Towards Automated Protocol Reverse Engineering Using Semantic Information

Georges Bossert*† Frédéric Guihéry* Guillaume Hiet†

*AMOSSYS
Rennes, France
first.last@amossys.fr

†SUPELEC
Cesson-Sévigné, France
first.last@supelec.fr

ABSTRACT

Network security products, such as NIDS or application firewalls, tend to focus on application level communication flows. However, adding support for new proprietary and often undocumented protocols, implies the reverse engineering of these protocols. Currently, this task is performed manually. Considering the difficulty and time needed for manual reverse engineering of protocols, one can easily understand the importance of automating this task. This is even given more significance in today's cybersecurity context where reaction time and automated adaptation become a priority.

Several studies were carried out to infer protocol's specifications from traces. As shown in this article, they do not provide accurate results on complex protocols and are often not applicable in an operational context to provide parsers or traffic generators, some key indicators of the quality of obtained specifications. In addition, too few previous works have resulted in the publication of tools that would allow the scientific community to experimentally validate and compare the different approaches.

In this paper, we infer the specifications out of complex protocols by means of an automated approach and novel techniques. Based on communication traces, we reverse the vocabulary of a protocol by considering embedded contextual information. We also use this information to improve message clustering and to enhance the identification of fields boundaries. We then show the viability of our approach through a comparative study including our re-implementation of three other state-of-the-art approaches (ASAP, Discoverer and ScriptGen).

Keywords

Protocol Reverse Engineering; Contextual Clustering; Semantic Sequence Alignment

ASIACCS '14 Kyoto, Japan
ACM 978-1-4503-2800-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2590296.2590346.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

1. INTRODUCTION

Reverse engineering (RE) of communication protocols is used in many security contexts to retrieve the specification of undocumented protocols. Indeed, security products such as IDSes, application firewalls or honeypots often need to parse application level protocols to perform their analysis. As a matter of fact, reverse engineering of protocols can be a key aspect in the development of these security products to cover a wider range of protocols.

In addition, prior knowledge of protocol specification is highly useful to evaluate both the compliance and robustness of security products. To perform such evaluation security auditors need to generate realistic traffic that contains legitimate and malicious patterns. Tools used to simulate this network activity should therefore precisely implement the targeted protocols. For example, vulnerability analysis and fuzzing tools can leverage protocol specifications to introduce invalid and unexpected patterns (*e.g.* integer or buffer overflows, wrong state transitions, *etc.*) in messages sent to a targeted application. This approach, also called *smart-fuzzing*, requires an accurate understanding of the message format and protocol state machine in order to obtain useful results. Protocol RE is also highly valuable in the context of malware analysis. Indeed, in order to identify leaked information and third party controllers (*e.g.* C&C botnet servers), and in the aim of taking down the latest without getting detected, malware analysts have to understand protocol internals and behaviour.

For all these needs, *i.e.* parser generation, compliance and robustness audit of network security products and malware analysis, our work aims at enhancing automated protocol RE. More precisely, in this article, we improve the reverse engineering of undocumented protocols relying on protocol traces. Unlike previous work, we use contextual information and their semantic definition as a key parameter in both the processes of message clustering and field partitioning. We also detect complex linear and nonlinear relationships between fields value, size and offset using correlation-based filtering. Besides, our multi-step pre-clustering phase reduces the required computation time of the main clustering phase. Furthermore, the experiments we conducted demonstrate that our solution is effective even with few samples.

We have implemented our approach in an open-source protocol RE framework we made available. We successfully applied our approach on a representative set of different text and binary protocols traces. We compared the results obtained using our approach with those of three state-of-the-art tools in the field of trace based protocol inference: ASAP [13], ScriptGen [16] and Discoverer [6]. We conducted experiments on well-known protocols such as FTP and SAMBA. We also compared the effectiveness of these different approaches on two unknown protocols: 1) the P2P protocol used by the ZeroAccess botnet and 2) the protocol used in a publicly available VoIP Commercial product. In summary, this paper makes the following contributions:

- We propose a new trace-based approach to infer protocol vocabulary. It relies on different pre-computing steps that enhance format inference and message clustering.
- We extend the Needleman & Wunsch [19] sequence alignment algorithm to leverage message semantic definition.
- We propose a correlation-based approach to automatically infer relationships between message fields.
- We present the results of an experimental comparative study. We compared our work with three other state-of-the-art solutions. reverse engineering framework.

The remainder is structured as follows: we detail our problem scope and the related work in section 2 and our approach as well as novel techniques for the extraction of protocols specifications in section 3. Section 4 is dedicated to our comparative study. Finally, section 5 concludes this paper.

2. PROBLEM SCOPE & RELATED WORK

A communication protocol defines the rules allowing one or more entities (or actors) to communicate [11]. Its specification are mostly made of 1) the vocabulary, *i.e.* the set of messages and their format; 2) the grammar, which defines all the rules of procedure that represents the valid sequences of messages, usually modelled as state machines. Subsequently, inferring a protocol requires learning both its vocabulary and its grammar. The RE framework we developed covers these two aspects of protocol inference. However, we focus in this article on the first problem, *i.e.* improving the message format reverse engineering.

Previous works on message format reverse engineering fall into two families depending on whether they analyse an implementation of the protocol [3, 4, 5] or rather some communication samples [16, 2, 6, 13, 27, 26, 12]. The first approach infers message format by monitoring a given implementation to analyse how it parses and generates these messages. This requires to instrument or debug the software implementation. This solution cannot be easily automated, mostly due to its intrinsic complexity. Besides, such automation seems hardly possible if counter-measures such as obfuscation, code compression, anti-debugging or anti-instrumentation solutions are used. We thus focus on the second approach, *i.e.* trace-based approach. However, we do not consider these two families as completely orthogonal and foresee to combine our approach with dynamic binary analysis.

Proprietary protocols, and more specifically malware protocols, can use cryptography to protect message exchanges. Such mechanisms may limit the capabilities of protocol inference algorithms. Some solutions exist to bypass existing cryptographic methods by extracting clear text before encryption [14, 28]. However, the difficulty appears when proprietary protocols use strong or custom encryption. We believe this problem is relevant to the cryptanalysis field and cannot be resolved automatically. Indeed, even with access to the process memory, and thus to clear-text buffers, understanding the cryptographic algorithms is still mandatory for traffic simulation and fuzzing. Therefore, we developed our approach for unencrypted communications and let the relevant decryption work for cryptanalysts.

As mentioned in section 1, inferring message format is usually achieved in two main steps: message clustering and fields partitioning. Regarding the first step, previous works tend to split messages in tokens using n-grams [13, 27], delimiters [6] or sequence alignment algorithms [16]. A syntactic message clustering is then performed by comparing the type (binary or ASCII) and/or the value of each token. Message clustering uses the result of this comparison to find similar messages. However, this type of algorithm tends to have a high computational complexity. Pre-clustering steps can reduce the size of input data and thus the execution time. It can also improve the quality of the whole clustering process.

Once similar messages are identified, an alignment algorithm is often used as part of the fields partitioning step. However, current algorithms [6, 13, 16, 27] only rely on syntactic, *i.e.* value and type (ASCII or binary) comparisons, to delimit the static and dynamic parts of the common format of two messages. Following this approach, we cannot dissociate two consecutive static (or dynamic) fields, which have different semantics but share the same type. For instance, in DHCP messages, the Server IP Address (`SIAddr`) and the Gateway IP Address (`GIAddr`) are stored in consecutive dynamic fields that have the same type (binary) but different meanings. Consequently, these two fields must be separated, which is not possible using syntactic approaches.

Moreover, some information of a given semantic, such as an IP address, may be split over different fields by syntactic approaches. For example, if the prefix of observed IP addresses is always the same, *e.g.* addresses belonging to the same network, these approaches split this information into two consecutive fields: the first denoting the network part and the second the machine address in the network. In the worst case, information of different semantics could be associated to the same fields provided they share the same type. For example, a size field getting aligned with an IP address.

To address these issues, inherent to syntactic approaches, we propose an inference algorithm that takes into account the semantic associated with message parts. In particular, our approach does not limit the semantic definition of a message part neither to its data type (ASCII or binary) [6] nor to its appearance probability [26] but instead leverages contextual and environmental information to improve both message clustering and message format inference.

Message format RE must also infer relationships between fields, as they are common in protocols and participate in fields definition. These relationships can be simple as for the HTTP "content-length" field or more complicated as for the DNS "number of questions" attribute which represents

the number of field blocks in the message. Due to computational complexity, existing solutions only consider equality relationships [6, 16]. However, to detect simple and complex relationships, an inference algorithm should consider both linear and nonlinear relationships between fields. Indeed, relationships between fields are also elements of semantic and thus must be considered during sequence alignment and message clustering. Compared to previous works, that do not take into consideration such semantic information while computing the message format, our approach provides more accurate clusters.

Besides, previous works often include an evaluation of their solution but they do not make some experimental comparison of their own contributions against others. Such comparative study is very useful to improve existing solutions by allowing researchers to clearly identify the limitations of each approach. However, such study is difficult to conduct due to the lack of public datasets and because tools used in previous work are not often available. To the best of our knowledge, no previous work has attempted to perform such an experimental comparative study of existing trace-based message format inference approaches.

3. OUR APPROACH

The intuition behind our approach is that message classification and format inference are more effective if they also rely on the semantic definition of messages rather than only on their syntax, *i.e.* the sequence of static and dynamic fields. In our approach, we identify the semantic associated to a given part of a message. We then take this information into account both identify fields boundaries and to cluster messages.

3.1 The Big Picture

As illustrated in figure 1, we take as inputs some traces containing application sessions and use them to infer the protocol specifications. The idea is to consider the semantic definition at every steps of message clustering and partitioning. Thus, our approach relies on several sub-steps participating in pre-clustering steps (*cf.* blocks 1, 2 and 3 in figure 1), the main clustering step (*cf.* blocks 4 and 5), the merging step (*cf.* block 6) and the inter-symbols relationships inference step (*cf.* block 7). The different pre-clustering steps aim at computing homogeneous clusters, which is mandatory to obtain good results during the sequence alignment. However, this approach can generate to many similar clusters. Thus, the goal of the merging step is to combine clusters that share the same format.

3.2 Collecting Semantic Information

Our approach not only requires messages belong to different application sessions but also semantic information associated to these sessions. By semantic information, we refer to 1) the definition of actions performed by the implementation during the capture, each denoting the execution of a specific feature of the protocol, *e.g.* "List Directory", "Read File", "Write File" in an FTP session and 2) the contextual data accessed by the implementation while executing these actions. We use actions in the Sessions Slicing step detailed in section 3.3.1 while we leverage contextual data in our Contextual Clustering step detailed in section 3.3.3. To collect these semantic information, we use three different solutions, depending on the control we have on the collection process.

The first solution applies when we have access to an implementation of the protocol that exposes either a graphical or a command line interface. In that case, we first identify the different input parameters offered by the interface. Based on this, we establish various scenarios each implying different actions to perform with arbitrary predefined parameters. We then capture the communications resulting from the automatic execution of these scenarios either achieved with scripts or with the use of graphical interface testers such as Sikuli [29]. This solution is the most effective to collect both the actions performed by the implementation and the contextual data.

We use the second solution when we have no control over the implementation of the protocol but we can monitor its execution. This situation arises when reversing protocols used by malware. This solution relies on the instrumentation of the OS on which the implementation is executed. In practice, we use sandboxes, such as Cuckoo [10], to capture network traffic in parallel with any useful contextual information related to the application, such as names of accessed files, network parameters or system calls. To retrieve the actions associated to the generated traffic, we identify the different actions based on captured system calls. Indeed, we believe that a signature of successive system and function calls can distinctly represents a specific program action on a system. For example, we have successfully instrumented the Android Dalvik Virtual Machine with the Substrate Framework [8] to intercept and collect, at runtime, any environmental information accessed by an application, such as Android version number, phone contacts, SMS providers. We also monitor specific API calls denoting the execution of actions, *e.g.* the creation of activities, the activation of devices such as GPS or bluetooth, reading and writing personal user information.

If we have no access to the implementation, *e.g.* when only traces are provided by third parties, we have to manually specify contextual information. In addition, environmental information is also automatically extracted from files metadata, such as IP addresses, hostnames and TCP/UDP port number.

3.3 Semantic-based Message Clustering

In this section, we describe the whole process of message clustering. We detail the different steps of clustering. The goal is to compute a common abstraction model, *i.e.* a symbol, for similar messages.

3.3.1 Session Slicing (Step 1)

This first step in our clustering process consists in slicing sessions into frames denoting different actions. We call these frames, action frames. This step is based on the two following heuristics. At first, we believe that initiators, *i.e.* actors sending the first message, and non-initiators of a communication can use two different subsets of the vocabulary. Secondly, we observe that most of the different messages are linked to specific actions. Hence, identifying these action frames allows us to find and pre-cluster messages implied in the same action but captured in different application sessions or at different times in the same session.

To perform this step, message timestamps are compared against the start time of each action. Messages with a timestamp superior or equal to the start time of "action `N`" and

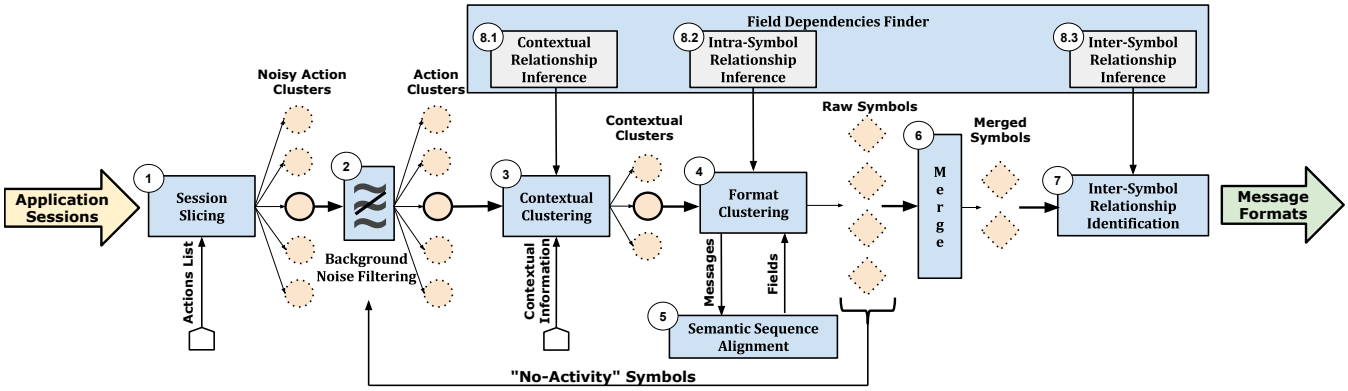


Figure 1: System overview

inferior to the start time of "action `N+1`" are clustered together.

When no information is available regarding the underlying actions performed by the application while traces were captured, we rely on a statistical analysis to detect action frames. We approximate action frame boundaries following the modifications of the inter-arrival time. Originally developed by U. Gargi [9] to cluster collections of pictures following their timestamps, this approach defines the following heuristics: 1) a long interval with no information usually marks the end of an action; and 2) a sharp upward change in the frequency of information inter-arrival time usually marks the beginning of a new action.

Finally, this step produces different *action clusters* for each type of action and each for each type of actor (initiator or not). We compute an additional cluster for messages generated during a no-activity period. This cluster corresponds to an application session explicitly identified. We use this cluster to filter background noise as described in the following section.

3.3.2 Background Noise Filtering (Step 2)

Some messages are not correlated to a particular action and can occur at any time, for example the PING/PONG messages used in IRC protocol. These messages correspond to background noise. As they have their own format, which is generally different from the format of messages generated by user actions, we need to filter them from *action clusters*. To do so, we rely on sent and received messages during a period of no activity.

Filtering background noise is achieved in two steps. First we execute the contextual clustering (step 3) and format clustering (step 4) on messages belonging to the no activity cluster. As detailed in the following sections, these algorithms create a symbol for each type of messages belonging to background noise. In a second step, we filter each *action cluster* using these symbols. If a message can successfully be parsed with a noise symbol, we remove it from the *action cluster*.

3.3.3 Contextual Clustering (Step 3)

Action clusters can still suffer from imprecision, as a single action frame can contain messages of different formats. For instance, a single user action such as the connection to an SMB share directory generates 16 different messages with smbclient[1].

The contextual clustering step refines *action clusters*. The main idea is to regroup messages that embed the same contextual data, such as host addresses, timestamps or usernames. To achieve this, we rely on contextual data collected during the capture or extracted from the capture files metadata. We search occurrences of these data in every messages of a given action frame. We finally subdivide action clusters by grouping messages that share the same sequence of contextual data occurrences.

This contextual information is searched in messages using different encoding (*e.g.* little-endian, big-endian, ASCII, UTF-16) and common transformations (*e.g.* CRC, Gzip, Base64). This step corresponds to the contextual relationship inference (*c.f.* block 8.1 in figure 1) which produces, for each message, a contextual signature, *i.e.* an ordered sequence of contextual information. For example, given two contextual data: an observed hostname "target.org" and the IP address of the implementation "192.168.0.103", it identifies two occurrences in message "1034|.attack192.168.000.103-443target.org".

Sometimes, some part of a given message can correspond to different contextual data. This situation is more frequent with short contextual data. For example, if the contextual data "1034" that denotes the ID of the message is also collected, our contextual relationship inference wrongly identifies it at two different places in the previous message: at the beginning of the message but also straddling the IP address of the implementation.

To address this situation, we use two resolutions scenario. The first applies when one contextual data is entirely embedded in the other. In that case, we give priority to the longest one. The second applies when the occurrences of two contextual data partially overlap. In that case, we use a disjunction in the contextual signature. For example, the contextual signature of the previous message is "ID;IP or ID;HOSTNAME".

We then use a greedy approach to cluster messages together if they share compatible contextual signatures, *i.e.* two signatures are said compatible if they are equal or if one is a possible realisation of the other. For example, the following message "2342|.attack192.168.100.101443example.org" which semantic signature is "ID;IP;HOSTNAME" is a valid realisation of the previous signature.

[1]smbclient is a "client to access SMB/CIFS resources on servers" and is developed by the Samba Team.

Finally, we compute the most precise common signature of each resulting cluster and iterate over its messages to tag their half-bytes. This last step consists in identifying each half-byte that correspond to each part of the contextual signature. In our first example message, we tag the eight first half-bytes with the tag "ID", half-bytes between the 26th and the 60th half-bytes with both tags "ID" and "IP" while the last half-bytes are tagged with "HOSTNAME". These tags are used by the format clustering to promote a semantic alignment between messages.

3.3.4 Format Clustering (Step 4)

Contextual clusters should be refined for two reasons: 1) to manage messages carrying no contextual information; and 2) to dissociate messages that include the same contextual information but have a different format. The format clustering step corresponds to the final stage of classification and is applied on each *contextual cluster*. Unlike the two previous steps, this clustering compares the alignment quality between messages to compute clusters.

We propose to extend both the Needleman and Wunsch (NW) [19] sequence alignment algorithm and the UPGMA [24] hierarchical clustering algorithm. Our modifications take into account the semantic in both the alignment and the clustering phase. In the remainder, we give some details about these modifications.

We first propose an extension of the NW algorithm to produce a semantic-aware common alignment between messages. In fact, NW can be applied on a symbol, which represents the common alignment of a set of messages. In the following, we use the term of message to both refer to messages and symbols. In the original version of NW, the alignment of two messages m_1 and m_2 is performed in two steps. First, a matrix F including a column for each half-byte of m_1 and a row for each half-byte of m_2, is initiated. We denote $m[x]$ the half-byte of index x in message m. This matrix is then filled accordingly to the principle of optimality described by formula (1). It uses a gap penalty d and a similarity function S.

$$F_{i,j} = max(F_{i-1,j-1} + S(m_1[i], m_2[j]), F_{i,j-1} + d, F_{i-1,j} + d) \quad (1)$$

In previous works [2, 6, 16], the similarity function S is reduced to a simple function $v(a, b)$ returning a match or mismatch coefficient, respectively e and f:

$$S(a, b) = v(a, b) = \begin{cases} e, & \text{if } a = b \\ f, & \text{otherwise} \end{cases} \quad (2)$$

We propose to extend this syntactic comparison with the comparison of the semantic definition attached to each half-byte. Hence our function compares the value but also the semantic tags of each half-byte and preserves common semantic information if available. These semantic tags are computed and attached to half-bytes during the Contextual Clustering and every time an intra-symbol relationship is found.

We denote $\psi(a) = \langle T, \phi_a \rangle$, the multiset [25] of semantic tags attached to an half-byte a, with T the set of all semantic types and $\phi_a : T \to \mathbb{N}$, a function returning the multiplicity of a semantic tag in a. For example, $\psi(b) = \{\{IP, IP, Username\}\}$ means that IP and $Username$ semantic tags are attached to half-byte b. In this example the multiplicity of IP is two, *i.e.* $\phi(IP) = 2$. This situation may arise when the same semantic tag corresponds to different types of relationship. For example, an half-byte could correspond to both environmental and application information.

Now, suppose $\psi(a)$ and $\psi(b)$ respectively the multiset of semantic tags attached to half-byte a and b, we denote one includes the other with the relation:

$$\psi(a) \sqsubset \psi(b) \Leftrightarrow \forall e \in T, \phi_a(e) < \phi_b(e) \quad (3)$$

and we define a size function the following way:

$$\overline{\psi(a)} = \sum_{e \in T} \phi_a(e) \quad (4)$$

We compute the similarity between half-bytes a and b by comparing their values and their semantic tags. For the value comparison we keep the original $v(a, b)$ definition while for the semantic comparison we introduce two new semantic match and mismatch parameters: h and g.

Our experimentation has shown best results with the following parameter values: $d = 0$, $e = 5$, $f = -5$ $g = 6 \times e$ and $h = 6 \times f$.

Hence, as described in table 1, our similarity function returns a high score if the semantic tags match but the values differ and on the contrary, returns a low score if the values match but not the semantic tags.

$\psi(a) \cap \psi(b) = \oslash$	$S(a, b) = v(a, b) + h \times \overline{\psi(a)} + h \times \overline{\psi(b)}$
$\psi(a) = \psi(b)$	$S(a, b) = v(a, b) + g \times \overline{\psi(a)}$
$\psi(a) \sqsubset \psi(b)$	$S(a, b) = v(a, b) + g \times \overline{\psi(a)} + h \times \overline{\psi(b) \setminus \psi(a)}$
$\psi(a) \sqsupset \psi(b)$	$S(a, b) = v(a, b) + g \times \overline{\psi(b)} + h \times \overline{\psi(a) \setminus \psi(b)}$

Table 1: Similarity function $S(a, b)$.

Once the matrix F is computed using our new similarity function and following formula 1, a traceback is performed from the maximal score located in entry $F_{|m_1|+1,|m_2|+1}$ until $F_{1,1}$ is reached[2]. A diagonal traceback describes a perfect alignment between the two messages, while a vertical or an horizontal motion implies the addition of gaps in one of the two messages. Such traceback produces two messages m'_1 and m'_2 containing the necessary gaps to align messages m_1 and m_2 under the constraints introduced by their inner syntactic and semantic similarities.

We then execute a pairwise comparison to compute the symbol that corresponds to the common alignment between m'_1 and m'_2. If the two half-bytes are equal we create a static field in the common alignment, if not a gap. We also consider the semantic tags of m'_1 and m'_2 and only propagate the common ones in the newly created field. Finally, we merge consecutive half-bytes that share the same characteristic (type and semantic) to delimit fields.

As illustrated in figure 2, our semantic based alignment preserves the semantic definition while computing field boundaries. In this example, without our solution, email addresses get split among multiple fields and firstnames definition is lost in a bigger dynamic field.

We use this extension of NW in our modification of the UPGMA algorithm. UPGMA is a heuristic clustering algorithm that recursively joins the two nearest clusters. It relies on a matrix, denoting the pairwise similarity of clusters.

In the n^{th} iteration of the algorithm, we try to align each pair of symbols resulting from the $(n-1)^{th}$ iteration using

[2] $|m|$ denotes the number of half-bytes in a message m.

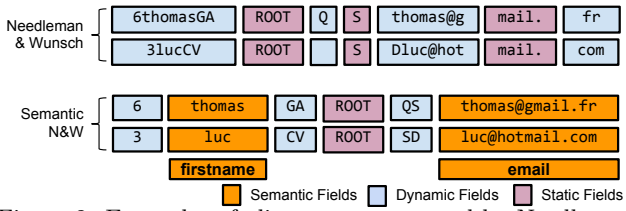


Figure 2: Examples of alignments computed by Needleman & Wunsch and our modified version of this algorithm.

our modified NW algorithm. We also compute a new symbol for each pair of

messages and search for intra-symbol relationships. A semantic tag labeled with the type of the relationship is attached to each corresponding half-byte. More details on the relationship inference are provided in the next section. After that, we compute the quality score of each possible merge using a dedicated function H. We finally merge the two symbols that maximize H into the new corresponding symbol provided by our modified NW algorithm. We stop this iterative merging process when the highest score falls below a specific threshold.

Originally, after each merge, the UPGMA matrix is recomputed using an equation which estimates the similarity of the new clusters based on previous ones. However, because we wanted to keep track of the semantic definition when merging clusters, we modified this original behaviour. In our work, after each round we realign messages participating in the new clusters and recompute a symbol to represent it. We then compute its quality score. It allows us for example, to detect that a semantic field will disappear if we merge two messages.

Hence, our UPGMA algorithm relies on $H(m_1, m_2)$. This function returns the quality score of s_{m_1,m_2}, a symbol representing the alignment of m_1 and m_2. This H function computes the euclidean norm of a vector composed of two measures, $Q(m_1, m_2)$ and $P(m_1, m_2)$:

$$H(m_1, m_2) = \|Q(m_1, m_2), P(m_1, m_2)\| \quad (5)$$

The first measure $Q(m_1, m_2)$ represents the syntactic and semantic similarity of the two messages and its value is extracted from the NW matrix F:

$$Q(m_1, m_2) = F_{|m_1|+1,|m_2|+1} \quad (6)$$

The second measure, denoted $P(m_1, m_2)$, evaluates the proportion of static half-bytes over the number dynamic fields in the symbol resulting from the NW alignment of m_1 and m_2.

3.3.5 *Merging Step & Inter-Symbol Relationship Identification (Steps 4 & 5)*

Format clustering produces *Raw Symbols*, each denoting a possible message format. However, this approach tends to produce a lot of redundant *Raw Symbols* corresponding to the same message format. To address this issue, we added a simple merging step that compares all the computed *Raw Symbols* and merges duplicate ones.

Finally, the last step of our approach consists in identifying inter-symbol relationships. As detailed in section 3.4, it searches for relationships between consecutive messages using a generic relationship inference. The same approach is also used during previous step to find contextual and intra-symbol relationship inside each message. Thus we can identify fields such as sequence ID or cookies. This last step in our approach returns symbols representing different message format, including the definition of their fields.

3.4 Fields Relationships Identification

In this section we present our approach to identify intra-symbol relationships (*i.e.* between fields that pertain to the same symbol) and inter-symbol relationships (*i.e.* between fields that pertain to two consecutive symbols). We consider three steps. At first, we generate a dataset that contains all combinations of field attribute couples. Then, we quickly eliminate bad candidates by means of a correlation approach, and finally we try to qualify the potential relationship that exists between the remaining field couples.

During the first step we generate a dataset with the following attributes for each field: its value, its size and its offset in current message. We then compute each possible combination of attributes couples as, for example, (field1.size, field3.value).

Then, we look in the dataset for correlations by leveraging the Maximal Information Coefficient (MIC) [21]. This coefficient retrieves many types of dependencies, including nonlinear ones. Moreover, it supports noisy datasets. This characteristic is useful as the clustering steps may have erroneously grouped messages of different formats, thus preventing us to find dependencies when looking for exact relationships. In order to differentiate linear from nonlinear dependencies (*e.g.* a size field from a CRC32 field), we combine the MIC score with r, the Pearson product-moment correlation coefficient. As demonstrated in [21], a value of $MIC - r^2$ close to zero indicates a linear dependency, whereas a score close to one tends to point out a nonlinear dependency.

The qualification step takes as input the best couple candidates considering their $MIC - r^2$ scores. We experimentally established that we obtain good results if we select couples between 0.8 and 1 for linear relationships and between 0 and 0.2 for non linear relationships. We do not consider scores between 0.2 and 0.8 as they generally lead to weak results in terms of relationships. We then evaluate each potential field couple under a set of specific relationships, in order to retrieve one that exactly applies, *i.e.* it should be valid for the entire set of messages.

In order to support the wide variety of encoding that exists in real protocols, we take into account different possible encoding and we try different combinations of endianness, signed number representation and byte interpretation (ASCII, decimal, hexadecimal and octal). In the current implementation, we consider the following basic relationships: size field, offset field, cookies and sequence number. We also consider the following complex relationships: SHA-1, CRC32 and the size of a repetition of a particular field or group of fields. The later can be found for example in the P2P ZeroAccess protocol, where a size field specifies the number of peer's IP address fields concatenated in a message.

4. COMPARATIVE STUDY

We evaluate our approach on various protocols and compare our contributions against state-of-the-art approaches. We conducted two different kinds of experiments: 1) on known protocols to compare inferred message formats with

their published specifications and 2) on unknown protocols to evaluate the effectiveness of the different approaches on more operational use cases.

For the first set of experiments, we selected a text protocol, *i.e.* FTP and a binary protocol, *i.e.* SAMBA, both often used in previous experiments [7, 3, 5, 6]. The second set includes two typical use cases of protocol reverse engineering to cover more operational contexts: the P2P protocol used by a recent botnet known as ZeroAccess [23] and a proprietary and undocumented VoIP protocol, Ventrilo[3].

In the remainder, we first give some key insights over the compared tools and then present the datasets, the metrics and the implementations we used in the study. We then conclude with a discussion on obtained results.

4.1 Choice of Compared Tools

Few works have been proposed to reverse protocols using trace-based approaches and among them, only one, ASAP, provides a publicly available implementation. As implementing these tools is time-consuming, we decided to retain the most representative ones while still covering the different types of approach. Thus, Discoverer uses a syntactic alignment approach, ScriptGen uses an inferred automaton and ASAP relies on statistics over message bytes. We consider that other works use the same types of approaches and are less advanced or outdated by the tools we selected. Moreover, the selected tools are often cited in scientific articles of the domain. Though ASAP is less popular, the tool is publicly available and other works that follow the same type of approach are very similar and not more popular.

Among works we excluded from our comparative studies are Roleplayer [7], Veritas [27] and PRISMA [12]. Roleplayer uses the same approach than Discoverer to find dynamic fields in a message. Veritas [27] is another work that we consider similar to ASAP, *i.e.* it uses a statistical analysis to infer protocols. Another work we did not consider in our comparative study is PRISMA [12]. In this extension of ASAP, authors consider the grammar of the protocol in the vocabulary inference process, which is similar to ScriptGen approach. In the following, we give more details about the retained tools.

ASAP, published by T. Krueger *et al.* in [13], focuses on message clustering. ASAP splits messages in tokens and searches for the most representative ones. To do so, they filter out keywords that have extreme (high and low) frequency of appearance. A Non-Negative Matrix Factorisation [15] is then performed to cluster similar messages. Finally, a template is extracted from each cluster to represent the message format associated with each cluster. However, the template is coarse-grained and not precise enough, especially to parse messages.

ScriptGen, developed by C. Leita *et al.* [16], includes features related to the vocabulary and the grammatical inference. It was initially designed to generate honeypot scripts[4]. ScriptGen differs from others because it uses the protocol automaton to identify similar messages. It passively builds an FSM by replaying the various sessions provided in traces. Messages that appear in the same state of the FSM are clustered together. Clusters are then subdivided following two main heuristics: 1) the number of bytes sent in response to a message; and 2) the result of Region Analysis algorithm execution. This algorithm is applied in two steps, first it clusters messages following a UPGMA execution coupled with a sequence alignment algorithm, then it subdivides obtained clusters following messages values.

Discoverer, by W. Cui *et al.* [6] tries to reverse unknown protocols following three main steps: tokenization, recursive clustering and merging. The tokenization process splits messages in ASCII and binary tokens to cluster messages that have the same token structure, *i.e.* the same sequence of token types. Then, the recursive clustering divides obtained clusters by identifying "format distinguisher" fields among them. To mitigate over-classification problems, the last step merges similar message formats by using a type-based sequence alignment.

4.2 Datasets

Our comparative study relies on six datasets: two of them (① and ②) correspond to a well-known text protocol (FTP), two of them (③ and ④) to the well-known SAMBA binary protocol (SMB), one to a P2P botnet protocol (⑤) and one to a typical commercial proprietary product (⑥). Table 2 summarises the different dataset characteristics.

#	Protocol	Source	# Msg	# True Format
①	FTP	Generated	1717	40
②		LBNL	2328	46
③	SMB	Generated	2650	32
④		Company	937	22
⑤	ZAccess	Public	883	4
⑥	Commercial	Laboratory	482	15

Table 2: Summary of datasets used in our comparative study. The first column denotes the dataset identifier. Last columns denote the number of true formats and the number of messages in the dataset.

To compare the best results of each tool, we use two kind of datasets for each known protocol (FTP and SMB): a calibration dataset to empirically compute the optimal parameters of each tool and an evaluation dataset to compare them.

To create the calibration datasets ① and ③, we used the first solution detailed in section 3.2 and wrote client scripts executing various actions with predefined parameters on a server. For instance, the FTP script executes more than 10 different actions, including a connection attempt with a bad password, some directory listing and downloading multiple files. Each calibration dataset includes 20 application sessions containing the same actions but with different contextual parameters, *ie.* usernames, filenames, IP addresses, hostnames. Thus, we annotated the captured pcaps with the executed actions and contextual data used to generate them.

To create the evaluation datasets, we used traces captured in both academic and professional environments. The realistic FTP dataset (②) is a subset of traces published by LBNL [20], collected in an university network. We arbitrary considered the first 1000 packets in three different days of capture (days 10, 11 and 12) to produce a dataset of reasonable size. The second realistic dataset (④) comes from a full day of SMB traffic captured in a company network. Users agreed to participate and behaved in a normal way. We selected the first 1000 packets. Obtained dataset is composed

[3]Ventrilo is a VoIP group voice communications: `http://www.ventrilo.com/`

[4]See Honeyd project: `http://www.honeyd.org/`

of 937 SMB packets covering 22 different true formats. By true format, we hereby refer to the format detailed in protocol specifications.

For anonymity reasons, the LBNL dataset only includes pcap files without a precise definition of the context in which it was captured. Thus, we should have used the last solution proposed in section 3.2 to obtain necessary semantic information. In that case, parameters used on the evaluation datasets would not reflect the same quality as those used on calibration datasets. Returned results would therefore be difficult to interpret as various factors would have influenced them. Thus, to ensure consistency between parameters used for calibration and evaluation, we extracted from evaluation network traces the same types of contextual data than the one we used for calibration. We followed the same approach on the SMB datasets.

Finally, we applied the four approaches on more realistic reverse engineering situations: i) the P2P communication protocol used by ZeroAccess [23] botnet and ii) a subset of the protocol used by a commercial VoIP product. To create the dataset of ZeroAccess traces (⑤), we used the second version of the malware, provided by K. McNamee [18]. We deployed this malware in a confined and controlled network infrastructure. We then allowed our sample to connect with other botnet members through its P2P protocol (used to retrieve the P2P directory). To capture the traffic, we used a network probe implementing the deobfuscation algorithm previously detailed by K. McNamee [18]. The obtained dataset includes 883 messages for four true formats. Regarding contextual data, we used the last solution detailed in section 3.2 and used pcaps meta-data. Even though it does no bring a lot of contextual data, it still provides good results as messages generated by this P2P protocol often denotes network related information, such as the IP addresses of its peers.

For the last dataset (⑥), we considered the protocol of a typical VoIP commercial product. To obtain traces of this protocol, we relied on a freely available implementation of this protocol and automatised its execution. In accordance with the solution we detailed in section 3.2, we selected a subset of the application features and established a scenario of 10 different actions, such as sending text messages, configuring personal data, disconnecting from or connecting to the server. We also arbitrary defined the values of each action parameters and stored them as contextual data. We then played this scenario three times and captured the generated traffic to create a dataset of 482 messages.

To ensure the reproducibility of these experiments, we stored and archived these datasets, but only some of them are made public. Generated datasets (① ③), FTP realistic dataset (②) and ZeroAccess dataset (⑤) are available for download. However, the realistic dataset of SMB packets captured from a company network cannot be published due to embedded sensitive information, and the one extracted from the commercial product cannot be published due to intellectual property restrictions. We also archived all the identified contextual information for each protocol that we summarised in table 5a.

4.3 Metrics

To measure and compare the effectiveness of message formats inference algorithms we need to define metrics. We reviewed all the metrics used in previous works experiments but no consensus emerged. For instance, some works report similar metrics but with different names (*i.e.* [6] and [5]), some compute their own measures [27, 1] and others only use qualitative metrics in their experiments [4]. We select two metrics and propose a new one to cover our needs in the evaluation of message clustering and field partitioning: the correctness, the conciseness and the precision. Correctness and conciseness are both used in [6] and are closed to the metrics used in [17, 5]. Figure 3 illustrates the three metrics we use.

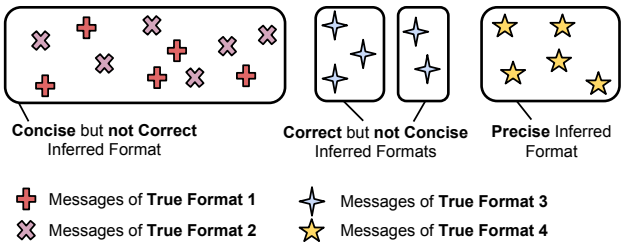


Figure 3: Illustration of our three metrics: conciseness, correctness and precision.

To define the conciseness and the correctness of a clustering algorithm we consider M the set of messages, F_{inf} the set of inferred formats and F_{true} the set of true formats. We also denote the function $I : M \rightarrow F_{inf}$, which defines the inferred format of a message and the function $T : M \rightarrow F_{true}$, which defines the true format of a message. We finally define two functions, $N_{con} : F_{true} \rightarrow \mathbb{N}$ and $N_{cor} : F_{inferred} \rightarrow \mathbb{N}$:

$$N_{con}(f) = |\{I(m)\ \forall m \in M \text{ such that } T(m) = f\}| \quad (7)$$

$$N_{cor}(f) = |\{T(m)\ \forall m \in M \text{ such that } I(m) = f\}| \quad (8)$$

Those two metrics are related to the mapping between true formats and inferred formats. Intuitively, the clustering is correct if it computes homogeneous clusters. It means that every cluster, *i.e.* inferred format, must contain only messages that share the same true format. In this case $N_{cor}(f) = 1$. Heterogeneous clusters decrease the correctness and in this case $N_{cor}(f) > 1$. Conversely, the clustering is concise if each true format is described by at most one inferred format. When messages corresponding to the same true format are clustered into different inferred formats, conciseness decrease and $N_{con}(f) > 1$. The clustering is correct if $N_{cor}(f)$ remains low (ideally equal to one) for a large number of true formats. It is concise if $N_{con}(f)$ remains low (ideally equal to one) for a large number of inferred formats. We thus define Conciseness (respectively Correctness) as the Cumulative Distributed Function of N_{con} (respectively N_{corr}): $Con(n) = P(N_{con} <= n)$ and $Cor(n) = P(N_{cor} <= n)$.

The overall shape of such CDF curves characterises the correctness or the conciseness of a given clustering approach. However, two points of such curves are of particular interests. The first one corresponds to $Cor(1) = p(N_{cor} = 1)$, *i.e.* the proportion of homogeneous clusters and the second one to $Con(1) = p(N_{con} = 1)$, *i.e.* the proportion of true format that correspond to at most one inferred format. We thus expect such values to be high.

Correctness and conciseness must be analysed together to understand the quality of the alignment. For example, an

inference algorithm that classifies every messages into different clusters will be described as very accurate but unconcise. For this reason, we also use ROC curves to compare the different tools, by taking into account $Corr(1)$ and $Con(1)$.

We use an additional metric to evaluate the quality of the classification: the precision. To measure it, we compute the number of inferred formats that perfectly match a true format. We call them precise formats or precise clusters.

To use these metrics, we need to known the true format associated with each message. To identify the true format of well-known protocols, such as FTP and SMB, we used the results of Wireshark. More precisely, we extract the needed information out of a PDML file generated by Wireshark. This file includes the description of all the fields in captured messages. We then use a protocol-specific PDML parser that returns the value of some key fields. We assume that all the messages that share the same key values correspond to the same true format. To select these key fields we refer to the official specifications of the protocols and only select important fields. For instance, for the SMB protocol we consider the value of the "SMB Command" field, the value of optional "subCommands" fields and the value of the "status" field. Unlike previous work [6], we believe optional fields do not participate in the definition of a true format. For the two other protocols, we manually create specific parsers based on previous works published by other researchers that reversed such proprietary protocols.

4.4 Implementations

In this section, we describe the different implementations we developed for the experimentation phase.

Our open source framework for reverse engineering of communication protocols is licensed under GPLv3. Freely available, its sources can be downloaded from a git repository and some packages for Linux platforms are provided. At the time of writing (December 2013), the source code of the framework comprises more than 50,000 lines of code, mostly in Python, some specific parts being implemented in C for performance purpose. It offers data models and basic algorithms to build, edit, visualise and simulate a communication protocol. We therefore implemented our approach and the others as plugins to reduce duplicated code and to simplify their comparisons.

The implementation of our approach as a plugin, we named Netzob, corresponds to only 500 lines of python since most of the computation codes are provided by our framework.

ASAP authors provide a publicly available implementation in R[5]. Thanks to the help of the authors, we developed a wrapper to execute ASAP implementation as a clustering plugin in our framework. This allows us to use the original ASAP code without inserting flaws. As recommended by authors, we uses Sally [22] to tokenize messages. Output clusters returned by ASAP are then transformed into symbols as presented in their article [13].

Unfortunately, implementing ScriptGen and Discoverer was much more difficult since neither source code nor implementation are publicly available, even for the scientific community. Among the two, Discoverer was the most difficult to re-implement as documentations and articles give too few details on some specific points such as on their merging strategy used in last step. In addition, both the authors of Discoverer and ScripGen did not publish the dataset they used to evaluate the effectiveness of their tools. Without publicly available datasets, it is thus difficult to validate our implementation of these approaches. However, we try to be as accurate as possible and check carefully our implementations of these approaches. Moreover, the results we obtained are similar to those described in the authors articles.

Each approach exposes parameters to the user that can highly impact the overall quality of their results when applied on a certain type of protocol. Thus, to ensure a fair comparison, we use the calibration datasets to compute the best parameters value and use them on their respective evaluation datasets. To identify these parameters, listed in figure 5b in appendix, we first established a large variation range for each of them and then compared the ROC curve of results brought by all possible combinations. For realistic datasets with no calibration, we only retained the best results given all the possible combination of parameter values.

4.5 Experimental Results

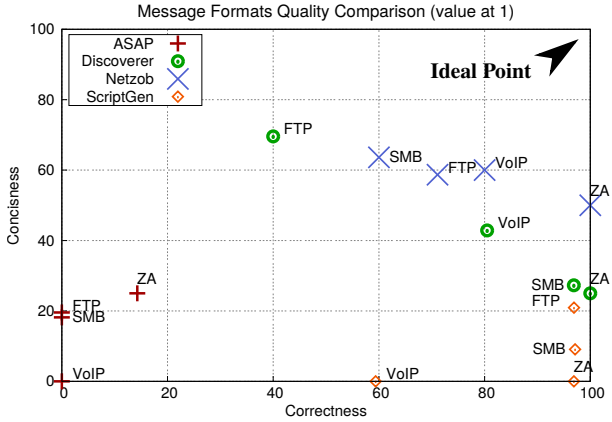


Figure 4: ROC Curve used to compare the quality of the inferred message clusters of ASAP, ScriptGen, Discoverer and Netzob. Best results are close to the top right corner.

In this section, we present the conclusions of our experimental comparative study of ASAP, ScriptGen, Discoverer and Netzob.

We expect results to be both concise and correct. This means that the ideal point of our ROC curves is the upper right corner of the graph, as illustrated in figure 4. It is also important that the tools balance concision with correctness. Indeed, incorrect clustering makes the tool useless as the user has to manually cluster the different messages. Conversely, a lost in conciseness entails an important problem: it generates too many symbols, which make the results difficult to interpret. Moreover, these symbols can be used to infer the grammar of the protocol and then to develop protocol generators. The size of the inferred protocol grammar automaton depends on the number of inferred symbols. A lost in conciseness will thus result in inefficient protocol generators. Concerning the ROC curve, this means that good results should be as close as possible to the upper right corner, *i.e.* the "Ideal Point" on figure 4, and near the diagonal that goes from origin to that upper right corner.

First of all, general results depicted in figure 4 show that the compared approaches fall into two categories. On one

[5]ASAP sources: `https://github.com/tammok/PRISMA/`

hand, ASAP and ScriptGen obtained poor results and always suffer from the overfitting problem. Moreover, ASAP correctness is quite low meaning that most of its inferred message formats denotes multiple true formats. On the other hand, Discoverer and Netzob show better results with a clear advantage for Netzob which results are always nearer the ideal point.

The precision of the clustering is also another revealing measure of this distinction. Indeed, only Discoverer and Netzob infer precise clusters, *i.e.* inferred clusters that perfectly match true formats. Despite the use of calibration datasets to optimize their parameters value, ScriptGen and Discover inferred clusters never matched a true format. Indeed, none of their inferred message formats is accurate enough to support the automatic generation of a protocol parser.

We also observe in figure 4 that ASAP, Netzob and ScriptGen tend to be stable as they provide similar results for the different datasets. However, Discoverer is quite unstable. On SMB and ZA datasets, it suffers from the overfitting problem whereas on the FTP dataset it obtains a good conciseness but lower correctness (about 40%).

Our comparative study shows that ASAP does not return good results on the datasets we used. For instance, applied on the FTP realistic dataset (②), only 20% of the true formats match a unique inferred format (*c.f.* figure 4). We believe ASAP is not appropriate to infer precise specifications of a protocol. An approach solely based on a statistical analysis of keyword or n-grams in messages, does not appear sufficient to cluster them.

Another interesting point in our comparative study results is that ScriptGen creates far too much clusters. For instance, on the SMB realistic dataset (④) made of 937 messages for 22 true formats, ScriptGen infers 906 different message formats. Indeed, most of the computed clusters contain a single message. Figure 5c depicts this low conciseness problem when applied on SMB dataset: it needs more than 100 inferred formats to cover 100% of the true formats. The reason why ScriptGen over-classifies is that it clusters messages according to their position in a session. This is not efficient, because in realistic datasets users often behave differently in each session which brings different message formats at similar position in sessions. Besides, when a classification error occurs at the beginning of the session, it affects the classification quality of the all following messages.

Obtained results confirm that ScriptGen was not designed to achieve a complete reverse engineering of a protocol. Indeed, it seems more appropriate for the inference of the very first exchanges of a communication. As stated by its authors, ScriptGen is more adapted to build an agnostic honeypot.

As illustrated in figure 4, our comparative study shows that among all the tools, Netzob always infers the best message formats. Indeed, it always computes message formats with a minimum correctness of 60% and a minimum conciseness of 50%. Whereas Discoverer lack of conciseness on binary protocols produces hundred of inferred message formats for a single true format.

Precision	FTP (②)	SMB (④)	ZA (⑤)	VoIP (⑥)
Discoverer	4.34%	22.72%	25%	6.66%
Netzob	**34.78%**	22.72%	**50%**	**26.6%**

Table 3: Number of precise clusters identified by Discoverer and Netzob.

In addition to its stability, Netzob also computes the highest rate of precise clusters. Table 3 shows that Netzob always infers at least (and often more) precise clusters than Discoverer. From our point of view, inferring precise clusters is important for an algorithm. Not only that precise clusters represents perfectly inferred format but they can also help the expert to correct the other inferred message formats. That is because in most of the protocols, the different messages shares common aspects such as encoding functions or delimiters. The expert can apply these information on other inferred format to improve them.

5. CONCLUSION

In this paper, we proposed a complete and automated approach for trace-based message formats reverse engineering. Our approach relies on novel techniques that leverage contextual information and correlation means to enhance message clustering as well as field boundaries and relationship identification. We implemented our approach in a publicly available framework, and demonstrated its efficiency against both standard and unknown protocols. Moreover, we compared our approach against three other state-of-the-art approaches (Discoverer, ASAP and ScriptGen). The experimentation shows that it provides better overall results, in addition to extracting fields semantic.

Based on these results, network security products editors can rely on an approach that automates the creation of protocol parsers, thus providing fast and reactive response adapted to today's cybersecurity context. Same goes with the field of malware analysis, in the aim of speeding up, for example, the take down of botnets.

6. REFERENCES

[1] J. Antunes, N. Neves, and P. Verissimo. Reverse engineering of protocols from network traces. In *Proceedings of WCRE*, 2011.

[2] M. A. Beddoe. Network protocol analysis using bioinformatics algorithms. In *Toorcon*, 2004.

[3] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of CCS*, 2009.

[4] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic extraction of protocol format using dynamic binary analysis. In *Proceedings of CCS*, 2007.

[5] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol specification extraction. In *Proceedings of SSP*, 2009.

[6] W. Cui. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of USENIX Security Symposium*, 2007.

[7] W. Cui, V. Paxson, N. C. Weaver, and Y. H. Katz. Protocol-independent adaptive replay of application dialog. In *Proceedings of NDSS*, 2006.

[8] J. Freeman. Hacking a closed ecosystem. In *O'Reilly Android Open Conference*, 2011.

[9] U. Gargi. Consumer media capture: Time-based analysis and event clustering. Technical report, HP Laboratories Palo Alto, aug 2003.

[10] C. Guarnieri, M. Schloesser, J. Bremer, and A. Tanasi. Cuckoo sandbox - open source automated malware analysis. In *Black Hat USA*, 2013.

[11] G. J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., 1991.

[12] T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012.

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

[14] F. Leder and P. Martini. Ngbpa next generation botnet protocol analysis. In *Emerging Challenges for Security, Privacy and Trust*, volume 297 of *IFIP Advances in Information and Communication Technology*. Springer Berlin Heidelberg, 2009.

[15] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*. MIT Press, 2000.

[16] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *Proceedings of ACSAC*, 2005.

[17] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Procedings of NDSS*, 2008.

[18] K. McNamee. Malware analysis report - new c&c protocol for zeroacess/siref. Technical report, Kindsight Security Lab, 2012.

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

[20] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proceedings of SIGCOMM*, 2003.

[21] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti. Detecting novel associations in large data sets. *Science*, 334, 2011.

[22] K. Rieck, C. Wressnegger, and A. Bikadorov. Sally: A tool for embedding strings in vector spaces. *Journal of Machine Learning Research*, 2012.

[23] J. Shearer. Trojan.zeroaccess threat report. Technical report, Symantec, 2011.

[24] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 1958.

[25] A. Syropoulos. Mathematics of multisets. In *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, 2001.

[26] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo. A semantics aware approach to automated reverse engineering unknown protocols. In *Proceedings of ICNP*, 2012.

[27] 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 ACNS*, 2011.

[28] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5(2):32–39, mar 2007.

[29] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: Using gui screenshots for search and automation. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 183–192, New York, NY, USA, 2009. ACM.

APPENDIX

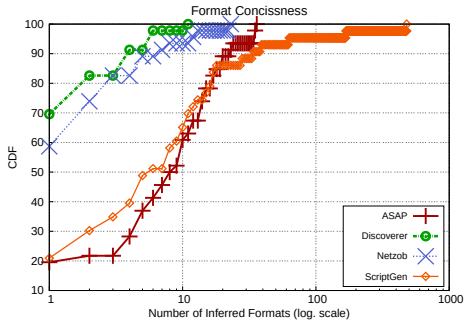
A. EXPERIMENTAL PARAMETERS

Protocols	Identified Actions	Identified Contextual Information
FTP	Connect with a bad password, successful connection, list current directory, invalid move directory, valid move directory, download file, upload a file, close connection	Client username, client password, server hostname, current directory, downloaded filename, uploaded filename, moving directory name, invalid directory name, directory listed.
SMB	Connect with a bad password, successful connection, list available shares, invalid move in a directory, list a directory, download file, upload a file, close connection	Client username, client password, server hostname, downloaded filename, uploaded filename, moving directory name, listed filenames, server domain name, server os, server version, server shares.
ZAccess	Receive a new peer address, propagate peer list.	IP addresses and UDP ports found in the pcap.
Commercial VoIP	Client 1 connects, client 2 connects, client 1 sends a message to client2, client 2 sends a message to client 1, client 1 changes its configuration, client 2 changes its configuration, client 1 disconnects, client 2 disconnects.	Server IP, server hostname, client IP, client phonetic names, client description, client comment message, client comment url, client messages.

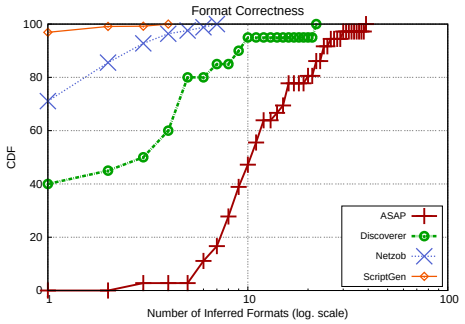
(a) Identified Semantics

Tools	*Parameters*	FTP ① ②	SMB ③ ④	ZAccess ⑤	Commercial VoIP ⑥
ASAP	Ngram Length	1	2	1	1
	Ngram Type	Text	Bin	Bin	Bin
	Ngram Delimiters	Extended	x	x	x
Discoverer	Min. Text Segments	2	2	2	2
	Min. Cluster Size	20	60	10	12
	Max. Distinct Values	10	5	20	350
ScriptGen	Macro-Clustering	0.9	0.7	0.6	0.6
	Micro-Clustering	0.5	0.5	0.4	0.4
Netzob	Similarity	0.9	0.6	0.5	0.8
	Matrix	10	10	10	-1

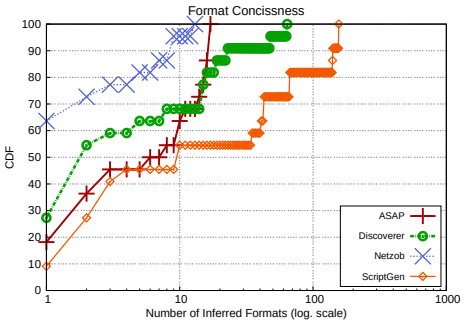
(b) Parameters used to configure each approach considered in our comparative study



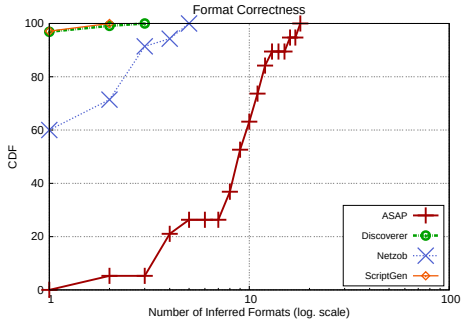
(a) FTP Conciseness



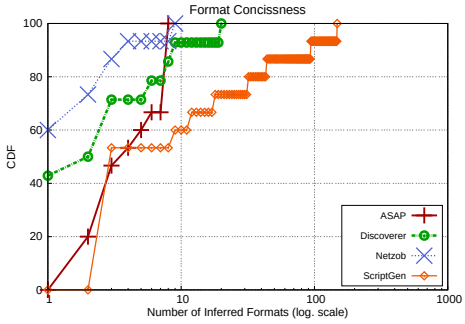
(b) FTP Correctness



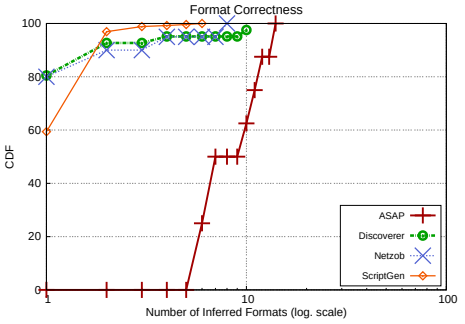
(c) SMB Conciseness



(d) SMB Correctness



(e) Commercial Product Conciseness



(f) Commercial Product Correctness

Figure 5: Comparative study details.