2009 Third International Conference on Network and System Security

Automatic Network Protocol Automaton Extraction

Ming-Ming Xiao[†‡], Shun-Zheng Yu[†], Yu Wang[†]
[†] Department of Electronics and Communication Engineering, Sun Yat-Sen University
[‡]Information College, Zhongkai University of Agriculture and Engineering
Guangzhou, P.R.China
{xmingm, hf98403}@gmail.com
syu@mail.sysu.edu.cn

Abstract—**Protocol reverse engineering, the process of (re)constructing the protocol context of communication sessions by an implementation, which involves translating a sequence of packets into protocol messages, grouping them into sessions, and modeling state transitions in the protocol state machine, is well-known to be invaluable for many network security applications, including intrusion prevention and detection, traffic normalization, and penetration testing, etc. However, current practice in deriving protocol specifications is either mostly manual or focusing on automatic reverse engineering the message format only and leaving the protocol state machine inverse undone. Although regular expressions offer superior expressive ability and flexibility, application protocols are described by regular expression manually based on sufficiently understanding protocol itself. At present there is not an effect method to realize classification, recognition and control automatically for the known applications and the unknown applications in future. In this paper a novel approach is presented to model network application specification. In this work, the whole automatic protocol reverse engineering is realized through accomplishing the protocol state machine, and then the FSMs are translated to corresponding regular expressions to enrich and update the pattern database. This approach uses grammatical inference and is motivated by the observation that an implementation of the protocol is inherently a state transition process, the state machine model the essence exactly. The important significance is to describe various state protocols with a common method through modeling the protocol state transition, including known and unknown ones. This approach had been implemented in the system and evaluated using real-world implementations of three different protocols: HTTP, SMTP, FTP, and compared the extracted protocol to the corresponding other newly system, such as l7-filter.**

Keywords- protocol reverse engineering; automaton inference; regular expression; Protocol analysis

I. INTRODUCTION

To date, there are 600 network protocols, 2000 network application in use on Internet and on enterprise network, moreover, various new applications even emerge in endlessly. Application-level protocol specifications are significantly useful for so heavy security tasks. For example, penetration testing can leverage protocol specifications to generate network inputs to an application to uncover potential vulnerabilities. For network management, protocol specifications can also be used to identify protocols and tunnelings in monitored network traffic [1]. Additionally, intrusion detection systems (e.g., Snort [2] and Bro [3]) perform deep packet inspection with the knowledge of application protocol. Finally, there are a lots of proprietary, closed protocol (e.g., QQ), and vicious application, such as worm, virus, need to be derived the protocol specification. Protocol reverse engineering aims to extract the application-level protocol used by an implementation, without access to the protocol specification. Thus, protocol reverse engineering is an essential tool for the above network security applications. Unfortunately, the protocol specifications for the above tasks are illustrated as documentation or reversed engineering manually. Such efforts are painstakingly time-consuming and error-prone.

To address the limitation of manual protocol analysis, the automatic protocol reverse engineering was proposed to tackle the problem. For example, the author in [1] proposed to extract the protocol message format with clustering method from network traces. Otherwise, Polyglot [4] and AutoFormat [5] propose to extract the field information of messages from observing the execution of a program during its processing execution traces. Furthermore, the author in [6] presented a novel approach to analyze multiple messages to extract the format specification and additional semantics for different types of messages. Finally, in [7] said that the grammatical inference approach presented only provides information that can assist an informed human analyst in protocol reverse engineering. In summary, there are much work to be carried out for the automatic protocol reverse engineer, but we can found that they all focus on reverse engineering the message format specification and leave the protocol state machine inference to be undone.

On the other hand, this deep packet inspection has performed by comparing against packet content to sets of strings traditionally. State-of-the-art systems, such as Snort, Bro, 3Com's TippingPoint X505, however, are substituting regular expressions for string sets, due to their superior expressiveness. Moreover, layer 7 filters also integrated the regular expressions [8] into the open source system available

978-0-7695-3838-9/09 $26.00 © 2009 IEEE
DOI 10.1109/NSS.2009.71

for the Linux operating system. At present, although regular expressions offer superior expressive power and flexibility, application protocols are described by regular expression manually based on sufficiently understanding protocol itself. Additionally, regular expression is typically complied into deterministic finite automata (DFA) during implementing. So, we can conclude that there is not an effect method to realize classification, recognition and control automatically for the known applications and the unknown applications in future at present.

In this work, we attempt to accomplish the rest part of automatic protocol engineering, i.e., realize protocol state machine inference. We propose to introduce grammatical inference to induce the protocol state machine representing the essence of protocol with the message format and semantics of protocol implementing, and then translate the FSM to corresponding regular expressions to update the pattern database. The important significance is to describe various state protocols with a common method through modeling the protocol state transition, including known and unknown ones.

Approach: Our approach uses grammatical inference and is based on a unique intuition—the way that an implementation of the protocol is inherently a state transition process, the state machine model the essence exactly. From as early as in the fifties [9], Grammatical Inference was already conceived as a technique used to learn syntax from example sentences. Most of the research efforts are concentrated on the learning of finite automata (i.e. regular grammars) [10]. It has been used so far to plenty of fields extensively, and also been applied to information extraction.

ECGI algorithm, one of GI techniques, proposed and used by Rulot and Vidal [11] [12] infers from positive examples a non-deterministic loop-free finite automaton. It can be considered as a Grammatical Inference "heuristic". It works in an incremental fashion: for each new example x, the current automaton A tries to recognize x. If this is not possible, A is modified for accepting x, by adding new states and new transitions. The techniques are fairly well known in the GI community and have been proven useful in a number of fields. Such as, it had been applied to the field of Automatic Speech Recognition [11] [12] [13]. In [14] [15] had presented the application of ECGI in image processing. Moreover, in the field of biology and medicine ECGI had been proposed for the classification of human chromosomes [16]. Additionally, ECGI technique also applied to the field of music processing [17] [18] [19]. Whereas the anthor in [7] have demonstrated the applicability of two grammatical inference algorithms for two specific protocols, he have not established generality of the approach. In this work, we introduce GI method to induce the protocol state machine with the message format and semantics extracted from execution trace, so that we can realize a common method to model various protocols.

Scope of the problem: The protocol reverse engineering is a hard work, which consist of two stages for implementation. During the first phase, we have to extract the message formats and related semantics along protocol session processing, which represent the essence feature of application protocol itself. In the second stage, we do our best to realize the protocol state machine derived from the order in which the message of different types can be sent in execution trace, which reflect the transition of protocol state, i.e., inherent structure of protocol. In this work, we concentrate on deriving the protocol state machine due to 1) it is an unwonted field of protocol reverse engineering at prsent, and 2) as we discussed above that it is only FSM to represent the essence of protocol implementing.

Contributions: In summary, this work makes the following contributions:

- We present a novel approach for the automated derivation of protocol state machine. As discussed above, the extraction of protocol state machine is an unreached goal in the protocol reverse engineering at present, we attempt to accomplish the rest part work of it.
- We reveal a new method to model common protocol. In essence the process of protocol implementing is the course of state transition of protocol. It can be concisely said that the key element of protocol trigger the state transformation between message in each protocol session. A series of ordered state form the protocol state machine which represents the essential structure of the application protocol.
- We introduce grammatical inference algorithm to model protocol state machine. The grammatical inference was applied to a number of fields successfully. The result of grammatical inference is regular grammar, i.e., the finite state machine, which exactly is a nice model for the application protocol.
- We applied our techniques to a set of real-world applications that implement complex protocols such as HTTP, SMTP, and FTP. Our results show that we can automatically generate specifications that can be used to parse messages of certain types.

The remainder of the paper is organized as follows. In Section Ⅱ we describe the approach and system architecture. Then, in Sections Ⅲ we present our techniques to derive the message format and the protocol state machine. We evaluate our approaches in Section Ⅳ and summarize related limitation in Section Ⅴ. Finally, we conclude in Section Ⅵ.

II. APPROACH AND SYSTEM ARCHITECTURE DESIGN

In the following section we present our approach for the problem mentioned above. Then introduce the overview of the system architecture.

Our method, using Grammatical Inference(GI) for protocol reverse engineering, is based on the fact — Grammatical Inference (GI) aims at learning models of languages from examples of sentences of these languages, and sentences can be any structured composition of primitive elements or symbols, though the most common type of composition is the concatenation. From this point of view, GI find applications in all those many areas in which the objects or processes of interest can be adequately represented as strings of symbols [19]. In our task, i.e., automatic protocol reverse engineering, network signature information shown the protocol structure would be represented as primitive strings of symbols, which satisfy the demand of GI.

To enable GI for automatic protocol reverse engineering, it is essential that protocol implementing behavior was characterized as primitive strings. During the stage, we reference the method in [4] to extract the information about the field boundaries and the keywords that form the basis of message format upon the program execution. The method can only indicate the field boundaries but not message boundaries. In our goal to reflect the state transition between messages in a session, i.e., sessions can contain multiple messages, so we need to identify the message boundaries as well.

The system architecture in our design is shown in Fig. 1. As depicted in the figure, our system design mainly consist of two stages in summary. The first stage is the message format extraction, which is implemented by execution monitor [20] and messages format extraction model [4]. It takes as input the program's binary and the application data, and dynamically monitors how the program processes the application data. The output of the execution monitor is an execution trace that contains a record of all the instructions performed by the program. And then the execution trace is analyzed to indicate the field boundaries and the keywords extraction during message format extraction. With respect to message boundaries, in this work we use tools that are based on libPCap [21] to indicate message boundaries. LibPCap has a number of implementations in a variety of languages. The tools are embedded in our system.

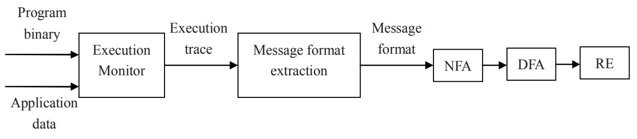


Figure 1. System architecture overview

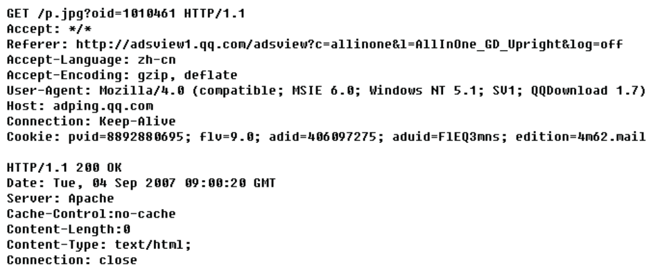
After finishing message boundaries and message format extraction, during the second stage in our framework, we select only specific key format of the execution trace to learn the finite state machine with GI algorithm thereby providing significant complexity reduction. At first, it takes as the input the selected format information to learn the nondeterministic finite automata (NFA) directly which reflect the protocol state transition context, and then transfers the NFA to DFA so that generate a compact finite state machine (FSM), and finally translate the DFA into regular express (RE) due to its power expressive capabilities.

III. AUTOMATIC NETWORK PROTOCOL EXTRACTION

The previous section only simply demonstrated the method and the system framework. In this section, we describe the detailed technique of our solution. We first reference related method to realize message format extraction, and determine the message boundaries in the session context with our tools. Based on these precondition, we introduce the GI algorithms to learn the DFA for recognizing the protocol application. We last transfer the DFA into RE to enrich the signature database because of its strong expressive ability.

A. Protocol Message Extraction

A protocol is an agreement, or a standard language, between two entities that need to communicate. In [1] [6], the authors summarize that most application protocols have a notion of an application session, which allows two hosts to accomplish a specific task. An application session consists of a series of individual messages. These messages can have different types. Each message type is defined by a certain message format specification. A message format specifies a number of fields, for example, length fields, cookies, keywords, or endpoint addresses (such as IP addresses and ports). The structure of the whole application session is determined by the protocol state machine, which specifies the order in which messages of different types can be sent. A simple example of application protocol session from real-world data is shown in Fig. 2. In the session there are two hosts to interact each other with HTTP protocol. There are two messages to accomplish the task in the simple example.



```
GET /p.jpg?oid=1010461 HTTP/1.1
Accept: */*
Referer: http://adsview1.qq.com/adsview?c=allinone&l=AllInOne_GD_Upright&log=off
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 1.7)
Host: adping.qq.com
Connection: Keep-Alive
Cookie: pvid=8892880695; flv=9.0; adid=406097275; aduid=F1EQ3mns; edition=4m62.mail

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2007 09:00:20 GMT
Server: Apache
Cache-Control:no-cache
Content-Length:0
Content-Type: text/html;
Connection: close
```

Figure 2. A session of application protocol

As discussed previously in Fig. 1, we use dynamic data tainting to track the bytes of the messages that are read by the application. Similar to previous systems that use tainting [22], [23], [24], each input byte receives a unique label. Then, we keep track of each labeled value as the program execution progresses. As a result of the dynamic data tainting process, an execution trace is produced for each message. This trace contains all operations that have one or more tainted operands [6].

In the next stage, we analyze the execution trace to locate the field boundaries and the keywords. This phase consists of four modules: the separator, direction field, keyword and message format extraction modules [4]. First, the direction field and the separator extraction modules take care of finding the boundaries of variable-length fields. Next, the keyword extraction module takes as input the separators and the execution trace and outputs the keywords. Finally, the message format extraction module takes care of finding the boundaries of fixed-length fields and of combining all previous information to generate the message format. It takes as input the previously found separators, direction fields and keywords, as well as the execution trace, and outputs the message format. The detailed techniques mentioned above can be found in [4].

B. *Message Boundaries Indication*

Once we have finished the message format extraction and the field boundaries location, another problem resolved about our task is to indicate the boundaries between messages in each session. The goal of our task is to use the captured information to extract an FSM, and the FSM codifies the features of the protocol that are used in the execution traces. So the execution traces must consist of protocol interaction sessions, and each session should be a sequence of message. The learning of protocol state machine is determined by the structure of the whole application session, which reflects the order of different type message sent. In [4], as discussed above, it can implement the message format extraction and the field boundaries indication successfully, but it can not indicate the boundaries between messages in each session. In this work we use tools that are based on libPCap to differentiate the message for each session in execution trace. LibPCap has a number of implementations in a variety of languages. We embedded libPCap in our system during processing phase of protocol message format extraction. So we can determine the occasion of protocol state transition perfectly, which builds the essential base for the coming learning of FSM.

C. *Message Format Selection*

As discussed above, after we implemented the protocol message format extraction from execution trace, we can gather abundance information about message format, e.g. the field start position in the message, the field length, the field boundary, the field type the field keywords, separator, and so on. In Fig. 2, it is an example of HTTP interaction session extracted from real world data. According to the message format extraction methods mentioned above, we can summarize the partial message format extracted for Fig. 2 in TABLE Ⅰ.

TABLE I. THE PARTIAL MESSAGE FORMAT EXTRACTION

Separator	Keyword
0x0d0a (‘CRLF’) 0x2f (‘/’) 0x2e (‘.’) 0x20 (‘ ’) 0x3a20 (‘: ’)	‘GET’ ‘Host’ ‘User-Agent’ ‘Accept’ ‘Accept-’ ‘close’ ‘Connection’ ‘HTTP/’ ‘HTTP/1.1’ ‘Keep-Alive’ ‘200’

It is unrealistic to include the all message format during the training stage of FSM. In practice, it is only some pivotal signatures which trigger the transition of protocol states from our observation of view. So another challenge in this work is of how to determine the key message format for protocol state machine inference. We select only specific key format of the execution trace to learn the finite state machine with GI algorithm thereby providing significant complexity reduction. For example, we would select the keyword representing interaction method and response code, e.g. ‘GET’, ‘HTTP/’, ‘200’, as key format of protocol session for HTTP session, combining the boundaries information between messages. So we must adopt heuristic for the sake of accomplishing our goal.

D. *Automaton Inference*

After gathering the structure information of application protocol, in the following stage, we launch on learning the corresponding protocol state machine with these materials. When we have hold the order in which the messages of different types can be sent in execution trace, i.e. the boundaries of message, combining the message format, we would adopt related methods to inference the protocol state machine. In this work, we impose the grammatical inference (GI) to realize our goals.

Grammatical Inference was already conceived as a technique used to learn syntax from example sentences as early as in the fifties [9]. Most of the research efforts are focused on the learning of finite automata (i.e. regular grammars) [10].

Error Correcting Grammatical Inference (ECGI) is a heuristic learning method that was introduced to obtain structural, finite-state models of (unidimensional) objects from samples of these objects. ECGI capitalizes on error-correcting techniques not only to learn the structural models and to train the assumed error-model parameters, but also to recognize new objects with the learned models.

Technical descriptions of ECGI have been presented elsewhere [11] [12] [14] and only the most important features will be outlined here.

Let R+ be a sequence of training strings representing a given class of objects. Initially, a trivial finite state grammar is built that only generates the first string of R+. Then, for every new string that cannot be parsed with the current grammar, the grammar is updated by adding certain non-terminals (states) and rules (transitions). In order to establish such rules, a standard error correcting scheme is adopted to determine a string in the language of the current grammar that is closest to the input string in an error-correcting sense. This is achieved through a Dynamic Programming error-correcting parsing procedure which also yields the corresponding optimal sequence of both non-error and error rules used in the parse. From these results, the current grammar is updated by adding a number of rules (and non-terminals) that permit the new string, along with other adequate generalizations, to be accepted [16]. At the end of the process, the inferred automaton accepts all the examples, but not only the examples: it has generalized to a (finite) language, grossly speaking, composed of sentences resulting of concatenations of sub-sentences of the examples.

Let $G=(N,\Sigma,P,S)$ be a characteristic (non-stochastic) regular grammar or finite-state acceptor supplied by ECGI. Here, N is the set of Non-terminals (or states), Σ is the set of terminal symbols, P is a set of regular Rules (or transitions) and S is the starting nonterminal (or state).

The error rules of G can be defined as follows:

For each rule in P, each $a, b \in \Sigma$, define the error rules associated to P by :

- Insertion (of a) :
  $A \rightarrow aA, \forall (A \rightarrow bB) \in P$, $A \rightarrow ab, \forall (A \rightarrow b) \in P$;

- Substitution (of a by b) :
  A →aB, ∀(A →bB)∈P, A→a, ∀(A→ b)∈P;
- Deletion (of b) :
  A → B, ∀ (A→bB)∈P, A → ε , ∀(A→ b)∈P.

This building step is a key step in which heuristics are implemented. Fig. 3 shows an example of this inference process. Bold lines represent consolidated transitions (rules) and states (non-terminals) of the successively inferred acceptors. At each step, the error-rules used in the error-correcting parsing are shown as dashed arrows, while the corresponding added rules and non-terminals are shown with continuous (thin) lines. The whole procedure infers a grammar which depends on the order of the examples.

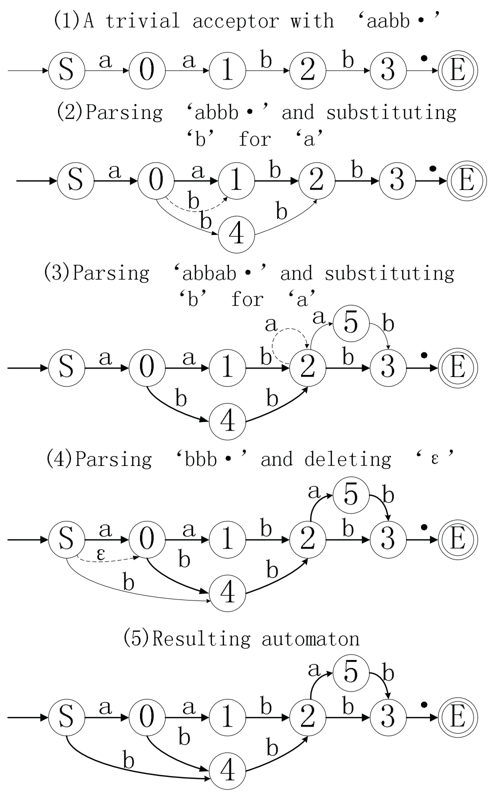

(1)A trivial acceptor with 'aabb•'
(2)Parsing 'abbb•' and substituting 'b' for 'a'
(3)Parsing 'abbab•' and substituting 'b' for 'a'
(4)Parsing 'bbb•' and deleting 'ε'
(5)Resulting automaton
S
0
1
2
3
4
5
E
a
b
•
ε

Figure 3. ECGI trace for the sequence of learning strings R_{+}={aabb,abbb,abbab,bbb}.

In order to apply the methods outlined above, the object patterns required both for learning and recognition must be described by strings of primitives, each primitive representing the appropriate local features of the corresponding object structure. Once object shapes are adequately represented as strings of the appropriate primitives, these strings can be straightforwardly supplied to ECGI both for learning and recognition.

Application protocol data are basically sequences of events, after implementing message format extraction and message boundaries indication, as mentioned in preceding section, these structure information are represented as strings of the appropriate primitives. Once again, for the example depicted in Fig. 2, as discussed in section Ⅲ, we can generate the following rule based on the key format of the protocol session during the processing of automaton inference:

$$A \xrightarrow{\text{GET HTTP/}} B \text{ , } B \xrightarrow{\text{HTTP/ 200}} C$$

Then, we apply the same analysis repeatedly to all the other sessions of samples recursively, we can create the whole FSM. In this work, we walk two steps to accomplish the FSM learning processing. At first, we realize the NFA with ECGI algorithm. In the second step, we transfer the NFA into a DFA, the more detailed content about the processing would be demonstrated in section Ⅳ.

E. Retro-RE

Finite automata are a natural formalism for regular expressions (RE). Because of the power expressive capabilities of RE, the pattern was described as RE rather than the set of strings any more. Once we got the DFA inferred from the execution trace of application implementing, we can convert the DFA to regular expressions, so that the pattern database would be updated. As mentioned previously, the structure of application protocol is represented as DFA, the structure inherently present as a hierarchical (tree) structure in practice. So we can construct an analytical tree to describe the conversion from DFA to regular expressions. The leaves of the analytical tree are labeled by the alphabet characters of rules upon the DFA, and each leaf is connected to the whole of the tree by a set of operator in the DFA, i.e., { |,*, • ,}, for concatenation, the detailed analytical tree can be shown in next section.

IV. Evaluation

In this section, we present the experimental evaluation of our approach. We have evaluated our system extensively using 9 different programs implementing 3 different protocols (HTTP, SMTP, FTP) as shown in TABLE Ⅱ. In order to take maximum statistical advantage of the data available, the experimentation was carried out following a procedure of the type known as "leaving-k-out" or "Cross-Validation" [25]. For this goal, we can find that program http1 in TABLE Ⅱ consist of program http2 & http3. It is similar to smtp1 and ftp1 for the same goal.

As discussed above, we introduce the approaches in [4] [6] to extract the message format from execution trace, and then select the key format for inducing the FSM thereby providing significant complexity reduction.

After we use tools embedded libPCap to differentiate the message for each session in execution trace，i.e, to indicate the boundaries of message, we gather these key format information represented as string of the appropriate primitives, and the message boundaries is the interface of protocol state transformation, then we can be straightforwardly supplied these material to ECGI for learning the FSM of protocol. As mentioned in preceding section, we impose two steps to accomplish the FSM learning processing. At first, we induce the NFA with ECGI algorithm. Fig. 4 is the partial result of NFA for HTTP protocol with Thompson automaton structure [26]. We only illustrate the part of the result of learned NFA for conciseness, the detailed results are shown in the following TABLE Ⅲ.

TABLE II. TEST CASE SUMMARY

Program	Type	#Session	OS
http1	HTTP server	9101	Win
http2	HTTP server	2423	Win
http3	HTTP server	6678	Win
smtp1	SMTP server	116	Win
smtp2	SMTP server	50	Win
smtp3	SMTP server	66	Win
ftp1	Serv-U FTP Server	6901	Win
ftp2	Serv-U FTP Server	2100	Win
ftp3	Serv-U FTP Server	4801	Win

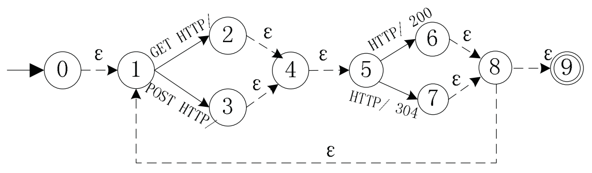


Figure 4. The incomplete Thompson NFA for HTTP

At second step, we convert the NFA to DFA with DFAClassical algorithm [27]. The corresponding DFA of NFA in Fig. 4 is shown in Fig. 5.

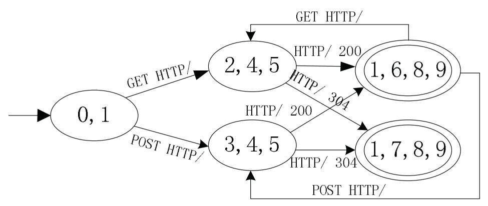


Figure 5. The corresponding DFA in Fig. 4

As mentioned in section Ⅲ, the structure of application protocol is represented as DFA, the structure inherently present as a hierarchical (tree) structure in practice. So we can construct an analytical tree to describe the conversion from DFA to regular expressions. The analytical tree of the DFA in Fig. 5 is illustrated in Fig. 6.

From the analytical tree, we can derived the corresponding RE as ((GET)(HTTP/)((HTTP/200)|(HTTP/)(304)))*|((POST)(HTTP/)((HTTP/200)|(HTTP/)(304)))* .

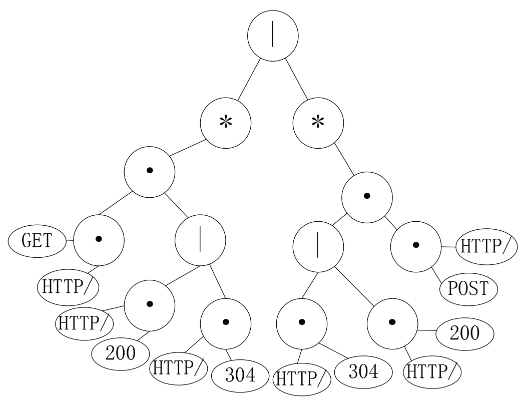


Figure 6. The analytical tree for the DFA in fig. 5

We implement all the cases indicated in TABLE Ⅱ with our inference algorithm, we can generate the following rule during the processing of automaton inference. The derived DFA are summarized in the TABLE Ⅲ. The left part of rule is the current state, indecated all by 'A', the right part one is the key code triggering state transfer.We have not illustrated the whole FSM derived due to the limitation of space in the paper.

TABLE III. THE SUMMARY OF INFERRED DFA

Execution trace	Inferred rules	#States	protocol
http1 (http2+http3)	A→GET HTTP/;A→HEAD HTTP/;A→POST;A→OPTION;A→HTTP/ 200; A→HTTP/ 304; A→HTTP/ 302; A→HTTP/ 202; A→HTTP/ 400; A→HTTP/ 403; A→HTTP/ 404; A→HTTP/ 503; A→HTTP/ 204; A→HTTP/ 408; A→HTTP/ 301; A→HTTP/ 301; A→HTTP/ 500; A→HTTP/ 206; A→HTTP/ 502	37	HTTP
smtp1 (smtp2+smtp3)	A→220; A→EHLO; A→HELO; A→250; A→AUTH LOGIN; A→334; A→235; A→MAIL FROM; A→RCPT TO; A→DATA; A→354; A→Date; A→QUIT; A→221; A→553	32	SMTP
ftp1 (ftp2+ftp3)	A→220; A→USER; A→331; A→PASS; A→230; A→530; A→QUIT; A→221; A→CWD; A→250; A→TYPE; A→PORT; A→PSAV; A→227;A→200; A→SIZE; A→213; A→RETR; A→150; A→226; A→LIST; A→REST;350; A→500; A→550; A→125; A→450	49	FTP

For verifying the accuracy of the derived DFA, we implement the procedure of "Cross-Validation" to the inferred DFA. We drive the derived DFA with the original execution trace as input. The summary of ("Cross-Validation") result for recognition rate of the derived DFA is illustrated in TABLE Ⅳ. From the result we can obtain high recognition rate for extracted DFA. The recognition rate arise somewhat decrease along with the increase of the protocol complexity. We can find that the derived DFA is considerable stable for the training set in the same group, as well as ones in hybrid train cases.

TABLE IV. THE RECOGNITION RATE FOR INFERRED DFA

DFA / Test trace	http1	http2	http3	smtp1	smtp2	smtp3	ftp1	ftp2	ftp3
http1	95.90%	95.77%	95.69%	0	0	0	0	0	0
http2	95.82%	95.88%	95.84%	0	0	0	0	0	0
http3	95.77%	95.75%	95.79%	0	0	0	0	0	0
smtp1	0	0	0	92.57%	92.43%	91.98%	0	0	0
smtp2	0	0	0	92.42%	92.46%	92.37%	0	0	0
smtp3	0	0	0	92.33%	92.30%	92.44%	0	0	0
ftp1	0	0	0	0	0	0	81.27%	81.03%	80.76%
ftp2	0	0	0	0	0	0	81.36%	81.54%	80.36%
ftp3	0	0	0	0	0	0	81.34%	80.21%	81.65%
http1+smtp1	94.69%	94.56%	94.49	1.17%	1.16%	1.16%	0	0	0
http1+ftp1	54.54%	54.46%	54.42%	0	0	0	35.05%	34.94%	34.83%
Smtp1+ftp1	0	0	0	1.53%	1.53%	1.52%	79.93%	79.69%	79.42%

We also compare the derived RE in this work with l7-filter RE, the result was shown as TABLE Ⅴ. It is found that there is considerable difference between the two cases. It is owing to the difference of the extracting process. The l7-filter RE is extracted manually based on discerning the protocol in advance, nevertheless, in this work, the RE is derived from DFA which is inferred automatically by capturing the key elements during the implementing of the application protocol. We believe that the result obtained in our approach represent the essence of protocols much more.

TABLE V. THE DERIVED RE VS. L7-FILTER RE

RE / Protocol	derived RE	L7-filter RE
HTTP	((GET)(HTTP/)((HTTP/200)|(HTTP/)(304)))*| ((POST)(HTTP/)((HTTP/200)|(HTTP/)(304)))*| ((HEAD)(HTTP/)((HTTP/200)|(HTTP/)(304)))*| ((OPTION)(HTTP/)((HTTP/200)|(HTTP/)(304)))*	http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9] [\x09-\x0d -~]*(connection:|content-type:|content-length: |date:)|post [\x09-\x0d -~]* http/[01]\.[019]
SMTP	220((EHLO)(250)(AUTH)(LOGIN)(334)(==)(334) (==)(235)((MAIL FROM)(250)(RCPT TO)(250)(DATA)(354)(Date)(250))*(QUIT)(221)| (HELO)(553)(QUIT)(221))	^220[\x09-\x0d -~]* (e?smtp|simple mail) userspace pattern=^220[\x09-\x0d -~]* (E?SMTP|[Ss]imple [Mm]ail) userspace flags=REG_NOSUB REG_EXTENDED
FTP	220(USER)(331)(PASS)((230)|(530))((QUIT)(221) | (((TYPE)(200)|(PSAV)(227))(PORT)(200) ((REST)(350)|(SIZE)(213)|(LIST))(150)(226))*)	^220[\x09-\x0d -~]*ftp

V. DISCUSSION

Protocol reverse engineering is an arduous task. In this work, we attempt to extract the protocol state machine from the execution trace of application protocol implementing, and we present the primarily evaluation result. There are some limitations lying in our work. At first, our work only derived a few protocol, the other protocol are currently not extracted. Moreover, we can only obtain the structure from the messages given to our analysis. We got nothing about hidden part of protocol state machine. Additionally, the accuracy of the derived FSM in this work is deficient for the more complex protocol, especially for the unknown protocol. Because the extracted FSM overly depend on the veracity of the corresponding message format extraction, and the key message selected. It is a challenge to determine a metric for the selection of the key massage in this work. It deserves to pay out about how to improve and measure the accuracy.

We hope our work give an explicit direction for the work in the future.

VI. CONCLUSION

It is well-known to be valuable about protocol reverse engineering for many network security applications, including intrusion prevention and detection that performs deep packet inspection and traffic normalization, penetration testing that generates network inputs to an application to uncover potential vulnerabilities, and so on. Unfortunately, current bread-and butter approach to determine protocol specification depends on analyzing manually. Current technique of protocol reverse engineering rests on the stage of message extraction automatically. Whereas RE was extensively applied to newly system, such as l7-filter, Bro, Snort, and so on, RE also are generated manually based on analyzing the implementing context of application protocol in advance. In this work, we applied the grammatical inference to model protocol specification as FSM from the extracted message formats, furthermore we transform the FSM into more expressive form, RE, to enrich and update the pattern database. Our experiments with real-world protocols and server applications demonstrate that the derived FSM take on well differentiating capability for different types of application protocols. We believe that the techniques that we introduce in this paper will be useful for related security policy.

ACKNOWLEDGMENT

This work has been supported by the Key Program of NSFC-Guangdong Joint Funds of China and by the National High Technology Research and Development Program of China under grants U0735002 and 2007AA01Z449, respectively, for which we are grateful. We would like to thank all members of the network group for helpful discussions. We would also like to thank the anonymous reviewers for their insightful comments to improve this paper. We are thankful to everyone's help.

REFERENCES

[1] G. Eason, W. Cui, J. Kannan, and H. Wang, "Discoverer: Automatic Protocol Reverse Engineering from Network Traces," In 16th Usenix Security Symposium, August 2007 , pp. 199-212.

[2] M. Roesch, "Snort: Lightweight intrusion detection for networks," In Proc. 13th Systems Administration Conference (LISA), USENIX Association, November 1999, pp. 229–238.

[3] "Bro: A System for Detecting Network Intruders in Real-Time," http://www.icir.org/vern/bro-info.html

[4] J. Caballero and D. Song, "Polyglot: Automatic Extraction of Protocol Format using Dynamic Binary Analysis," In ACM Conference on Computer and Communications Security (CCS), October 2007, pp. 317-329 .

[5] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic Protocol Format Reverse Engineering through Conectect-Aware Monitored Execution", In 15th Symposium on Network and Distributed System Security (NDSS), February 2008.

[6] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda, "Automatic Network Protocol Analysis," Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), Feb. 2008, pp. 1-18.

[7] Mark E. DeYoung, "Dynamic Protocol Reverse Engineering: A Grammatical Inference Approach," Master's thesis. Air Force Inst. of Tech., Wright-Patterson AFB, OH. Graduate School of Engineering and Management.

[8] J. Levandoski, E. Sommer, and M. Strait, "Application Layer Packet Classifier for Linux". http://l7-filter.sourceforge.net/.

[9] Chomsky (N.) et Miller (G. A.). Pattern conception. Rapport technique, AFCRC-TN-5757, 1957. (ASTIA Document AD 110076).

[10] Jacques Chodorowski, Laurent Miclet, "Applying grammatical inference in learning a language model for oral dialogue," Lecture Notes in Computer Science, Volume 1433, 1998.

[11] Rulot (H.), Vidal (E.), "An efficient algorithm for the inference of circuit-free automata," Syntactic and Structural Pattern Recognition, 1988, pp. 173-184.

[12] Rulot (H.), Prieto (N.), Vidal (E.), "Learning accurate finite-state structural models of words: the ecgi algorithm," In : ICASSP'89, 1989, pp. 643-646.

[13] N. Prieto, E. Vidal, "Automatic Learning of Structural Language Models", Proceedings ICASSP-91, 1991, pp. 789-792.

[14] E. Vidal, H. Rulot, J. M. Valiente, and G. Andreu, "Recognition of Planar Shapes Through the Error-Correcting Grammatical Inference Algorithm (ECGI)," Research Report DSIC-II/32/91, Univ. Politécnica de Valencia (Spain), 1991.

[15] E.Vidal,, H.Rulot,, Valiente, J.M. ,Andreu, G., "Application of the error-correcting grammatical inference algorithm(ECGI) to planar shape recognition," IEEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives, Apr 1993.

[16] E.Vidal, M.J.Castro, "Classification of Banded Chromosomes using Error-Correcting Grammatical Inference (ECGI) and Multilayer Perceptron," VIIth National Symposium on Pattern Recognition and Image Analysis, 1997.

[17] P Cruz-Alcazar, E Vidal-Ruiz, "A study of Grammatical Inference Algorithms in Automatic Music Composition and Musical Style Recognition," ICML-97 Workshop on Automata induction, grammatical, 1997.

[18] PP Cruz-Alcazar, E Vidal-Ruiz, "Learning Regular Grammars to Model Musical Style: Comparing Different Coding Schemes," Grammatical Inference: 4th International Colloquium, ICGI-98, 1998.

[19] PP Cruz-Alcazar, E Vidal-Ruiz, "Modeling musical style using grammatical inference techniques: a tool for classifying and generating melodies," Third International Conference on Web Delivering of Music, 2003.

[20] H. Yin, D. Song, E. Manuel, C. Kruegel, and E. Kirda, "Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis," ACM Conference on Computer and Communications Security, Alexandria, VA, October 2007.

[21] JWS, Tcpdump public repository, http://www.tcpdump.org/.

[22] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, "Understanding Data Lifetime via Whole System Simulation," In Usenix Security Symposium, 2004.

[23] J. Crandall and F. Chong, "Minos: Control Data Attack Prevention Orthogonal to Memory Model," In 37th International Symposium on Microarchitecture (MICRO), 2004.

[24] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-End Containment of Internet Worms," In 20th ACM Symposium on Operating Systems Principles (SOSP), 2005.

[25] S.J.Raudys and A.K.Jaiii, "Small Sample Size Effects in Statistical Patten Recognition: Reconmendations for Practitioners," IEEE Trans on PAMI, vol.13, no.3, March 1991, pp. 252-264.

[26] K. Thompson, "Regular expression search algorithm," Communications of the ACM, 11:419-422, 1968.

[27] A. V. Aho, R. Sethi, and J. D. Ullman, Compilers—Principles, Techniques and Tools. Addison-Wesley, 1986.