Survey on Network Protocol Reverse Engineering Approaches, Methods and Tools

Baraka D. Sija[1] , Young-Hoon Goo[1], Kyu-Seok-Shim[1], Sungyun Kim[2], Mi-Jung Choi[2] and Myung-Sup Kim[1]
Department of Computer and Information Science, Korea University, Sejong, Korea[1]
Department of Computer Science, Kangwon National University, Chuncheon, Korea[2]
{sijabarakajia25, gyh0808, kusuk007, tmskim} @korea.ac.kr[1], {kyun, mjchoi} @kangwon.ac.kr[2]

***Abstract*— A network protocol defines rules that control communications between two or more hosts on the Internet, whereas Protocol Reverse Engineering (PRE) defines the process of extracting the structure, attributes and data from a network protocol. Enough knowledge on protocol specifications is essential for security purposes, network policy implementation and management of network resources. Protocol Reverse Engineering is a complex process intended to uncover specifications of unknown protocols. The complexity of PRE, in terms of time consumption, tediousness and error-prone, has led to short and diverse outcomes of Protocols Reverse Engineering approaches. This paper, surveys outputs of 9 PRE approaches in three divisions with methodology analysis and its possible applications. Moreover, in the introductory part we provide a general PRE literature in great depth.**

Keywords— Network Protocols; Unknown Network Protocols; Protocol Reverse Engineering; PRE Outputs; Survey

I. INTRODUCTION

There is a vast number of network protocols and network applications that are active on the Internet, however only a part of it is well-known. One and major challenge of Protocol Reverse Engineering (PRE) is that it is mainly done manually and since manual PRE is extremely tedious and time consuming, sometimes it may take several years to uncover certain protocol specifications. Efficient PRE is prevented by numerous obstacles but the major obstacle is lack of protocol reverse engineering knowledge [1].

In the following paragraphs of this section, we discus several key terminologies which are commonly used in PRE and their relations.

Protocol Reverse Engineering is an entire process in which protocol parameters, formats and semantics are inferred in the absence of formal specifications [2]. PRE developed approaches, methods and tools use either *execution traces* or *network traces* as their inputs to analyze protocols formats or Protocol Finite State Machines (PFSMs). An *execution trace* is a sequence of instructions output that is executed during a single run of an application between multiple communicating hosts whereas a *network trace* is a ground truth traffic extracted from well-known tools such as Wireshark and Microsoft network monitor in cap or Pcap format.

When two or more machines communicate in the Internet all transmissions are grammatically and semantically controlled. From this point of view, protocol reverse engineering is conducted to infer such unknown syntaxes and semantics of a protocol. *Syntax inference (inferring protocol syntaxes)* is the process in which protocol grammars are inferred, where field boundaries, offset locations and endianness are discovered. It is the process in which the protocol rules that were used to format the messages involved in communication between two hosts are identified [3]. *Semantic inference (inferring protocol semantics)* is the process in which the data content that were exchanged between two communicating machines together with its meaning are inferred. In this process, protocols that are oriented in data structures such as Domain Name System (DNS) are inferred as *binary protocols* while text strings oriented protocols such as HTTP are classified as *text protocols*.

PRE results can be presented in two ways, either *Protocol Format or PFSM* for both binary and text protocols. The PFSM defines timing, orders or states in which fields in a message or messages in flow are exchanged between two hosts whereby a *protocol format* is a structural presentation of how fields are bounded in a message semantically with independent syntaxes. A *field* in a message composition of contiguous sets of associated bits(bytes) that contain semantic message data [2]. A protocol can have several fields from which some may be variable or fixed, as well as their lengths. A *keyword* is a word that differentiate one field from another semantically. In TCP/IP protocols source IP/port, destination IP/port, checksum, data offset, sequence number and acknowledgement are examples of fields keywords. In text protocols fields, key-values or values are separated by *delimiters* (*separators* [4]). A delimiter is a non-alphabetic symbol such as '#', ':', ';', ',', 'LF' or hexadecimal such as "0x0D0A, 0x00, 0x5C, etc.".

Although several evaluation metrics exist, *correctness, conciseness* and *coverage* are mainly involved in evaluation of PRE methods [2]. Correctness measures how accurate the reverse engineered protocol matches the true specifications of a protocol. Conciseness measures how many reverse engineered messages or states represent a single true message or that state

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00513, No. 2017-0-00158)

whereas coverage measures the quantity of reverse engineered protocol messages or states as compared the true protocol specifications.

Objectives, interests and applications of PRE may vary from one approach to another, such as involvement in IDSs, deep packet inspection (DPI), efficient fuzzing, identifying and analyzing botnets command and control message and integration and software compliance [2]. Of these motivations, PRE mostly relies on network and the Internet security purposes. Protocol formats and Protocol Finite State Machines(PFSMs) presentations are significant since they offer network administrators whole views of protocols for analysis and detection of abnormalities for security.

The rest of the paper is structured as follows: In Section II, we analyze approaches that Reverse Engineer the protocols formats. Section III, presents approaches that Reverse Engineers PFSMs. In Section IV, we present approaches that focus on Reverse Engineering neither protocol formats nor PFSMs directly. Conclusive analysis and future work are given in section V.

II. PRE FOR PROTOCOL FORMAT

PRE and protocol analyzers are such necessary for numerous applications in modern networks, however what outputs are resulted and what contribution do they bring, is more important. The following three sections, presents three divisions of PRE outputs as summarized in Table 1.

A protocol format is the general structure of how different fields of a protocol appear, when such a protocol is involved in real environment of network communications. Although some approaches, methods or Tools claim to reverse engineer a protocol format from a single message extracted from either side of the two communicating network machines, several messages from both sides need to be extracted and critically analyzed for complete uncovering of a protocol format.

TABLE 1. THREE DIVISIONS OF PROTOCOL REVERSE ENGINEERING OUTPUTS

Approach, Method Tool or Author	OUTPUTS		
	ProtoForm	*PFSM*	*OTHER*
Tupni [4]	○		
ReFormat [5]	○		
J. Cai et al [6]	○		
ReverX [7]		○	
PEXT [8]		○	
A. Trifilo et al [9]		○	
ASAP [10]			○–Semantics
ScriptGen [11]			○–Dialogs
PowerShell [12]			○–Scripts

A. Tupni

Tupni [5], is a tool that reverse engineers an input format with record sequences, record types and input constraints. Different format specifications can be generalized by the tool over multiple inputs. When aggregated over multiple protocol source files, it can derive more complete format specifications.

- Tupni Architecture

Tupni architecture has three major modules; ***F-I*** *(Field Identification* whose raw input is ***i*** (network message), ***I-RS*** *(Identification of Record Sequences)* whose input is ***S*** *(Sequence of Fields) and* ***I-RT*** *(Identification of Record Types)* whose input is ***S*** (Sequence of Records) as shown in Fig.1. An output ***O,*** is a Sequence of Record Types.

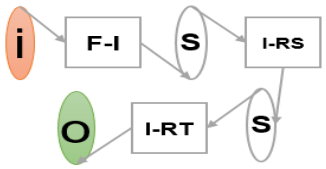

Fig.1. Tupni; Summarized Architecture Overview

The tool analyzes a single run of a parsing application on a target input. For a single run of the application, the sequence of instructions that is executed during this run is called the execution trace of the run. Each execution trace is associated with the list of binaries that were loaded during the run and the base addresses at which the binaries were loaded. Byte positions in the input are referred as offsets. The term position is used to identify instructions in the execution trace. For example, a '*mov'* instruction in the application binary may appear at multiple positions in the execution trace [5]. Sequences of contiguous positions in the execution trace are termed as subsequences. After several runs, Tupni finally aggregates the results of its analysis of the same application on different inputs to uncover the format.

B. ReFormat

ReFormat is a system that focus on deriving the message format for even encrypted messages [6]. An assumption is taken that after you decrypt a message a normal reverse engineering process takes place. Thus, ReFormat divides in two major steps, decryption of messages and reverse engineering.

- ReFormat Architecture

ReFormat [6] takes four key phases which are Execution Monitor, Phase Profiler, Data Lifetime Analyzer and Format Analyzer as shown in Fig.2. In Execution Monitor phase, the application execution is monitored and an execution trace is collected with records of how an application decrypts a message. In phase profiler, the execution trace is analyzed to identify both message decryption and normal protocol processing. Next, data lifetime analysis is done to locate buffers containing the decrypted messages. The last phase conducts dynamic data flow analysis on the buffers located in the previous step to uncover the format of the decrypted messages.

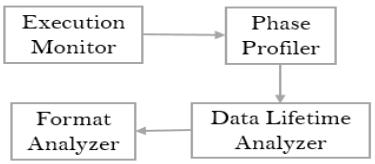

Fig. 2. ReFormat System Architecture

C. J. Cai et al

J. Cai et al. determine the optimal length of protocol keywords and recover message formats of Internet protocols by maximizing the likelihood probability of message segmentation and keyword selection [7]. Through a hidden semi-Markov model, J. Cai et al. attempt to model the whole protocol message format. Based on Hidden Markov Model functions, an affinity propagation mechanism based on clustering technique is introduced to determine type of messages. In the beginning the raw data set is collected using tshark belonging to one protocol. Next, *session are messages are formed, HsMM is modeled* with *message segmentation* and *type inference.*

The contribution of this approach is the *HsMM modeling* where message formats are generated from an algorithm based on the Baum-Welch method, performed to re-estimate the parameters of the HsMM protocol model. In the *message segmentation* phase, the re-estimated HsMM model is applied to determine the optimal length of protocol keywords and divide message into field sequence. The final stage is *message type inference*, where protocol messages are clustered by affinity propagation mechanism and each cluster represents a message type.

III. PRE FOR PROTOCOLS FINITE STATE MACHINES(PFSM)

Protocol Finite State Machine *(PFSM)* is one and important presentation of a protocol transitions in PRE. It simply defines orders, states and transitions of fields in a message or messages in a flow between two or more communicating machines.

A. ReverX

ReverX is a methodology that automatically infers a specification of a protocol from real environment network traces, that will generate automata for the protocol language and state machine. Since ReverX focuses on only protocols sample interactions, it is a well-suited approach for uncovering the message formats and protocol states of closed protocols and automating most of the processes that specifies open protocols.

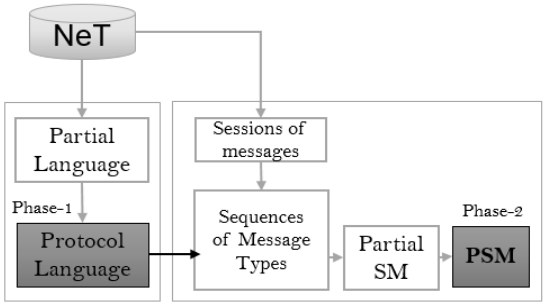

Fig. 3. ReverX Architecture Overview

ReverX is divided in two phases, Generalized *Protocol Language* and reduced *Protocol State Machine* as indicated in Fig. 3. In the first phase, the construction of a Prefix Tree Acceptor (PTA) is done from the protocol messages of the network traces (Partial Language), which is then generalized with the intention of producing an FSM that could accept the same message types in different payloads (Protocol Language) [8]. In the second phase, it deduces the protocol state machine from the causal relations among different messages present in the network traces and resorts to a more accurate and highly reduced PFSM.

B. PEXT

In PEXT (Protocol EXTraction) [9], networked applications protocols are reverse engineered by raw packets of an application captured at runtime. Packets are first captured from distinct execution traces and grouped to distinct classes. Identical packets are grouped into their individual flows and then identical flows are extracted. These flows which are restricted to contain at least two packets, form initial states. Since each state is restricted to contain packets of the same flow, distinct states are classified. Identical flows are identified and labeled as states with specific IDs. To this point, longest common substring algorithm (LCS) applies to all leftover flows to derive all the other states. Finally, each packet that does not belonging to any state yet becomes a single state to form a general minimized state machine.

C. A. Trifilo et al

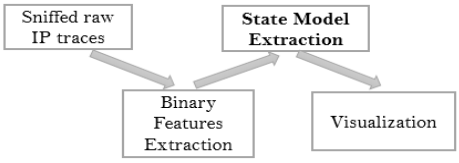

Fig. 4. Workflow Overview of A. Trifilo et al

In [10], A. Trifilo et al propose an approach that derives the protocol state machines from network traces. As shown in Fig. 4, raw traffic data are sniffed and filtered to select the protocol to be reverse engineered and then stored in a standard Tcpdump file. The extraction of binary features is done on only target fields of a protocol message. For example, in a HTTP request "GET /d3/ko/ HTTP/1.1" only the field "GET" is necessary to understand the basic logic of the protocol as it defines the type of action requested. In this approach, a statistical analysis of several flows based on the "Variance of the Distribution of Variances" (VDV) is used to achieve the reduction to a subset of interested features in each byte of the binary protocol message. Through a State Splitting Algorithm, the PFSM is constructed from the selected features [10].

IV. PRE FOR OTHER OUTPUTS

Apart from Protocol Formats and Protocol Finite State Machine (PFSM), which are two and major focus outputs of PRE, this section summarizes three works that do not directly focus on Reverse Engineering either Protocol Format or PFSMs.

A. ASAP

ASAP (**A**utomatic **S**emantics-aware **A**nalysis of network **P**ayloads) is a framework for protocol semantics inference and analysis from network traffic [11]. The method maps network payloads to a vector space and identifies communication

templates corresponding to base directions in the vector space. ASAP is applicable in different security applications such as automatic discovery of patterns in honeypot data, analysis of malware communication and network intrusion detection.

B. Script-Gen

Script-Gen, is a tool for automatically generating scripts from two communicating machines.

Fig. 5 shows four modules of the Script-Gen framework, where TCP based protocol messages that are exchanged between a client and a server are first extracted by Tcpdump. To this point TCP streams that correctly handle retransmissions and reordering are reconstructed. Next, the extracted messages are used as building blocks to build a state machine where the built states lead to a generation of redundant and highly inefficient state machine. To control such redundant, thresholds that limit the number of outgoing edges for each state are applied.

The State Machine Simplifier is the core module in which raw state machine and semantics are analyzed and achieved through two algorithms, the PI and the Region Analysis, newly introduced by Script-Gen [12]. From the two algorithms, simpler state machine for scripts generation is obtained.

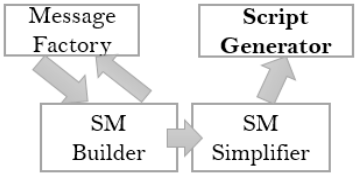

Fig. 5. ScriptGen Summarized Structure

C. PowerShell

Microsoft Windows PowerShell[13] is a tool that has led to several exploit frameworks such as *PowerSploit*, *PowerView* and *PowerShell Empire*. However, only some of these frameworks investigate network traffic for exploitative potential. The tool includes several network analysis and network traffic related capabilities to explore capturing, analysis and identification of protocols without installation of any other supporting tool in Microsoft Windows environment. *PowerShell* may investigate protocols that indicate potential vulnerabilities within a network environment, for both attackers and defenders.

The scripts generated by PowerShell can currently fully support IPv4 traffic and protocols where by the environment for IPv6 are also being developed [13]. In general, a script produced by PowerShell, provides an easy method to identify many of the protocols in different vulnerable conditions and is useful to both network defenders and penetration testers for identification of network protocol based vulnerabilities.

V. Conclusion and Future work

Protocol Reverse Engineering (PRE) is an increasingly important field in modern network environment and security. In a summarized way, this paper discusses PRE outputs of 9 approaches, methods and tools, dividing them in three divisions, PRE for Protocol Format, PRE for Protocol Finite State Machine (PFSM) and Other outputs which can either be semantics-contents or scripts. One of the major motivations of this paper is how risk are the Internet users' privacy, when their packets will be involved and controlled by an unknown or undocumented protocol, without knowing?

Many and different research on PRE have been done and promising outputs have been obtained, however due to frequent environment changes that lead to current protocols updates or inventions of new protocols, PRE as well need to be conducted in modern and highly automated fashion to match the evolving of protocols and network environment changes.

References

[1] Dennis Yurichev, 2013~2016, Reverse Engineering for Beginners

[2] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. 2015. A survey of automatic protocol reverse engineering tools. ACM Comput. Surv. 48, 3, Article 40 (December 2015), 26 pages. DOI: http://dx.doi.org/10.1145/2840724.

[3] XiangDong Li and Li Chen, "A Survey on Methods of Automatic Protocol Reverse Engineering", 2011 Seventh International Conference on Computational Intelligence and Security, 685 - 689, DOI: 10.1109/CIS.2011.156 .

[4] Mingming Xiao, Shunzheng Yu, Yu Wang. "Automatic network protocol automaton extraction". The 3 International Conference on Network and System Security (NSS 09), 2009: 336-343.

[5] Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz. 2008. Tupni: Automatic reverse engineering of input formats. In 15th ACM Conference on Computer and Communications Security (CCS'08). ACM, New York, NY, 391–402. [DOI:10.1145/1455770.1455820].

[6] Wang Z., Jiang X., Cui W., Wang X., Grace M. (2009) ReFormat: Automatic Reverse Engineering of Encrypted Messages. In: Backes M., Ning P. (eds) Computer Security – ESORICS 2009. ESORICS 2009. Lecture Notes in Computer Science, vol 5789. Springer, Berlin, Heidelberg

[7] Jun Cai, Jian-Zhen Luo, and Fangyuan Lei, "Analyzing Network Protocols of Application Layer Using Hidden Semi-Markov Model", Mathematical Problems in Engineering, Volume 2016 (2016), Article ID 9161723, 14 pages http://dx.doi.org/10.1155/2016/9161723

[8] Jo˜ao Antunes, Nuno Neves, and Paulo Verissimo. 2011. Reverse engineering of protocols from network traces. In 2011 18th Working Conference on Reverse Engineering (WCRE), 169,178. DOI:10.1109/WCRE.2011.28 (ReverX)

[9] Maxim Shevertalov and Spiros Mancoridis. 2007. A reverse engineering tool for extracting protocols of networked applications. In 14th Working Conference on Reverse Engineering (WCRE'07). 229–238. DOI:10.1109/WCRE.2007.6

[10] Antonio Trifilo, Stefan Burschka, and Ernst Biersack. 2009. Traffic to protocol reverse engineering. In 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 1–8. DOI:10.1109/CISDA.2009.5356565

[11] T. Krueger, N. Kramer, and K. Rieck, "ASAP: automatic semantics-aware analysis of network payloads", In Proceedings of ECML/PKDD, 2011.

[12] Corrado Leita, Ken Mermoud, and Marc Dacier. 2005, "ScriptGen: An automated script generation tool for HoneyD" In 21st Annual Computer Security Applications Conference (ACSAC'05), 200–214. DOI:10.1109/CSAC.2005.49

[13] David R Fletcher Jr, "Identifying Vulnerable Network Protocols with PowerShell", 2017, SANS Institute Reading Room site