A Survey on Methods of Automatic Protocol Reverse Engineering

Xiangdong Li
School of Computer Science
Zhongyuan University of Technology
Zhengzhou, China
e-mail: lixiangdong2510@126.com

Li Chen
Computer Center
Henan University of Economics and Law
Zhengzhou, China
e-mail: chlit123@yahoo.com.cn

***Abstract*—Protocol reverse engineering is necessary for many security applications, and has been an important and active research topic. This paper makes a survey on automatic protocol reverse engineering, especially on the goals, methods, tools and their achievements. The targets and obstacles in automatic protocol reverse engineering are identified. Then the evolution of the technique and tools proposed for protocol reverse engineering are reviewed chronologically. Finally we summarize those difficulties and problems that are still open to solve, and envision the prospects of this field.**

Keywords-protocol; reverse engineer; binary analysis; tainted analysis; network-based; program-based;survey

I. Introduction

A protocol is a set of rules and conventions prescribing how participant components should communicate. Protocols play an important role in networks and information security, and have pervasive and profound influence on security applications.

Many security applications value and depend much on the knowledge of network protocol specifications. Network-based firewalls require the knowledge of protocol specifications to understand the content of network communication sessions. Intrusion detection systems (IDS) or vulnerability-specific filters require knowledge of protocols to perform deep packet inspection and take corresponding actions. Network management software utilizes such knowledge to correctly recognize and classify monitored network traffic. Fuzz testing can take advantage of such knowledge to improve fuzzing process by generating “malicious” inputs more efficiently, or generating more effective mutations that have wider or deeper code-coverage; Such knowledge is also highly valuable in a variety of security assessment, vulnerability discovery and forensic applications.

But in many cases, protocol specifications are kept secret by their owners or inventors for reasons including pursuing profits and keeping technology advantages. In addition, protocols are constantly evolving with new features and functionalities that lead the previously known specifications incomplete or not applicable.

Therefore, the knowledge of specifications of non-publicly “open-source” protocols is traditionally obtained by reverse engineering, mostly a manual process which is time-consuming and error-prone. To address this challenge, a number of methods have been proposed, and prototype systems have been developed to allow for efficient or even automatic protocol reverse engineering.

In this paper, we make a survey on automatic protocol reverse engineering, especially on the goals, methods, tools and their achievements. We identify the targets and obstacles in automatic protocol reverse engineering, trace the evolution of the techniques, summarize the existing problems, and envision the perspectives of this field.

II. Goals of Protocol Reverse Engineering

The goals of protocol reverse engineering may vary a little with different applications. In packet filtering and traffic monitoring, establishing the knowledge message formats and semantics of the target protocol is fairly enough. However, in fuzz testing and intrusion detection systems, it is better to know more about the protocol.

What we already know about the protocol also determines the goal of reverse engineering. If the protocol specification is completely open as those proposed and released by IETF in RFC documents, or the protocol is open source, then reverse engineering of the protocol is almost unnecessary except that we want to know whether there are some variants between the implementation and the specification. But for most cases in practice, neither of the above is available. Usually, we only have access to some kind of binary executable of the protocol. For example, we need to reverse engineer a file transfer protocol different from the well-known one specified by RFC959[1]. What we have known about the protocol is that it can carry out file transfer tasks on networks, that its communication traffic can be captured by tools like Wireshark[2], and that, for most, we obtain a binary executable program of its client. That is a typical scenario for protocol reverse engineering.

In general, protocol reverse engineering is about capturing unpublished or undocumented features of an implemented protocol. Next, we will analyze what we can get easily, and what we have to dig hard to reveal a protocol features.

A. Knowledge of Protocol Functions

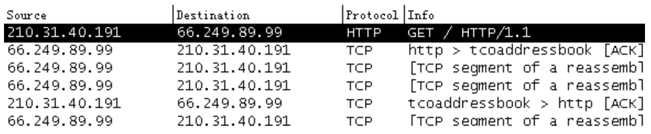
Usually, before starting to reverse engineering, we roughly know the main function of the target protocol. But if the binary executable of the protocol is available, the first thing for a researcher to do to know more exactly about its functions, is to run and test the protocol personally, watch and record inputs and responds.

Protocol functions are sensible via protocol interface. The protocol input and output at user interface provides useful information about meaning and formats of message fields such as user names, IP addresses, website URLs and other prompting strings. These are visible to users researchers, and helpful in the later stage of reverse engineering.

For some purpose such as vulnerability discovering, reading the protocol user-guide carefully is very helpful, because it reveals a lot to reserve engineering researchers. However, protocol interface and user manual is far from enough. There are many more functions which are within protocol binaries and important for programmers or hackers, but difficult to know or find. For examples, back doors and covert channels which exist in a protocol are not easy to find by anyone who just uses it.

B. *Knowledge of Protocol Messages*

When a protocol is running，among the protocol parties there are many message exchanges which are invisible to human eyes but visible to network packet capture tools such as tcpdump and Wireshark. These tools are also called network packet analyzers, which run silently, intercepting and displaying the communications sent on a shared network. Figure 1 is part of the Wireshark screen showing part of packets captured when a user (with IP address 210.31.40.191) visits Google's homepage (with IP address 66.249.89.99).



```
Source          Destination     Protocol Info
210.31.40.191   66.249.89.99    HTTP     GET / HTTP/1.1
66.249.89.99    210.31.40.191   TCP      http > tcoaddressbook [ACK]
66.249.89.99    210.31.40.191   TCP      [TCP segment of a reassembl
66.249.89.99    210.31.40.191   TCP      [TCP segment of a reassembl
210.31.40.191   66.249.89.99    TCP      tcoaddressbook > http [ACK]
66.249.89.99    210.31.40.191   TCP      [TCP segment of a reassembl
```

Figure 1. Packets captured when visiting Google's homepage

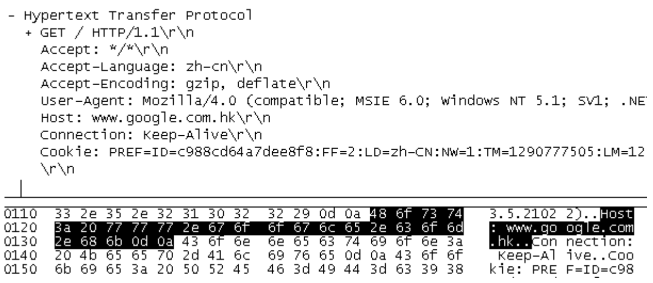
If enough packets are captured, hints and clues about what really happens behind the protocol interfaces will be accumulated and eventually lead to reveal the protocol specification.

C. *Knowledge of Protocol Syntax*

Protocols use mechanisms for participants to identify and response with each other, as well as rules of formatting messages that specify how data is packaged into messages sent and received. We call these rules as protocol syntax. One of the main goals of reverse engineering a protocol is to reveal and establish its syntax, that is, to determine the location and lengths of fields within protocol packets.

There are many protocol decoders (or dissectors) in Wireshark allowing users to examine protocol contents in details, as shown in Figure 2. These decoders are each built up based on their protocol syntaxes.

But for unknown protocols, we have to derive their syntaxes by reverse engineering.



```
- Hypertext Transfer Protocol
  + GET / HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: zh-cn\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NE
    Host: www.google.com.hk\r\n
    Connection: Keep-Alive\r\n
    Cookie: PREF=ID=c988cd64a7dee8f8:FF=2:LD=zh-CN:NW=1:TM=1290777505:LM=12
    \r\n

0110  33 2e 35 2e 32 31 30 32  32 29 0d 0a 48 6f 73 74   3.5.2102 2)..Host
0120  3a 20 77 77 77 2e 67 6f  6f 67 6c 65 2e 63 6f 6d   : www.go ogle.com
0130  2e 68 6b 0d 0a 43 6f 6e  6e 65 63 74 69 6f 6e 3a   .hk..Con nection:
0140  20 4b 65 65 70 2d 41 6c  69 76 65 0d 0a 43 6f 6f    Keep-Al ive..Coo
0150  6b 69 65 3a 20 50 52 45  46 3d 49 44 3d 63 39 38   kie: PRE F=ID=c98
```

Figure 2. Protocol details decoded based on protocol syntax in Wireshark

D. *Knowledge of Protocol State-machine*

Protocols running on networks can be classified as stateless or state-based. In a stateless protocol, such as HTTP, the server answers client requests without retaining any client information. On the other hand, in a state-based protocol, such as FTP, the server remembers that the user logged into the server and which directory the user was working with. There are distinct states in FTP conversations, including log-in, log-out, and file transfer requests.

The state machine of a protocol is an essential component in the protocol's specification that characterizes all possible legitimate states and sequences of messages. A state machine of a protocol can be viewed as:

- a set of input messages,
- a set of output messages,
- a set of states,
- a function that maps states and input to output,
- a function that maps states and inputs to states (which is called a state transition function) , and
- a description of the initial state.

In this paper, we focus on protocols running on application layer of networks. This is just for the sake of simplicity, and protocols on other layers can be analyzed and reverse engineered in the way discussed in this paper.

III. METHODS OF PROTOCOL REVERSE ENGINEERING

A. *Manual work*

Originally, protocol reverse engineering was done manually, and it is so for most cases. By "manually", we do not mean that no tools are used, but that the work is done with high human involvement and is not done automatically. In addition to intuition and experience, protocol analyzer tools such as tcpdump [3] or Wireshark are usually necessary. Practices show that manual method is cumbersome and time-consuming.

B. *Network-based Methods*

Network-based methods are the first type of technique widely used in supporting some kind of automatic protocol reverse engineering. They work mainly on capturing and analyzing communication traffic. They are effective in identifying keywords, field delimiters, length fields and specific values. They rely only on captured messages, are the most widely used, and applicable for reverse engineering most of unknown protocols.

C. *Program-based Methods*

Program-based methods are also called binary analysis, which mainly work on protocol binaries. They can be further divided into static and dynamic methods, which both are widely used in ordinary software reverse engineering. But for protocol reverse engineering, there are specially designed techniques in these methods.

D. *Hybrid Methods*

For those methods that are both network-based and program-based, we call them hybrid. They are more complicated and more powerful, and will play more important role in future in our view.

And if other new techniques are added to the above third type of methods, we will still categorize them to the hybrid.

IV. TOOLS AND ACHIEVEMENTS

In this section, we will look back at the tools developed automatically carrying out protocol reverse engineering, and evaluate their achievements and drawbacks. The type of the method used(network-based, program-based, or hybrid) is put after the name of tools or projects in a sub-section title.

A. *PI (Network-based)*

In 2004, Marshall Beddoe, inspired by algorithms found in the bioinformatics field, launched and finished the Protocol Informatics Project (PI Project) [4] . The purpose of PI Project is, by analyzing a large amount of packets captured on networks, to obtain structure information of messages of the target protocol. The target protocol can be known and unknown.

In order to identify fields in protocol packets, samples are aligned using multiple string alignment algorithms and their common sequences are analyzed to understand the beginning and the end of fields in the packet. In PI Project, two bioinformatics algorithms, the Needleman Wunsch algorithm [5] and the Smith Waterman algorithm [6], are used in global and local alignment respectively.

A prototype of PI had been coded in python, freely available and can be downloaded online at http://www.4tphi.net/~awalters/PI/PI.html.

In paper [4] HTTP is taken as an example of a "black-box" protocol to be reverse engineered. some HTTP messages were reversely analyzed for illustrative purposes, and it was claimed that the same principle applies to any byte stream that contains structure.

B. *ScriptGen (Network-based)*

In 2005, to alleviate problems existing in Honeyd[7], Corrado Leita, Ken Mermoud and Mar Dacier proposed ScriptGen[8], a tool automatically generating new scripts for a honeypot server like honeyd. ScriptGen includes four functional modules: Message Sequence Factory, State Machine Builder, State Machine Simplifier, and Script Generator.

It extracts messages exchanged between a client and a server from the tcpdump file, by adding all observed message sequences one by one, to build a state machine which may have a lot of redundant and highly inefficient states. Then this "raw" state machine was analyzed, by introducing some sort of semantics using PI we mentioned above and the Region Analysis algorithm, to obtain a much simpler state machine where incoming messages are not recognized as simple sequences of bytes but instead as sequences of typed regions that must fulfill certain properties.

ScriptGen is designed to create honeyd-compatible emulators(scripts) for any protocol using a given sample of interaction without any a priori knowledge of its behavior.

But the scripts it can generate are simple and primitive compared with functions of a protocol. In addition, the tree-structured state-machine it generates, built only for decoding messages observed before, is far from the real state-machine of the protocol.

C. *RolePlayer and Discoverer(Both Network-based)*

In 2006, Weidong Cui, Vern Paxson, Nicholas C. Weaver, et al presented RolePlayer[9], a universal protocol emulator which, working on messages of an application session, can mimic both the client side and the server side of the session. Using byte-stream alignment algorithms to compare different instances of a session to determine which fields it must change to successfully replay, the system does not require any specification about the protocol it mimics.

RolePlayer works on two stages: preparation and replay. During preparation, RolePlayer first searches for endpoint-address, argument fields, length fields, and possible cookie fields in each sample message. During the second stage, it replays by sending message with updated value in fields which are identified by comparing received messages with the corresponding ones in the primary dialog.

As an automatic tool, RolePlayer had been stated to be successfully used to replay both the client and server sides for a variety of application-layer protocols, including NFS, FTP, and CIFS/SMB file transfers.

In 2007, Discoverer[10] was proposed. Different from RolePlayer, Discoverer was claimed as a tool for automatically reverse engineering the protocol message formats of an application from its network trace. The key achievement of Discoverer is that it can infer protocol idioms by dissecting the formless byte streams into text and binary segments for clustering messages with similar patterns. Discoverer was evaluated and calibrated over traces of a representative set of protocols consisting of one text protocol (HTTP) and two binary protocols (RPC and CIFS/SMB).

D. *Polyglot and AutoFormat(hybrid)*

In 2007，Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song introduced an automatic method of extracting protocol message formats, and implemented a tool, Polyglot[11]. Polyglot use a hybrid method, which works both on analysis of protocol traces and binary programs.

By monitoring the execution of the application binary that receives the protocol messages dynamically, Polyglot can distinguish and identify direction fields, separators, and keywords of the protocol. This technique, called tainted analysis, can also be used for exploitation exploring, worms detecting, XSS checking and fingerprint generation.

In 2008，Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang proposed AutoFormat[12], which made improvement on Polyglot by revealing the inherently “non-flat”, hierarchical structures of protocol messages. AutoFormat have been evaluated with more than 30 protocol messages from seven protocols, including two text-based protocols (HTTP and SIP), three binary-based protocols (DHCP, RIP, and OSPF), one hybrid protocol (CIFS/SMB), as well as one unknown protocol used by a real-world malware.

E. Prospex(hybrid)

In 2009，P. M. Comparetti and others designed and implemented Prospex[13]. The distinctive progress of Prospex is its capability of automatically reverse engineering state machine of stateful network protocols. Prospex had been applied to one malware protocol (Agobot C&C), two text-based protocols (SMTP, SIP), and one binary protocol (SMB). Furthermore, it is shown that how protocol specifications generated by Prospex are used to automatically generate input for a stateful fuzzer, allowing to discover security vulnerabilities in real-world applications. The inferred state machine for Agobot command and control protocol is illustrated as in Figure 3[13].

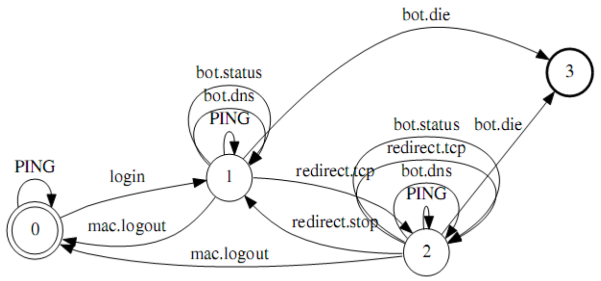


Figure3. Inferred state machine for Agobot

Compared with ScriptGen[8], Prospex reveals and produces much better protocol state machines by adopting Exbar algorithm[14] to exact minimal consistent DFA.

The researches and tools discussed above mainly focus on reverse engineering protocols that exchange plain-text messages, but in reality, there are many protocols used for security purposes of confidentiality, integrity, and non-repudiation. These security protocols usually put their messages in encrypted-text forms which impose extra difficulties to reverse engineering. We briefly mention below some progresses in this direction of research.

F. Reformat(hybrid)

In 2009, Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace published their research result ReFormat[15]. This is the first time the encrypted messages of protocols were automatically reverse engineered. ReFormat leverages generic techniques of “dynamic tainted analysis” , and relies on another general technique, i.e., data lifetime analysis, to locate the decrypted memory buffers.

ReFormat had been evaluated with a variety of protocol messages from real-world protocols such as HTTPS, IRC, MIME and the unknown protocol used by an malware agobot. In experiments, ReFormat accurately identified the decrypted message.

G. Rewards(program-based)

In 2010, Zhiqiang Lin et al [16] proposed REWARDS, a reverse engineering technique, to automatically reveal program data structures and infer their syntactic and semantic definitions from binaries. REWARDS is also based on dynamic analysis. The significant contribution of REWARDS is that by involving a backward type resolution procedure, REWARDS get the types of the variables in-memory recursively resolved. Then, based on the type information derived, REWARDS is able to reconstruct in-memory data structure layout. Another thing needed to mention is that REWARDS is totally program-based, and more targeted on memory image forensics and binary fuzzing for vulnerability discovery.

V. PROBLEMS AND PROSPECTS

Although a lot of researches and achievements have been made in protocol reverse engineering, there are still some problems unsolved. In this section, we look into these difficulties and look at the vistas in the future.

A. Problems

Firstly, most studies in this area are limited to extract field formats of plain-text messages, which are contained within a single application layer protocol data unit(PDU). But, as we have mentioned, there are protocols that put their messages in encrypted forms, or spit and put one message into several PDUs. In these scenarios, existing methods are no longer effective or insufficient for the goals we set up.

Secondly, for those program-based methods such as ReFormat which claim to have capability of reverse engineer encrypted messages, they are still very preliminary for security protocol reverse engineering. For example, the main limitation of ReFormat is that the method used to identify decrypted data is relatively simple. When a protocol involves a large number of arithmetic instructions in processing tasks other than decrypting messages in the program, or deliberately misleading a lot of mixed arithmetic instructions, it will lead ReFormat fail to locate and identify decrypted data.

B. Prospects

The research of protocol reverse engineering, in our observation, will make more progresses in two directions. One is combining network-based method more efficiently with program-based method. That is, for example, using network-based information as hints to make dynamic analysis of protocol binaries, and vice versa. Second is applying general methods to more specific tasks, for instance, improving program-based method for vulnerability discovering, or making network-based methods better to use in fuzz testing.

ACKNOWLEDGMENT

This work was partially supported by the Key Technologies R&D Program of Henan Province of China (No.092102210029, 092102310038), and by Zhengzhou Key Lab of Computer Network Security Assessment.

REFERENCES

[1] IETF. FILE TRANSFER PROTOCOL (FTP). http://www.ietf.org/rfc/rfc959.txt

[2] WIRESHARK. http://www.wireshark.org/.

[3] Tcpdump team. Tcpdump/Libpcap. www.tcpdump.org. Viewed on July 1st , 2011.

[4] M. Beddoe. The Protocol Informatics Project [EB/OL]. http://www.4tphi.net/~awalters/PI/PI.html. 2004-09-14/2011-01-19

[5] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:444-453, 1970.

[6] Smith, T.F and Waterman, M.S. Identification of common molecular subsequences. Journal of Molecular Biology, 147: 195-197, 1981.

[7] N. Provos. A Virtual Honeypot Framework[R]. Center for Information Technology Integration, University of Michigan. 2003.10. Also appeared in Proceedings of the 12th USENIX Security Symposium, pp:1-14, 2004.

[8] C. Leita, K. Mermoud, M. Dacier. ScriptGen: an automated script generation tool for honeyd[C]. Proc. Of the 21st Annual Computer Security Applications Conference. Tucson, AZ, USA, 2005: 202-214.

[9] W. Cui, V. Paxson, N. Weaver, and R.H. Katz. Protocol-Independent Adaptive Replay of Application Dialog[C]. Proceedings of the 13th Annual Symposium on Network and Distributed System Security (NDSS'06), San Diego, California. February 2006.

[10] W. Cui, J. Kannan and H. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces[C]. The 16th USENIX Security Symposium, 2007, 199-212.

[11] J. Caballero, Heng Yin, Zhenkai Liang, Dawn Song. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis[C]. Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, Virginia, USA, 2007:317-329.

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

[13] Comparetti P. M., Wondracek G., Kruegel C., Kirda E. Prospex: Protocol specification extraction[C]. Proceedings of the 2009 30th IEEE Symposium on Security and Privacy. Oakland, California, USA, 2009:110-125.

[14] K. J. Lang. Faster Algorithms for Finding Minimal Consistent DFAs[R]. NEC Research Institute,1999. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.7130&rep=rep1&type=pdf.

[15] Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, Mike Grace. ReFormat: Automatic Reverse Engineering of Encrypted Messages. Springer LNCS, 2009, Vol. 5789, 200-215.

[16] Zhiqiang Lin, Xiangyu Zhang, Dongyan Xu. REWARDS: Automatic Reverse Engineering of Data Structures from Binary[C]. In Proceedings of the 17th Network and Distributed System Security Symposium (NDSS'10), San Diego, CA, February 2010.