Building an Automaton Towards Protocol Reverse Engineering*

João Antunes and Nuno Neves
{jantunes,nuno}@di.fc.ul.pt

University of Lisboa,
Faculty of Sciences, LASIGE—Portugal

Abstract. The communication between computer systems is dictated by network protocols, which determine how the network components interact with each other. Knowing the specification of a network protocol can greatly improve the security and dependability of both the design of the protocol and the applications implementing it. The specification can be used, for example, to verify if the application's implementation is correct and in accordance, or even to aid in the creation of specific firewall rules or IDS filters to block messages that do not comply with the defined standard.
However, the protocol specification is not always available, which makes assessing the correctness and security of such protocols difficult. Protocol reverse engineering has been used to overcome this problem, by deducing the specification of closed protocols from their utilization alone and without any assumption about their structure or operation. In this paper, we present two different approaches, based on sequence alignment techniques, to build an automaton of a network protocol from network traces.

Resumo. A comunicação entre sistemas computacionais é ditada pelos protocolos de rede que determinam a forma como os componentes de rede interagem entre si. Conhecer a especificação do protocolo de rede pode melhorar profundamente a segurança e confiabilidade, tanto no desenho do protocolo, como nas aplicações que o concretizam. A especificação pode ser usada, por exemplo, para verificar se a concretização da aplicação está correcta e em conformidade, ou ainda para ajudar na criação de regras específicas de firewall ou filtros de IDS para bloquear mensagens que não estejam em conformidade com o padrão definido.
No entanto, a especificação do protocolo nem sempre está disponível, o que dificulta a avaliação da correcção e segurança destes protocolos. Para resolver este problema, foi proposto usar engenharia de reversão para deduzir a especificação de protocolos fechados a partir apenas da sua utilização e sem qualquer suposição sobre a sua estrutura ou funcionamento. Neste trabalho, apresentamos duas abordagens diferentes, com base nas técnicas de alinhamento de sequências, para construir um autómato de um protocolo através do tráfego de rede.

* This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

1 Introduction

Network protocols are essential in today's interconnected world. The communication between computer systems is dictated by a set of rules that regulates the syntax, semantics, and synchronization of the exchanged messages. Since it determines how the network components interact with each other, they are of crucial importance in security-related contexts. Most exploits, for instance, take advantage of vulnerabilities present in network protocols, or at least use them as a vehicle to explore vulnerabilities present in interconnected applications, such as servers or critical infrastructures. Knowing the specification of a network protocol can improve the security and dependability of both the design of the protocol and the applications that implement it. For example, one can verify if a network server correctly complies to the specification of the protocol by creating test cases covering the protocol space and then comparing its responses (and internal state) with the expected ones. The protocol specification can even be used to create firewall rules, denying messages not fully complying with the standard, or to provide intrusion detection systems (IDS) with the capability of performing deep packet inspection.

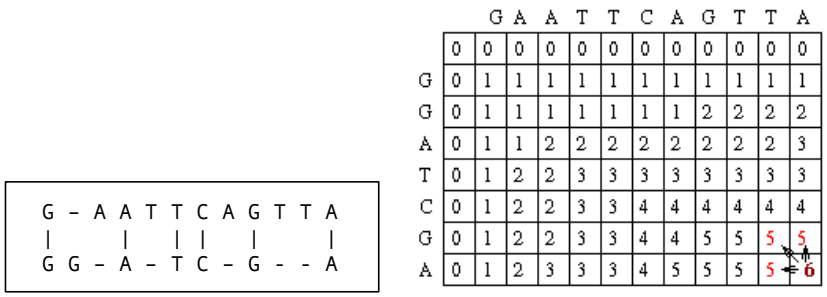
However, some Commercial Off-The-Shelf (COTS) components rely on closed protocols for their execution, sometimes to avoid the integration with other third-party components, other times for security reasons. Nonetheless, security through obscurity is not a safe principle, as a sufficiently motivated attacker will eventually discover the hidden flaws.

Protocol reverse engineering has been used to address this problem, by deducing the specification of closed protocols from their utilization alone and without any assumption about their structure or operation. In this paper, we present two different approaches to build an automaton of a network protocol from network traces. Our approaches are based on sequence alignment techniques, taken from the field of bioinformatics, which try to find the optimal alignment between the automaton and the protocol messages. The alignments are then used to further extend the automaton, which represents the syntax rules of the protocol.

The rest of this paper is organized as follows: Section 2 provides some background about sequence alignment algorithms and techniques. The proposed approaches to automata generation through sequence alignment are addressed in Section 3. Section 4 provides some preliminary evaluation results with FTP network traces. In Section 5 we present some related work. And finally, we conclude in Section 6.

2 Sequence Alignment

Sequence alignment is used in bioinformatics to identify similarity regions in sequences of DNA, RNA, or protein. Sequences are represented as a series of nucleotide and amino acid residues that are aligned, based on the residues similarities, within a matrix. Gaps can be introduced in any of the sequences so that similar residues can be aligned in consecutive columns.



(a) Alignment of the two sequences. (b) Substitution matrix.

Fig. 1. Pairwise sequence alignment using dynamic programming.

Figure 1a shows the alignment of two DNA sequences. Vertical lines represent matches between residues, whereas horizontal lines depict gaps. Gaps and mismatches represent the edit operations required to transform one sequence into the other.

2.1 Pairwise Sequence Alignment

Finding the best alignment between two sequences is an optimization problem in which the optimal solution of the overall problem can be deduced from the optimal solutions of many overlapping subproblems. Most sequence alignment algorithms are based on dynamic programming that makes use of such properties by memorizing previously computed subsequence alignments in a substitution matrix. Figure 1b shows the final substitution matrix of a sequence alignment algorithm using dynamic programming. A two-dimensional matrix is constructed with one column for each residue in one sequence and one row for each residue in the other sequence. Thus, if we are aligning sequences of length n and m, the running time of the algorithm is $O(n \times m)$. Each cell in the matrix has the value of the best-aligned subsequence as well as a reference to the adjacent cell from which that value was computed. When the algorithm reaches the last cell of the matrix, it backtraces through the referenced cells.

Usually, alignment algorithms fall into one of two categories, global or local alignment, in which the Needleman-Wunsch algorithm [1] and the Smith-Waterman [2] algorithm are the best examples. Local alignment algorithms try to identify similarity regions within the sequence, as opposed to the global alignment that attempt to align every residue in the entire sequences.

2.2 Multiple Sequence Alignment

Ideally, one would like to align more than two sequences. However, a straightforward dynamic programming algorithm in a k-dimensional matrix (where k is the number of sequences) would be computational unfeasible even for a modest number of sequences—the complexity to align k sequences of length n would be

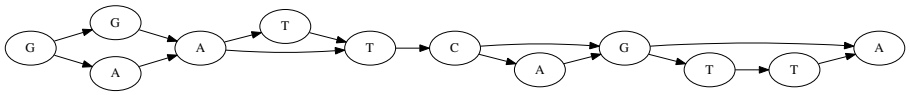


Fig. 2. Partial order graph representing a merged sequence alignment.

$O(2^k n^k)$. Hence, over the last 30 years several approaches have been developed to provide multiple sequence alignment, such as progressive alignment strategies, evolutionary trees, alternative alignment representations, probabilistic alignment, etc., resulting in a very large number of alignment applications, each with their own strengths and weaknesses [3]. Most techniques use heuristics for finding sub-optimal multiple alignments, such as computing all $\binom{k}{2}$ optimal pairwise alignments between every pair of sequences and then combine them together using other heuristics. For instance, the results of multiple sequence alignments can be merged into one reduced consensus sequence that shows which residues are most abundant at each position of the alignment. Clustal, for instance, uses pairwise alignment as a building block for multiple k-way alignments [4]. A phylogenetic tree is built from the similarities between each pair of sequences (pairwise alignment). This progressive multiple alignment heuristic then combines each alignment iteratively as a linear consensus sequence, starting with the closest related groups in the tree towards the root. Consensus sequences, or profiles, allow multiple alignments to be achieved by only performing one pairwise alignment for each new sequence.

2.3 Partial Order Alignment (POA)

Consensus sequences are an incomplete representation of the alignment of multiple sequences. Information about less frequent residues found in early sequence alignments will be lost and cannot be used later when more similar sequences appear. Therefore, the order in which the sequences are aligned is determinant to the quality of the final multiple sequence alignment.

One solution that preserves information of the previous alignments is to represent the consensus sequence as a complete partial-order graph. Partial order alignment (POA) performs multiple sequence alignment by aligning each new sequence (in any order) with a partially ordered graph in which individual sequence letters are represented by nodes, and directed edges are drawn between consecutive letters in each sequence [5]. As each new sequence is aligned, even the less frequent residues are represented, so no information is lost. Thus, instead of a single incomplete consensus sequence, several subsequences are used (one for each chain of consecutive residues). Figure 2 shows the alignment of the sequences from the previous examples, merged into a partial-order graph. As more sequences are aligned, unmatched residues will create new branches. POA extends the dynamic programming method of aligning two sequences by replacing one of the linear sequences with the partial order nodes. Branches in the graph are resolved by grafting a copy of the bifurcations across the matrix, forming additional dynamic programming matrix surfaces that are joined exactly as the individual branches

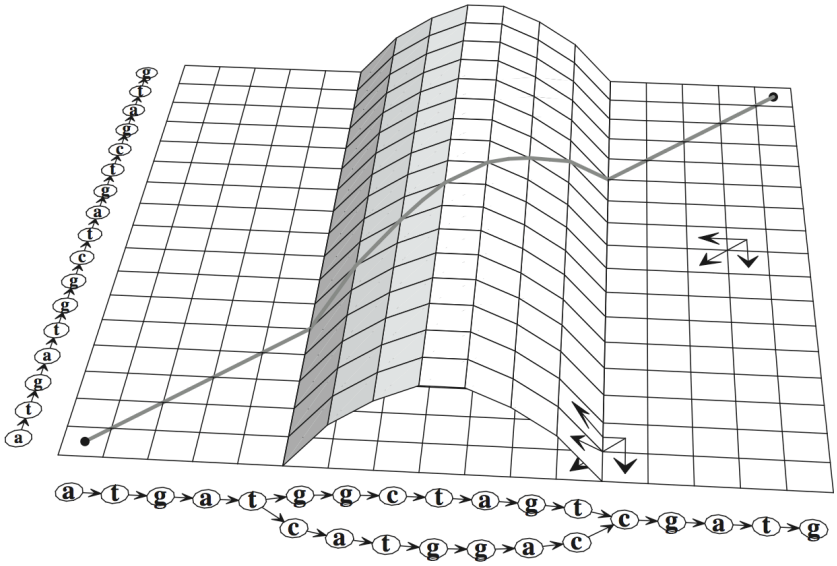


Fig. 3. Substitution graph for the partial order alignment (image taken from [5]).

are joined in the partial order graph. Figure 3 shows the complete substitution matrix for a partial order alignment. The branch in the graph is depicted in the matrix by an alternative sub-matrix, which is then calculated as in the original dynamic programming algorithms.

This method has the advantage of merging similar nodes and sequences into one larger partial order graph, and executing only one pairwise alignment for each new sequence, though all branching subsequences must be visited.

3 Building an Automaton From Protocol Messages

A protocol is a set of rules that dictates the communication between two or more entities. Messages accepted by a given protocol must follow some very specific syntax rules. In this sense, a network protocol can be seen as a formal language whose syntax rules are defined through a formal grammar. In formal language theory, languages are completely deprived of any semantic meaning, and only their syntax formation rules define the words that are accepted. The formal grammar is the set of formation rules that describes how symbols of the alphabet can be combined to form words that are accepted by the language. In another words, the formal grammar of a network protocol describes how bytes can be combined to form acceptable messages.

An equivalent and perhaps more comfortable way of modeling a network protocol is through a finite-state machine automaton. In particular, a deterministic finite automaton is one of the most practical models of computation, since there are trivial linear time, constant-space, and online algorithms to simulate them on

a stream of input. In fact, the automata theory is closely related to formal language theory as the automata are often classified by the class of formal languages they are able to recognize.

The purpose of our work is to provide a solution to construct an automaton that recognizes a specific network protocol based on a sample of its messages. Each state of the automaton represents the sequence of symbols (i.e., bytes of the message) accepted so far, and the remaining symbols required to accept the entire word (i.e., the message of the protocol). Formally, the automata can represented by the 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a set of all possible states[1], Σ is the alphabet (2^8 possible symbols) of the protocol, δ is the transition function that tells the automaton which state to go to next given a current state and a current symbol (byte), q_0 is the initial state, and F is the set of final states.

Graphs can also be used to represent automata, with nodes denoting states and edges describing the transition functions. To our purposes, representing the automaton as a directed graph or a finite-state machine is equivalent. Graphically, however, the directed graph offers a more clear and simple representation because edges have no labels and each state is represented by the symbol that leads to it.

The partial order alignment (POA) algorithm briefly described earlier, successively aligns new sequences with a growing partial order graph. In order theory, a partial order set formalizes the intuitive concept of an ordered set, which can be represented through a directed acyclic graph (DAG). A DAG is a directed graph in which there are no directed cycles, i.e., in other words, a DAG flows in a single direction where each state can only be visited once.

3.1 Greedy Approach

Our first approach was to extend the POA algorithm, used in DNA multiple sequence alignment, to generate an automaton from protocol messages that would describe the language of the protocol. A pairwise sequence alignment algorithm is used to identify the best automaton path, i.e., the ordered set of nodes that provides a better scoring alignment, and to align the new sequence with the automaton. Figure 4 shows the DAG of the multiple sequence alignments of the previous examples, being aligned with a new sequence. The original pairwise sequence alignment algorithm resorts to a matrix of scores. Instead, our solution avoids matrixes by addressing each column of the matrix as an individual vector of scores, calculated from the adjacent vector of scores. A vector of scores is calculated for each node, so that each position of the vector represents the total score of the best alignment sequence that aligns the symbol of that node with the symbol at that position of the sequence. Additionally, the order in which the vectors are actually calculated is reversed, so that the alignment algorithm starts backtracing the alignment at the first node (instead of the last cell of the matrix). This optimization also relieves the need for additional references (i.e., parent's references on each child).

[1] In the context of protocol reverse engineering, we refer to *state* as the state of the automaton that accepts/rejects network messages, and not the various states that the protocol might have, such as waiting for login, authenticated, etc.

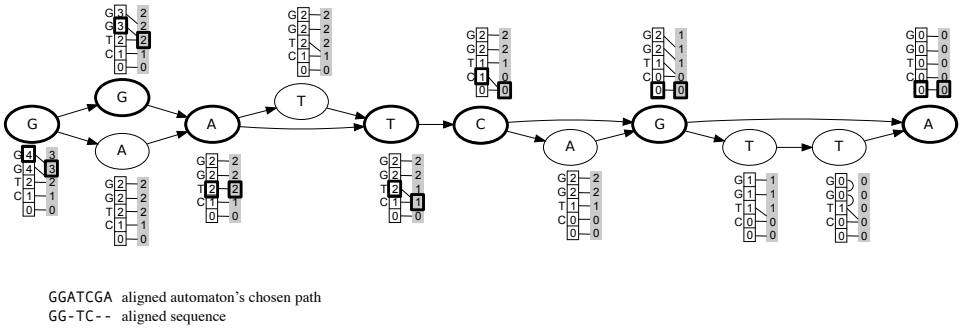


Fig. 4. Aligning a new sequence with the automaton.

This approach uses the Needleman-Wunsch algorithm as a pairwise sequence alignment, but other dynamic programming alignment algorithms, such as the Smith-Waterman, could also be adapted in the same way. The alignment algorithm starts by calculating the vector of scores from the first node, which recursively calculates the adjacent vectors of scores. In the end, the first cell of the first vector of scores provides the remaining cell references for the complete alignment. Pseudocode 1.1 depicts the recursive algorithm to calculate the vector of scores. Each cell of the vector also holds a reference of the adjacent cell from which that value was calculated, i.e., diagonal, right or bottom. The corresponding recurrence (adapted from the Needleman-Wunsch algorithm) for the score $S_{i,j}$ of an optimal alignment between node i and the j-th character of the sequence is given as follows:

$$S_{i,j} = max \begin{cases} S_{i+1,j} + 0 \\ S_{i,j+1} + 0 \\ S_{i+1,j+1} + 1 \end{cases}$$

This score function attributes a score of zero for each additional gap penalty and one for each match. The sequence alignment provides information about the nodes of the automaton that are best aligned with the sequence, i.e., which path is closest to the new message. The maximum value chosen for the vector of scores of node i directs the path of the alignment—in the first case a gap is introduced in the sequence, the second case adds a gap before the next node of the automaton, and in the third case the node i and the j-th character of the sequence are aligned.

The next step is then to incorporate the new sequence in the automaton. The POA algorithm tries to minimize the number of different nodes; hence, it follows a greedy approach by merging every aligned node that shares the same symbol. In the same example, merging the sequence `GGTC` with our automaton would not produce new nodes because all symbols of the sequence were already perfectly aligned in the graph. However, a sequence with a mismatched node, such as `GGTU`, would generate a new edge from `T` (any of the first two, as they would score the same) to a newly created node `U`.

This approach is especially useful to detect common subsequences, wherever they are placed in the messages. For example, consider the IMAP protocol in

```
vectorScores
    Input: Node ← Node of the automaton
    Input: Sequence ← Sequence to be aligned
    Output: Scores ← Vector with the best scores of the children

    // base case
    Scores ← vector of zeros of length #Sequence
    if Node has no children then
        return Scores

    // recursion for every child
    foreach Child of Node do
        Temp ← PairwiseAlignment(Child, Sequence, vectorScores(Child,
              Sequence))
        foreach i of Temp
            if Temp[i] is best than Scores[i] then
                Scores[i] ← Temp[i]
    return Scores
```

Pseudocode 1.1. Algorithm for the calculation of the vector of scores.

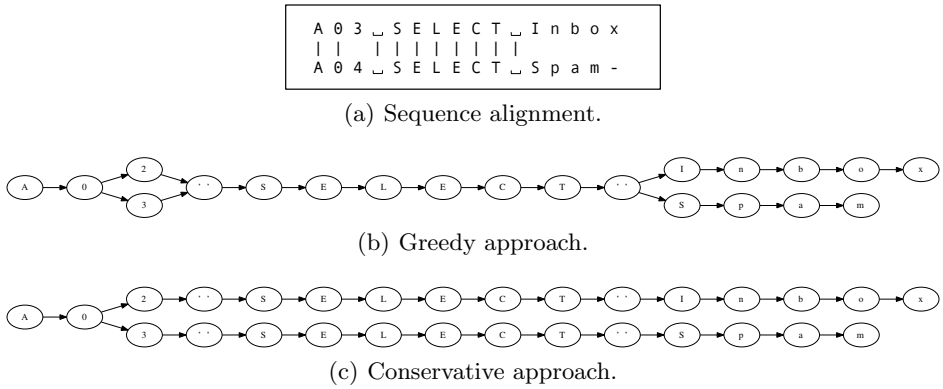
which every command is prefixed with a distinct alphanumeric tag (e.g., `A03 SELECT Inbox`, `A04 SELECT Spam`, etc.). The most relevant part of the message (i.e., the command `SELECT`) appears *after* the variable tag parameter. Hence, similar command names will always be aligned, independently of the remaining of the message. Figure 5b shows the greedy approach while merging the alignment of the two IMAP messages of Figure 5a. The graph evidences the common patterns of both messages, in particular the `SELECT` command.

While messages previously processed are always accepted by the automaton, a greedy merge might be too optimistic and generate transitions allowing messages that do not belong to the language to be accepted. One could imagine a protocol that uses a special prefix to confer different parameters to the same commands, making the syntax of those messages different altogether.

3.2 Conservative Approach

Our other approach to merge the aligned sequences is to produce a conservative automaton, which would *only* accept sequences that were previously aligned, therefore avoiding any false positives. For instance, the automaton of Figure 4, which was constructed from two sequences: `GAATTCAGTTA` and `GGATCGA`, is prone to false positives because it would also accept the sequences `GAATTCGA` or `GGATCAGTTA` (which were never merged before).

In order to keep the automaton strictly correct, i.e., it only accepts messages already received, our conservative approach only branches new nodes when the alignments start to differ (after a common prefix) or ever cease to differ (upon a common suffix). Hence, our solution is to do a greedy merge only for common



(a) Sequence alignment.

(b) Greedy approach.

(c) Conservative approach.

Fig. 5. Two approaches to merge a sequence alignment into the automaton.

prefixes and suffixes, and create new nodes and transitions for the remaining cases. This approach will produce a conservative automaton, with a higher number of states than the greedy one and much more specific (and less permissive) as to which messages it accepts.

Figure 5c shows the conservative approach with two separate paths for the same command `SELECT`, as opposed to the merged path in the greedy approach.

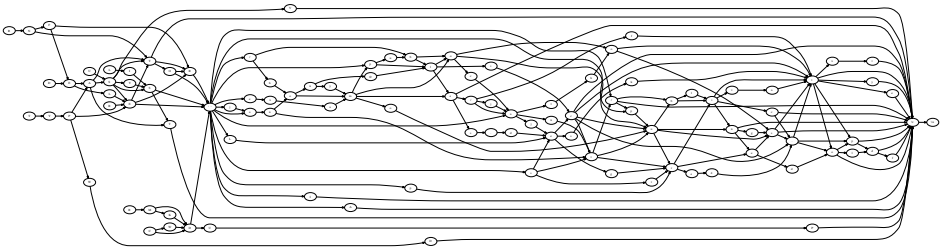
4 Preliminary Evaluation

To evaluate the impact of the conservative and greedy approaches we used the methodology presented here to create two separate automata from the network traces of three FTP sessions. The FTP sessions were composed of 103 messages containing 14 different FTP commands. We then evaluated the resulting automata with respect to the number of nodes and edges of the resulting graphs.

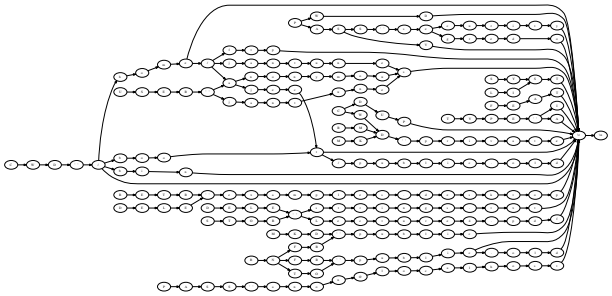
The automata generator prototype was implemented in Java and supports both the greedy and the conservative merging approaches. The program application reads a packet capture file containing the network traces. It filters out all network interactions from other protocols and not coming from the client, so that it only processes FTP messages from the client, i.e., the TCP messages from the FTP session coming from the client. Currently, each network packet is regarded as a single protocol message, disregarding any issues related with the streaming nature of the TCP[2]. The application then iteratively feeds each packet to the automaton, obtaining a sequence alignment, which it will integrate according to the predefined strategy, i.e., greedy or conservative.

This preliminary evaluation tries to reveal the strengths and weaknesses of either approach. Figure 6 shows the resulting directed graphs of the automata. The greedy approach, which optimistically merges every aligned node sharing the same symbol, generated a much smaller automaton with only 110 nodes as

[2] Although TCP is stream-oriented, it is very common that each application-layer message corresponds to a single TCP message, unless the payload is too large.



(a) Greedy approach (110 nodes, 199 edges, 135 branches).



(b) Conservative approach (225 nodes, 242 edges, 32 branches).

Fig. 6. Preliminary evaluation of three FTP sessions with 103 client messages.

opposed to the 225 nodes of the conservative approach. However, 68% of the edges of the greedy automaton are actually branches, thus increasing the complexity of the graph, whereas the conservative automaton the branches account for only 13% of the edges.

A greedy approach will result in smaller but more complex automata, susceptible of false positives. The conservative automata, on the other hand, guarantees that only valid messages are accepted at the cost of some additional separate paths, i.e., several single-transition nodes.

5 Related Work

Protocol reverse engineering is a traditionally manual and tedious process. Aided by network packet analyzers, reverse engineers carefully analyze each packet looking for common and recognizable patterns, such as special delimiters and text fields [6]. The process of reverse engineering a protocol also depends on the number (and quality) of protocol interactions. The more complete the collection of network traces is, more confidence can be placed in the resulting protocol specification.

Since some of the tasks associated with the manual process could be automated or at least automatically assisted, several techniques started to emerge to ease the reverse engineering process. PDB, for instance, is a Protocol Debugger tool that

operates as a transparent network proxy, allowing the operator to set breakpoints on specific packets or events, and inspect and modify individual packets [7].

More recently, new works appeared in the literature trying to infer the protocol specification automatically. Protocol informatics, for instance, applied concepts from bioinformatics to construct a rough description of the protocol [8]. The tool employs sequence alignment algorithms to group similar messages together in phylogenetic trees to automatically identify fields in network packets. The tree is traversed to guide a progressive sequence alignment that will produce a consensus sequence for each cluster of similar messages. As the authors notice, using consensus sequence alone is subject to loss of information as only the most prevalent characters are preserved.

Discoverer is another tool that operates on network traces to perform clustering and type-based sequence alignment to infer message formats [9]. It first tokenizes each message in to binary and text segments to cluster messages of similar format. Clusters are then further divided in a refining process where messages are re-compared and their fields analyzed for cross-field dependencies. This allows some special fields to be identified, such as length fields, or fields whose value differentiates the format of the remaining message. To avoid over-classification, i.e., the creation of small irrelevant clusters, the tool then compares the field structure of the inferred message formats to merge messages with identical format in the same cluster.

Other techniques focus on the binary programs that implement the protocol. Some researches have proposed to use static analysis to generate possible inputs that a program can accept [10]. Beside the fact that it is undecidable to statically determine the complete set of inputs for a program, this approach also suffers from significant scalability issues.

Other researches have also proposed to use dynamic analysis on applications that implement the protocol. Dynamic analysis also circumvents the problems inherent to static analysis, such as memory aliasing and indirect jumps. These tools closely monitor the program's execution while processing the input messages, resorting to dynamic taint analysis to identify the relevant sections responsible for processing and parsing the network packets [11, 12]. One of the limitations of these tools is that they cannot generalize message formats over multiple message samples.

6 Conclusion

In this paper, we have presented two different approaches to build an automaton of a network protocol from network traces. Our solution is based on sequence alignment techniques, taken from the field of bioinformatics, which try to find the optimal alignment between the automaton and the protocol messages. The alignments are then used to further extend the automaton, which represents the syntax rules of the protocol.

The preliminary results of our evaluation with FTP network traces show the main differences in generating a greedy automaton (which tries to minimize the number of nodes with similar symbols) and a conservative one (where new nodes

are created only if they do not introduce any spurious paths). Our results show that a greedy strategy will identify common patterns in the messages, independently of their position, but it will generate more complex automata, i.e., with several bifurcations. On the other hand, the conservative approach will produce simpler automata, but with alternative paths whenever the new message diverges with the automata.

On this work, we have focused on bioinformatics algorithms to generate finite-state machines that would accept protocol messages. The next step towards the protocol reverse engineering will be focused on the refining the automaton and on producing some sort of protocol specification that could be used on other applications, such as intrusion detection systems or fuzzers. We intend to further complement our work with other bodies of research, such as automata reduction, information theory, and formal language theory, to ameliorate and extend the generation of the automata.

References

1. Needleman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol **48**(3) (1970) 443–453
2. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology **147** (1981) 195–197
3. Simossis, V., Kleinjung, J., Heringa, J.: An overview of multiple sequence alignment. Current protocols in bioinformatics (2003)
4. Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T., Higgins, D., Thompson, J.: Multiple sequence alignment with the clustal series of programs. Nucleic acids research **31**(13) (2003) 3497
5. Lee, C., Grasso, C., Sharlow, M.: Multiple sequence alignment using partial order graphs (2002)
6. Beardsley, T.: Manual protocol reverse engineering. BreakingPoint Systems (2009) http://www.breakingpointsystems.com/community/blog/manual-protocol-reverse-engineering.
7. Rauch, J.: PDB: The protocol debugger. In: BlackHat USA, BlackHat (2006)
8. Beddoe, M.A.: Network protocol analysis using bioinformatics algorithms (2005) http://www.4tphi.net/ awalters/PI/PI.html.
9. Cui, W., Kannan, J., Wang, Helen, J.: Discoverer: automatic protocol reverse engineering from network traces. In: Proceedings of the USENIX Security Symposium, Boston, MA, USA (2007) 199–212
10. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: Automatic protocol replay by binary analysis. In: Proceedings of the ACM conference on Computer and communications security, ACM New York, NY, USA (2006) 311–321
11. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: Proceedings of the ACM Conference on Computer and Communications SecurityC, ACM New York, NY, USA (2007) 317–329
12. Wondracek, G., Comparetti, P., Kruegel, C., Kirda, E., Anna, S.: Automatic network protocol analysis. In: Proceedings of the Annual Network and Distributed System Security Symposium. (2008)