ticket
Msg2	Token7	State	Msg7	Token1	Http
Msg3	Token1	Appid	Msg7	Token2	Ticket
Msg3	Token2	Http	Msg7	Token3	Jsessionid
Msg3	Token3	Response type	Msg7	Token4	MOD_AUTH_ CAS
Msg3	Token4	State	Msg7	Token5	Http
Msg4	Token1	Uuid	Msg7	Token6	Http
Msg4	Token2	Code	Msg8	Token1	MOD AUTH CAS
Msg4	Token3	State	Msg8	Token2	Http
Msg5	Token1	Code	Msg8	Token3	Http
Msg5	Token2	State	Msg8	Token4	Jsessionid
			Msg8	Token5	Http

2) TOKEN MAPPING

Table 7 shows the VF type and regular expression defined by SPIO. Fig. 35 shows the data type relation, where Code,

TABLE 7. VF type and regular expression.

VF Type	Regular expression
Int	/^[0-9]*$
Float	/^[1-9]d*.d*|0.d*[1-9]d*$
Double	/^[-//+]?//d+(//.//d*)?|//.//d+$
Http	/^http:\/\/[A-Za-z0-9]+\.[A-Za-z0-9]+ [\/=\?%\- &_~`@[\]\':+!]*([^<>\"\"])*$/
CASTGC	/^ TGT- [1-9]\d{5}|-[A-Za-z0-9]\w{48,52}|
Ticket	[0-9]{11,15}|-[A-Za-z0-9] \w{4}$ /^ST-|\d{5}|-[A-Za-z0-9] \w{15,19}| [0-9]{12,16}|[A-Za-z]\w{4}|-cas $
Jsessionid	/^ [A-Za-z]{5}|-[A-Za-z0-9]\ w{4,48}|!-[1-9]\d{9}$
Code	/^-[A-Za-z0-9]\w{30,34}$
Uuid	/^-[A-Za-z0-9] \w{5,7}$
State	/^ [0-9a-z]\ w{20,24}$
Mod_Auth_CAS	/^ MOD_AUTH_ST-|d{4,6}|- |[A-Za-z0-9]\w{31,35}|-cas $
Username	/^20[0-1]\w{1}|[0-9]\w{1}$
Appid	Fixed value
Response_type	Fixed value
Type	Fixed value

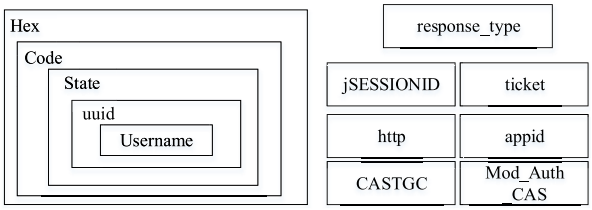


FIGURE 35. Data type rationales.

State, uuid, and username and so on are special data types in the CAS protocol.

Table 8 lists the weight of the data type. Other VFs including ''appid,'' ''Type,'' ''response_type'' and so on are fixed values, and the weight of them is set to 0. ''CASTGC,'' ''MOD_AUTH_CAS,'' and ''ticket'' have specific start characters in the implementation specification, and it is easy to identify them with regular expressions. Therefore, the weight of these VF types is set to 1 or 0. However, ''username,'' ''uuid'' and ''State'' have similar data type. Hence when the minimum type ''uuid'' matches the ''Code,'' the weight is set to 0.3. When the minimum type ''uuid'' matches the ''State,'' the weight is set to 0.7.

3) MSG MAPPING

The weight between the trace Msg and the ontology Msg is calculated by Equation (4). For instance, in the green trace, Msg4 is ''uuid = a84Fb2&last = 404&_= 4786wx_ code = hieo923fp3domlhcnnvdbpiathkpnka&code = hie7o92 3fp3d omlhcnnvdbpiathkpnka&state = 99efebf38249d3dgd7b 58g.'' The matched result with Msg4 is shown in Table 9. First, we use Equation (2) to calculate the Key weight. When the distance ratio is lower than 0.7, the Key weight is specified as 0. Then, according to the VF weighting module, the VF weight is obtained by Table 7, and then the Token weight is calculated by Equation (3). After that, we apply the greedy algorithm to select the maximum Token weight one by one. Finally, the Token weights are added by Equation (4) to get the Msg weight in Msg4.

4) FLOW MAPPING

In order to analyze the red trace and the green trace, the flow mapping method takes the red trace and the CAS-OAUTH implementation ontology as input and produces the matching weight. Table 10 shows the matched weights of the red trace to the CAS-OAUTH implementation ontology, where ''Omsg'' is CAS-OAUTH implementation ontology Msg and ''Tmsg'' is trace Msg. Weight selection is based on the greedy selection algorithm. In the first method, it selects the global maximum 7.071, the red trace ''Tmsg5'' matches the ''Omsg8,'' so ''Omsg8'' is removed; in the second method, it selects the global maximum 5.893, the ''Tmsg3'' matches the ''Omsg6,'' and the ''Tmsg4'' matches ''Omsg7,'' so ''Omsg6'' and ''Omsg7'' are removed; in the third method, it selects the global maximum 4.243, the ''Tmsg1'' is matched with the ''Omsg1,'' so ''Omsg1'' is removed; in the fifth method, the ''Tmsg2'' is matched with the ''Omsg5,'' so ''Omsg5'' is removed. The matching results of the red trace and the CAS-OAUTH implementation ontology are shown in Table 11. Since the weight of the red trace matching the CAS-OAUTH implementation ontology is low, it is impossible to match the red trace with the CAS-OAUTH implementation ontology.

Similarly, the green trace is matched to CAS-OAUTH implementation ontology, and the matched weights are shown in Table 12. The matched results are shown in Table 13. Since the weight of the green trace matching the CAS-OAUTH implementation ontology is high, the green trace is fully matched with the CAS-OAUTH implementation ontology.

The matched results of the red trace and the CAS-SSO implementation ontology are shown in Table 14, and the red trace is fully matched with the CAS-SSO implementation ontology. Since the weight of the red trace matching the CAS-SSO implementation ontology is high, the red trace is fully matched with the CAS-SSO implementation ontology.

D. IMPLEMENTATION SECURITY ANALYSIS

The mapping analysis method is used to analyze the mapping of the CAS-OAUTH and CAS-SSO. The result shows that the CAS-OAUTH trace consists of 39 ontology tokens and 70 non-ontology tokens. The CAS-SSO trace consists of 22 ontology tokens and 46 non-ontology tokens. The analytical conclusions of both two traces indicate that the Msg sequence is fully consistent with the implementation ontology and the number of ontology Token is equal to the implementation ontology. The type of VF in the two traces is consistent with the regular expression generated by implementation ontology. Moreover, the non-ontology token analysis method shows that there is an information leakage threat for the Ticket in the 7th and 8th messages of the CAS-SSO

TABLE 8. Weight of the data type.

Minimum Data Type in Trace	Data Type in Ontology								
Key	Http	CAST GC	Ticket	Jsessionid	Code	Uuid	State	Mod_Auth_CAS	User Name
Http	1	0	0	0	0	0	0	0	0
CASTGC	0	1	0	0	0	0	0	0	0
Ticket	0	0	1	0	0	0	0	0	0
Jsessionid	0	0	0	1	0	0	0	0	0
Code	0	0	0	0	1	0	0	0	0
Uuid	0	0	0	0	0.3	1	0.7	0	0
State	0	0	0	0	0.7	0	1	0	0
Mod_Auth_CAS	0	0	0	0	0	0	0	1	0
Username	0	0	0	0	0.3	0.7	0	0	1

TABLE 9. Matching result with green trace Msg4 and CAS-OAUTH implementation ontology.

CAS-OAUTH Implementation ontology		Green Trace Msg4					Matching Result
		Uuid	Last	Wx_code	Code	State	
Msg3	Appid	0	0	0	0	0	
Msg3	Http	0	0	0	0	0	0.354
Msg3	Response_type	0	0	0	0	0	
Msg3	State	0.7	0	0	0	1.414	
Msg4	Uuid	1.414	0	0	0	0	
Msg4	Code	0	0	1.152	1.414	0.7	4.243
Msg4	State	0 . 7	0	0	0	1.414	
Msg5	Code	0.3	0	1.152	1.414	0.7	
Msg5	State	0.7	0	0	0	1.414	
Msg5	CASTGC	0	0	0	0	0	1.131
Msg5	Http	0	0	0	0	0	
Msg5	Http	0	0	0	0	0	

TABLE 10. Matched weights of the red trace to the CAS-OAUTH implementation ontology.

Red Trace	CAS-OAUTH Implementation ontology							
	Omsg1	Omsg2	Omsg3	Omsg4	Omsg5	Omsg6	Omsg7	Omsg8
Tmsg1	4.243	1.818	0.354	0	0.913	3.182	2.122	2.546
Tmsg2	1.521	0.652	0.354	0	0.913	0.761	0.761	0.913
Tmsg3	4.243	1.818	0.354	0.1	1.877	5.893	3.771	2.546
Tmsg4	4.243	1.818	0.354	0	1.131	3.771	5.893	4.526
Tmsg5	4.243	1.818	0.354	0	1.131	2.122	5.893	7.071

TABLE 11. Matching result with red trace and CAS-OAUTH implementation ontology.

Red Trace	Tmsg1	Tmsg2	Tmsg3	Tmsg4	Tmsg5
CAS-OAUTH	1	5	6	7	8

TABLE 12. Matched weights of the green trace to the CAS-OAUTH implementation ontology.

Green Trace	CAS-OAUTH Implementation ontology							
	Omsg1	Omsg2	Omsg3	Omsg4	Omsg5	Omsg6	Omsg7	Omsg8
Tmsg1	4.243	1.818	0.354	0	1.131	2.122	2.122	2.546
Tmsg2	4.243	12.726	1.414	0.471	2.546	3.771	2.121	2.546
Tmsg3	0.471	1.818	4.243	0.471	1.131	0.236	0.236	0.283
Tmsg4	0	0.202	0.354	4.243	1.131	0	0	0
Tmsg5	1.521	0.808	1.414	1.885	7.071	2.125	0.943	1.131
Tmsg6	4.243	3.233	0.354	0	2.546	8.485	3.771	2.546
Tmsg7	4.243	1.818	0.354	0	1.131	2.122	8.485	7.071
Tmsg8	4.242	1.818	0.354	0	1.131	2.122	5.893	7.071

protocol trace. We also find this information leakage threat in the 12th and 13th messages of the CAS-OAUTH protocol trace.

Figs. 36 and 37 show the results of the non-ontology token analysis method. The Ticket appears in the Location field and Get field. It indicates that Ticket is sent along with

TABLE 13. The mapping between the green trace to the CAS-OAUTH implementation ontology.

Green Trace	Tmsg1	Tmsg2	Tmsg3	Tmsg4	Tmsg5	Tmsg6	Tmsg7	Tmsg8
CAS-OAUTH	1	2	3	4	5	6	7	8

TABLE 14. The mapping between the red trace to the CAS-SSO implementation ontology.

Red Trace	Tmsg1	Tmsg2	Tmsg3	Tmsg4	Tmsg5
CAS-OAUTH	1	2	3	4	5

```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Fri, 22 Feb 2019 07:40:55 GMT
Connection: close
Cache-Control: no-cache
Cache-Control: no-store
Pragma: no-cache
Location: http://ehall.scuec.edu.cn/login?service=http://ehall.scuec.edu.cn/new/index.html&ticket=ST-1113910-BaL73HKEjIdfT1QPYsQk1550821255950-aKbn-cas
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Length: 485

<html><head><title>302 Moved Temporarily</title></head>
<body bgcolor="#FFFFFF">
<p>This document you requested has moved temporarily.</p>
<p>It's now at <a href="http://ehall.scuec.edu.cn/login?service=http://ehall.scuec.edu.cn/new/index.html&amp;ticket=ST-1113910-BaL73HKEjIdfT1QPYsQk1550821255950-aKbn-cas">
http://ehall.scuec.edu.cn/login?service=http://ehall.scuec.edu.cn/new/index.html&amp;ticket=ST-1113910-BaL73HKEjIdfT1QPYsQk1550821255950-aKbn-cas</a>.</p>
```

FIGURE 36. The 7th message of CAS-SSO.

```
GET /login?service=http://ehall.scuec.edu.cn/new/index.html&ticket=ST-1113910-BaL73HKEjIdfT1QPYsQk1550821255950-aKbn-cas HTTP/1.1
Host: ehall.scuec.edu.cn
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3610.2 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://open.weixin.qq.com/connect/qrconnect?appid=wx04c41c417652bc43&redirect_uri=http://id.scuec.edu.cn/authserver/callb
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: amp.locale=zh_CN; zg_did=%7B%22did%22%3A%20%22167cfa85a4b67e-0bda32c2b8467e-3f674706-1fa400-167cfa85a4cf9d%22%7D; UM_disti
Connection: close
```

FIGURE 37. The 8th message of CAS-SSO.

the URL. Get field and Location field is the plaintext in the HTTP protocol and is generally used to query resources and jump to another website. However, according to the SPIO, the ticket is an important authorization credential for establishing a session between the client and the CAS application server. The Get field and location field could be saved in the cache by the browser, and the attacker can get the ticket by searching the cache. Therefore, the login module of a university information system has a risk of data leakage.

We also notice that the number of non-ontology token in the trace is larger than the ontology token. There are two reasons for this phenomenon: (1) Many tokens may traverse in the same connection multiple times. (2) CAS protocol and the other protocol are mixed in transmission. The CAS protocol implementation of the user login module is not standard.

E. DISCUSSION

We discuss in detail our approach from four aspects:

(1) The past protocol reverse engineering methods merely infer the protocol specification, but in our work, it is used to analyze the consistency of security protocol trace and SPIO to find the vulnerabilities of network and system.

(2) In the past, researchers used purified network traces to promote the correctness, but purified network traces are hard to obtain. In our work, unpurified network traces are used. We process it by B-spline such that the noise can be resisted and reduced. The B-spline points out the special characters used for the K-means method to classify the unpurified network traces.

(3) As for complexity, PI project [35] uses multiple sequence alignment in which computational complexity is O (L^n) (L-is the length of the trace, n is the number of the trace). [15,17-19] adopts the n-grams method to analyze the protocol format. The computational complexity of the n-grams is O (K^w) (K is the number of keywords and W is the number of n-grams for the given corpus). We use two-way sequence alignment method to replace multiple sequence alignment method and reduce the computational complexity from O (L^n) to O (L^2), which is a considerable improvement on the efficiency.

(4) Regarding the separator identification assumption, Field Hunter [21] uses two assumptions: (1) Separators often appear at the beginning and end of the IF and are usually shorter in length, (2) Separators are a nonalphanumeric sequence of 1 or 2 characters. Field Hunter uses assumption (2) to filter separators. But we only adopt assumption (1) and then use VF region to filter separators.

(5) Malicious traffic detection methods [9]–[14] find an attack when an attack is launched, while our proposed SAMSPI method can find in advance the vulnerabilities of

the network and system before an attack is launched to take advantage of these vulnerabilities.

VI. CONCLUSION AND FUTURE WORKS

Under the condition of not being able to obtain SPI, in this paper, we analyze the SSPI based on the unpurified security protocol traces and SPI specifications. First, the SPIOF is presented based on a seven-step method and an ontology framework. It describes the attributes of the ontology terms. Second, FAMUNT method is presented using separator inference method. It produces the protocol format from the unpurified network trace. Third, the mapping method of SPT2SIO is presented based on the greedy algorithm. It generates SPT2SIO mapping. Fourth, the SAMSPI method is presented to analyze the consistency of CAS trace and check the token leakage. Finally, FSIA software is implemented to analyze the CAS-SSO protocol and CAS-OAUTH protocol of login module of a university, the result shows that there is a risk of leakage of the Ticket in the CAS protocol and a large number of the non-ontology token are mixed with ontology token in the same connection.

In the future, we will take the token dependencies and the protocol state machine into consideration to analyze the security of SPI to a finer degree.

REFERENCES

[1] K.-S. Min, S.-W. Chai, and M. Han, "An international comparative study on cyber security strategy," *Int. J. Secur. Appl.*, vol. 9, no. 2, pp. 13–20, Sep. 2015.
[2] M. Aizatulin, A. D. Gordon, and J. Jürjens, "Extracting and verifying cryptographic models from C protocol code by symbolic execution," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, Chicago, IL, USA, Oct. 2011, pp. 331–340.
[3] B. Meng, X. He, J. Zhang, L. Yao, and J. Lu, "Security analysis of security protocol Swift implementations based on computational model," *J. Commun.*, vol. 39, no. 9, pp. 178–190, Sep. 2018.
[4] J. Jürjens, "Automated security verification for crypto protocol implementations: Verifying the Jessie project," *Electron. Notes Theor. Comput. Sci.*, vol. 250, no. 1, pp. 123–136, Sep. 2009.
[5] J. Lu, L. Yao, X. He, C. Huang, D. Wang, and B. Meng, "A security analysis method for security protocol implementations based on message construction," *J. Appl. Sci.*, vol. 8, no. 12, p. 2543, Nov. 2018.
[6] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "ReFormat: Automatic reverse engineering of encrypted messages," in *Proc. Eur. Symp. Res. Comput. Secur.*, vol. 5789, Berlin, Germany, Sep. 2009, pp. 200–215.
[7] J. Zeng and Z. Lin, "Towards automatic inference of kernel object semantics from binary code," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*, vol. 9404, Kyoto, Japan, Dec. 2015, pp. 538–561.
[8] M. Li, Y. Wang, P. Xie, and Z. Huang, "Reverse analysis of secure communication protocol based on taint analysis," in *Proc. Commun. Secur. Conf.*, Beijing, China, May 2014, pp. 1–8.
[9] G. Sun, L. Liang, T. Chen, F. Xiao, and F. Lang, "Network traffic classification based on transfer learning," *Comput. Elect. Eng.*, vol. 69, pp. 920–927, Jul. 2018.
[10] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Comput. Netw.*, vol. 109, no. 2, pp. 127–141, Nov. 2016.
[11] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, and B. Yang, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Inf. Sci.*, vols. 433–434, pp. 346–364, Apr. 2018.
[12] Y. Wang, X. Yun, Y. Zhang, L. Chen, and T. Zang, "Rethinking robust and accurate application protocol identification," *Comput. Netw.*, vol. 129, pp. 64–78, Dec. 2017.
[13] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, "Data exfiltration: A review of external attack vectors and countermeasures," *J. Netw. Comput. Appl.*, vol. 101, pp. 18–54, Jan. 2018.
[14] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android malware leveraging text semantics of network flows," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1096–1109, May 2018.
[15] T. Krueger, N. Krämer, and K. Rieck, "ASAP: Automatic semantics-aware analysis of network payloads," in *Proc. Int. Workshop Privacy Secur. Issues Data Mining Mach. Learn.*, Barcelona, Spain, Sep. 2010, pp. 50–63.
[16] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo, "Biprominer: Automatic mining of binary protocol features," in *Proc. 12th Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Barcelona, Spain, Oct. 2011, pp. 179–184.
[17] Y. Wang, Z. Zhang, and L. Guo, "Inferring protocol state machine from real-world trace," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, Ottawa, ON, Canada, Sep. 2010, pp. 498–499.
[18] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Gou, "A semantics aware approach to automated reverse engineering unknown protocols," in *Proc. 20th IEEE Int. Conf. Netw. Protocols*, Austin, TX, USA, Oct./Nov. 2012, pp. 1–10.
[19] J.-Z. Luo and S.-Z. Yu, "Position-based automatic reverse engineering of network protocols," *J. Netw. Comput. Appl.*, vol. 36, no. 3, pp. 1070–1077, May 2013.
[20] Z. Zhang, Z. Zhang, P. P. C. Lee, Y. Liu, and G. Xie, "ProWord: An unsupervised approach to protocol feature word extraction," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr./May 2014, pp. 1393–1401.
[21] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafó, "Towards automatic protocol field inference," *Comput. Commun.*, vol. 84, pp. 40–51, Jun. 2016.
[22] J.-Z. Luo, S.-Z. Yu, and J. Cai, "Method for determining the lengths of protocol keywords based on maximum likelihood probability," *Chin. J. Commun.*, vol. 37, no. 6, pp. 119–128, Jun. 2016.
[23] S. Tao, H. Yu, and Q. Li, "Bit-oriented format extraction approach for automatic binary protocol reverse engineering," *IET Commun.*, vol. 10, no. 6, pp. 709–716, Apr. 2016.
[24] Z. Xiaoming, Q. Qian, W. Weisheng, W. Zhanfeng, and W. Xianglin, "IPFRA: An online protocol reverse analysis mechanism," in *Proc. Int. Conf. Cloud Comput. Secur.*, Haikou, China, Sep. 2018, pp. 324–333.
[25] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 1083–1097, Apr. 2018.
[26] X. Luo, D. Chen, Y. Wang, and P. Xie, "A type-aware approach to message clustering for protocol reverse engineering," *Sensors*, vol. 19, no. 3, p. 716, Feb. 2019.
[27] Y.-H. Goo, K.-S. Shim, M.-S. Lee, and M.-S. Kim, "Protocol specification extraction based on contiguous sequential pattern algorithm," *IEEE Access*, vol. 7, pp. 36057–36074, 2019.
[28] *Enterprise Single Sign-On for All*. Accessed: Jan. 2015. [Online]. Available: https://ereo.github.io/cas/4.2.x/protocol/CAS-Protocol.html
[29] C. W. Royer, M. O'Neill, and S. J. Wright, "A Newton-CG algorithm with complexity guarantees for smooth unconstrained optimization," *Math. Program.*, vol. 19, pp. 1–38 Jan. 2019.
[30] J. Nocedal and S. J. Wright, *Numerical Optimization*. Berlin, Germany: Springer, 2006.
[31] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, Apr. 2007.
[32] K. Thompson, "Programming techniques: Regular expression search algorithm," *Commun. ACM*, vol. 11, no. 6, pp. 419–422, Jun. 1968.
[33] R. Yang, W. C. Lau, and T. Liu, "Signing into one billion mobile app accounts effortlessly with OAuth2.0," in *Proc. Blackhat Eur.*, 2016, pp. 1–10.
[34] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems," in *Proc. ACM Conf. Comput. Commun. Secur.*, Raleigh, NA, USA, Oct. 2012, pp. 378–390.
[35] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," *Toorcon*, to be published.

XUDONG HE was born in 1991. He received the M.S. degree from the School of Computer Science, South-Central University for Nationalities, China, where he is currently a Research Assistant. His research interests include security protocol implementations and reverse engineering.

JIABING LIU was born in 1994. He is currently pursuing the master's degree with the School of Computer Science, South-Central University for Nationalities. His research interests include blockchain and smart contract security.

CHIN-TSER HUANG received the Ph.D. degree in computer science from the University of Texas at Austin, Austin, TX, USA. He is currently a Professor with the Department of Computer Science and Engineering, University of South Carolina, where he is also the Director of the Secure Protocol Implementation and Development (SPID) Laboratory. His current research interests include network security, network protocol design and verification, secure computing, and distributed systems.

DEJUN WANG was born in 1974. He received the Ph.D. degree in information security from Wuhan University, China. He is currently an Associate Professor with the School of Computer, South-Central University for Nationalities, China. He has authored or coauthored more than 20 articles in international/national journals and conferences. His current research interests include security protocols and formal methods.

BO MENG was born in China, in 1974. He received the M.S. degree in computer science and technology and the Ph.D. degree in traffic information engineering and control from the Wuhan University of Technology, Wuhan, China, in 2000 and 2003, respectively, where he was a Postdoctoral Researcher in information security, from 2004 to 2006. From 2014 to 2015, he was a Visiting Scholar with the University of South Carolina. He is currently a Full Professor with the School of Computer Science, South-Central University for Nationalities, China. He has authored or coauthored more than 50 articles in international/national journals and conferences. He has also published two books *Automatic generation and verification of security protocol implementations* and *secure remote voting protocol* (Science Press), China. His current research interests include blockchain, and security protocols and formal methods.