Hindawi
Security and Communication Networks
Volume 2018, Article ID 8370341, 17 pages
https://doi.org/10.1155/2018/8370341

Review Article

A Survey of Automatic Protocol Reverse Engineering Approaches, Methods, and Tools on the Inputs and Outputs View

Baraka D. Sija, Young-Hoon Goo, Kyu-Seok Shim, Huru Hasanova, and Myung-Sup Kim

Department of Computer and Information Science, Korea University, Seoul, Republic of Korea

Correspondence should be addressed to Myung-Sup Kim; tmskim@korea.ac.kr

Received 5 August 2017; Revised 16 December 2017; Accepted 2 January 2018; Published 20 February 2018

Academic Editor: Zhe Liu

A network protocol defines rules that control communications between two or more machines on the Internet, whereas Automatic Protocol Reverse Engineering (APRE) defines the way of extracting the structure of a network protocol without accessing its specifications. Enough knowledge on undocumented protocols is essential for security purposes, network policy implementation, and management of network resources. This paper reviews and analyzes a total of 39 approaches, methods, and tools towards Protocol Reverse Engineering (PRE) and classifies them into four divisions, approaches that reverse engineer protocol finite state machines, protocol formats, and both protocol finite state machines and protocol formats to approaches that focus directly on neither reverse engineering protocol formats nor protocol finite state machines. The efficiency of all approaches' outputs based on their selected inputs is analyzed in general along with appropriate reverse engineering inputs format. Additionally, we present discussion and extended classification in terms of automated to manual approaches, known and novel categories of reverse engineered protocols, and a literature of reverse engineered protocols in relation to the seven layers' OSI (Open Systems Interconnection) model.

1. Introduction

High demand of reverse engineering protocols in the Internet comes as an outcome of rapid increase of Internet traffic volume. Network scholars, researchers, and organizations are currently focusing mainly on reverse engineering unknown or undocumented protocols specifications for security and networks maintenance purposes. Indeed, there are a vast number of network protocols and network applications believed to be active on the Internet, whereby only a part of it is well known or documented. One and major challenge of Protocol Reverse Engineering (PRE) is that it is mainly done manually. Since manual PRE is extremely tedious and time consuming, it may sometimes take several years for certain protocol specifications to be completely uncovered. Efficient PRE is prevented by numerous obstacles such as binary human unreadable protocols, high entropy data, dynamic protocol fields, and context based fields in which earlier fields affect the meaning or the form of following fields, encryption, and so on.

In the following paragraphs of this section, we define several key terminologies which are commonly used during Protocol Reverse Engineering. The PRE terminologies consecutively defined in detail are Protocol Reverse Engineering itself, execution trace and network trace, syntax inference (inferring protocol syntaxes) and semantic inference (inferring protocol semantics), binary protocol and text protocol, protocol finite state machine (PFSM) and protocol format, field, keyword, keyword value, delimiter (separator), key-value delimiter and value-value delimiter, correctness, conciseness, and coverage.

Protocol Reverse Engineering is the process in which protocol parameters, formats, and semantics are inferred in the absence of formal specifications [1]. Since protocol specifications of unknown protocols are mainly kept secret by developers or owners the only way to uncover the

specifications is through reverse engineering. Many reasons may apply to why the inventors are likely to hide the specifications of the protocol they develop, but all pertain to pursuing individual or organization profits. Some protocols in the Internet are developed solely for surveillance. Such kind of protocols is hard to uncover their specifications. Moreover, protocols are constantly evolving due to features and functionalities changes that lead to even harder and more complicated Protocol Reverse Engineering, as previously known protocol specifications turn incomplete and not applicable. Although Automatic Protocol Reverse Engineering methods are being proposed and developed, the process has been mostly manually done, which is error-prone and time consuming. For instance, it took the SAMBA project 12 years to generate a protocol specification for the Microsoft Server Message Block (SMB) protocol [1]. Therefore, maximization of accuracy, automation, and shortening of time for Protocol Reverse Engineering are the priorities for the modern security and Internet communication environment.

Execution trace and network trace are two main inputs of Protocol Reverse Engineering methods. An *execution trace* is a program code executed in a single run of an application between two or more communicating hosts, [1] while a *network trace* is a ground truth traffic captured from a network by tools such as Tcpdump, Wireshark, and Microsoft network monitor. Network traces appear in raw packets and are stored as *cap* or *pcap* files. Execution traces are efficient in identifying relevant fields of a protocol from the multiple and iterated instruction loops. Compared to static analysis execution traces are most likely applicable and effective for dynamic analysis.

Syntax inference and semantic inference are simply necessary stages for a successive Protocol Reverse Engineering process. *Syntax inference (inferring protocol syntaxes)* is the process in which protocol field boundaries, offset locations, and endianness are inferred. It is the process in which the protocol rules that were used to format the messages involved in communication between two hosts are identified [2]. *Semantic inference (inferring protocol semantics)* is obviously the following step after syntax inference. It is the process in which the data content that was exchanged between two communicating machines together with its meaning is inferred. For instance, in Hypertext Transfer Protocol (HTTP), the inferred semantic information can be a web page content, whereas in the case of Distributed Network Protocol Version 3 (DNP3) [1], the inferred semantic information is commands and control data. For known protocols, their syntaxes and semantics can easily be found and studied; however for the unknown and undocumented protocols their syntaxes and semantics need to be reverse engineered.

Binary protocol and *text* protocol are two major categories to which all protocols fall; however at special cases, certain protocols can be text-binary. A binary protocol is a protocol that is oriented in data structures such as Domain Name System (DNS). It is a protocol that is machines readable rather than human. A binary protocol is simply a protocol that is data structures oriented while a text protocol is a protocol that is oriented around text strings which are human readable. HTTP is an example of text protocol that is designed to be communicated as a flat stream of lines of text. For instance, in a HTTP response type instead of three characters "2," "0," and "0" the response type appears as integer value of "200."

Protocol finite state machine (PFSM) and protocol format (PF) are two possible target outputs after reverse engineering a protocol. The PFSM defines timing, orders, and states in which messages change between two hosts. It is the transition of states and order from one form to another syntactically and semantically according to a protocol. In this way, a protocol can be reverse engineered and fairly presented in finite state machine. A protocol format is the presentation of how a field boundary is structured, whereby each field has its own semantic keyword. A protocol format presents fields in priorities of positioning from left to right or right to left.

Field, keyword, keyword value, delimiter (separator), key-value delimiter, and value-value delimiter are terms used to express mostly text protocol formats. A *field* is a composition of contiguous sets of associated bits (bytes) that contain semantic message data [1]. A *field* in a protocol message can be identified from either constant or frequently appearing features such as bytes, data offset position, and weight. A protocol can have several fields from which some may be variable or fixed, as well as their lengths which may be variable or fixed. A *keyword* is a semantic word that differentiates one field from another semantically. Source IP/port, destination IP/port, checksum, sequence number, and message type are example of fields, whereas a *keyword value* is a unit value of that field. A *delimiter (separator* [3]*)* is a nonalphabetic symbol such as "#", ":", ";", ",", and "$" and hexadecimal delimiters such as "0x0D0A, 0x00, and 0x5C" are used to separate fields, key-values, and value-value. Therefore, there are three categories of delimiters for text protocols which are *field delimiters, key-value delimiters, and value-value delimiters*. For instance, in a message "TIME-OUT:60 # ALLOWED PORTS:4534,80,53", "#" is a field delimiter, ":" is key-value delimiter, and "," is value-value delimiter.

Correctness, conciseness, and coverage are three independent metrics that measure how successfully a method can reverse engineer a protocol [1]. *Correctness* measures how accurate the reverse engineered protocol matches the true specifications of a protocol. For instance, if a protocol has 10 fields and every time the PRE-method is tested about 7 fields are discovered, then the method correctness (accuracy) is termed as 70%. *Conciseness* measures how many reverse engineered messages or states represent a single true message or that state whereas *coverage* measures the quantity of reverse engineered protocol messages or states as compared to the true protocol specifications.

The remainder of the paper is structured as follows: In Section 2, we describe the background and motivation of the paper. Section 3 which is the main area of interest of this paper chronologically analyzes all the APRE approaches, methods, and tools' outputs based on their inputs, evaluating more appropriate and efficient inputs. Section 4 presents classified discussion in terms of manual to automated approaches, reverse engineered or analyzed protocols category, and a detailed literature of reverse engineered or analyzed protocols

based on the 7-layers OSI model. Section 5 presents conclusive remarks and potential future research directions of APRE based on Sections 3 and 4.

2. Background and Motivation

Objectives and interests of Protocol Reverse Engineering may vary from one approach to another. Major applications of APRE are Intrusion Detection Systems (IDSs), Deep Packet Inspection (DPI), efficient fuzzing, identifying and analyzing botnets command and control message, and integration and software compliance [1].

Protocol analyzers are highly utilized by modern IDSs to detect and protect against malicious application-layer communication on networks. In a diverse and high-volume Internet, APRE tools are essential for analyzing mainly application-layer network traffic that may contain unknown communications specifications. Unknown specifications of network traffic need to be analyzed and uncovered as they may be malicious and/or surveillance to the network. Protocol analyzers enhance identification and classification of application-layer network data through Deep Packets Inspection (DPI). Indeed, DPI is pre-IDSs as it has been and can be traditionally deployed in IDS architectures for identifying malicious payloads through pattern matching. Fuzzing is a testing technique that tests programs by receiving or sending data (inputs) from unsafe sources. Communications protocols can be effectively fuzzed for security vulnerability detection by APRE and protocols analyzers. Fuzzing plays an intercepted role with IDSs. Botnets are primary security concern in the Internet since they infect systems while users are unaware of them. They consist of hosts in networks with malicious applications that are monitored by a remote attacker. APRE and protocols analyzers play a key role in identifying and analyzing botnets commands and control messages sent by an attacker. When new hosts are connected to a network, they sometimes fail to comply with existing protocol standards. In this case, APRE and protocol analyzers can infer such differences and make software integration [1]. Conclusively, PRE is highly applicable for network security and for Internet users' privacy security.

APRE and protocol analyzers are such necessary for numerous applications in modern networks; however what method or approach leads to a successive and perfect APRE or protocol analyzer is a question at hand motivating this paper. Taking this question as the main area of interest we evaluate the outputs of approaches, methods, and tools in relation to their inputs along with methodologies applied.

3. Automatic Protocol Reverse Engineering Inputs and Outputs

High correctness (accuracy), conciseness, and coverage of APRE outputs are the major target and expectation of any approach, method, or tool. However, due to constant evolving of unknown network protocols, the evaluations and estimations of the three metrics are often not satisfactory. The major cause of unsuccessful APRE is mostly inappropriate inputs to the proposed approaches or developed methods and tools. Either network traces or execution traces should closely consider both sides (*the client* (the machine side that sends requests and receives responses (commands) from the server) and *the server* (the machine side that sends responses (commands) and receives requests from the client)) information in different levels from a single packets to session level. This analysis is essential for an efficient Protocol Reverse Engineering.

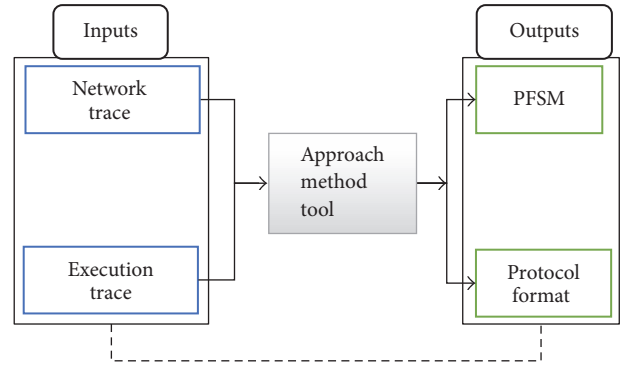

Figure 1: Major classification of APRE inputs and outputs.

Protocol Reverse Engineering outputs can be mainly categorized into two categories: protocol formats (PFs) and protocol finite state machines (PFSMs), as depicted in Figure 1. Approaches that focus on outputting PFSMs of unknown protocols are highly likely to infer a protocol formats as well, since PFSMs rely on syntax inference, a necessary protocol format stage. Therefore, approaches that focus on uncovering PFSMs have high chance of indirectly uncovering protocol formats too. Meanwhile, approaches that focus on outputting the protocol formats are most likely to uncover FPSMs as well, since they are bound to infer protocol syntaxes in discovering fields and formats, which is a preprocess for protocol semantic inference, a key and necessary PFSM step. Either PFSM or protocol format can be obtained from either network traces or execution traces; however, the accuracy may vary from either input.

Bossert et al. [26] points that, due to inappropriate inputs traces, many studies do not provide accurate results especially on unknown and complex protocols. And as many approaches or methods being too theoretical, only few of them have resulted in the publication of tools that would allow scientific community to experimentally validate and compare difference among approaches. One major reason for this challenge is the fact that network activities in laboratories never perfectly reflect that observed in the wild [26, 27]. In the following two subsections the outputs and inputs from different approaches, methods, and tools are analyzed along with the applied algorithms towards Automatic Protocol Reverse Engineering.

3.1. Distribution of Automatic Protocol Reverse Engineering Outputs. This section analyzes 39 different approaches, methods, and tools outputs' focus in relation to the algorithms and/or technique along with architectures applied.

Table 1: Approaches, methods, and tools that focus on reverse engineering PFs; *NetT* = network traces; *ExeT* = execution traces.

Approach, method, tool, or author	Year	Input format		Special remarks
		NetT	ExeT	
Discoverer [4]	2007	○		*PFSM for future work*
Polyglot [5]	2007		○	*Dispatcher [6] 1st work*
AutoFormat [7]	2008		○	~
Tupni [8]	2008		○	*PFSM for future work*
ReFormat [9]	2009		○	*Decryption before PRE*
Prospex [10]	2009	○		*Reverse engineers PFSM as well*
ProDecoder [11]	2012	○		~
Wang et al. [12]	2013	○		*PRE in wireless environment*
ProGraph [13]	2015	○		*Traffic classification*
Cai et al. [14]	2016	○		~
WASp [15]	2016	○		*IEEE.802.15.4 wireless protocols*

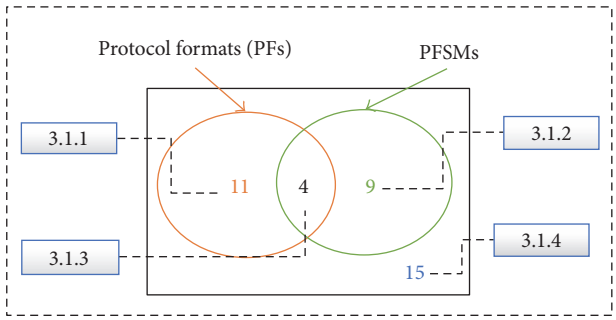

Figure 2: Four distributions of 39 APRE approaches' focus (Sections 3.1.1, 3.1.2, 3.1.3, and 3.1.4).

Among 39 approaches, 11, 9, 4, and 15 approaches focus on outputting only protocols formats, only protocols finite state machine, both protocols formats and protocols finite state machine, and those that do not directly focus on outputting protocols formats or protocols finite state machines, respectively, as depicted in Venn diagram (Figure 2).

3.1.1. Protocols Formats Focus. Reverse engineering unknown or undocumented protocols formats are technique that requires a lot of information and knowledge to completely and successively reverse engineer an unknown protocol. Yurichev [43] provides basic and important fundamentals such as integral datatypes, signed number representations, AND, AND, and OR as subtraction and addition, XOR (exclusive OR), Population Count, Endianness, Memory, CPU, and Hash functions towards reverse engineering. Many approaches do not consider in great depth the fundamentals such as those introduces by Yurichev [43], leading to poor PRE outputs and analysis. With chronological order as shown in Table 1, this section analyzes 11 approaches that focus on reverse engineering formats or specifications of unknown and undocumented protocols.

Discoverer [4] uses network traces to automatically reverse engineer protocol message formats. The tool is capable of inferring protocol idioms that can often be seen in message formats of different application protocols. Three modules are involved in Discoverer, *Tokenization and Initial Clustering, Recursive Clustering, and Merging* consecutively functioning.

In the *Tokenization and Initial Clustering* module, Discoverer clusters messages based on their token patterns assigned in 4-tuple information *(dir, class of token 1, class of token 2,. . ., class of token n)* where *dir* is the direction of the message C2S or S2C followed by the classes of all tokens in a given message. Message direction is considered because messages in opposite directions may have different formats. The 4-tuple example of token patterns can appear in the form like *client to server, text, binary,* and *text*. Discoverer applies its own developed formats aligning algorithm based on Needleman and Wunsch [44]. In general raw packets are utilized. From the raw packets, message fields' boundaries are identified with first-order structure giving to the unlabeled messages. The packets are reassembled in messages and then broken into a sequence of tokens fields. The tokens are used to define two major classes of messages: binary or text. Recursive Clustering is the second module where binary or text messages are further divided so that messages in each cluster contain the same formats. Lastly, messages of the same format are mitigated and classified into their respective clusters.

Polyglot [5] reverse engineers a protocol message format by program binaries through *shadowing* algorithm. The approach uses dynamic binary analysis with facts on how a protocol is implemented. Polyglot shadowing approach has two phases as depicted in Figure 3 {[5], Caballero et al. (2007)}. It extracts the protocol message format by processing messages at different time and outputs the formats for each received message. The execution monitor [45] is the key runner of polyglot architecture in the first phase. In this phase, all records of the running program are generated, while containing necessary information of a particular execution. Field boundaries and the keywords that forms message formats are analyzed in the second phase.

Through a protocol field tree generation algorithm, AutoFormat [7] reverse engineers structures of protocol messages based on individual extracted protocol fields. In *AutoFormat*, each field in a given message format is handled in its own

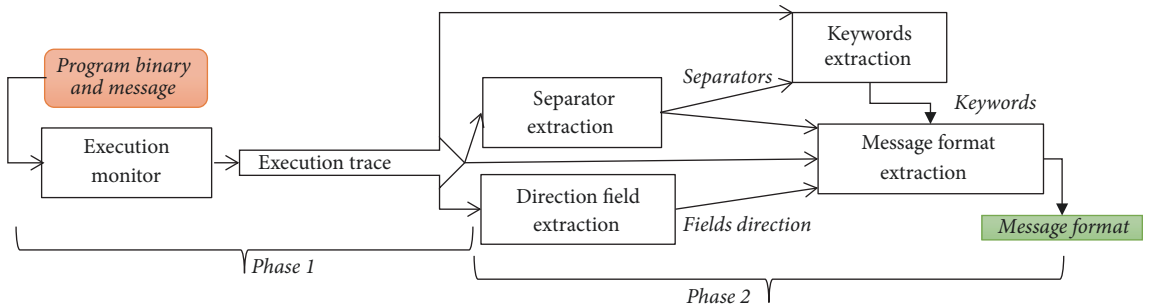

Figure 3: *Polyglot*, shadowing system architecture {Figure reproduced from Caballero et al. (2007), [under the Creative Commons Attribution License/public domain]}.

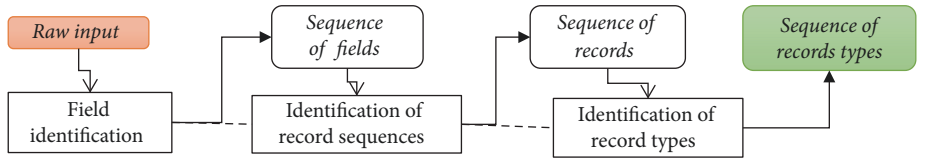

Figure 4: *Tupni* architecture {Figure reproduced from Cui et al. (2008), [under the Creative Commons Attribution License/public domain]}.

execution contexts at runtime. The execution contexts are collected from each message bytes that is annotated with its offsets to create whole protocol message format. One and major advantage of *AutoFormat* is that it can implement a binary of an unknown protocol to simply recognize its format perfectly. This is an accurate way to handle different messages and successfully uncover their formats. To do this the approach architecture divides in two components, the part dealing with executing contexts and the part that identifies each protocol field.

Tupni [8] is a tool capable of discovering protocol formats with several pieces of information, such as record sequences, types, and type of input constraints. From this information further generalization of the format specifications over multiple *inputs files* or *network messages* is done. Based on its own designed algorithm as shown in Figure 4 {[8], Cui et al. (2008)}, Tupni first identifies general fields, next records sequences from which records types are recorded. In the *fields' identification* stage, short sequences of partitioned bytes are utilized as inputs which carry necessary and correspondent information to basic fields of the input format. Accurate discovering of fields in Tupni is possible from observation of the mapped instruction loops in the CPU. In identification of record sequences, the tool applies its own developed algorithm to process unbounded sequences and identifying all the fields.

ReFormat [9] attempts to reverse engineer protocol specifications of encrypted application messages. The tool first decrypts all messages involved in a transaction and then reverse engineers a target protocol. In decryption, algorithms such as Triple-DES, AES, and RC4 are involved in all bytes of the original messages. *Prospex* [10] is a system that focuses on automatically extracting application-layer protocol specifications. From executed traces Prospex correctly infers message formats by the PAM (Partitioning Around Medoids) clustering algorithm. *ProDecoder* [11] is a tool that uses network traces to infer semantics of protocol messages and formats discovering. In ProDecoder an *n-grams* concept is applied where a relationship is checked between prefixes and suffixes among keywords. If a previous word suffix resembles a next word prefix, the two are joined to obtain and reveal possible protocol fields' semantics. For message clustering the tool applies IB (Information Bottleneck) algorithm to group similar messages. *ProGraph* [13] is a tool that focuses on reverse engineering a protocol from both its bits-level and bytes-level information from its own designed graphical model algorithm. An intrapackets exploitation is conducted to find dependencies among packets, especially in packet payloads where many fields can be identified. Such kind of information is assumed to be rich in the first few bytes of the packet payloads [46]. Moreover, ProGraph can determine the task carried out by an individual packet within the protocol traffic.

Cai et al. [14] aim at selecting fields keywords with their optimal lengths and finding about how protocol messages are segmented. Through a hidden semi-Markov model Cai et al. try to model an entire protocol message format. They collect training raw data set using tshark [47], in preapplication of the derived model, Hidden Semi-Markov Modelling (HsMM). After the raw data capturing process five modules, *session reconstruction, message reassembling, HsMM, message segmentation,* and *message type inference,* follow. In raw data traffic collection phase, the approach takes an assumption that all traffic captured belongs to the same protocol.

After raw data traffic has been collected, *session reconstruction,* the second phase follows where sessions are reconstructed according to the traffic 5-tuple information, which are *Source IP, Destination IP, Source port, Destination port, and L4 Transport protocol (TCP/UDP).* For TCP-based protocol traffic, a session starts at the packet with the SYN flag in TCP

Table 2: Approaches, methods, and tools that focus on reverse engineering PFSMs.

Approach, method, tool, or author	Year	Input format		Special remarks
		NetT	ExeT	
PEXT [16]	2007		○	*Limitation in extracting semantic information*
Xiao et al. [3]	2009	○	○	~
Trifilo et al. [17]	2009	○		~
Antunes and Neves [18]	2009	○		~
ReverX [19]	2011	○		~
Veritas [20]	2011	○		~
Zhang et al. [21]	2012	○		~
Laroche et al. [22]	2013	○		~
Meng et al. [23]	2014	○		~

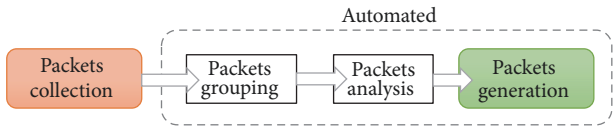

Figure 5: *WASp*: summarized architecture overview {Figure reproduced from Choi et al. (2016), [under the Creative Commons Attribution License/public domain]}.

header and ends when the FIN flag is acknowledged. For UDP protocol traffic, a session is defined as the packets shared by the same 5-tuple. In *message reassembling* phase, messages of TCP-based protocols are reassembled from packets according to their TCP sequence number and acknowledgement number while the messages of UDP-based protocols are reassembled based on their arrival time stamp of packets and the transmission direction of packets.

The key contribution of this approach is the *HsMM* phase. In this phase, the message formats are generated from an algorithm based on the Baum-Welch method, which is performed to reestimate the parameters of the HsMM-based protocol model. In the *message segmentation* phase, the reestimated HsMM model is applied to obtain optimal lengths for protocol keywords and to divide a single message into a sequence fields. The final stage is *message type inference*, where protocol messages are clustered using the affinity propagation mechanism and each cluster will represent a message type as the final output.

Wang et al. [12] present an approach that utilizes association rules among sequential features and identify unknown protocol formats in wireless environment. The approach uses captured binary data from an implemented protocol. The goal of this method is to uncover all vulnerabilities in Wi-Fi networks, where attackers may secretly exchange data and spread malicious codes. To extract 4-bit frequent sequences, determine fields, and discover possible associations among them, an *AC algorithm*, about mining association rules [48], is applied, where unknown possible protocol formats are built. WPAN automatic spoofer (WASp) [15] is a tool based on IEEE 802.15.4 wireless protocols, capable of understanding and reconstructing customized protocols to byte-level and generating packets that can be used for analysis and spoofing attacks. Since wireless protocols, especially IEEE 802.15.4 based, are not well studied for PRE, WASp provides key insights towards reverse engineering such protocols.

Figure 5 {[15], Choi et al. (2016)} summarizes four *WASp* phases, which are *packet collection, packet grouping, protocol analysis, and packet generation*, whereby packets capturing is conducted manually with the rest modules being fully automated.

Section 3.1.1 describes 11 approaches that focus on reverse engineering undocumented and/or unknown protocols and outputting their general formats. A diversity of techniques and algorithms towards reaching this goal is observed from received network traces *(packets, flows,* and *sessions)* and execution traces. However, approaches that utilize network traces as inputs are 7 out of 11.

3.1.2. Protocols Finite State Machines (PFSM) Focus. *PFSM* is an important presentation of a protocol transitions in PRE. It simply defines orders, states, and transitions of fields in messages and messages between two or more communicating machines. This section presents 9 approaches, methods, and tools that focus on Reverse Engineering Protocols Finite State Machines from received network traces or execution traces as shown in Table 2.

PEXT (Protocol EXTraction) [16] is a tool that analyzes captured packets and reverse engineers PFSMs of application-layer based protocols from its own designed algorithm. In PEXT, through LibPcap tools packets are first captured from different binary traces of specific features; second, all packets are grouped into distinct classes. Third, execution flows graphs are produced for all the traces using the Longest Common String (LCS) algorithm to find message states, and finally, all the separate diagrams are combined to form a PFSM. In this process, identical packets are grouped into their individual flows and then identical flows are extracted. These flows are restricted to contain at least two packets and then form initial states. Since each state is restricted to constitute same flow packets, distinct states are simply obtained. Identical flows are identified and labeled as states with specific IDs. From this step, the (LCS) applies to all remaining flows to reduce and minimize the states. To this point, a packet that does not belong to any state yet becomes a single packet state.

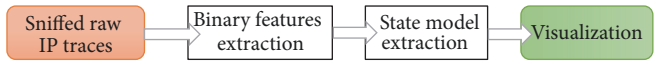

Figure 6: Trifilo et al. workflow design {Figure reproduced from Trifilo et al. (2009), [under the Creative Commons Attribution License/public domain]}.

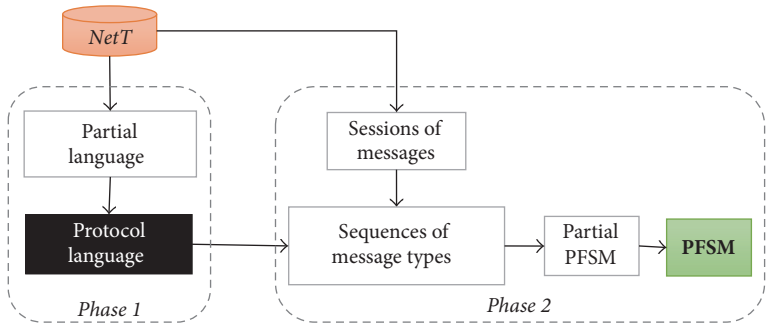

Figure 7: *ReverX* architecture overview {Figure reproduced from Antunes et al. (2011), [under the Creative Commons Attribution License/public domain]}.

Xiao et al. [3] introduce a grammatical inference algorithm to model network applications specifications by syntax inference, under an observation that an implementation of the protocol is inherently a state transition process. In this approach, various state protocols are described through modelling methods of protocol state transitions for both known and unknown protocols. The system architecture proposed by Yin et al. consists of two major stages, the *execution monitor* by [45] where execution traces are outputted and the *Protocol State Machine inference*. To obtain a PFSM in the second stage, messages between sessions are separated and the format of each message is inferred by delimiters obtained in the execution traces and then the automaton of the protocol is inferred as the order and transitions of messages indicate in the extracted execution traces.

Trifilo et al. [17] derives the PFSM of a protocol by analyzing logics from both sides captured network traces. Figure 6 {[17], Trifilo et al. (2009)} indicates relevant fields extraction procedures to determine the general protocol states and their transitions. As shown in Figure 6, in the first module, extracted raw data traffic is filtered to separate which protocol traffic will enter the system for PRE process. In the second module, fields' binary features are extracted, where only target and key protocol fields are identified. Trifilo et al. assume that key protocol fields contain the general meaning and logic of the protocol, while other fields are considered least. For example, in HTTP protocol request, "GET" is a key and necessary field to understand the protocol logic as it defines the type of requested action. To achieve this, a statistical analysis based on the "Variance of the Distribution of Variances" (VDV) is used. In the *state machine model extraction*, based on the features already extracted a PFSM for a target protocol is finally created.

Based on sequence alignment techniques [44], Antunes and Neves [18] build a PFSM of a network protocol from network traces. From the sequence alignment techniques, *Greedy algorithm* and *Partial Order Alignment* (POA), they generally focus on providing solutions to construction of automata that recognize specific protocols based on their sampled messages. *ReverX* [19] is a methodology that, from real network traffic, automatically infers protocol specifications and the PFSM. It is a suitable approach and a well-designed frame work for uncovering both message formats and PFSMs for unknown/undocumented protocols. *ReverX* is divided into two phases, Generalize *Protocol Language* and reduced *Protocol State Machine* as indicated in Figure 7 {[19], Antunes et al. (2011)}.

Veritas system [20] is a statistical based analysis method that can automatically infer target PFSM from captured real-world network traffic. The key feature of a tool is that it is based on statistical information of target protocol and it is designed to deal with both text and binary protocols. The system has four modules, collection of network traffic, analysis of captured packets, inference of messages, and state machines.

In first module, target network traffic is collected based on target transport layer port of an application. In the second phase, message units with high frequency are identified in an offline means and the K-S (Kolmogorov-Smirnov) test is applied to obtain optimal units for a message. The phase where message states are inferred, and from each protocol format message distinctive features are extracted to measure the similarity between message formats. Next, the PAM (Partitioning Around Medoids) clustering algorithm applies to classify similar messages in their respective clusters. Finally, the state machine inference phase infers a PFSM by building flows of a specific protocol from labeled states.

Zhang et al. [21] present an architecture of three components, packets analyzing stage, Protocol State Machines exploring stage, and the QSM *(Query-driven State Machine) Learner* [48]. The components describe a method for constructing and parsing real packets to their formats and generating packets queries for exploration to mine PFSMs. In the learning process, an interactive grammar inference

technique is involved to generate queries to the protocol implementation. The architecture functions as follows: first it uses Wireshark [49] to capture raw network traffic of a targeted protocol as implemented by ends user and then saves the captured sessions in pcap file. Second, the packets analyzing module converts all the packets to xml-based files that appropriately describe the packet structure. Third, the Protocol State Machines Space Explorer creates several sequences to communicate with the target protocol. Lastly, protocol formats and specifications are mined from the QSM Learner algorithm combined with EDSM algorithm [48].

Laroche et al. [22] explore a Linear Page Based Genetic Programming (NimsGP), an evolutionary approach, and propose a solution for analyzing and discovering PFSMs and specifications. By using a Genetic Programming (GP) based technique with a priori algorithm implementation, a given Protocol State Machine can be analyzed. By this method two text protocols, FTP and DHCP, are analyzed. Meng et al. [23] propose an approach capable of mining protocol state machines for unknown binary protocols. The methodology aligns corresponding fields and extracts the state relevant fields from binary protocol communication traces and finally the protocol state model is constructed based on the state relevant fields.

Four functional modules, communication data frame capture, alignment of corresponding fields, state relevant fields identification, and state machine reconstruction, are involved in the proposed architecture. In the communication data frame capture tools, such as Wireshark, are used to capture and filter the raw traffic. Next, corresponding fields are aligned by an improved progressive multiple sequence alignment technique. Third, in the state relevant fields identification module, only certain fields of a protocol frame are captured by the logic of the protocol. For example, flags fields are state relevant fields in TCP. A statistical analysis, "Distribution of the Distribution of the Variances (DDV)" for each field in the binary protocol frame is applied to identify state relevant fields. Finally, a Protocol State Machine is reconstructed based on the state relevant fields.

Section 3.1.2 describes 9 approaches that focus on reverse engineering undocumented and/or unknown protocols and outputting their general PFSMs. In this section as well, a diversity of techniques and algorithms towards reaching this goal is observed from received network traces *(packets, flows,* and *sessions)* and execution traces, whereby approaches that utilize network traces are much more than those utilizing execution traces. Out of 9 approaches, 7 utilize network traces based inputs and 1 approach utilizes both network trace and execution trace whereas 1 approach utilizes execution traces based input.

3.1.3. Both Protocols Formats and PFSM Focus. This section consists of four approaches that provide architectures and literatures on reverse engineering both general protocol formats and *PFSMs*. Table 3 summarizes all the four approaches chronologically with their input formats.

GAPA [24] is a protocol analyzer and open language that is designed to satisfy three key goals, *safety*, *real-time analysis and response*, and *rapid development of analyzers*. The language is basically developed for integrating it into other network monitoring and packets analyzing tools, such as Ethereal and Shield to allow rapid development of protocol analyzers. GAPA language (GAPAL) uses a standard protocol analyzer specification called *Spec* that takes care of three tasks, *specifying how to parse the message format used by a protocol (by BNF (Backus-Naur Form))*, *correctly tracking sessions and applications context*, and *performing analysis based on the message content and the application context*. A *Spec* can also potentially carry out decisions, such as terminating a connection depending on conditions involved whether satisfying or not.

Table 3: Approaches, methods, and tools that focus on RE both PFs and PFSMs.

Approach, method, Tool, or author	Year	Input format		Special remarks
		NetT	ExeT	
GAPA [24]	2005		○	*BNF based*
Biprominer [25]	2011	○		~
Netzob [26, 27]	2012	○	○	~
AutoReEngine [28]	2013	○		~

Biprominer [25] is a statistical nature tool that is designed to extract different formats of binary protocol messages and present its PFSM. Its architecture has three major parts, *the learning phase* (the learning process that provides several pattern messages called *cells*), *the labeling phase* (marks samples with cells obtained from the learning phase), and *the transition probability model building phase* (which gives a probabilistic description of reverse engineered protocol formats). Figure 8 {[25], Wang et al. (2011)} shows a whole architecture of Biprominer in three parts.

Netzob [26, 27] is a tool capable of inferring both PFs and PFSMs for complex protocols. The tool takes its inputs as communication traces and reverses the involved protocol vocabularies by considering embedded contextual information. From this information, clustering of messages and protocol fields' boundaries identification is improved. In vocabulary and grammar inference, *Netzob* semiautomatically undergoes three steps, clustering of messages and message fields partitioning, message fields characterization and abstraction of similar messages in symbols, and inferring the transition PFSM graph. Moreover, *Netzob* extends the Needleman and Wunsch [44] sequence alignment algorithm to leverage message semantic definition so that message classification and format inference are much effective from considering both protocol semantics and syntaxes, in the sequences of static and dynamic fields.

In *AutoReEngine* [28], PFSMs of application-layer protocols are automatically inferred from network traces. The support values and variance positions are applied to extract fields' keywords. Frequent strings in a given message are extracted to identify fields and keywords. For instance, since the field keyword "GET" has high frequency in HTTP sessions, it is considered as a field keyword. This is an *Apriori property* implementation (Agrawal and Srikant, 1994). *AutoReEngine*

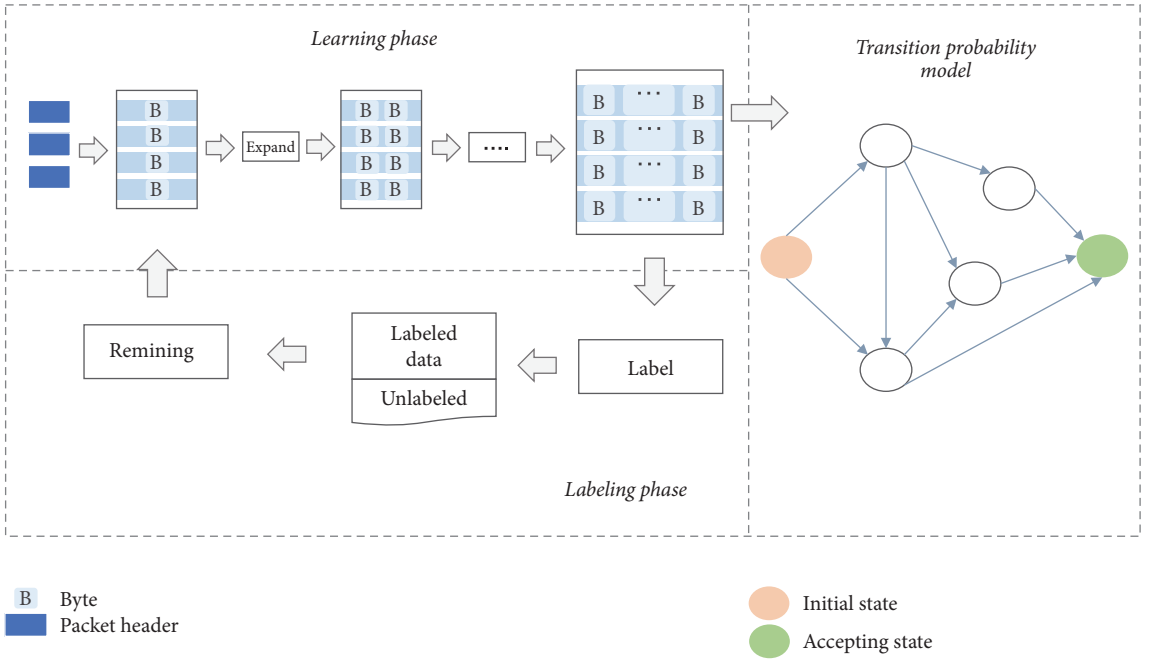

Figure 8: *Bipromine* architecture {Figure reproduced from Wang et al. (2011), [under the Creative Commons Attribution License/public domain]}.

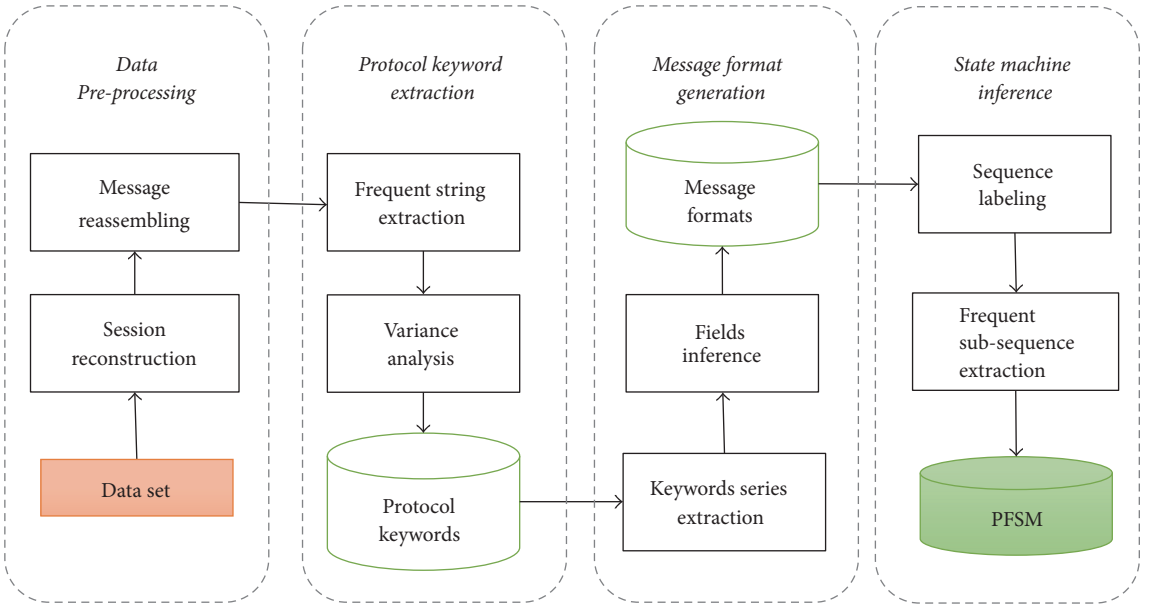

Figure 9: *AutoReEngine* architecture {Figure reproduced from Luo and Yu (2013), [under the Creative Commons Attribution License/public domain]}.

defines a message format as a series of frequent keyword appearing in a target protocol message format. State machines in AutoReEngine can be achieved through two steps, *labeling all messages according to whichever message format they belong to* and *applying the Apriori algorithm to extract frequent subsequence* for transitions identification. As indicated in Figure 9 {[28], Luo and Yu (2013)}, the architecture of *AutoReEngine* has four main modules, data preprocessing, second, extraction of protocol keywords, third, message formats generation, and lastly PFSMs inference.

Section 3.1.3 describes 4 approaches that focus on reverse engineering undocumented and/or unknown protocols and outputting both their possible general Protocol Formats and Protocol Finite State Machines (PFSM). Various techniques and algorithms are involved towards reaching this goal. Among 4 approaches shown in Table 3, 2 approaches utilize

Table 4: Approaches that focus on neither reverse engineering general PFs nor PFSMs.

Approach, method, tool, or author	Year	Input format		Special remarks
		NetT	ExeT	
ScriptGen [29]	2005	○		*Dialogs/scripts*
RolePlayer [30]	2006	○		*Dialogs/scripts*
Ma et al. [31]	2006	○		*App-identification*
Boosting [32]	2008	○		*Field(s)*
Dispatcher [6]	2009		○	*C&C malware*
ASAP [33]	2011	○		*Semantics*
Dispatcher2 [34]	2013		○	*C&C malware*
ProVeX [35]	2013	○		*Signatures*
PIP [36]	2014	○		*Keywords/ fields*
FieldHunter [37]	2015	○		*Fields*
RS Cluster [38]	2015	○		*Grouped-messages*
UPCSS [39]	2015	○		*Proto-classification*
PowerShell [40]	2017	○		*Dialogs/scripts*
ProPrint [41]	2017	○		*Fingerprints*
ProHacker [42]	2017	○		*Keywords*

network traces based inputs and 1 approach utilizes both network trace and execution trace based inputs whereas 1 approach utilizes execution traces based input.

3.1.4. Neither Protocols Formats Nor PFSMs Focus. Most of the approaches target directly on reverse engineering either general protocol formats or *PFSM*s. As shown in Table 4, this section presents approaches, methods, and tools that focus on reverse engineering directly neither PFs nor PFSMs, instead they focus on identifying either individual fields, conversations carried between two entities (scripts/dialogs), applications identification, signatures generation, or semantics inference.

ScriptGen [29], *RolePlayer* [30], and *PowerShell* [40] are approaches that focus on generating new scrips or dialogs between two hosts based on the network protocols involved, for attacks and network vulnerability detections. *RolePlayer* [30], given session samples of transmission or communication between two entities, can analyze application protocols in wide varieties. RolePlayer applies bytes streams technique to align and compare different session instances to determine which session side fields to replay. The method can heuristically detect network information such as addresses, ports, cookies, and length fields confined in a session. Meanwhile, PowerShell [40] is a tool that investigates, analyzes, and generates protocol catalogs or scripts models involved during two communicating machines without installing third-party tools, such as Wireshark and Tcpdump. The generated scripts allow evaluating an environment for any potential vulnerabilities.

ScriptGen [29] is a method consisting of four modules (factorization of message sequence, state machine building, state machine simplification, and generation of scripts). ScriptGen can generate scripts for exploiting possible attacks in networks.

In the first module, TCP protocols based messages that exchanged between the receiver and the sender sides are extracted by Tcpdump and all the TCP streams are reconstructed and reordered. In state machine building stage, states representing blocks are built. To avoid a high redundant among state machines, thresholds are defined and applied to control the complexity and limit the number of unsatisfying block edges for each state. Threshold application has a disadvantage, since when it is strictly implemented, the scripts produced may not represent real behaviors of the targeted server. Simplification of protocol states machines is the core module of *ScriptGen, where* analysis of raw state machines is done. In this stage target protocol key semantics are introduced from the two distinct algorithms, the PI algorithm [50] and the Region Analysis algorithm (developed by ScriptGen). Finally, a simpler and reduced state machine is obtained. The last module, Script Generator, creates some compatible *scripts* from the simplified PFSMs.

Interests of Protocol Reverse Engineering may vary from one approach and another. Ma et al. [31] use PRE techniques to identify application based on their applications-layer protocols. To achieve this goal, protocol models, product distribution model, and Markov process model are implemented with unsupervised algorithms, LCS, and Smith-Waterman algorithm. The approach can identify known applications regardless of their port numbers while unknown traffic applications can distinctively be identified from the known ones. Besides, Ma et al. utilize real-world traffic traces to evaluate each mechanism for applications' classification.

Boosting [32], PIP [36], and *FieldHunter* [37] are approaches solely focusing on identifying and reverse engineering protocol fields or fields of interest. *Boosting* [32] is an algorithm applied to extract specific fields of interest in an unknown protocol, from an active learning framework, in which the user presents the system with a small number of labeled instances and the system automatically generates sufficient features and classifiers of targeted fields or interested fields. *PIP* [36] is a Protocol Informatics Project that discusses four algorithms for similarities alignment when given two or more sequences. The algorithms show promising performance in protocol fields identification, especially text based protocols. PIP discusses four algorithms, which are Needleman-Wunsch [44] *(Sequence Alignment)*, BLOSUM&PUM *(Similarity Matrices)*, UPGMA *(Phylogenetic Trees)*, *and* Phylogeny *(Multiple Alignment)*.

FieldHunter [37] is a self-developed algorithm system, which can automatically extract fields of binary and text protocols and infer their types by computing statistical correlations of collected target application messages from multiple sessions. The computed statistical correlations here come from different messages or other associations in metadata such as *message length, client, or server IP addresses*. *FieldHunter* first studies well known protocols to identify fields boundaries and infer their types which is statistical and heuristic information gathering and can be applicable in many security applications. The general focus of the system is to identify Message Types, such as flags in DNS protocol

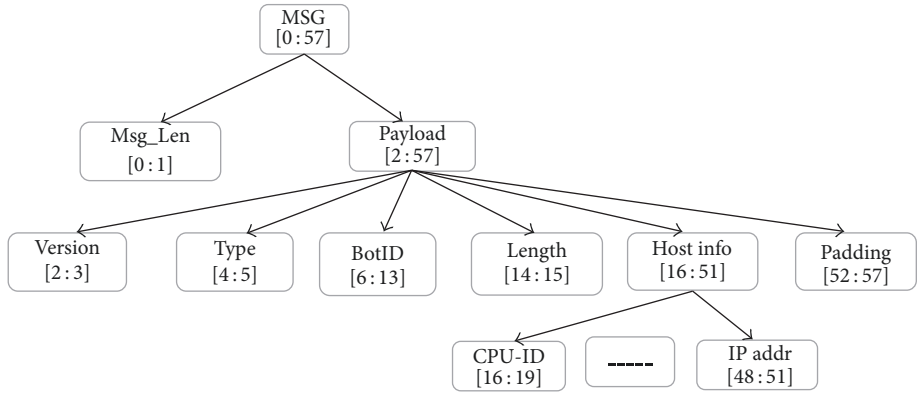

Figure 10: Dispatcher: message field tree for MegaD host information message {Figure reproduced from Caballero et al. (2009), [under the Creative Commons Attribution License/public domain]}.

or GET/POST keywords in HTTP, *Message Lengths*, mostly found in TCP protocols to delimit application messages in streams, *Host Identifiers* such as Client ID and Server ID, *Session Identifiers* such as cookies, *Transaction Identifiers* such as sequence/acknowledgement numbers, and *Accumulators* such as generic counters and timestamps.

FieldHunter is composed of twosteps modules, the *field extraction step and* the *field type inference step.* In the field extraction step, fields are extracted from target protocol messages in different techniques depending on whether the messages belong to textual or binary protocols. In the field type inference step both textual and binary protocols apply common statistical properties and heuristic techniques to infer types of messages. This message types' inference step is a potential step in discovering the whole protocol format and/or the PFSM of a target protocol. Thus, iterated application of FieldHunter system for single target undocumented protocol may result in a protocol general format or its PFSM.

Uncovering of the C&C used by a botnet in network hosts is essential for infiltration and security analytical measures. *Dispatcher* [6] 2009 and *Dispatcher2* [34] 2013 are two consecutive works that attempt to uncover the C&C used by a botnet in network hosts. They present techniques for extraction of message formats and semantics sent or received by the client side or the server side. Although, both Dispatcher [6] 2009 and Dispatcher2 [34] 2013 Protocol Reverse Engineering techniques focus on extracting message formats and protocol fields semantics for unknown protocols, Dispatcher2 [34] goes further and extracts a program that implements the protocol for even encrypted protocol traffic. An example of message format parsing and fields tree for C&C message generation and extraction by Dispatcher is shown in Figure 10 {[6], Caballero et al. (2009)}.

As Figure 10 {[6], Caballero et al. (2009)} shows, a single message in a formation of several fields that appear syntactically and semantically. In this example, a single message of captured protocol traffic is presented in a tree form and all the node leaves are fields of the message where child nodes represent subfields of their parents. The root node stands for the whole message with 58 bytes [6]. From offset 0 to offset 57, each field is situated at its own offset range carrying specified bytes and a meaning for the protocol. In this example, all fields clearly show what kind of information they do carry. For instance, in the payload, a subfield (6–13) indicates the Bolt ID while a subfield (16–51) indicates host information such as a CPU ID (16–19), an IP identifier (48–51), and other pieces of host related information (19–48).

A three-step framework for semantics analysis of network payloads is proposed by *ASAP* [33], where semantics of unknown protocols fields can be inferred. The ASAP framework divides into three major steps, extraction of network payloads alphabet, matrix factorization analysis, and construction of communication templates. ASAP focuses on protocol field semantics rather than syntaxes. *ProVeX* [35] is a designed system to detect and discover C&C botnets' traffic that is encrypted in their C&C protocols by probabilistically learning C&C inputs. To attain this goal, all packets captured from potential malware C&C protocols are first decrypted and their bytes are evaluated to generate the signatures to identify the protocol syntax of C&C botnets messages. ProVeX functions as follows: First, it *uses* an *n-gram* based analysis to find botnets which do not have characteristic payload strings in their encrypted network traffic data. Next, it reverse engineers and reimplements decryption routines of all bots, based on a fact that botnets are symmetric in encryption. Lastly, probabilistic C&C models and protocol syntaxes are inferred.

RS Cluster [38] is an unsupervised method capable of reverse engineering different protocols by implementing a rough set-based technique. The technique is suitable for clustering network traffic and group protocol messages according to their types and can analyze multidimension of uncertain information in multiple categorical attributes based on Rough Sets theory [51]. UPCSS (Unknown network Protocol Classification method based on Semi-Supervised learning) [39] is a semisupervised learning method, proposed to identify applications from unknown protocols by labeling small training sample set. Based on Erman's semisupervised approach, UPCSS is designed to detect unknown samples generated by unknown protocols with the help of flow correlation information and semisupervised clustering techniques.

ProPrint [41] is a network trace based protocol fingerprint inference approach that focuses on distinguishing individual application protocols. Fingerprints are set of bytes subsequences in packet payload which can act as application protocols classifiers. To attain this goal *ProPrint* builds a nonparametric and modified Bayesian statistical protocol language model and next uses a corresponding protocol language model to identify field boundaries in packet payload to segment each payload into a set of protocol feature words. ProPrint applies its own developed ranking algorithm to select true protocol fingerprints from the candidate protocol feature words. ProHacker [42] is a network traces based system that attempts to extract protocol keywords and identify the protocol trace from mixed Internet traffic. It uses the *n-grams* of protocol traces to predict statistical nature that can be captured from statistical language models. *ProHacker* in similar fashion to ProPrint [41] uses nonparametric Bayesian statistical model, a corresponding protocol keywords to extract protocol keywords and classify protocol traces by a semisupervised learning algorithm.

Section 3.1.4 describes 15 approaches that focus on neither reverse engineering general PFs nor PFSMs of unknown protocols. However, all the approaches provide detailed literature, techniques, and algorithms that end in uncovering both PFs and PFSMs. Among 15 approaches, 13 approaches utilize network traces based inputs and 2 approaches utilize execution traces based inputs.

3.2. Analysis on Automatic Protocol Reverse Engineering Inputs. *Automatic Protocol Reverse Engineering* inputs divide into two major categories, *network traces and execution traces*; however, they may vary as well from one approach to another in each single category. Section 3.1 describes 39 approaches, methods, and tools' outputs in four divisions, whereby 29 approaches utilize only *network traces*, 8 approaches utilize only execution traces, and 2 approaches utilize both *network* and *execution traces* as their systems inputs.

Regardless of the fact that network traces based approaches are often faced with significant challenges, such as reconstruction of application-layer messages and synchronization on the start of the message prior to any clustering algorithms or format extraction, we have found that many approaches, methods, and tools are utilizing more network traces to execution traces in reverse engineering unknown or undocumented protocols. A key drawback of network traces based approaches is that they rely solely on defined message boundaries to infer protocol format [1]. *ReFormat* [9] emphasizes that APRE based on network traces is limited by the lack of semantic information. Besides, [34] points that using dynamic program binary analysis (execution traces) is more efficient in analyzing unknown or undocumented protocols since execution traces are more efficient to infer specification of unknown and even encrypted protocols.

Most network traces based approaches have resulted in trustful methods or developed tools [26]. However, these methods are often neither suitable nor applicable in real operations. Only few of these methods are significant to allow scientific community to experimentally validate and compare them when taken into applications in real network environment. One major reason for this challenge is the fact that network activities in laboratories never perfectly reflect that observed in the wild [26]. Although more approaches use network to execution traces, execution traces are more appropriate inputs that results to more accurate reverse engineered protocol fields. Execution traces are binaries that indicate instructions loops' (commands) starts and ends of a communication, hence allowing effective Protocol Reverse Engineering.

4. Discussion

This section discusses APRE approaches, methods, and tools based on automated to manual approaches, text to binary, binary to hybrid, and other reverse engineered protocols. Additionally, the detailed literature of the approaches based on the 7-layer OSI model is discussed.

4.1. Automated to Manual PRE Approaches. In practice, almost all PRE approaches, methods and tools undergo manual techniques at either module in their systems. However, in presentation almost all PRE approaches claim to be full automated. Indeed, PRE is still a highly manual process, moreover an error-prone, tedious and time consuming [4], due to lack of basic PRE knowledge as such that given by Yurichev [43]. For open protocols such as HTTP and DNS, the specifications are open in RFCs (Request for Comments) while for unknown/undocumented protocols their specifications need to only be reverse engineered. Numerous obstacles such as binary human unreadable protocols, high quantities of high entropy data, dynamically sized message fields, context specific fields in which earlier fields in a message change the meaning of following fields, and the possibility of data protection schemes within the data such as encryption, prevent efficient Protocol Reverse Engineering.

4.2. Categories of Reverse Engineered or Analyzed Protocols. This section presents four divisions of the reverse engineered or analyzed protocols, *Text, Binary, Hybrid, and Others* (unknown/undocumented protocols which may be text, binary, or hybrid) by four approaches outputs categories presented in Section 3.1. Protocols are mostly categorized as text or binary; however in this paper we find four possible divisions of reverse engineered or analyzed protocols, whereby *Discoverer* [4] and *RolePlayer* [30] specifically categorize SMB and CIFS protocols as hybrid. However, in *Prospex* [10], *ProDecoder* [11], and *Netzob* [26] SMB protocol is termed as binary protocol, from which we conclude that SMB is a binary-hybrid protocol. A hybrid protocol is basically a crossed protocol or a protocol that is multifunctions, more efficient, and more reliable than previous versions or previous protocols such as HRP (Hybrid Routing Protocol). Text and binary protocols are well defined in the sixth paragraph of Section 1. Protocols that are unknown/undocumented protocols and that do not fall to either of the three, text, binary, or hybrid fall to the *other* category.

Table 5 shows four divisions of reverse engineered or analyzed protocols by approaches that focus on reverse

Table 5: Four divisions of reverse engineered or analyzed protocols by PFs approaches.

Approach, method, tool, or author	Protocols analyzed or reverse engineered			
	Text	Binary	Hybrid	Others (unknown/undocumented)
Discoverer [4]	HTTP	RPC	SMB, CIFS	*None*
Polyglot [5]	HTTP, Samba, ICQ	DNS, IRC	*None*	*None*
AutoFormat [7]	HTTP, SIP	DHCP, RIP, OSPF	SMB, CIFS	*None*
Tupni [8]	HTTP, FTP	RPC, DNS, TFTP	*None*	WMF, BMP, JPG, PNG, TIF
ReFormat [9]	HTTP, MIME	IRC	*None*	One unknown protocol
Prospex [10]	SMTP, SIP	SMB	*None*	Agobot (C&C)
ProDecoder [11]	SMTP	SMB	*None*	*None*
Wang et al. [12]	ICMP	ARP	*None*	*None*
ProGraph [13]	HTTP	DNS, BitTorrent, WeChat	*None*	*None*
Cai et al. [14]	HTTP, SSDP	DNS, BitTorrent, QQ, NetBios	*None*	*None*
WASp [15]	*None*	*None*	*None*	Smart plug & PSD systems

Table 6: Four divisions of reverse engineered or analyzed protocols by PFSMs approaches.

Approach, method, tool, or author	Protocols analyzed or reverse engineered			
	Text	Binary	Hybrid	Others (unknown/undocumented)
PEXT [16]	FTP	*None*	*None*	*None*
Xiao et al. [3]	HTTP, FTP, SMTP	*None*	*None*	*None*
Trifilo et al. [17]	*None*	TCP, DHCP, ARP, KAD	*None*	*None*
Antunes and Neves [18]	FTP	*None*	*None*	*None*
ReverX [19]	FTP	*None*	*None*	*None*
Veritas [20]	SMTP	PPLIVE, XUNLEI	*None*	*None*
Zhang et al. [21]	HTTP, SNMP, ISAKMP	*None*	*None*	*None*
Laroche et al. [22]	FTP	DHCP	*None*	*None*
Meng et al. [23]	*None*	TCP, ARP	*None*	*None*

engineering protocol formats while Table 6 shows four divisions as well, of reverse engineered or analyzed protocols by approaches that focus on reverse engineering PFSMs. Table 7 shows four divisions of reverse engineered or analyzed protocols by approaches that can reverse engineer both PFs and PFSMs. Lastly, Table 8 shows four divisions of reverse engineered or analyzed protocols by approaches that do not focus directly on either RE protocol formats or PFSMs.

Several challenges are mentioned in Section 1, such as binary human unreadable protocols, high entropy data, dynamic protocol fields, and context based fields in which earlier fields affect the meaning or the form of following fields, encryption, and so on. To this point there is no single approach that specifically deals with a particular challenge. However, we find that for binary human unreadable protocols, execution trace inputs utilization is a better choice to approach uncovered fields, their syntaxes, and semantics. For high entropy of data and dynamic protocol fields, collection of statistical large data set of the target protocol may be a better solution from which noise traffic data can be eliminated and fields' dynamic can easily be studied. For context based fields in which earlier fields affect the meaning or the form of the following field, application of association rules related algorithms may be effective.

Automatic Protocol Reverse Engineering approaches, methods, and tools are essential in modern network technology for parsing the protocols and for security in general. This paper sees the challenges of unskilled network administrators if these approaches, methods, and tools were not open. Furthermore, this paper sees a high risk of certain networks to be under surveillance while administrators are unaware. Attackers may take advantage of this unawareness and may figure out how to play with hosts in a network. Some traffic applications pretend to be certain protocols only for surveillance and finding possible vulnerabilities in a network to launch attacks. Since HTTP is the most used application protocol in the Internet by far, we observe high risks of web servers and web applications where most of these protocols are involved in one way or another. Being aware of how to reverse engineer protocols in a specific layer or in all the 7 layers of the OSI model in important, however, this paper finds that knowing more about how HTTP protocol operates in web servers and web applications is even more important. For instance, in HTTP protocol requests and responses, there are header fields such as *Cookie*, *User-Agent*, *Referer*, and *Host* that need to be well studied. Within these header fields there are distinctive feature vectors between normal and abnormal traffic that if deeply studied will enhance security in web servers and web applications.

Table 7: Four divisions of reverse engineered or analyzed protocols by approaches focusing both PFs and PFSMs.

Approach, method, tool, or author	Protocols analyzed or reverse engineered			
	Text	Binary	Hybrid	Others (unknown/undocumented)
GAPA Spec [24]	HTTP	*None*	*None*	*None*
Biprominer [25]	*None*	XUNLEI, QQLive, SopCast	*None*	*None*
Netzob [26, 27]	FTP, Samba	SMB	*None*	Unknown P2P protocol & VoIP commercial product protocol
AutoReEngine [28]	HTTP, POP3, SMTP, FTP	DNS, NetBIOS	*None*	*None*

Table 8: Four divisions of reverse engineered or analyzed protocols by approaches that do not directly focus on reverse engineering PFs or PFSMs.

Approach, method, tool, or author	Protocols analyzed or reverse engineered			
	Text	Binary	Hybrid	Others (unknown/undocumented)
ScriptGen [29]	HTTP	NetBios	*None*	DCE
RolePlayer [30]	NFS, FTP, HTTP, SMTP, TFTP	DNS	SMB, CIFS	*None*
Ma et al. [31]	FTP, SMTP, HTTP, HTTPS (TCP-Protos)	DNS, NetBIOS, SrvLoc (UDP-Protos)	*None*	*None*
Boosting [32]	*None*	DNS	*None*	*None*
Dispatcher [6]	HTTP, FTP, ICQ	DNS	*None*	*None*
ASAP [33]	HTTP, FTP, IRC, TFTP	*None*	*None*	*None*
Dispatcher2 [34]	HTTP, FTP, ICQ	DNS	*SMB*	*None*
ProVeX [35]	HTTP, SMTP, IMAP	DNS, VoIP, XMPP	*None*	Malware Family Protocols
PIP [36]	HTTP	*None*	*None*	*None*
FieldHunter [37]	MSNP	DNS	*None*	SopCast, Ramnit
RS Cluster [38]	HTTPS, POP3, SMTP, FTP	DNS, XunLei, BitTorrent, BitSpirit, QQ, eMule	*None*	MSSQL, Kugoo, PPTV
UPCSS [39]	IMAP, HTTP, SMTP, FTP, POP3	DNS, SSL, SSH	SMB	*None*
PowerShell [40]	*None*	ARP, OSPF, DHCP, STP	*None*	CDP/DTP/VTP, HSRP, LLDP, LLMNR, mDNS, NBNS, VRRP

4.3. Protocols Classification by 7 Layers of the OSI Model. The OSI model covers a large spectrum of typical communication functions from its electrical aspects with the physical layer to the application layer. Thus, in real network environment communication or transmission between two endpoints, seven protocols are possibly involved. This kind of protocols layering is known as protocol family. From this point of view if an approach considers every layer to uncover specifications of a target or unknown protocol, the process may turn extremely hard. Therefore, many approaches reverse engineer target protocols in a particular layer. However, given a communication or transmission, reverse engineering an involved protocol from layer 1 to layer 7 and vice versa would indicate a complete behavior and perfect protocol format. GAPA [24] attempts to consider protocol layering with a developed GAPAL Spec, whereas at each layer the language reverse engineers and connects pre- and postlayer protocols.

As a single universal protocol would be very hard to design and implement correctly, similarly a single unknown protocol would be very hard to reverse engineer when only one and specific layer is considered. The challenge to this impossibility is due to the fact that network protocols operate in very heterogeneous environments that consist of very different network technologies and sometimes having very rich set of applications. To ease protocols design a layered architecture of protocols is promoted by the OSI model and retained in very most network related communication protocols [52], which in reverse simplifies the whole process of reverse engineering both known and unknown protocols in a specific layer and in the whole OSI model.

In Table 9, we show a category of reverse engineered or analyzed protocols based on the OSI model with approaches for each layer. Application-layer protocols appear to be highly analyzed or reverse engineered, whereby 33 protocols are

Table 9: Classification of analyzed or reverse engineered protocols based on the 7-layer OSI Model.

OSI-layer	Analyzed/reverse engineered protocols	Related approach, method, or tool
7-application	HTTP, HTTPS, DNS, SIP, FTP, TFTP, IMAP, SSH, IRC, XMPP, SMTP, BitTorrent, MIME, SNMP, MSNP, POP3, DHCP, *SMB*, CIFS, Samba, XunLei, BitSpirit, QQLive, eMule, ICQ, SopCast, PPLIVE, PPTV, KAD, ISAKMP, SSDP, WeChat, Kugoo	[3–11, 13, 14, 16–22, 24–26, 28–30, 32–37, 39, 41, 42]
6-presentation	*SMB*, WMF, BMP, JPG, PNG, TIF, MIME, *SSL*	[8–11, 29, 39]
5-session	RPC, NetBIOS, NFS, *SSL*	[8, 14, 29, 30, 39, 40]
4-transport	TCP, UDP, SrvLoc	[17, 23, 31, 38]
3-network	ICMP, OSPF, RIP	[7, 12, 40]
2-data link	ARP, STP	[12, 15, 17, 23]
1-physical	*none*	[15]

analyzed or reverse engineered by 33 out of 39 approaches, methods, and tools presented in this paper. HTTP protocol appears in 21 approaches as the most analyzed protocol. WASp [15] which is explained in Section 3.1.1 is the only approach generally focusing on analyzing physical layer protocols, although no single specific protocol of the physical layer is mentioned as analyzed or reverse engineered.

5. Conclusion and Potential Future Research

This paper, discusses 39 approaches, methods, and tools that focus on automatic reverse engineering network protocols, especially unknown and undocumented ones. In Sections 3.1 and 3.2, we evaluate the approaches outputs in four divisions based on their inputs, network traces, and execution traces. We further present discussion about the reviewed approaches in terms of automated to manual, categories of analyzed or reverse engineered protocols in terms of text, binary, hybrid, or others (unknown or undocumented) and analyze in depth the seven layers of OSI model in relation to APRE along with classification for all analyzed or reverse engineered protocols based on the OSI model. Moreover, in the introductory part, Section 1, a detailed literature and general key terminologies of PRE are provided.

We now draw future and potential research directions of APRE based on Sections 3 and 4. Automatic Protocol Reverse Engineering is an increasingly important field in modern network communication environment and security. However, this paper finds that poor knowledge on reverse engineering unknown protocols is the first major drawback. Having little background on the PRE and APRE, many approaches propose frameworks and architectures that do not result into promising outputs. For instance, most approaches utilize network traces to execution traces. Utilization of network traces provide few syntax and semantic information of protocol messages, hence always outputting protocol specifications that have low correctness, coverage, and conciseness. From the reviewed approaches, those that utilize execution trace analysis result in promising outputs which almost match true specifications of the analyzed protocols. Additionally, almost all the reviewed approaches do not consider more that on layers of the OSI model. Since at every layer of the OSI model a single protocol is involved to handle a communication [53], to be accurate and efficient, Protocol Reverse Engineering approaches should consider and analyze messages at each layer *(from physical layer to application layer)* to uncover all protocols involved in the OSI model and draw a much wider and detailed format or behavior of a certain network and Internet transmission or communication.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A3A01018057) and by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea Government (MSIT) (no. 2017-0-00513).

References

[1] J. Narayan, S. K. Shukla, and T. C. Clancy, "A survey of automatic protocol reverse engineering tools," *ACM Computing Surveys*, vol. 48, no. 3, article 40, 2015.

[2] X. Li and L. Chen, "A survey on methods of automatic protocol reverse engineering," in *Proceedings of the 7th International Conference on Computational Intelligence and Security (CIS '11)*, pp. 685–689, December 2011.

[3] M.-M. Xiao, S.-Z. Yu, and Y. Wang, "Automatic network protocol automaton extraction," in *Proceedings of the 3rd International Conference on Network and System Security (NSS '09)*, pp. 336–343, October 2009.

[4] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: automatic protocol description generation from network traces," in *Proceedings of the USENIX Security Symposium*, 2007.

[5] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 317–329, ACM, November 2007.

[6] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 621–634, ACM, Chicago, Ill, USA, November 2009.

[7] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in *Proceedings of the 15th Symposium on Network and Distributed System Security (NDSS '08)*, February 2008.

[8] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: automatic reverse engineering of input formats," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, pp. 391–402, ACM, Alexandria, Va, USA, October 2008.

[9] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "ReFormat: automatic reverse engineering of encrypted messages," in *Computer Security—ESORICS 2009. ESORICS 2009*, M. Backes and P. Ning, Eds., vol. 5789 of *Lecture Notes in Computer Science*, pp. 200–215, Springer, Berlin, Germany, 2009.

[10] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: protocol specification extraction," in *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pp. 110–125, Berkeley, Calif, USA, May 2009.

[11] Y. Wang, X. Yun, M. Z. Shafiq et al., "A semantics aware approach to automated reverse engineering unknown protocols," in *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP '12)*, pp. 1–10, IEEE, Austin, Tex, USA, November 2012.

[12] Y. Wang, N. Zhang, Y.-M. Wu, B.-B. Su, and Y.-J. Liao, "Protocol formats reverse engineering based on association rules in wireless environment," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '13)*, pp. 134–141, Melbourne, Australia, July 2013.

[13] Q. Huang, P. P. C. Lee, and Z. Zhang, "Exploiting intra-packet dependency for fine-grained protocol format inference," in *Proceedings of the 14th IFIP Networking Conference (NETWORKING '15)*, Toulouse, France, May 2015.

[14] J. Cai, J. Luo, and F. Lei, "Analyzing network protocols of application layer using hidden Semi-Markov model," *Mathematical Problems in Engineering*, vol. 2016, Article ID 9161723, 14 pages, 2016.

[15] K. Choi, Y. Son, J. Noh, H. Shin, J. Choi, and Y. Kim, "Dissecting customized protocols: automatic analysis for customized protocols based on IEEE 802.15.4," in *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 183–193, Darmstadt, Germany, July 2016.

[16] M. Shevertalov and S. Mancoridis, "A reverse engineering tool for extracting protocols of networked applications," in *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE '07)*, pp. 229–238, October 2007.

[17] A. Trifilo, S. Burschka, and E. Biersack, "Traffic to protocol reverse engineering," in *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–8, July 2009.

[18] J. Antunes and N. Neves, "Building an automaton towards reverse protocol engineering," 2009, http://www.di.fc.ul.pt/~nuno/PAPERS/INFORUM09.pdf.

[19] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," in *Proceedings of the 18th Working Conference on Reverse Engineering (WCRE '11)*, pp. 169–178, October 2011.

[20] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, "Inferring protocol state machine from network traces: a probabilistic approach," in *Proceedings of the 9th Applied Cryptography and Network Security International Conference (ACNS '11)*, pp. 1–18, 2011.

[21] Z. Zhang, Q.-Y. Wen, and W. Tang, "Mining protocol state machines by interactive grammar inference," in *Proceedings of the 2012 3rd International Conference on Digital Manufacturing and Automation (ICDMA '12)*, pp. 524–527, August 2012.

[22] P. Laroche, A. Burrows, and A. N. Zincir-Heywood, "How far an evolutionary approach can go for protocol state analysis and discovery," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '13)*, pp. 3228–3235, June 2013.

[23] F. Meng, Y. Liu, C. Zhang, T. Li, and Y. Yue, "Inferring protocol state machine for binary communication protocol," in *Proceedings of the IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA '14)*, pp. 870–874, September 2014.

[24] N. Borisov, D. J. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo, "Generic application-level protocol analyzer and its language," MSR Technical Report MSR-TR-2005-133, 2005.

[25] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo, "Biprominer: automatic mining of binary protocol features," in *Proceedings of the 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '11)*, pp. 179–184, October 2011.

[26] G. Bossert, F. Guihéry, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, Kyoto, Japan, June 2014.

[27] G. Bossert and F. Guihéry, "Reverse and simulate your enemy botnet C&C," in *Proceedings of the Mapping a P2P Botnet with Netzob, Black Hat 2012*, Abu Dhabi, UAE, December 2012.

[28] J.-Z. Luo and S.-Z. Yu, "Position-based automatic reverse engineering of network protocols," *Journal of Network and Computer Applications*, vol. 36, no. 3, pp. 1070–1077, 2013.

[29] C. Leita, K. Mermoud, and M. Dacier, "ScriptGen: an automated script generation tool for Honeyd," in *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC '05)*, pp. 203–214, Tucson, Ariz, USA, December 2005.

[30] W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz, "Protocol-independent adaptive replay of application dialog," in *Proceedings of the 13th Symposium on Network and Distributed System Security (NDSS '06)*, 2006.

[31] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. Voelker, "Automatic protocol inference: unexpected means of identifying protocols," UCSD Computer Science Technical Report CS2006-0850, 2006.

[32] K. Gopalratnam, S. Basu, J. Dunagan, and H. J. Wang, "Automatically extracting fields from unknown network protocols," in *Proceedings of the 15th Symposium on Network and Distributed System Security (NDSS '08)*, 2008.

[33] T. Krueger, N. Krmer, and K. Rieck, "Asap: automatic semantics-aware analysis of network payloads," in *Proceedings of the ECML/PKDD*, 2011.

[34] J. Caballero and D. Song, "Automatic protocol reverse-engineering: message format extraction and field semantics inference," *Computer Networks*, vol. 57, no. 2, pp. 451–474, 2013.

[35] C. Rossow and C. J. Dietrich, "PROVEX: detecting botnets with encrypted command and control channels," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2013.

[36] M. Beddoe, "The protocol informatics project," 2014, http://www.4tphi.net/~awalters/PI/PI.html.

[37] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo, "Automatic protocol field inference for deeper protocol understanding," in *Proceedings of the 14th IFIP Networking Conference (Networking '15)*, pp. 1–9, May 2015.

[38] J.-Z. Luo, S.-Z. Yu, and J. Cai, "Capturing uncertainty information and categorical characteristics for network payload grouping in protocol reverse engineering," *Mathematical Problems in Engineering*, vol. 2015, Article ID 962974, 9 pages, 2015.

[39] R. Lin, O. Li, Q. Li, and Y. Liu, "Unknown network protocol classification method based on semi supervised learning," in *Proceedings of the IEEE International Conference on Computer and Communications (ICCC '15)*, pp. 300–308, Chengdu, China, October 2015.

[40] D. R. Fletcher Jr., *Identifying Vulnerable Network Protocols with PowerShell*, SANS Institute Reading Room site, 2017.

[41] Y. Wang, X. Yun, Y. Zhang, L. Chen, and G. Wu, "A nonparametric approach to the automated protocol fingerprint inference," *Journal of Network and Computer Applications*, vol. 99, pp. 1–9, 2017.

[42] Y. Wang, X. Yun, Y. Zhang, L. Chen, and T. Zang, "Rethinking robust and accurate application protocol identification," *Computer Networks*, vol. 129, pp. 64–78, 2017.

[43] D. Yurichev, 2013~2016, Reverse Engineering for Beginners.

[44] 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*, vol. 48, no. 3, pp. 443–453, 1970.

[45] H. Yin, D. Song, E. Manuel, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, Alexandria, Va, USA, October 2007.

[46] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the ACM CoNEXT Conference—4th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '08)*, December 2008.

[47] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '93)*, pp. 207–216, May 1993.

[48] P. Dupont, B. Lambeau, C. Damas, and A. Van Lamsweerde, "The QSM algorithm and its application to software behavior model induction," *Applied Artificial Intelligence*, vol. 22, no. 1-2, pp. 77–115, 2008.

[49] Wireshark: the world's most popular network protocol analyzer, http://www.wireshark.org/.

[50] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," 2005, http://www.insidiae.com/PI.

[51] L. J. Mazlack, A. He, and Y. Zhu, "A rough set approach in choosing partitioning attributes," in *Proceedings of the 13th ISCA International Conference (CAINE '00)*, pp. 1–6, New Orleans, La, USA, March 2000.

[52] A. Leon-Garcia and I. Widjaja, *Communication Networks: Fundamental Concepts and Key Architectures—Chapter 2, Applications and Layered Architectures*, Tata McGraw-Hill, 2004.

[53] G. Bossert, Exploiting Semantic for the Automatic Reverse Engineering of Communication Protocols, Supélec, NNT: 2014SUPL0027, 2014.