HAL
archives-ouvertes.fr

State of the art of network protocol reverse engineering tools

Julien Duchêne, Colas Le Guernic, Eric Alata, Vincent Nicomette, Mohamed Kaâniche

▶ **To cite this version:**

Julien Duchêne, Colas Le Guernic, Eric Alata, Vincent Nicomette, Mohamed Kaâniche. State of the art of network protocol reverse engineering tools. Journal of Computer Virology and Hacking Techniques, Springer, 2018, 14 (1), pp.53-68. <10.1007/s11416-016-0289-8>. <hal-01496958>

HAL Id: hal-01496958
https://hal.inria.fr/hal-01496958

Submitted on 11 Apr 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

State of the art of network protocol reverse engineering tools

J. Duchêne · C. Le Guernic
E. Alata · V. Nicomette · M. Kaâniche

Received: date / Accepted: date

Abstract Communication protocols enable structured information exchanges between different entities. A description, at different levels of detail, is necessary for many applications, such as interoperability or security audits. When such a description is not available, one can resort to protocol reverse engineering to infer the format of the messages exchanges or of a model of the protocol. During the past 12 years, several tools have been developed in order to automate, entirely or partially, the protocol inference process. Each of those tools has been developed with a specific application goal for the inferred model, leading to specific needs, and thus different strengths and limitations. After identifying key challenges, the paper presents a survey of protocol reverse engineering tools developed in the last decade. We consider tools focusing on the inference of the format of individual messages or of the grammar of sequences of messages. Finally, we propose a classification of these tools according to different criteria, that is aimed at providing relevant insights about the techniques used by each of these tools and comparatively to other tools, for the classification of messages, the inference of their format or of the grammar of the protocol. This classification also permits to identify technical areas that

J. Duchêne and C. Le Guernic
DGA Maîtrise de l'Information
BP 7
35998, RENNES CEDEX 9, France
E-mail: julien.duchene@intradef.gouv.fr, colas.le-guernic@intradef.gouv.fr

C. Le Guernic
Laboratory High Security, INRIA team TAMIS, Rennes, France
E-mail: colas.le-guernic@inria.fr

J. Duchêne, E. Alata, V. Nicomette and M. Kaâniche
LAAS-CNRS, Univ. de Toulouse, CNRS, INSA, Toulouse, France
E-mail: jduchene@laas.fr, ealata@laas.fr, nicomett@laas.fr, kaaniche@laas.fr

are not sufficiently explored so far and that require further development in the future.

Keywords Reverse engineering · Protocol inference · Data structure inference · Network trace analysis · Binary application analysis

1 Introduction

Communication protocols allow several components to exchange messages in a consistent way. Protocols are widely used in networks and telecommunications domains. A protocol may be published in an open standard or proprietary, thus unknown from the final users. Protocol reverse engineering is mainly useful in this second case, in the context of undocumented and non-standardized closed protocols.

Reverse engineering of protocols consists in deriving a model of the communications established between several components that implement this protocol, without any *a priori* knowledge of this protocol.

The Samba project is a popular example of protocol reverse engineering projects [42]. It offers an open-source implementation of *SMB/CIFS* protocols for Linux clients, enabling Linux and Windows systems to inter-operate. At the beginning of the project, in 1992, it was mainly based on manual reversing, a tricky and time consuming work, whose success is tightly linked to the skills of the analyst. Moreover, keeping pace with a protocol evolutions was a real challenge.

Supporting **interoperability** is not the only motivation for using reverse engineering. **Simulation of network protocols** [31,22,41,11] is another application domain. A network simulator is useful to quickly prototype some specific tests of a protocol whereas performing such tests on a real implementation may be tedious and in some cases practically impossible. It is also useful to carry out some statistical experiments and sensitivity analyses. Finally, a network simulator may replay, in various environments, network traces and possibly adapt them. Replaying network packets is relevant *e.g.*, to analyze a network attack or to develop honeypots that can interact with attackers in order to record and analyze their behavior.

Software security audits is another relevant application domain of protocols reverse engineering [26,13,23]. This application domain is closely linked to the previous one but its main goal differs: a component is solicited under various scenarios to check whether it correctly handles communications whatever their context. Thus, a model of the protocol may be used in order to develop smart fuzzers useful for testing the robustness of the protocol implementation. These fuzzers can generate messages towards a component by relaxing some constraints on some message fields. Messages that trigger a vulnerability can be used to build signatures integrated in a network intrusion detection system.

Malware protocol analysis [8,10,14] also relies on protocol reverse engineering. Indeed, a lot of malware, such as bots for example, use network protocols to communicate with their Command & Control server. Reverse

engineering these protocols is useful to identify some crucial information regarding the location of the botnet master, a date, an imminent attack, attack targets, *etc*, and as a consequence, allows to anticipate an attack occurrence and react accordingly.

Finally, protocol reverse engineering can also be used to support **network protocol conformance testing**. It consists in checking whether a software correctly implements a network protocol whose specification is known. Reverse engineering enables to get a model of the protocol, from the implementation of the software under study and to check whether this model is compliant with the specification of the protocol or not.

During the last decade, several protocol reverse engineering tools have been developed. A brief summary of the state of the art is reported in [32], by Li and Chen in 2011. However, the review presented in this paper is incomplete and does not provide a comparative and synthetic analysis of the different approaches developed so far. The state of the art proposed by Narayan *et al.* [37] in 2015 is more complete and presents recent tools developed for reverse engineering. Compared to these papers, our contributions consist in first discussing the main challenges that have to be addressed by protocol reverse engineering tools, then reviewing how these challenges are tackled by each tool. A classification is then proposed providing a synthetic comparative analysis of 19 state of the art tools. This analysis also highlights the open issues that still need to be addressed. In addition, compared to [37], we extend the scope of the study to tools aimed at reverse engineering complex data structures.

The different tools surveyed in this paper are discussed according to the techniques used for the inference of message format and/or the grammar of the protocol. The inference process is based on the analysis either of network traces (called network inference) or on application execution traces (called application inference). Our classification also distinguishes whether the inference relies on a passive approach (i.e., using an initially observed data set) or on an active approach (i.e., the observations are based on a controlled execution of the system). In particular, the following conclusions have been derived from the comparative analysis of 19 protocol reverse engineering tools developed since 2004:

1. Protocol reverse engineering tools based on network traces have been subject to a large amount of research with significant advances in this area. Automated approaches are now available to support the various steps for the reverse engineering process: from data collection up to the generation of model describing the protocol.
2. Protocol reverse engineering tools based on network traces generally focus of the classification of protocol messages and on the inference of a grammar of the protocol. On the other hand, protocol reverse engineering tools based on application execution traces mainly address the inference of the format of the messages exchanged by the application, without initially classifying the messages using e.g., alignment techniques.

3. New active approaches have been developed to perform both message format and protocol grammar inference using both network and application inference.
4. Since 2010, an increasing research effort targeted the reverse engineering of complex data structures, e.g. to improve the debugging of binary applications.

The paper is organized as follows. Section 2 presents the terminology associated to protocol reverse engineering. Section 3 is dedicated to the different challenges. The evolution and the comparative analysis of protocol reverse engineering tools and associated techniques are detailed in Section 4. Section 5 provides a classification of these tools. Finally, Section 6 proposes some perspectives to this research work.

2 Terminology

Several studies [22,41,13,4,21,5] propose definitions of important notions for protocol reverse engineering. These terms aim at defining both the components that are involved in the communication as well as the structure of the communication and the exchanged messages. To our knowledge, there is no general consensus on these definitions. In the following, we define the main concepts used in this paper, most of them correspond to the most frequently used terms in the publications cited above.

A *protocol* is composed of *messages*, which correspond to protocol data units (*PDU*). The messages are grouped into *classes of messages*, identified by a *type*. A message is composed of *fields*, encoded according to a *format*. Messages are exchanged according to a *grammar*. Let us note, that, in the rest of the paper, we consider that a data structure, as used in various applications, also corresponds to a message format.

To illustrate these terms, let us consider the example of the TCP protocol for the establishment of a connection, which uses the three classes of messages: {`Syn, Syn-Ack, Ack`}. The grammar of the protocol stipulates that, when component A establishes a connection with component B: 1) A must send the `Syn` message, 2) B must respond with the `Syn-Ack` message, and 3) A must finalize the establishment of the connection with the `Ack` message. The format of `Syn` messages is given in Figure 1. This message class corresponds to a set of messages that are identified by the fact that the *Syn* bit of the `flags` field is set to 1 and the other bits are set to 0, which corresponds to the value 0x2 for the `flags` field. The categorization of a message in the `Syn` class does not depend on the other fields of the message, whose values may change from one communication to another.

Protocol reverse engineering techniques are most of the time classified into two categories: protocol reverse engineering based on the analysis of messages exchanged between two components (which we call *network inference*), or, protocol reverse engineering based on the analysis of the application itself (which we call *application inference*). A sequence of messages exchanged between

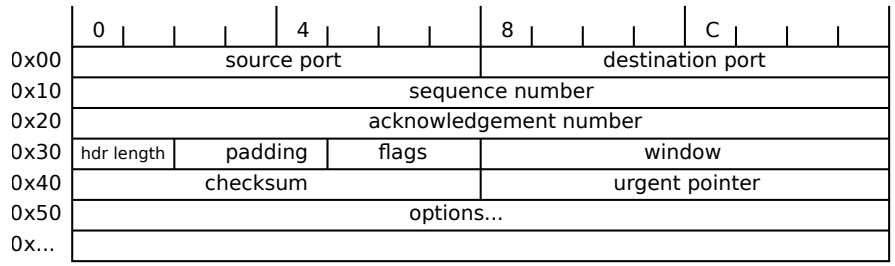

Fig. 1 TCP SYN packet format

two components is called a *network trace*. An application inference consists in analysing the code (source or binary) or in analysing an *execution trace* (a sequence of binary instructions). These two categories of protocol reverse engineering techniques can also be differentiated according to the inference type: passive or active. *Active inference* stimulates the system in order to discover information or to validate previous information. At the opposite, *passive inference* is only based on captured messages, without stimulating the system. Thus, a tool can perform: *network passive inference*, *network active inference*, *application passive inference* or *application active inference*.

3 Protocol reverse engineering challenges

Protocol reverse engineering raises several challenges, not all of them are relevant for each study. This section first presents the different steps of the reverse engineering of protocols. The challenges associated to each step are then discussed.

The preliminary phase of protocol reverse engineering should be dedicated to the identification and characterization of the environment. This phase is important but is out of the scope of this paper. Based on the knowledge of the environment, the analyst can start the observation step, which consists in installing in the environment appropriate means for collecting network or execution traces. The next step consists in sanitizing these traces in order to obtain the relevant messages of the protocol under study. The last step consists in carrying out the inference of the message format or the protocol grammar, from the messages obtained at the previous step. These different steps are executed in an iterative process and usually rely on the expertise and the intuition of the analyst. The success of the reverse engineering activity heavily relies on the intuition and the expertise of the analyst, supported by the efficiency of the tools used to automate the inference. As our paper is focused on protocol reverse engineering tools, we do not identify steps which specifically require an analyst intervention or can benefit from it. As there are few tools targeting earlier steps, this work usually has to be performed manually.

Figure 2 illustrates the protocol reverse engineering process, and highlight the main challenges associated to each step. These challenges are presented in the following subsections.

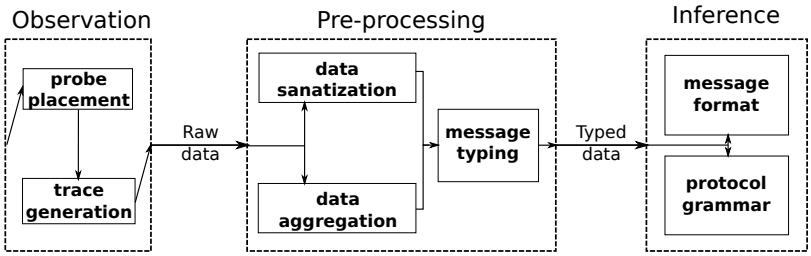

Fig. 2 Protocol reverse engineering steps with associated challenges

3.1 Observation step

The inference is based on the gathering of a set of traces thanks to the observation of a communication channel. Two challenges are associated to this observation step: **probe placement** and **traces generation**.

Probe placement is essential to capture data required for protocol reverse engineering. Challenges for network inference differ from application inference. For network inference, probe placement may become quite difficult if for instance, applications use several protocols to communicate, each protocol on different channels. It is necessary to identify the corresponding channels. Furthermore, one protocol may use different channels, thus, one probe per used channel is required. This induces a synchronization challenge to keep pace with the order of messages. A weak probe placement may lead to message leaks, and thus to an incomplete inference. Some application domains may not suffer from this incompleteness but others where replay is a goal may lead to irrelevant results. If a protocol under study is encapsulated in encrypted channels, the placement of the probes may become tricky. For instance, it may be necessary to implement the probe in the application itself, at the interface of the cryptographic libraries, in order to be able to gather decrypted messages. One solution consists in setting a breakpoint just after the decryption routine, in order to extract the plain message.

This challenge is more difficult for application inference which is based on the analysis of execution traces. Thus, not only the probe must produce the sequence of instructions of the application during messages processing, but also the history of the system state execution. Many solutions exist (debuggers, instumentation of the binary, virtualisation...) but each of these solutions has to address different challenges: the application may have some code protection (code control integrity, anti-debug,...), or simply to deal with a slowdown of the application which may lead to a loss of packets due to timing errors.

The quality of collected (network or execution) traces depends on the probes placement, and also on the observation duration. A too short observation phase may lead to a trace which does not contain a set of all the possible message sequences. Again, such a trace may lead to an incomplete inference. Thus, it is important to identify when the trace is sufficiently rich to be exploitable. This depends on the components involved in the protocol as well as on their communication frequency. In some cases, to obtain more quickly some appropriate traces, the analyst may adopt an active approach, by stimulating the system.

3.2 Pre-processing step

The pre-processing of collected traces is closely linked to the previous step. It may be difficult to achieve if the analyst could not place the probe ideally. In that case, collected traces may contain irrelevant information that have to be filtered out. Furthermore, messages relevant to protocol reverse engineering may be 1) encapsulated in another protocol; 2) split into several packets transferred in multiple exchanges or 3) observed among messages belonging to other protocols.

Thus, a first challenge in this pre-processing step is to correctly sanitize traces in order to reconstruct appropriate messages. This corresponds to **data sanitization**. This step is required for both network and application inference. Nevertheless, the latter has to cope with an additional challenge which is related to the identification in the execution trace of control structures (jumps, loops, tests...), that reflect the encapsulated and iterative structure of some messages.

When messages are transferred in several packets, another challenge consists in aggregating the traces in order to reconstruct the appropriate messages for the analysis. For network inference, if the observation is performed on the TCP layer, thus TCP segments have to be gathered in order to reconstruct messages of an application protocol (ISO 7 layer). For application inference, if the execution trace of the complete message is split in several traces, these traces have to be assembled. Furthermore, (network or execution) traces may have data from several messages, thus, the data associated to each message have to be isolated. These two operations are referred to as **data aggregation**.

After the reconstruction of appropriate messages, they are grouped into classes of messages. This clustering phase is required in order to compare the messages that are semantically identical. This phase consists in finding a function allowing the identification of the type of the message, from a sequence of bytes. This **typing** function must take into account how close a message is to another.

3.3 Inference step

Message format inference is aimed at identifying, from messages of a same type, their structure whereas the **protocol grammar inference** is aimed at reconstructing, from sequences of typed messages, rules describing the exchanges of message classes.

For both message format and protocol grammar inference, it is also important to identify dependencies between the different fields of a message or between messages themselves.

The ultimate goal of protocol reverse engineering process is to obtain a specification for message format or protocol grammar. This specification is represented by a model. It is necessary to choose a sufficiently expressive model so that it can faithfully reflect the original specification. For instance, some complex message formats or protocol grammars may have a tree or recursive structure. Such a structure means that a message (in case of protocol grammar inference) or value of a field (in the case of message format inference) depends on other messages or other fields. This dependency is difficult to express with finite state machine for instance. Thus, the choice of an appropriate model is a major challenge. A bad choice may lead to a specification without any generalization[1] capabilities. Moreover, as observations are most of the time partial, tools have to adopt approaches that are sufficiently robust so that they can generalize the models while minimizing the over-approximation.

4 Chronological review of inference tools

Several significant advances have been achieved in protocol reverse engineering in the past decade. The tools discussed in this paper are based on two major contributions. The first tool *PI Project* [4] used bioinformatics algorithms for inferring message formats from network traces. The second tool, *Polyglot* [13], presented a new dynamic binary analysis technique for inferring message formats from an execution trace generated by an application upon the processing of selected input data. Table 1 gives an overview of the chronological evolution of protocol reverse engineering tools that are categorized according to the type of inference applied. It can be noticed that the majority of the tools focus on message format inference, which is a prerequisite for protocol grammar inference. Furthermore, let us notice that active inference approaches (tools are in bold in the table) have emerged only recently compared to passive approaches.

4.1 Inference tools based on network traces

In [4,3], Beddoe presents ***PI Project***[2], a tool for message format reverse-engineering based on network traces. Protocol grammar inference is not cov-

[1] Model generalization offers the capability to define the format of a class of messages instead of a single instance of a message.

[2] available at `http://www.4tphi.net/~awalters/PI/PI.html`

Table 1 Contribution evolutions of protocol inference, classified by inference type. Tools performing active inference are in bold.

	Network based inference		**Application based inference**	
	Message Format	Protocol Grammar	Message Format	Protocol Grammar
2004	PI Project [4,3]			
2005	ScriptGen [31]	ScriptGen [31]		
2006	**RolePlayer** [22]		FFE/x86 [33] **Replayer** [41]	
2007	Discoverer [21]		Polyglot [13] Rosetta [11]	
2008			AutoFormat [35] Tupni [23] Prospex [49] **ConfigRE** [45]	
2009			ReFormat [48] Dispatcher [10] **Fuzzgrind** [16]	Prospex [20]
2010	ASAP [29]	Veritas [46]	REWARDS [36]	**MACE** [17]
2011	ReverX [2] Netzob [8]	ReverX [2] Veritas [47]	Howard [44]	**MACE** [18]
2012	Netzob [26,25]	**Netzob** [26,25] PRISMA [28]	ARTISTE [9]	
2013			Dispatcher [12]	
2014	**Netzob** [7]	**Netzob** [7]	**AFL** [50]	
2015			ARGOS [51]	

ered. This tool uses the *Needleman & Wunsch* [38] algorithm to align byte sequences of message pairs. The optimal alignment permits to identify common parts of two messages that correspond to the same fields. This result is used to build a tree representing the hierarchical classification of messages based on their similarity degree (most similar messages are located on neighboring nodes in the tree). The UPGMA [39] algorithm is used for this purpose. However, the user has to manually partition the tree to identify message classes. A large majority of *(network trace based analysis)* tools developed later are built on concepts introduced in this tool.

Soon later, ***ScriptGen*** [31,30] was developed by Leita *et al.* to generate a set of scripts that simulate network protocols implemented in a server for the *Honeyd* honeypot. Thus, unlike *PI Project*, *ScriptGen* requires protocol grammar inference. It starts with a pre-processing phase consisting in: i) filtering out irrelevant network packets, keeping only TCP packets corresponding to the protocols of interest, and ii) reconstructing protocol messages from these packets. The partitioning of messages into classes is carried out automatically considering a threshold value. The protocol grammar inference is based on regular languages. The resulting model is a deterministic finite automata that contains only the most frequent responses returned by the server.

At the same period, Cui *et al.* presented ***Roleplayer*** [22] that is aimed at replaying communications in different environments, in particular to study the details of a network attack scenario. Replaying a communication involves the adaptation of a network trace to the characteristics on the environment (e.g., changing the IP address, the sequence number, *etc.*). This requires in particular a precise knowledge of message fields, their conformance to the environment and their dependencies. To address this challenge, *Roleplayer* applies an active inference approach during which communications are initiated by the tool in order to identify such conformance. In addition, by applying an alignment algorithm, the tool is able to learn the dependencies between the fields of a message (e.g., size, cookies values, *etc.*). However, fields encapsulation cannot be identified.

To cope with the difficulties to correctly identify message fields, a new approach by Cui *et al.* is implemented in the tool ***Discoverer*** [21]. Instead of using *Needleman & Wunsch* algorithm, it is assumed that the delimiters of message fields are known (space, CR/LF, tab, comma, *etc.*) which simplifies significantly the problem. After separating message fields, the tool applies a hierarchical and recursive classification on message fields which has the advantage of identifying field encapsulations. Finally, a set of heuristics are applied to identify field dependencies.

While most of protocol reverse engineering research address message format inference, ***Veritas*** [46,47] developed by Wang *et al.* focused on protocol grammar inference. For this tool, only message classes are required, without any information on message format. These classes are identified based on the most frequent byte sequences observed in message headers (the first n bytes). It is assumed that the header of a message is located at the beginning of a message and that this header contains specific byte sequences characterizing

message classes. This assumption, which is generally valid, enables to decrease significantly the amount of data to be processed. This step is followed by protocol grammar inference itself. Authors assume that messages sent depend only on the last message observed and not on the entire history of the communication. The model built by this tool is an automata labelled with the inferred message classes and the probability to obtain such a message. A threshold is used to remove message sequences occurring rarely. It is noteworthy that with this approach, *Veritas* discards sequences that might correspond to unusual data or behavior of the server. In this case, the inferred grammar is partial and does not include all the behaviors included in the observed communications.

Similarly, ***ASAP*** [29] developed by Krueger *et al.* focuses on messages classification rather than their exact format inference. It was developed to improve malware and data collected from honeypots analysis, and to help intrusion detection systems design. Observed messages are split into basic tokens, based on predefined delimiters for text messages or with fixed size tokens for binary messages. This results in the construction of an alphabet that is used to characterize the network payload which is mapped into a vector space. Communication templates (composed of conjunctions of tokens) corresponding to base directions in this vector space are then identified. These templates give insights into the semantics of a typical communication. Beyond the specific techniques used e.g., to identify basic tokens, *ASAP* mainly differs from other tools by the use of an algebraic approach. However, message format is not precise and does not highlight possible links between the identified tokens and their order of occurrence.

Later on, *ASAP* authors extended their tool with capabilities enabling protocol grammar inference associated to a network payload. This resulted in the development of ***PRISMA***[3] [28]. This tool uses globally the same strategy as *ASAP*. However, the algebraic analysis of messages is replaced by a classification of messages based on a distance. A hidden Markov chain is then built by the tool based on the observed sequences of message classes. This model is well suited to describe protocol state changes that do not necessarily result in the emission of messages.

Unlike the previous approaches, ***ReverX***[4] [2] developed by Antunes *et al.* uses the same approach to infer message format and protocol grammar. Similarly to *Discoverer*, this tool splits the messages into fields based on a predefined set of delimiters. An automata with loops is then generated to model possible sequences of fields. Loops are useful to generalize the model and include non observed messages. Such automata models message format. A similar approach is used to build protocol grammar, from the observed sequences of message classes. This tool has interesting features, in particular regarding the possibility to generalize the format of the messages which is a major challenge in protocol reverse engineering. However, the method used to split the messages into fields is not suitable for the analysis of binary messages.

[3] available at `https://github.com/tammok/PRISMA`

[4] available at `https://github.com/jasantunes/reverx`

An additional relevant tool is ***Netzob***[5] that was the subject of several research works with multiple evolutions during the recent years. It has been first developed to support the modeling of botnets [8]. Authors used a variant of the Mealy machine to model protocol grammar, by including information about the time elapsed between two observed messages. Such a grammar is learned using an active approach based on the L^* algorithm. The automata is then made non-deterministic by including the probability to send a message, given a state and a received message.

The techniques used for message clustering and message format inference are presented in [26,25]. In particular, similarly to *PI Project* [4], the *Needleman & Wunsch* algorithm is used for message format inference and classification. Authors introduced in addition an heuristic, named *orphan reduction*, to perform a local alignment of two messages, without considering their entire content. The inference of the message semantics is also performed, based on similar heuristics to those used in *Discoverer* [21]. However, the different attributes inferred are preserved in the message format (e.g., the encoding), as such information can help to refine the inference. Furthermore, protocol grammar inference is based on an active approach, while a passive approach is used for inferring the format of the messages. In [7], major improvements on message format inference results are obtained using an active approach. Specific data are inserted into the application implementing the protocol and are then searched in the messages. In addition, time correlation is used in order to associate messages and application actions. These two means enable them, on one hand, to identify message fields with dynamic content, and on the other hand to associate a semantics to each message class. However, this approach requires some extra work by the analyst which may not be easy to automate. Finally, the paper also presents a comparative analysis of the results with those provided by *PRISMA* [28] and *ScriptGen* [31].

Additional results are presented in [6], including in particular, a model formalization used for messages format and protocol grammar inference, and the development of active techniques for grammar inference. These techniques consist in decomposing the learning of the automata into sub-automata related in the actions performed on the system. This leads to a dramatic reduction of the Mealy machine learning time. It is noteworthy that the tool *Netzob* includes plugins to export the results under different formats, as well as for integrated network communications fuzzing and simulation. This tool with *AFL* one are the only ones to be publically available and maintained nowadays. Furthermore, they both were used for real-world reverse-engineering, on the malware *PHP/bot* for instance.

4.2 Inference tools based on application execution traces

FFE/x86 presented by Lim *et al.* in [33] is the first tool that was designed to infer message format based on application analysis. The objective was to obtain

[5] available at `https://www.netzob.org/`

the format of the data produced by an application by means of a static analysis of the binary code. To our knowledge, *FFE/x86* remains the only tool that adopts this approach. Moreover, it is one of the very few tools (with *Rosetta*, *Replayer* and *Dispatcher*) that perform their analysis based on the outputs of the application. More precisely, it is able to derive, from the graph of function calls, a hierarchical automata representing the format of the messages. Then, two static analysis techniques, *VSA (Value Set Analysis)* and *ASI (Aggregate Structure Identification)*, implemented in the tool, are used in order to obtain an over-approximation of the values of the output data. The result of these analyses is directly included in the model of the format of the messages. The output of the model is presented under the form of a regular expression which is much easier to exploit for a human analyst than a hierarchical automata.

At the same time, Newsome *et al.* published the ***RePlayer*** [41] tool that was designed to replay communications in different environments. In the same way as *RolePlayer* [22], the authors derive from the messages, the constraints identifying the data linked to the environment. To perform the replay, the authors use the constraints formula associated to a solver (STP) in order to generate the messages that would have been produced in a novel environment. However, the message format is not actually inferred. As a consequence, even if the tool performs this task, it is difficult to generalize its usage to other application domains.

The ***Polyglot*** tool published shortly after by Caballero *et al.* [13] adopts a different approach for the inference of messages based on the analysis of execution traces. This approach and the associated analysis techniques were most of the time reused in the other tools. However, the authors do not take into consideration the classification of messages as well as the inference of the protocol grammar. Thus, the tool does not provide a message format per message class. It uses heuristics in order to obtain the message semantic. Overall, they correspond to the identification in the execution trace of patterns that represent that semantic. The first heuristics focuses on the fields `length`. For instance, an execution trace that indicates that the value of a field is used as an upper bound for a second field lets the analyst assume that the first field corresponds to the length of the second one. The second step enables to extract the `delimiters` and the `keywords` of the message. For that purpose, if a byte of the message is compared to a static value, this information is backed-up in a table so-called `token table`. By analyzing this table, the delimiters and the keywords are identified. The final step consists in gathering all this information in order to compose the message format. This approach improves the quality of the message format inference compared to the approaches adopted in the previous publications, namely those based on the identification of keywords and the splitting of the message into fields. However, even in this approach, the message format does not reveal the fields encapsulation. The authors of *Polyglot* have identified an important limitation of their tool: if the message format is not strict, by allowing several delimiters for instance (HTTP allows the use of spaces or tabulations as delimiters), then the execution trace does

not necessarily correspond to the execution patterns used in the heuristics and this flexibility of the format is not inferred in the message format.

The ***Rosetta*** [11] tool of Caballero *et al.* directly derives from the research work of *Polyglot* and *Replayer*. Its main objective, likes *Replayer*, is to replay communications in different environments. The authors extend the identification of fields to the output messages of the application, by analyzing the return values of the system calls that are used to build these output messages. They also identify the fields with `hash` or `checksum` type, even if they do not directly depend on the environment. Finally, they propose a new technique to identify the fields, so-called `session fields`, that indicate an inter-message dependency, by checking if some data of a received message are used to build one of the messages sent.

Three message format inference tools, developed in parallel, build on the research work of *Polyglot* : *AutoFormat*, *Prospex* and *Tupni*. The first one, ***AutoFormat*** [35] from Lin *et al.*, relies on the intuition that different fields of a message are processed by different portions of binary code. Thus, the execution context associated to the processing of each of the fields is unique. This tool splits the messages according to the order in which the different portions of the code are executed, while also taking into account the encapsulation. The message is split according to a tree in which each node represents a field. In that way, the fields may contain sequences of fields. Finally, the tool is able to identify parallel fields (separated by a "|" in BNF notation), which are generally processed in portions of code that share the same execution context history, i.e., the same stack of function calls. *AutoFormat* is the only tool that: i) does not need to analyze loops, and ii) could analyze an application that uses recursive functions to process the fields of a message. Let us note also that this tool introduces a new challenge which consists in evaluating the distance between several portions of code.

In the second tool, ***Prospex*** [49], Wondracek et al. present inference techniques of message format. They extend the analysis of table of tokens of *Polyglot* [13] in order to identify a hierarchy of fields. Moreover, the authors rely on the *Needleman & Wunsch* algorithm, applied to sequences of fields, in order to generalize the formats obtained. This research work was pursued by Comparetti *et al.* in [20] for the inference of protocols grammar. The main contribution concerns the classification of messages using three metrics. The first one, derived from the research work of *Discoverer* [21], defines a distance between two messages based on the alignment of sequences of fields. The second metric measures the similarity between execution traces of messages, taking into account that similar messages are very likely to be processed by the same portions of binary code. Finally, the last metric measures the impact of messages on the system, notably, upon sending of data on the network, opening files, creating directories, *etc.* The proposed message classification algorithm uses the average of these three metrics as a distance between two messages. The grammar of the protocol is represented by a finite state automata which is built with traditional language theory techniques.

After the publication of *AutoFormat*, Cui *et al.* have presented ***Tupni*** [23]. This tool derives from the research works of *Discoverer* [21], in which the authors encountered difficulties to infer the semantic of fields during the reverse of format messages based on network observation. The tool first splits the message into different fields according to the instructions accessing these portions of the message. Then, the fields belonging to a repeated sequence (fields separated by `*` or `+` in BNF notation) are identified from the analysis of loops in the program. The third step identifies the constraints on the fields of the message, either by using the heuristics of *Polyglot* (static values or `length` fields) and *Rosetta* (inter-messages dependency), or by using symbolic predicates similar to those of *Replayer*. The results were generalized through the analysis of several messages. Moreover, this tool can be associated to *ShieldGen* [24] in order to automatically create attack signatures for unknown vulnerabilities. It may also enable the inference of files format by considering files as received messages.

This approach was adopted by ***ConfigRE*** [45] of Wang *et al.*, that consider the configuration file of an application as a message received by this application. This tool can thus be seen as a message format inference tool based on application traces. Its main goal is the reverse of configuration of access control used in an application, for instance the configuration of accesses to the files of an apache server. This tool presents specificities related to its application domain. It first splits the file into fields and tries to infer the semantic of fields. The splitting into fields depends on the known delimiters. Moreover, taking into account the application domain (access control), the tool introduces specific matched delimiters such as: { and } or `(` and `)` or `<Directory>` et `</Directory>`. The authors adopt a specific heuristic of conditional jumps analysis in order to identify these elements. Furthermore, they analyze the interaction between the tainting of the configuration file during its memory loading and the tainting of the different test requests to identify the fields semantic. This tool adopts an active approach to build a semantic tree of the configuration file. Finally, it modifies the fields of the configuration file whose semantic is `permission` thanks to random values, then analyzes the new execution trace and identifies the legitimate values that these fields may take.

At the opposite of the previous tools that take into account nonencrypted protocols only, ***ReFormat*** [48] defined by Wang *et al.* focus on inferring the format of ciphered messages. For that purpose, it must firstly identify the ciphering functions and, in the execution trace, a memory area where the message is deciphered. In that way, it identifies the functions possessing a large number of arithmetic instructions which most of the time are cryptographic functions. In their tool, the authors use the notion of `lifetime` of the data [19], in order to identify the outputs of the cryptographic functions corresponding to the unciphered message. Finally, they use their tool in combination with *AutoFormat* [35] in order to automatically perform message format inference. However, this tool only focuses on inputs data.

Caballero *et al.* continued the research work of *Polyglot* and *Rosetta* by including the enhancements from *Tupni*, *AutoFormat*, *Prospex* and *ReFormat*. The resulting tool is ***Dispatcher*** [10,12]. which enables to identify the encapsulation of fields thanks to a deeper analysis of the table of tokens proposed in *Polyglot*. The different semantics of the inferred fields have also been enlarged. This contribution is detailed in [14] along with a more deeper formalization of the inferred message format. However, this approach still suffers from the following weaknesses: it does not make the distinction between the fields which are useful to the transmission of messages (`length, offset, hash, checksum, delimiters...`) and those which are useful for the analysis (`port, IP address, timestamps, name of files...`). The latter fields directly depend on the application domain while the other ones are common to all inferences. *Dispatcher* was mainly designed to allow the infiltration of botnets. However, in many cases, malware use ciphering and obfuscation. Accordingly, the techniques introduced in *Reformat* [48] have to be extended to identify the ciphering functions and unciphered messages in the bot *MegaD*. These extensions enabled also to address the following challenges: multiple points of deciphering, live deciphering of portions of message, identification of ciphering functions. Nevertheless, the use of *Dispatcher* is limited by the absence of results regarding the classification of messages and the absence of the protocol grammar inference. Also, [10] does not include any information regarding the implementation of these contributions.

To our knowledge, ***MACE*** [17,18] is the only tool actually performing protocol grammar inference for a concolic execution (concrete and symbolic) of the application. Concolic execution enables to investigate the different values that the protocol messages may take and then to build the associated grammar. Firstly, the analysts must supply the abstraction functions of the input and output messages. This problem was partially solved in [18], in which only the abstraction of output messages is actually required. This tool used the L^* [1] algorithm to infer the protocol grammar. The authors analyzed the protocol of *MegaD* botnet and discovered portions of the protocol that were not observable when using usual analysis techniques. Even if this tool does not allow to infer the format of the messages (only their type), it produces instances of concrete messages in order to reach a particular level of execution. This is useful to assess the exploitability of a vulnerability. It also enabled to detect vulnerabilities in different applications (three in Samba 3.3.4, three in Vino 2.26.1 and one in RealVNC 4.1.2).

As presented in section 1, smart fuzzers use a model to perform deep security testing. In this context, ***Fuzzgrind***[6] [16,15] tries to automatically infer a model of messages format with a symbolic execution of the application, using *Valgrind* [40] framework. This symbolic execution allows to register the execution paths of the application. The main assumption is that a class of messages corresponds to one of these paths. As a result, the constraints on the data that guide the execution of a path reflect the format of the class of messages.

[6] available at `https://github.com/Grindland/Fuzzgrind`

The ***AFL***[7] [50] fuzzer infers models based on paths exploration of the binary code. However, in this case, this exploration rely on pratical execution and genetic algorithm to mutate entries. The quality of the mutation is directly derived from the path exploration. This tool is used by a wide community which discovered many different vulnerabilities.

Following these research works, some techniques have been proposed for the inference of data structures, in particular in tools like *REWARDS*, *Howard*, *ARTISTE* and *ARGOS*. Lin *et al.* are the first authors to adapt the message format inference to the data structure inference in the ***REWARDS*** [36,34] tool. It enables to perform *forensics analysis of the memory* as well as fuzzing. For that purpose, the authors performed the same analysis of the execution trace as in *Dispatcher* [10] to extract the data structure format (corresponding to messages for *Dispatcher*) and the semantic of fields. However, they also had to analyze the memory at some specific moments of the execution. So, they extended their analysis by adding a backtrack analysis of the execution trace in order to identify the origin of the elements in memory.

The second tool, ***Howard*** [43,44], from Slowinska *et al.*, has been designed to improve the debugging of binary applications. It is able to store the allocated buffers dynamically, and identifies the way they are accessed in order to identify their splitting into different fields. This step is similar to research works in *FFE/x86* [33] or *Tupni*[23]. However, its dynamic analysis leads to more precise results. Indeed, there is no over-approximation of the values taken by the fields of the structure. Moreover, it includes an improvement of the inference of arrays presented in *REWARDS* and *Polyglot* [13] and it propagates in the same way as *REWARDS* the structures derived from the known functions. Finally, *Howard* can detect and prevent memory corruptions by instrumenting the application, sometimes manually, to secure the pointers dereferencing.

Soon after, in collaboration with the authors of *REWARDS*, Caballero et al. have developed ***ARTISTE*** [9]. This tool extends some of the techniques presented in *Dispatcher* [10] and *REWARDS* [34] for the data structure and message format inference. It can infer complex structures such as trees or double-linked lists. It also includes a novel technique allowing to identify the similar structures allocated in different areas of the program. Finally, it improves the inference speed of simple structures.

Finally, Zeng and Lin have developed ***ARGOS*** [51] in order to improve the understanding and monitoring of operating systems kernels. They used previous techniques presented in *ARTISTE* to propagate the semantic inference for the kernel structures. Moreover, they benefit from the execution context, i.e., the actions performed by the kernel such as the process creation or the reading of files, in order to classify the different objects of the kernel, and give them a comprehensible meaning for a human analyst.

[7] available at `http://lcamtuf.coredump.cx/afl/`

5 Classification

In the previous section, the different tools as well as the associated approaches and methodologies were described chronologically. In this section, we propose a classification of these tools according to four important criteria, that correspond to: 1) the method used to classify messages, 2) the model used to describe message format, 3) the model used to describe protocol grammar and 4) the passive (observing traces without purposely sending inputs to the application) or active (sending inputs to the application and using these specific inputs for the analysis) inference method. This classification should enable to have better insights into the functional capabilities offered by each of these tools comparatively to other existing tools, and also to identify technical areas that are not sufficiently explored so far and that require further development in the future.

The classification of messages is systematically achieved by the tools operating on network traces but more seldom by the tools operating on application execution traces. Messages classification is most of the time done automatically but some tools adopt or refine this classification manually.

Regarding the inference models used, the most important point is their ability to represent all the characteristics of the protocol specification. Message format is often more complex to analyze due to the dependencies that may exist between fields. Three different models can be distinguished:

- *Template*: specific pattern describing the general structure of the format of a message
- *Annoted tree*: a hierachical template represented as a tree in order to describe the encapsulation of fields
- *Finite State Machine (FSM)*: a more powerful representation which can be represented under different forms (Regular Expression, Deterministic Finite Automata, Non Deterministic Finite Automata, Mealy Machines, Hidden Markov Chains, *etc.*).

5.1 Classification of inference tools based on network traces

Tables 2, 3, 4 and 5.1 summarize the different approaches used by protocol reverse engineering tools based on network traces. Network inference is mainly based on: 1) sequence alignment techniques to decompose messages into fields and cluster them into message classes, 2) some heuristics to infer the semantics of the message strings and intra-message dependencies, and 3) an approach based on regular languages for the inference of the protocol grammar. Several tools use similar techniques. Recently, new significant approaches have been adopted by different tools. In particular, the active approach used by *RolePlayer* and *Netzob* allows them to improve the decomposition of messages into fields. Nevertheless, only *ReverX* uses the same technique for message format and protocol grammar inference. Finally, *PRISMA* and *Netzob* use

probabilistic models to take into account non-determinism of some exchanges, mainly due to environment influence on some transactions.

As regards the inference of messages format (Table 3), at the exception of *Veritas* which does not cover this step, overall, the model used for this purpose generally corresponds to templates. Some of the associated inference techniques support model generalization. Unfortunately, such templates are not expressive enough to model important features such as dependencies between messages and encapsulations (the corresponding column of Table 3 reflects this shortcoming). Indeed, to model such characteristics, either the model must be enriched with semantic information, or the tool must use a richer model such as context-free grammars. These grammars are well suited to model encapsulation, such as in XML or JSON type messages for instance. However, the learning complexity for these models is more important than the ones used in the tools so far [27]. *Netzob* tackles this challenge using some heuristics in order to retrieve some specific dependencies. Still, this is an open issue which is worth exploring.

Table 2 Inference tools based on network traces: message classification strategies

Tools	**Analysis method**	**Messages classification technique**
PI Project	Passive	Alignment, manual classification
ScriptGen	Passive	Alignment, automatic classification
RolePlayer	Active	Alignment, automatic classification
Discoverer	Passive	Delimiters, automatic classification
Veritas	Passive	Recursive decomposition in fixed size strings, automatic classification
ReverX	Passive	Delimiters, automatic classification
ASAP	Passive	Delimiters or decomposition, automatic classification
PRISMA	Passive	Delimiters or decomposition, automatic classification
Netzob	Active	Alignment, automatic classification, allow analyst interaction

5.2 Classification of inference tools based on application execution traces

Tables 6, 7, 8 and 9 summarize the different approaches used by protocol reverse engineering tools based on application traces analysis. Tables 10 and 11 identifies the tools performing data structure inference. These tables emphasize that *FFE/X86* is the only tool performing a static analysis of the binary

Table 3 Inference tools based on network traces: message format strategies

Tools	**Message Format**		
	Dependencies between fields	*Model used*	*Model Generalization*
PI Project	no	Template	yes
ScriptGen	no	Template	yes
RolePlayer	no	Template	no
Discoverer	yes without encapsulation	Annotated Tree	yes
Veritas	-	-	-
ReverX	no	FSM	yes
ASAP	no	Template	no
PRISMA	no	Template	no
Netzob	yes	Annotated Tree	yes

Table 4 Inference tools based on network traces: protocol grammar inference strategies

Tools	**Protocol Grammar Model**
PI Project	-
ScriptGen	FSM (Deterministic)
RolePlayer	-
Discoverer	-
Veritas	FSM (Probabilistic/Deterministic)
ReverX	FSM (Deterministic)
ASAP	-
PRISMA	FSM (Hidden Markov chains)
Netzob	FSM (Non Deterministic Mealy machines)

code, while the other tools adopt a dynamic approach, mostly in a passive way. Moreover, tools mostly focus on message format inference and do not consider message classification, which constitutes a limit to perform protocol grammar inference. As a consequence, only *Prospex* and *MACE*, which classify messages, are able to infer protocol grammar. As regards the inference of message format, only *MACE* does not cover this step. The model used by the other tools to support the inference process consists of specific templates or message field annotated trees. The proposed approaches take into account possible dependencies between message fields, and for most of them the analysis is performed based on application inputs.

Let us note finally that an extension of these tools to address new application domains (file format inference, data structure inference) has been proposed in several tools (*Tupni*, *REWARDS*, *Howard*, *ARTISTE* and *ARGOS*).

Obviously, limitations in this area are more important than the ones in the previous case of inference based on network traces, particularly because of the lack of a clear delimitation of the messages (a message can be stored in stack memory, heap memory, in registers, *etc.*). The main limitation is the lack of message classification. This is an interesting perspective to explore in future research.

Table 5 Summary of major contributions for network inference tools and their usage context.

Tool	Contribution	Use case or solved challenge
PI Project	Network inference algorithm based on Needleman & Wunch with manual clustering	Automatic message spliting into fields
ScriptGen	Protocol grammar inference based on language theory, micro/macro clustering	Automatic generation of scripts for honeypots when network trace is not clean
Roleplayer	Active inference of environment dependant fields	Replay network traces in another environment
Discoverer	Automatic hierarchical and recursive classification of messages based on well-known delimiters + fields dependencies identification based on heuristics	Fields encapsulation + Fields dependencies identification such as length fields or session cookies
Veritas	Message clustering based on their header for protocol grammar inference	Reduce amount of data to treat + protocol grammar inference without knowing its message format
ASAP	Clustering based on algebraic metric, message format containing keywords	New metric for message clustering and identification of keywords
PRISMA	Hidden Markov chain for protocol grammar representation	Representation of states which do not lead to the emmission of a message
ReverX	Language theory message format inference	Message format generalisation when it is correctly split into fields
Netzob	Active procotol grammar inference + analyst interraction	Learning of sequences of message using an active approach
Netzob	Active message format inference + time corelation for message clustering	Improving the accuracy of the message format, reductionof the required time for Mealy machine inference, lots of interraction between analyst, application under study and Netzob

Table 6 Inference tools based on applications execution traces: analysis method and message classification strategies

Tools	**Analysis method**	**Message classification technique**
FFE/x86	static	-
Replayer	active/dynamic	-
Polyglot	passive/dynamic	-
Rosetta	passive/dynamic	-
AutoFormat	passive/dynamic	-
Prospex	passive/dynamic	Alignment and application behavior
Tupni	passive/dynamic	-
ConfigRE	active/dynamic	-
Dispatcher	passive/dynamic	-
MACE	active/concolic	Manual abstraction of produced messages
Fuzzgrind	active/symbolic	-
AFL	active/dynamic	-

Table 7 Inference tools based on execution application: message format inference strategies

Tools	**Message Format**			
	Inputs or *Outputs*	*Dependencies between fields*	*Model used*	*Model generalization*
FFE/x86	Outputs	yes	FSM (Regular expression)	yes
Replayer	Both	yes	Template	no
Poloyglot	Inputs	yes	Template	no
Rosetta	Both	yes	Template	no
AutoFormat	Inputs	yes	Annotated tree	yes
Prospex	Inputs	yes	Annotated tree	yes
Tupni	Inputs	yes	Annotated tree	yes
ConfigRE	Inputs	yes	Annotated tree	yes
Dispatcher	Both	yes	Annotated tree	no
MACE	-	-	-	-
Fuzzgrind	Inputs	yes	No visualization	-
AFL	Inputs	yes	Interractive model of memory representation	yes

6 Conclusion

In recent years, many protocol reverse engineering tools have been developed. They have been applied to different application domains and have been extended to infer complex data structures in the most recent work. While the pre-processing of collected traces still requires a significant investment of the analyst, the inference step itself is increasingly automated and no longer requires human intervention.

Research is still focused on the inference of the message format. Indeed, there are a few tools that infer the protocol grammar and most of these tools are based on network traces analysis. This is due to a large volume of complex data to be processed in order to infer the protocol grammar based on the

Table 8 Inference tools based on execution applications traces: protocol grammar inference strategies

Tools	Protocol Grammar Model
FFE/x86	-
Replayer	-
Polygloy	-
Rosetta	-
Rosetta	-
AutoFormat	-
Prospex	FSM (Deterministic)
ConfigRE	-
Dispatcher	-
MACE	FSM (Deterministic Mealy machines)
Fuzzgrind	-
AFL	-

analysis of application traces. In addition, some message formats and protocol grammars may be very complex to analyze due to encapsulation and strong dependencies between fields. However, the models used to infer the message format and the protocol grammar are equivalent to finite automata and they may not be expressive enough to represent all the characteristics of the original specification.

While significant advances have been achieved so far in the area of protocol reverse engineering, there are still some open questions that need to be further explored to address the challenges discussed in Section 3. Furthermore, it would be interesting to address the inference of more complex data such as algebraic grammars or to identify unencrypted data when parsing messages. For this last challenge, other approaches based on the measurement of the entropy of the data could be explored. Finally, a large majority of the studied tools rely on message clustering techniques. Such techniques have been presented in these different tools, but the classification of messages has not been considered as a relevant challenge by these tools up to now. We believe that a comparative study of the various clustering techniques could greatly improve the protocol reverse engineering methodology.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2), 87–106 (1987). DOI 10.1016/0890-5401(87)90052-6
2. Antunes, J., Neves, N., Verissimo, P.: Reverse Engineering of Protocols from Network Traces. In: 2011 18th Working Conference on Reverse Engineering (WCRE), pp. 169–178. IEEE, New York, NY, USA (2011). DOI 10.1109/WCRE.2011.28
3. Beddoe, M.: Network Protocol Analysis using Bioinformatics Algorithms. `http://www.4tphi.net/~awalters/PI/pi.pdf` (2004)
4. Beddoe, M.: Protocol Informatics Project. `http://www.4tphi.net/~awalters/PI/PI.html` (2004)
5. Bohlin, T., Jonsson, B.: Regular Inference for Communication Protocol Entities. Technbical Report 2008-024, Department of Information Technology, Uppsala University, Uppsala University, Sweden (2008)

Table 9 Summary of major contributions for applicative inference tools and their usage context.

Tool	Contribution	Use case or challenge
FFE/x86	Binary static analysis from a synchronisation point	Inference of produced messages when application execution is not available
RePlayer	Replay network traced by identifying constraints dependent on the environment	Replay network trace in another controlled environment
Polyglot	execution trace analysis to infer message format of received messages	Message format inference when a client is available
Rosetta	Previous work *Polyglot* extended to produced messages: system call API backward analysis	message format inference of produced messages
AutoFormat	Message spliting based on code location execution	Message treated by a recursive function + new message spliting technique
Prospex	Needleman & Wunsch algorithm used for message format generalisation + interraction between application/system used as metric for message clustering	message format generalisation + new metric for mesasge clustering
Tupni	Previous work integration + generalisation of model by multiple message analysis	Message format generalisation
ConfigRE	Dynamic active analysis of file treatment	Configuration file inference containing apaired delimiters
ReFormat	Automatic cryptographic interface identification by arythmetic instruction analysis	Crypted messages inference when a client is available
Dispatcher	*ReFormat* extension to generated messages + message format model formalisation	Botnet (agobot) automatic infiltration: multiples cryptographic interface, no server side access for message
MACE	Concolic execution for path exploration	Automatic path exploration in the code treatment of an application inputs
Fuzzgrind	Symbolic execution for path exploration	Automatic path exploration in the code treatment of an application inputs
AFL	Genetic algorithm path exploration	Automatic path exploration in the code treatment of an application inputs

6. Bossert, G.: Exploiting Semantic for the Automatic Reverse Engineering of Communication Protocols. phdthesis, Suplec (2014)
7. Bossert, G., Guihery, F., Hiet, G.: Towards automated protocol reverse engineering using semantic information. In: Proceedings of the 9th ACM symposium on Information, computer and communications security, pp. 51–62. ACM, Kyoto, Japan (2014). DOI 10.1145/2590296.2590346
8. Bossert, G., Hiet, G., Henin, T.: Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion Detection Systems. In: 2011 Con-

Table 10 Tools inferring data structures

Tools	Analysis Method	Classification based on :	Structure inference	
			Type	*Generalization*
REWARDS	passive dynamic	Execution context	simple	no
Howard	passive dynamic	-	simple	no
ARTISTE	passive dynamic	Execution context	recursive	yes
ARGOS	passive dynamic	-	kernel	no

Table 11 Summary of major contributions for data structure inference tools and their usage context.

Tool	Contribution	Use case or challenge
REWARDS	Data structure inference by backward analysis from well-known API	Forensic analysis of an execution trace
Howard	Automatic typing of allocated simple data structure	automatic typing of dynamically allocated structures
ARTISTE	Dereferenced pointers analysis in an execution trace	Complexe data structure inference: chained lists, trees, tables...
ARGOS	System call analysis for automatic classification and analysis of dynamically allocated structures in Linux kernel	Help a human to understand and analyse how kernel structures are structured and accessed

ference on Network and Information Systems Security (SAR-SSI), pp. 1–8. IEEE, La Rochelle, France (2011). DOI 10.1109/SAR-SSI.2011.5931397
9. Caballero, J., Grieco, G., Marron, M., Lin, Z., Urbina, D.: ARTISTE: Automatic Generation of Hybrid Data Structure Signatures from Binary Code Executions. Technical Report TR-IMDEA-SW-2012-001, IMDEA Software Institute, Madrid, Spain (2012)
10. Caballero, J., Poosankam, P., Kreibich, C., Song, D.: 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, New York, NY, USA (2009). DOI 10.1145/1653662.1653737
11. Caballero, J., Song, D.: Rosetta: Extracting protocol semantics using binary analysis with applications to protocol replay and NAT rewriting. Technical Report CMU-CyLab-07-014, Carnegie Mellon University, Pittsburgh, USA (2007)
12. Caballero, J., Song, D.: Automatic protocol reverse-engineering: Message format extraction and field semantics inference. Computer Networks **57**(2), 451–474 (2013). DOI 10.1016/j.comnet.2012.08.003
13. Caballero, J., Yin, H., Liang, Z., Song, D.: 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, New York, NY, USA (2007). DOI 10.1145/1315245.1315286
14. Caballero Bayerri, J.: Grammar and model extraction for security applications using dynamic program binary analysis. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (2010)

15. Campana, G.: Fuzzgrind : un outil de fuzzing automatique. In: Symposium sur la Scurit des Technologies de l'Information et de la Communication, SSTIC. SSTIC, Rennes, France (2009)
16. Campana, G.: Fuzzgrind: an automatic fuzzing tool (2009)
17. Cho, C.Y., Babi , D., Shin, E.C.R., Song, D.: Inference and Analysis of Formal Models of Botnet Command and Control Protocols. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10, pp. 426–439. ACM, New York, NY, USA (2010). DOI 10.1145/1866307.1866355
18. Cho, C.Y., Babi, D., Poosankam, P., Chen, K.Z., Wu, E.X., Song, D.: MACE: model-inference-assisted concolic exploration for protocol and vulnerability discovery. In: Proceedings of the 20th USENIX conference on Security, SEC'11, p. 19. USENIX Association, Berkeley, CA, USA (2011)
19. Chow, J.: Understanding Data Lifetime. Ph.D. thesis, Stanford University, Stanford, CA, USA (2006). AAI3219247
20. Comparetti, P., Wondracek, G., Kruegel, C., Kirda, E.: Prospex: Protocol Specification Extraction. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 110–125. IEEE, Berkeley, USA (2009). DOI 10.1109/SP.2009.14
21. Cui, W., Kannan, J., Wang, H.J.: Discoverer: automatic protocol reverse engineering from network traces. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, SS'07, pp. 14:1–14:14. USENIX Association, Berkeley, CA, USA (2007)
22. Cui, W., Paxson, V., Weaver, N., Katz, R.H.: Protocol-independent adaptive replay of application dialog. In: Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (2006). URL `http://research.microsoft.com/apps/pubs/default.aspx?id=153197`
23. Cui, W., Peinado, M., Chen, K., Wang, H.J., Irun-Briz, L.: 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, New York, NY, USA (2008). DOI 10.1145/1455770.1455820
24. Cui, W., Peinado, M., Wang, H., Locasto, M.: ShieldGen: Automatic data patch generation for unknown vulnerabilities with informed probing. In: IEEE Symposium on Security and Privacy, 2007. SP '07, pp. 252–266. IEEE, Oakland, USA (2007). DOI 10.1109/SP.2007.34
25. Guihery, F., Bossert, G.: The future of protocol reversing and simulation applied on ZeroAccess. In: 29C3: 29th Chaos Communication Congress '12. C-3, Hambourg, Germany (2012)
26. Guihery, F., Bossert, G.: Netzob : un outil pour la rtro-conception de protocoles de communication. In: Symposium sur la Scurit des Technologies de l'Information et de la Communication, SSTIC. SSTIC, Rennes, France (2012)
27. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)
28. Krueger, T., Gascon, H., Krmer, N., Rieck, K.: Learning Stateful Models for Network Honeypots. In: Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence, AISec '12, pp. 37–48. ACM, New York, NY, USA (2012). DOI 10.1145/2381896.2381904. URL `http://doi.acm.org/10.1145/2381896.2381904`
29. Krueger, T., Krmer, N., Rieck, K.: ASAP: Automatic Semantics-Aware Analysis of Network Payloads. In: C. Dimitrakakis, A. Gkoulalas-Divanis, A. Mitrokotsa, V.S. Verykios, Y. Saygin (eds.) Privacy and Security Issues in Data Mining and Machine Learning, no. 6549 in Lecture Notes in Computer Science, pp. 50–63. Springer Berlin Heidelberg, Barcelona, Spain (2010). DOI: 10.1007/978-3-642-19896-0_5
30. Leita, C.: SGNET : automated protocol learning for the observation of malicious threats. Ph.D. thesis, Universit de Nice (2008). URL `http://www.eurecom.fr/publication/2709`
31. Leita, C., Mermoud, K., Dacier, M.: ScriptGen: an automated script generation tool for Honeyd. In: Computer Security Applications Conference, 21st Annual, pp. 12 pp.–214. IEEE, Tucson, USA (2005). DOI 10.1109/CSAC.2005.49
32. Li, X., Chen, L.: A Survey on Methods of Automatic Protocol Reverse Engineering. In: 2011 Seventh International Conference on Computational Intelligence and Security (CIS), pp. 685–689. IEEE, Hainan, China (2011). DOI 10.1109/CIS.2011.156

33. Lim, J., Reps, T., Liblit, B.: Extracting Output Formats from Executables. In: 13th Working Conference on Reverse Engineering, 2006. WCRE '06, pp. 167–178. IEEE, Benevento, Italy (2006). DOI 10.1109/WCRE.2006.29
34. Lin, Z.: Reverse Engineering of Data Structures from Binary. Ph.D. thesis, PURDUE UNIVERSITY, West Lafayette, Indiana (2011)
35. Lin, Z., Jiang, X., Xu, D., Zhang, X.: Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (2008)
36. Lin, Z., Zhang, X., Xu, D.: Automatic Reverse Engineering of Data Structures from Binary Execution. In: Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (2010)
37. Narayan, J., Shukla, S.K., Clancy, T.C.: A survey of automatic protocol reverse engineering tools. ACM Computing Surveys (CSUR) **48**(3), 40 (2015)
38. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology **48**(3), 443–453 (1970). DOI 10.1016/0022-2836(70)90057-4
39. Nei, M., Tajima, F., Tateno, Y.: Accuracy of estimated phylogenetic trees from molecular data. Journal of Molecular Evolution **19**(2), 153–170 (1983)
40. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: J. Ferrante, K.S. McKinley (eds.) Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007, pp. 89–100. ACM (2007). DOI 10.1145/1250734.1250746. URL `http://doi.acm.org/10.1145/1250734.1250746`
41. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: automatic protocol replay by binary analysis. In: Proceedings of the 13th ACM conference on Computer and communications security, CCS '06, pp. 311–321. ACM, New York, NY, USA (2006). DOI 10.1145/1180405.1180444
42. Samba Team: Opening windows to a wider world. `http://www.samba.org`
43. Slowinska, A., Stancescu, T., Bos, H.: Dynamic data structure excavation. Technical Report IR-CS-55, Vrije Universiteit Amsterdam, Amsterdam, Danemark (2010)
44. Slowinska, A., Stancescu, T., Bos, H.: Howard: A Dynamic Excavator for Reverse Engineering Data Structures. In: Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (2011)
45. Wang, R., Wang, X., Zhang, K., Li, Z.: Towards Automatic Reverse Engineering of Software Security Configurations. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS '08, pp. 245–256. ACM, Limerick, Ireland (2008). DOI 10.1145/1455770.1455802
46. Wang, Y., Zhang, Z., Guo, L.: Inferring Protocol State Machine from Real-World Trace. In: S. Jha, R. Sommer, C. Kreibich (eds.) Recent Advances in Intrusion Detection, no. 6307 in Lecture Notes in Computer Science, pp. 498–499. Springer Berlin Heidelberg, Ottawa, Canada (2010). DOI: 10.1007/978-3-642-15512-3_32
47. Wang, Y., Zhang, Z., Yao, D.D., Qu, B., Guo, L.: Inferring Protocol State Machine from Network Traces: A Probabilistic Approach. In: J. Lopez, G. Tsudik (eds.) Applied Cryptography and Network Security, no. 6715 in Lecture Notes in Computer Science, pp. 1–18. Springer Berlin Heidelberg, Nerja, Spain (2011)
48. Wang, Z., Jiang, X., Cui, W., Wang, X., Grace, M.: ReFormat: Automatic Reverse Engineering of Encrypted Messages. In: M. Backes, P. Ning (eds.) Computer Security ESORICS 2009, no. 5789 in Lecture Notes in Computer Science, pp. 200–215. Springer Berlin Heidelberg, Saint Malo, France (2009)
49. Wondracek, G., Comparetti, P.M., Krgel, C., Kirda, E.: Automatic network protocol analysis. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (2008). DOI http://www.isoc.org/isoc/conferences/ndss/08/papers/13_automatic_network_protocol.pdf
50. Zalewski, M.: American fuzzy lop. `http://lcamtuf.coredump.cx/afl/technical_details.txt`
51. Zeng, J., Lin, Z.: Towards Automatic Inference of Kernel Object Semantics from Binary Code. In: 18th International Symposium, RAID 2015, vol. 9404, pp. 538–561. Springer, Kyoto, Japan (2015). DOI 10.1007/978-3-319-26362-5