

The future of protocol reversing and simulation with Netzob

29C3, Hamburg

georges.bossert@supelec.fr

frederic.guihery@amossys.fr

Abstract

Have you ever been staring at binary or hexadecimal data flows extracted from an USB channel ? Don't you remember yourself searching for some patterns and other similarities in this fuc***g mess of zeros and ones grabbed from a binary configuration file ?

Did you know you were not a alone and that others including Rob Savoye [**Sav09**] and Drew Fisher [**Fis10**] have already described the main difficulties of the RE¹ operations. Both of them have called for the creation of a tool which would help the expert in its process. For about these last 2 years, we've been working to respond to this call to create this kind of tool. Hence, this article will describe a new semi-automated RE process based on Netzob². This tool simplifies the manipulation of binary flows, finds relations, deduce data formats, infer grammatical definition and other few little things ;)

Introduction

There are many reasons why an I.T. Advance User would engage himself in a RE operations. For example, some wants to understand how their favorite game store their player's profile while others wants to use their USB device on an originally unsupported OS. In addition to these common usages, security auditors (and evaluators) often use RE process in their work. **This article will discusses usage of RE by security auditors and evaluators.**

In recent years, the field of security analysis systems or software was extended with new approaches and new tools based on the fuzzing. Compared to more traditional techniques (static and dynamic analyzes binary, potentially combined with the analysis of the source code) that require specialized skills, resources and time, these new tools offer many advantages : relative simplicity of implementation, semi-automated approach, rapid acquisition of results, etc... . However, experience shows that to be truly effective, an effective fuzzing analysis requires a good knowledge of the target, and in particular of the communication protocol. This fact limits the effectiveness and completeness of results obtained in the analysis of products implementing

¹"RE" is a common acronym for Reverse Engineering.

² See project website: <http://www.netzob.org>

proprietary protocols or undocumented.

On the other hand, in the analysis of security products such as firewall or network intrusion detection (NIDS), an evaluator often needs to generate realistic traffic in order to assess the relevance and reliability of the tested product (that is, its ability to limit false positives and false negatives). This operation is complex because it requires the complete characterization and control over the generated traffic. Hence, an evaluator can't generate traffic if he doesn't have access to the specification of the protocol.

This is to meet the many needs of reverse engineering protocols that Netzob was developed. This tool is a framework to infer protocols. It supports the expert in his reverse engineering work by offering a set of semi-automated features to reduce the analysis time and ultimately improve its understanding of the targeted protocol. We organize the remainder of the article as follows: after presenting terminology and our assumptions in Section 2, we detail Netzob's design and functionalities in Section 3.

Terminology and design assumptions

Few days of bibliography on the RE field illustrated the huge differences between the academic world and the "applied" world. The first one regroups multiple work and papers while in the applied world, experts are specialized ninjas computing CRCs and other format trans-coding with their eyes.

On the academic field, most of the researchers consider A. Beddoe as the initiator of the automated RE with its tool named PI³ published in [Bed04]. Following years brought multiple papers including those of W. Cui and al. [Cal.06d][Cui07] and of C. Leita [LMD05]. Each of them include new algorithms and next-gen complicated approaches, but none of them effectively brought to the applied world a useable tool. This resulted to an important dis-synchronization between the researchers and the security experts we'll try to tackle with this article.

Definition of a Communication Protocol

A communication protocol can be defined as the set of rules allowing one or more entities (or actors) to communicate. Applied to the field of computer networking, protocols have been the subject of many standardization activities, particularly from the OSI model, which establishes, among other things, the principle of protocol layers. However, few studies have attempted to give a formal and generic definition of a protocol. We refer here to the definition provided by Gerard Holzmann in his reference book "*Design and Validation of Computer Protocols*" [Hol91]. According to the author, a protocol specification consists of five distinct parts:

1. The **service** provided by the protocol.
2. The **assumptions** about the environment in which the protocol is executed.
3. The **vocabulary** of messages used to implement the protocol.

³See the "Protocol Informatics" Project : <http://www.4tphi.net/~awalters/PI/PI.html>

4. The **encoding (format)** of each message in the vocabulary.
5. The **procedure rules** guarding the consistency of messages exchanges.

In our work, we seek to infer the three last elements of the specification since there are mandatory to configure the fuzzing process and to generate valid traffic. Subsequently, we consider protocol inference requires to learn both:

1. The set of messages and their format, also called **vocabulary of a protocol**.
2. All the rules of procedure that we name **grammar of a protocol**.

These two parts of a protocol are detailed in the followings.

Definition of the Vocabulary of a Protocol

Within Netzob, the vocabulary consists of a set of symbols. **A symbol represents an abstraction of similar messages**. We consider the similarity property refers to messages having the same role from a protocol perspective. For example, the set of *TCP SYN* messages can be abstracted to the same symbol. An *ICMP ECHO REQUEST* or an *SMTP EHLO* command are other kinds of symbols.

A symbol structure follows a format that specifies a sequence of expected fields (e.g. TCP segments contains expected fields as sequence number and checksum). Fields have either a fixed or variable size. A field can similarly be composed of sub-elements (such as a payload field). Therefore, by considering the concept of protocol layer as a kind of particular field, it is possible to retrieve the protocol stack (e.g. TCP encapsulated in IP, itself encapsulated in Ethernet) each layer having its own vocabulary and grammar. The identification of the different fields allows to define the symbol format.

In our model, a field has different attributes: some of them concern every fields, others are specific to the field type and only relevant for a visualization purpose. **The domain defines the authorized values of a field, under a disjunctive normal form⁴**. An element of the domain can be:

- A static value (e.g. a magic number in a protocol header).
- A value that depends on another field, or set of fields, within the same symbol. We call this notion intra-symbol dependency (e.g. a CRC code).
- A value depending on another field, or set of fields, that are part of a previous symbol transmitted during the same session. We call this notion inter-symbol dependency (e.g. an acknowledgment number).
- A value depending on the environment. We call this notion environmental dependency (for example, a timestamp).
- A random value (e.g. the initial value of the TCP sequence number field).

Besides, fields have some interpretation attributes, notably for visualization and data seeking purposes. We associate a field content with a unit size (the size of atomic elements that compose the field, such as bit, half-byte, word, etc.), an endianness, a sign and a representation (i.e. decimal, octal, hexadecimal, ASCII, DER, etc.). Optionally, a semantic may be associated to a field (such as an IP address, an URL, etc.). As an example, the Illustration 1 depicts different format and visualization attributes of Botnet SDBot C&C

⁴A Disjunctive Normal Form (DNF) is a logical formula built with OR of AND : ((a AND b) OR (c AND d)).

messages.

Format	:	{1,20}	!	{1,20}	@	{7,15}	PRIVMSG	{1,20}	::download	{10,100}	□	{2,80}
Visualization	str	str	str	str	str	IP	str	str	str	str	URL	str
Messages	:	master	!	qster	@	192.168.163.3	PRIVMSG	micztu	::download	http://Y7jgkl.net/dump.png	□	E:\\dump.png
	:	master	!	stkp	@	172.16.18.52	PRIVMSG	supze	::download	http://gth856782.info/file.bin	□	E:\\file.bin
	:	master	!	fplr	@	10.10.11.7	PRIVMSG	utpb	::download	http://vxABC23.xxx/drop.dll	□	E:\\drop.dll
	:	master	!	qster	@	192.168.163.3	PRIVMSG	micztu	::download	http://p4l3xrvtqw.eu/config.txt	□	E:\\config.txt

Illustration 1: Format of SDBot C&C messages. The first line depicts the symbol format. The second line corresponds to the visualization attributes. These attributes are optionally overloaded by semantic characteristics.

Definition of the Grammar of a Protocol

The grammar represents the ordered sets of messages exchanged in a valid communication. Applied to the ICMP protocol, its grammar include a rule which state that an *ICMP ECHO REPLY TYPE 8* always follow an *ICMP ECHO REQUEST TYPE 8*. Another example of a grammar is the set of rules which describe the ordered symbols sent between two actors of a TCP session. **These rules can be represented using an automata which define the states and the sent and received symbols on each transitions** (see example on Illustration 2).

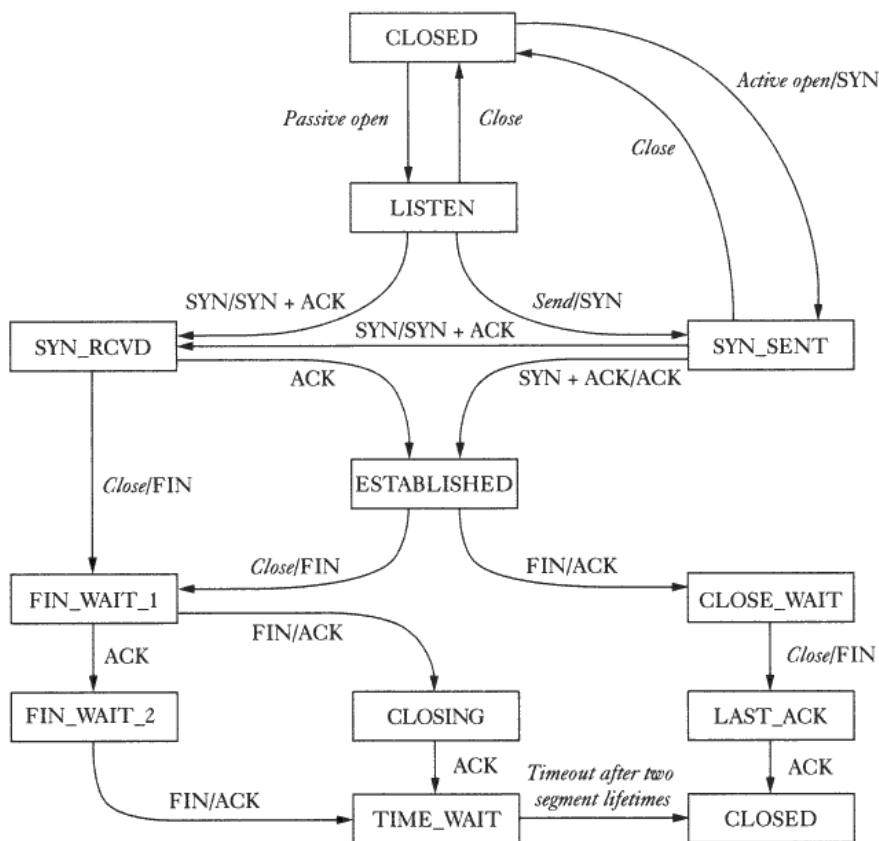


Illustration 2: TCP State Transition Automata (<http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html>)

Within Netzob, the grammar is defined with our own mathematical model named Stochastic Mealy Machine with Determinist Transitions (an SMMDT). This extension of traditional Mealy Machines allows to have multiple output symbols on the same transition and to represent multiple answers to the same command. In addition to this feature, our model also include the reaction time for each couple of request and reply symbols. The interested reader can contact us⁵ for any questions regarding the model, we've some nice equations available ;)

Few Assumptions

Our objective is to infer the vocabulary and the grammar of a communication protocol in order to audit (using fuzzing approaches) and to evaluate (using traffic simulator). These operations do not require previous knowledge on the protocol but still the followings assumptions are made :

1. The inferring process requires initial input data. Hence the expert must obtain examples of multiple communication channels hosting the targeted protocols. It can be any data flow from an USB channel (e.g. external devices) to network flows (e.g. Botnets' C&C) through inter-process calls (e.g. API Hooking) or file protocols (e.g. configuration files, ...).
2. The input data should include the largest diversity of included environmental values (IP addresses, pseudos and other user data, hostnames, date and time, ...).
3. The expert must have access to an implementation which uses the targeted protocol. It must be possible to automate its execution and to reinitialize it to its initial state. We recommend snapshot mechanism included in VMWare or VirtualBox for this.
4. Naturally, the targeted protocol should not include encrypted or compressed content. If some exists, the expert can use included entropy measures to identify some of them and use solutions like API Hooking to get the clear data.

Inferring Vocabulary and Grammar with Netzob

In this Section, we describe the learning process implemented in Netzob to semi-automatically infer the vocabulary and the grammar of a protocol. This process, illustrated in picture 3, is performed in three main steps:

6. Clustering messages and partitioning these messages in fields.
7. Characterizing message fields and abstracting similar messages in symbols.
8. Inferring the transition graph of the protocol.

⁵You can contact authors on contact@netzob.org.

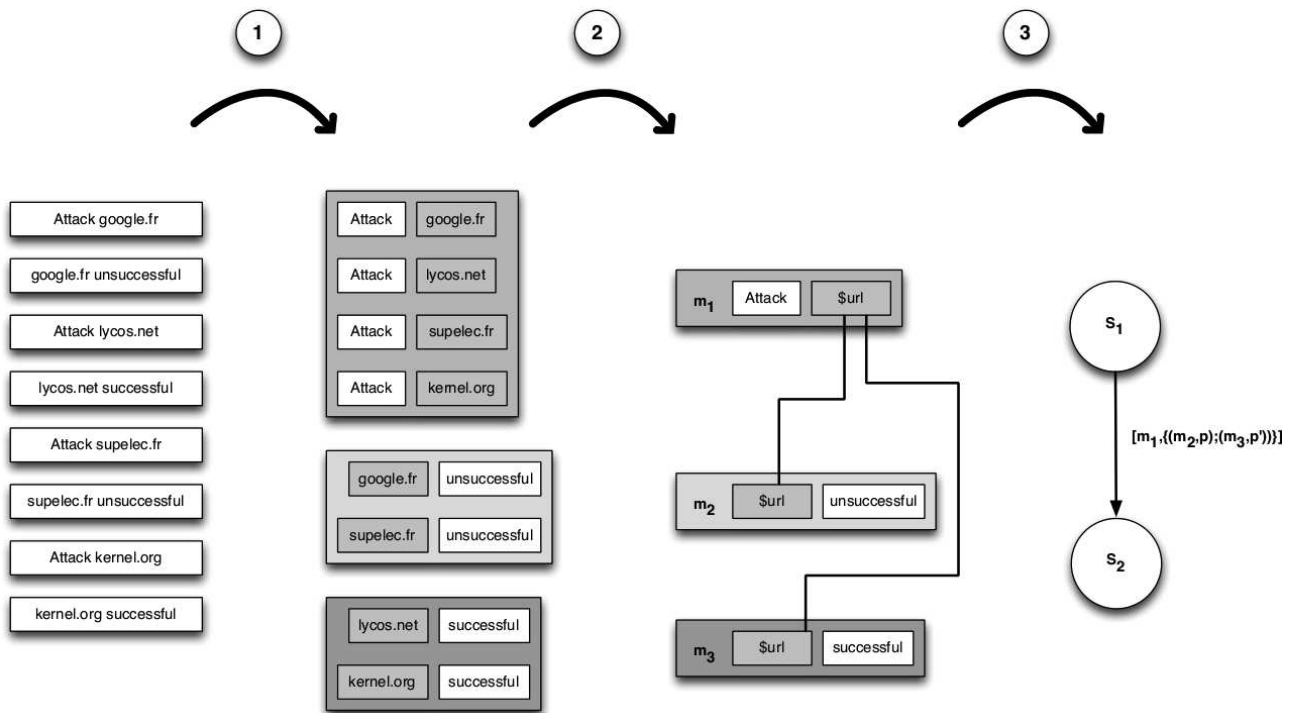


Illustration 3: Steps of protocol inference in Netzob

Step 1: clustering Messages and Partitioning in Fields

To discover the format of a symbol, Netzob supports different partitioning approaches. In this article we describe the most accurate one, that leverages sequence alignment processes. This technique permits to align invariants in a set of messages. The Needleman-Wunsh algorithm [NW70] performs this task optimally. Needleman-Wunsh is particularly effective on protocols where dynamic fields have variable lengths (as shown on picture 4).

	Type	Visualization format
150 Opening BINARY mode data connection for string	(.{12,40}) string	((.{6,14}) bytes).. string decimal string
150 Opening BINARY mode data connection for	tb.php	(5292 bytes)..
150 Opening BINARY mode data connection for	rss.php	(3965 bytes)..
150 Opening BINARY mode data connection for	coords.txt	(365 bytes)..
150 Opening BINARY mode data connection for	access_marsToMai.log	(1392891 bytes)..

Illustration 4: Example of sequence alignment on FTP messages

When partitioning and clustering processes are done, we obtain a relevant first approximation of the overall message formats. The next step consists in determining the characteristics of the fields.

If the size of those fields is fixed, as in TCP and IP headers, it is preferable to apply a basic partitioning, also provided by Netzob. Such partitioning works by aligning each message by the left, and then separating successive fixed columns from successive dynamic columns.

To regroup aligned messages by similarity, the Needleman-Wunsh algorithm is used in conjunction with a clustering algorithm. The applied algorithm is UPGMA [Smi58].

Step 2 : characterization of Fields

The field type identification partially derives from the partitioning inference step. For fields containing only invariants, the type merely corresponds to the invariant value. For other fields, the type is automatically materialized, in first approximation, with a regular expression, as shown on figure 3. This form allows to easily validate the data conformity with a specific type. Moreover, Netzob offers the possibility to visualize the definition domain of a field. This helps to manually refine the type associated with a field.

Some intra-symbol dependencies are automatically identified. The size field, present in many protocol formats, is an example of intra-symbol dependency. A search algorithm has been designed to look for potential size fields and their associated payloads. By extension, this technique permits to discover encapsulated protocol payloads.

Environmental dependencies are also identified by looking for specific values retrieved during message capture. Such specific values consist of characteristics of the underlying hardware, operating system and network configuration. During the dependency analysis, these characteristics are searched in various encoding.

Step 3: inferring the Transition Graph of the Protocol

The third step of the learning process discovers and extracts the transition graph from a targeted protocol. This step is achieved by a set of active experiments that stimulate a real client or server implementation using successive sequences of input symbols and analyze its responses.

Each experiment includes a sequence of selected symbols according to an adapted version of Angluin's L^* algorithm [Ang87] that originally applies to DFA⁶ machines. This choice is justified by its effectiveness to infer a deterministic transition graph in polynomial time. Its use on a protocol's grammar requires:

- The knowledge of input symbols of the model (also called the vocabulary).
- That clients or servers fail in some obvious way, for instance by crashing, if the submitted sequence of symbol is not valid.
- The capability of resetting clients/servers to their initial states between experiment.
- That we can temporary ensure the protocol's grammar to be deterministic.

⁶DFA stands for "Deterministic Finite Automaton" also known as "Deterministic Finite State Machine".

The inferring process implies a Learner who initially only knows the input symbols extracted from the previous step. It tries to discover $L(M)$ that represents the language generated by the model M by submitting queries to a Teacher and to an Oracle.

Among the possible queries, the original algorithm considers the following:

- **Membership queries**, which consist in asking the Teacher whether a sequence of symbols is valid. The Teacher replies yes or no.
- **Equivalence queries**, which consist in asking if the language of an hypothesized model is equal to the language of the inferring model. If not, the Oracle supplies a counterexample.

This algorithm progressively builds a hypothesized model based on results brought by submitted membership queries. Once the learned model is considered stable by the algorithm, an equivalence query is made. If the query is successful, the model is considered equal to the inferred model, otherwise more membership queries are submitted according to the returned counterexample.

Finally, we consider that environmental variations may impact output symbols. Therefore, we confine the client/server in a restrictive environment to limit its indeterminism on output symbols. For example, we use network whitelist-based filtering to limit Internet accessibility to only identified necessary websites.

Netzob Implementation

This chapter presents the implementation-specific aspects of Netzob, which is **distributed under the GPLv3 license**. At the time of writing (September 2012), the source code comprises about 33.000 lines of code, mostly in Python some specific parts being implemented in C for performance purposes. All algorithms described in this article have been reimplemented.

The source code is publicly available⁷ on a git repository and packages for Debian, Gentoo, ArchLinux and Windows are provided. Currently, Netzob works on both x86 and x64 architectures and includes the following modules (as shown on picture):

⁷See: <http://www.netzob.org>

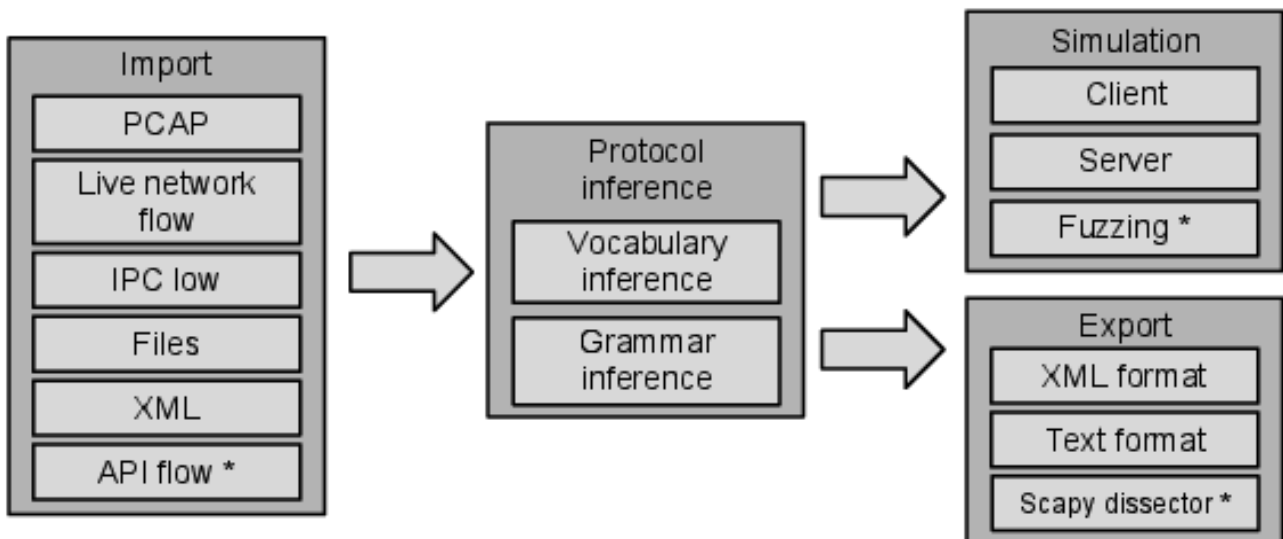


Illustration 5: Architecture of Netzob

- **Import module:** Data import is available in two ways: either by leveraging the channel-specific captors, or by using the XML interface format. As communication protocols are omnipresent, it is fundamental to have the possibility to capture data in a maximum number of contexts. Therefore, Netzob provides native captors and easily allow the implementation of new ones. Current work focuses on data flow analysis, that would permit to acquire clear messages before they get encrypted. Besides, Netzob supports many input formats, such as network flows, PCAP files, structured files and Inter-process communication (pipes, socket and shared memory).
- **Protocol inference modules:** The vocabulary and grammar inference methods constitute the core of Netzob.
- **Simulation module:** One of our main goal is to generate realistic network traffic from undocumented protocols. Therefore, we have implemented a dedicated module that, given vocabulary and grammar models previously inferred, can simulate a communication protocol between multiple bots and masters. Besides their use of the same model, each actors is independent from the others and is organized around three main stages. The first stage is a dedicated library that reads and writes from the network channel. It also parses the flow in messages according to previous protocols layers. The second stage uses the vocabulary to abstract received messages into symbols and vice-versa to specialize emitted symbols into messages. A memory buffer is also available to manage dependency relations. The last stage implements the grammar model and computes which symbols must be emitted or received according to the current state and time.
- **Export module:** This module permits to export an inferred model of a protocol in formats that are understandable by third party software or by a human. Current work focuses on export format compatible with main traffic dissectors (Wireshark and Scapy) and fuzzers (Peach and Sulley).

Conclusion

In this article, we introduced Netzob, an open source tool dedicated to the reverse engineering and simulation of communication protocols. Besides its original goal of inferring undocumented protocols, Netzob is also used in operational contexts in a french security lab. The tool has been successfully applied for the reimplementation of undocumented protocols and vulnerability analysis of proprietary protocols. The project also receives regular external contributions.

As future work, different directions are followed. We are currently addressing advanced fuzzing as an extension of the traffic simulator module. We also aim at generating NIDS rules, that would leverage model checking approach for protocol recognition. This would allow a more efficient detection of botnets than most current approaches, that are based on pattern matching. Netzob will also be extended to support the generation of protocol parsers, allowing the manipulation of inferred protocols in third-party-products. Finally, this community project will continue to support up-to-date academic researches while being available in operational context.

Bibliography

- [Bed04] Beddoe, M. A. (2004): *Network Protocol Analysis using Bioinformatics Algorithms*.
- [Cal.06d] Cui, W., Paxson, V., Weaver, N. C. & Katz, Y.H. (2006): *Protocol-Independent Adaptive Replay of Application Dialog*.
- [Cui07] Cui, W. (2007): *Discoverer: Automatic protocol reverse engineering from network traces*.
- [LMD05] Leita, C., Mermoud, K. & Dacier, M. (2005): *ScriptGen: an automated script generation tool for honeyd*.
- Holzmann, G. J. (1991): *Design and validation of computer protocols*. Prentice-Hall, Inc.
- [NW70] Needleman, S. B. & Wunsch, C.D. (1970): *A general method applicable to the search for similarities in the amino acid sequence of two proteins*. *Journal of Molecular Biology*, 48, 443-453.
- [Smi58] Sokal, R. R. & Michener, C.D. (1958): *A statistical method for evaluating systematic relationships*. *University of Kansas Scientific Bulletin*, 28, 1409-1438.
- [Ang87] Angluin, D. (1987): *Learning regular sets from queries and counterexamples*. *Inf. Comput.*, 75, 87-106.