Learning Behavioral Fingerprints From Netflows Using Timed Automata

Gaetano Pellegrino*, Qin Lin†, Christian Hammerschmidt‡ and Sicco Verwer§
*†§Delft University of Technology, Delft, the Netherlands
‡University of Luxembourg, Luxembourg
Email: *g.pellegrino@tudelft.nl, †q.lin@tudelft.nl, ‡christian.hammerschmidt@uni.lu, §s.e.verwer@tudelft.nl

***Abstract*—We present a novel way to detect infected hosts and identify malware in networks by analyzing network communication statistics with state-of-the-art automata learning algorithms. The automata encode patterns of short-term interactions in known malicious hosts, and are used to obtain small but effective fingerprints of machine behavior. We showcase the effectiveness of our system, named BASTA[1] (Behavioral Analytics System using Timed Automata), on a public dataset containing Netflow traces of real-world botnet malware. Compared to a deep packet inspection of communication content, Netflows are easy and cheap to collect and analyze, and preserve a greater degree of privacy. Even though the high level of abstraction in Netflow data makes it more difficult to utilize it, BASTA shows very impressive results achieving high accuracy in several settings while returning few false positives. It is also capable of detecting infections of previously unseen malware.**

I. Introduction

Botnets pose a significant threat to cyber-security. Bots are zombie computers, remotely controlled by a malicious entity, and are used for attacks, spam, phishing and information exfiltration [1], [2]. Despite recent research, detecting and countering botnets is still considered an unsolved problem [3]. In Feily's survey [4], three categories of botnet detection methods are distinguished: signature-based [5], anomaly-based [6] and DNS-based [7], [8]. In signature-based detection, some characteristic like hashes are calculated, either on the malware binary, or from resource usage or from packet content capture. These characteristics serve as signatures used to identify the same malware or packets in the wild. Botnet developers counter this detection using techniques such as code obfuscation, encryption, and polymorphic code [9], makings signature-based detection increasingly ineffective. DNS-based detection techniques rely on anomalies in the DNS traffic, caused by the infected hosts' need to locate and communicate with a command and control server, which is usually hosted by a Dynamic DNS provider. This type of detection often wrongly considers hosts as malicious, e.g., due to fake-domains and reconnaissance poisoning [4]. These false positives make DNS-based techniques fairly unreliable. The last category, anomaly-based detection methods are often behavior-based and monitor the run-time execution behavior of malware, which is much more difficult to conceal [10]. Consequently, there has been a large amount of research devoted to the development of effective behavior-based malware detection and analysis tools, see, e.g. [11], [12]. Behavior-based malware detection or analysis applies machine learning techniques in order to automatically learn models from data such as network traffic. In state machine learning, an instance of generative learning, is of particular interest: it can detect a botnet in new data, but its generative property also allows to infer and analyze the logical structure underlying the observed traffic. Depending on the data, in some cases it is even possible to infer a state machine diagram communication protocol used by the botnet, see [13].

In this paper, we introduce BASTA (Behavioral Analytics System using Timed Automata), which uses probabilistic deterministic real-time automata (PDRTAs) to obtain identity fingerprints of hosts from timed network traffic streams. It is a behavior-based system, and learns models from Netflow traces instead of full packet contents. Packet content is typically used as information source to identify the basic event types/messages used in communication protocols. Netflows only contain information on the sources and targets of flows, the amount of data transmitted, the network protocol used, and the timing of the flows. This makes it much harder to infer a botnet's communication protocol. If successful, however, this approach opens up the road to many new applications of this technology because Netflow traces are widely available, while access to the content of messages is typically restricted due to proprietary or privacy related issues.

BASTA models specify behavior over timed events. We are especially interested in this timing information because it can be very important for determining network traffic behavior [14], [15]. In addition, PDRTAs can be learned efficiently from unlabeled data [16], making them an ideal candidate for modeling network traffic behavior. The development of BASTA is driven by the practical need of network administrators that run a wide network composed of many hosts that require monitoring. A common action taken after identifying an infection is a hard reset of the given machines in order to restore it to a trusted state. This operation is often an expensive one. In contrast to most Intrusion Detection Systems (IDS) that label individual packets/flows as suspicious, BASTA is focused on ranking hosts using an indicator of suspiciousness based on all of the outgoing and incoming flows. This indicator models the overlap in communication behavior between a given candidate host and a known infected machine. A low indicator means that frequencies of behavioral patterns

[1]Meaning "enough" in Italian.

observed in the entire set of flow records match the expected frequencies obtained from a known infected machine. A high indicator means that these frequencies do not match, and thus that the traffic shows no sign of this infection. Although we initially developed BASTA in order to detect such known infections, we demonstrate in this paper that many of the patterns that it learns from Netflow records are generic: initial results demonstrate that BASTA is capable of detecting new infections from unknown malware.

Our contribution is BASTA, a system encompassing:

- a technique to learn PDRTAs from Netflow data;
- a formal definition of behavioral fingerprints, and methods to obtain fingerprints for malware detection;
- two methods for monitoring PDRTA fingerprints in new Netflow data: detecting occurrences of suspicious states (called fingerprint-based) and comparing observed frequencies with expected frequencies (called error-based);
- a method to use fingerprints as an indicator to rank hosts according to their suspiciousness;
- an empirical study on effectively using fingerprints generated from PDRTAs on botnet data for detecting infections in real data and synthetic data using a public dataset.

In our empirical study on publicly available Netflow data, we obtain a high detecting rate, catching 100% of infected hosts in some scenarios, while maintaining a low false positive rate. These results are surprising, considering the low detection rates that have been reported using existing techniques on the same data [17]. Although the obtained results are not directly comparable due to the fact that these existing methods try to label every individual flow, our results do indicate that assigning labels to hosts rather than flows is a very promising direction for botnet detection. In addition, they show that timed automata are very effective tools for capturing the behavioral patterns in all of this data. In noised settings, the default settings used by BASTA can produce too many false positives to be used by network administrators directly. Since BASTA is a ranker of hosts, however, an admin can opt to only inspect the most suspicious ones, i.e., the hosts that are most likely malicious. Furthermore, since BASTA is a machine learning tool geared toward learning useful models from network traffic, it can be used as a replacement of standard machine learning methods used by malware detection frameworks such as for instance DISCLOSURE [18]. When using BASTA as a base detection system, such frameworks will continue to make use of many other tricks such as filters on IP range and port usage in order to further reduce the number of false alarms.

The remainder of this paper is organized as follows. We discuss related work and provide an example use-case as motivation for BASTA in Section II. We introduce PDRTAs in Section III together with a brief discussion of the learning algorithm for identifying PDRTA from positive data. Section IV introduces our infection fingerprinting system based on PDRTAs, and Section V addresses its evaluation on real Netflow data samples. Section VI presents our conclusions.

II. Related Work

Previous work on botnets ranges from tracking and identification to infiltration and take-overs of botnets. Tracking using behavioral models can make use of a variety of data. Recently, Barabosch [19] observed behavior of botnet malware on a system level, including CPU and memory usage as well as kernel events and processes. In very related work, Babić et al. showed how to capture and analyze malware behavior observable in system call dependency graphs as tree automata [20]. If a physical, or at least system level access to a botnet host is not available, it might be possible to interact with an infected host. A prime example of such work was by Cho et al. [13], who inferred a Mealy machine model of the C&C communication protocol of the MegaD botnet through interaction with a C&C server. In addition, they demonstrated how formal analysis can be applied to the inferred model for botnet defense. If neither system level access nor direct interaction is possible, network communication can be observed. Instead of relying on analyzing packet content, summary statistics of communication can be used. Cisco's Netflow protocol has attracted a wide interest in academia as well as industry and led to applications ranging from network monitoring and measurement to application detection and intrusion detection [3]. A very related work is [18], where the authors described a complete, Netflow based, botnet detection framework called DISCLOSURE. It was composed by several different modules, including a detection module based on Random Forest, filters that remove uninteresting traffic, and a false positive reduction module. BASTA may be thought of as an alternative to the detection module that is geared towards modeling network traffic. Reverse engineering of protocols from passively collected network traces can be found in the work of Comparetti et al [21], who presented the Prospex tool for learning deterministic finite state automaton (DFA) models. Leita et al. developed ScriptGen for the automatic construction of scripts for HoneyD [22] based on learning simple finite state machines from network traces. These scripts were then used to emulate services and fool attackers into believing they connect to genuine ones. Wang et al. [23] and Krueger et al. [24] learnt probabilistic variants of state machines (Markov chains and hidden Markov models (HMM) respectively) from network traces, which can be visualized to aid reverse engineering and used to detect anomalies. In [25], it was shown how HMMs learned from network packet traces can be used to identify distributed denial of service attacks. Beyond analyzing the individual flows of network traffic, communication graphs can be extracted from origin and destination fields of Netflows and analyzed by themselves. In Nagaraja [26], a topology was extracted from observed botnet traffic and a graph-theoretic technique was applied to the communication graphs to extract bots communicating peer-to-peer. In [27], properties of the communication graphs are analyzed and used to attack peer-to-peer botnets.

III. State Machine Learning

State machines are key models for the design and analysis of computer systems [28]. Learning such machines from software data dates back to 1998 [29]. The formal problem of learning automata from data has been studied much longer [30]–[32]. It is one of the best studied problems in grammatical inference and many learning algorithms have been developed [33]. Some of these have been used for malware detection and analysis [13], [21]–[24]. These algorithms require discrete events as input. To obtain events from network packets or system calls generated by malware, either the messages need to be reverse engineered, e.g. using [34], or clustered into abstract events, see, e.g., [23], [24]. The resulting symbolic sequences are then provided to a state machine learner that will learn the behavioral structure underlying these sequences.

A. PDRTAs

PDRTAs are a probabilistic version of Real Time Automata [35], and they are probabilistic automata that include guards on the transition timings (inter-event) times. Formally, the events are modeled by timed strings $(a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$, where t_i denotes the time delay between the occurrences of the ith and $i-1$th events. The PDRTA model defines a probability distribution over such timed strings, having a Markov property in the distribution over events, and a semi-Markov property in the time guard.

Definition 1: A PDRTA is a 4-tuple $\langle \mathcal{A}, \mathcal{E}, \mathcal{T}, \mathcal{H} \rangle$, where

- $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0 \rangle$ is a 4-tuple defining the machine structure: Q is a finite set of states, Σ is a finite set of event types (symbols), Δ is a finite set of transitions, and $q_0 \in Q$ is the start state;
- $\mathcal{E}$ and $\mathcal{T}$ are the event and time probability distributions, respectively. $\mathcal{E} : (Q, \Sigma) \to [0, 1]$ returns the probability of generating/observing a given event in a given state. $\mathcal{T} : (Q, H) \to [0, 1]$ returns the same but for a given time range $[v, v'] \in \mathcal{H}$, where $\mathcal{H}$ is a finite set of non-overlapping intervals in $\mathbb{R}_+$.

A transition $\delta \in \Delta$ in a PDRTA is a tuple $\langle q, q', a, [m, m'] \rangle$, where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol and $[m, m']$ is a temporal guard.

Figure 1 illustrates a PDRTA inferred from Netflow data. On the timed string $\langle Q - TCP, 500 \rangle \langle TCP, 50 \rangle \langle TCP, 200 \rangle$, it goes from state 1, via state 4 and 7, to state 2, its probability is computed as: $0.28 \cdot 0.21 \cdot 0.59 \cdot 1.0 \cdot 0.86 \cdot 1.0 = 0.03$. It can also represent a distribution over $(\Sigma, \mathcal{H})^*$ by adding final probabilities.

Note that in a PDRTA the states are defined by their event-time value distributions and their transitions to future states. These cannot be directly observed in data but have to be estimated using a learning method. PDRTAs are therefore similar to a timed variant of the HMM [36].

We learn PDRTAs instead of regular automata or Markov models because time information is important for characterizing network traffic. In PDRTAs, the influence of time values on the string probabilities is equal to that of all the other data

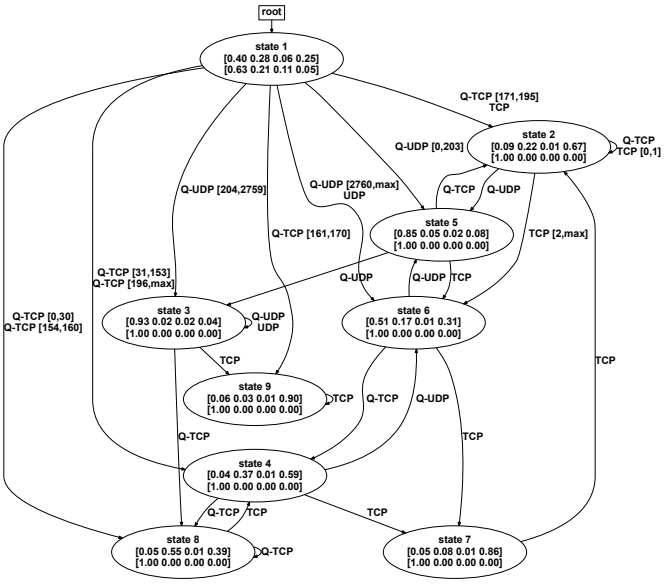

Fig. 1. A PDRTA inferred from a sample of malicious Netflow traces in Scenario 9 of our dataset. States 2, 3, and 9 are spamming states. The botnet initiates many Quick (short duration) UDP/TCP flows. Edges are labeled with events and temporal guards. The latter are omitted if they are empty. The states contain two distributions: one for events, one for time intervals. The 4 event types in order: Quick UDP, Quick TCP, Other UDP, Other TCP. Breaks-points of the time intervals are: 485, 981, and 1660 ms.

contained in abstract events. Other types of stateful models such as Hidden Markov Models or Mealy Machines have been used for detection purposes [17], [21], but cannot be used to infer time constraints. We focus on deterministic automata because identifying non-deterministic automata is harder [37].

B. Learning PDRTAs

We use a recent state machine learning algorithm, RTI+ [16] to learn malicious behaviors from Netflow data, and then use these as fingerprints for detection.

Here we briefly review this algorithm. RTI+ is based on state-merging [38]. An untimed probabilistic state-merging algorithm starts by building a large tree-shaped automaton called prefix tree from a sample of input strings. Every state of this tree can be reached by exactly one untimed string and therefore encodes exactly the input sample. The algorithm then greedily merges pairs of states (q, q') in this tree, forming a smaller and smaller machine. When the target machine is deterministic, for every event $e \in \Sigma$ the states reached from q and q' have to be merged as well (the determinization process). By iteratively applying these merges, the algorithm generalizes over the sample and learns the structure of the target machine used to generate the sample. The algorithm uses a heuristic to decide merges and avoids overfitting. In an unsupervised setting, the merge heuristic is determined using statistics. A merge between q and q' is considered good if the future behavior after reaching q is similar to that after reaching q', which can be tested using, e.g., a likelihood-ratio test [16]. This essentially tests the Markov property, i.e., whether future behavior is independent of being in state q or q'. When these futures are significantly different, the merge is considered

inconsistent and will not be performed. In addition to state merges, RTI+ is capable of performing transition splits [16]. In the prefix tree, the temporal guards include all possible time values. A split of a transition $\delta = \langle q, q', a, [m, m'] \rangle$ at time point t creates two new transitions $\langle q, q_1, a, [m, t] \rangle$ and $\langle q, q_2, a, [t+1, m'] \rangle$. The target states q_1 and q_2 are the roots of two new prefix trees that are reconstructed based on the input sample. In this way, RTI+ can learn temporal constraints in addition to the machine structure. For more details such as pseudo code and complexity analysis, the reader is referred to [36].

IV. Intrusion Fingerprinting

Blacklists, containing addresses of known malicious hosts detected over time, are one of the simplest and most common tools network administrators have and use. From passively observing hosts on these lists, it is possible to learn behavioral models for any communication between hosts within the network under observation (NUO) and a known blacklist entry.

We consider all blacklisted hosts as infected by some malware. By using models learned with BASTA from such known infected hosts, we can detect malicious behavior in communications between either trusted partners, or trusted partners with an unknown source.

This achieves two equally important goals. Firstly, we can discover new malicious hosts outside of the NUO, allowing the network administrator to extend the blacklist by additional entries. Equally important, we detect potentially infected hosts within the NUO, effectively using the model as a behavioral fingerprints generator. We call this use case "Infection Fingerprinting". In the following, we describe our methodology for obtaining behavioral fingerprints of infections. Our goal is to identify a PDRTA from the Netflows coming from a known infected host, and then to use this model for creating network-specific behavioral fingerprints. To learn PDRTAs on Netflow data, it is necessary to convert the data into a set of timed strings. Such strings are constructed in two steps: first every Netflow record is mapped to a event-time value pair, and second the resulting sequence is decomposed into a set of small subsequences captured at regular and predetermined time intervals. We then use RTI+ to learn a PDRTA.

A. Netflow Data

Netflows are sequences of packets passing on a given network link, from a source host to a destination host. As such, they are univocally defined by the couple (source-address, destination-address) and characterized by several properties derived from the aggregation of packet-based features. The Netflow features we use in our system are listed in Table I.

We use Netflow records instead of packet captures because these are frequently logged by network operators and much easier to obtain. Furthermore, they preserve privacy of the communication: in contrast to network packets, Netflows do not contain the content or format of messages. The downside of learning from Netflows is that the learned machines define behaviors on a high abstraction level. In the following sections, we show that these high level machines are very powerful behavioral models, capable of detecting other infected hosts with very few false positives.

TABLE I
Netflow features, with type and examples.

FEATURE	TYPE	VALUES
source-ip	string	147.32.84.193
start-time	timestamp	2011-08-17 15:51:08.499
protocol	string	TCP, UDP
duration	float	0.103, 2.696
direction	string	$\rightarrow$, $\leftarrow$, $\leftrightarrow$
total-packets	integer	9, 1
total-bytes	integer	1030, 66, 43

B. Obtaining Timed Events

Clustering of Netflows basically consists of assigning a numerical code to each flow based on the features in Table I such that similar Netflows receive the same code. We obtain these using a simple attribute mapping for each feature. Protocol type and direction are assigned a progressive non-negative number for every possible value v, for example, for protocol type, we assign 0 if v = TCP, 1 if v = UDP, 2 if v = ICMP, etc. The source-ip feature is only used to distinguish flows from each other, and the timestamp is used to compute time values. We use percentiles for clustering other numerical features, i.e. duration, total-packets and total-bytes. The ELBOW methods are applied to select the "optimal" number of bins [39]. The experiments show that within-cluster sum of squares (WCSS) has a "break point" at number of cluster-5, i.e. 20th, 40th, 60th, and 80th percentiles. We assign values accordingly: 0 if v is before the 20th percentile, 1 if v is after the 20th and before the 40th, etc. We then compute the event types from Netflows using Algorithm 1, where $M_i : v \rightarrow \mathbb{N}$ denotes the attribute mapping for feature i.

Algorithm 1: Netflow encoding using attribute mappings:

Input: a Netflow $n = \langle a_0, a_1, \cdots, a_k \rangle$ with k features
Input: an attribute mapping M_i, $i = 0, 1, \cdots, k$
Output: integer code for n
$code \leftarrow 0$;
$spaceSize \leftarrow \prod_{i=0}^{k} |M_i|$;
for $i \leftarrow 0$ ***to*** k **do**
  $code \leftarrow code + M_i(a_i) \times \frac{spaceSize}{|M_i|}$;
  $spaceSize \leftarrow \frac{spaceSize}{|M_i|}$;
return *code*;

Algorithm 1 uses attribute mappings to encode a Netflow Note that $|M_i|$ denotes the number of values for feature a_i. For example, a simplified scenario where every Netflow has only two features: protocol and total-packets. For the protocol we observe only two possible values, namely TCP and UDP, and the attribute mapping will assign 0 to the former and 1 to the latter. For the total-packet feature let assume we have gathered values: $\{1, 1, 1, 5, 12, 14, 14, 18, 23, 31\}$ and assume we are interested in the 20th and 80th percentiles. We first need to find out the ordinal ranks of such values in the above collection, using the formula: $r(p) = \left\lceil \frac{p}{100} \times N \right\rceil$, where

p is the required percentile, and N is the collection size. Therefore $r(20) = \left\lceil \frac{20}{100} \times 10 \right\rceil = 2$ and $r(80) = \left\lceil \frac{80}{100} \times 10 \right\rceil = 8$. Percentiles are the collection values corresponding to the ordinal ranks, thus the 20th percentile is 1 and the 80th percentile is 18, they induce an attribute mapping to $M_{total-packets}(v) = \{0 \text{ if } v \leq 1, 1 \text{ if } 1 < v \leq 18, 2 \text{ else}\}$. where v is the total-packets attribute value for any given Netflow . Hence the code associated to the instance $\langle$TCP, 14$\rangle$ is: $0 \times \frac{6}{2} + 1 \times \frac{3}{3} = 1$. And the code assigned to the instance $\langle$TCP, 33$\rangle$ is 2. Time values are obtained from Netflow data by calculating the delays between to consecutive events. For each host of interest, we compute the time differences of its consecutive flows in milliseconds. Table II shows the results of this process applied to five example Netflow records.

C. Sliding Timed Frames

As mentioned in Section III-B, RTI+ learns PDRTAs from sets of timed strings. In this section we address the task of obtaining timed strings from Netflows , achieved in two stages: At first stage we group Netflows by source address (source-ip feature in Table I) because we are interested in modeling per-host behavior. By doing so, we collect a plain sequence of Netflows for each monitored host, which is translated in timed events as showed in the previous section. At second stage we slide a time window of fixed duration over the sequence obtained at stage one. For every window w, we create a timed string by concatenating all events that occur within the duration of w. The window duration used in the experiments is 20 milliseconds, which creates two flows from the data in Table II: $(1, 0)(52, 5)(150, 12)$ and $(52, 5)(150, 12)(150, 2)(44, 5)$. Each flow represents a snapshot of 20 milliseconds of timed events produced by a given host, which we call a "Sliding Timed Frame".

D. Recognizing a Host as Infected

After obtaining a PDRTA A from a malicious host M, we use it to evaluate other hosts C. Intuitively, we compare the expected behavior of a malicious host M, given by the model A with the observed behavior in C. A symptom of C describes how the behavior of C fits the behavior of M. Formally, a symptom is an internal state of A together with the input needed to reach it:

Definition 2: An infection symptom is a couple $\langle q, t \rangle$, where

- $q \in Q_M$ is a state of $\mathcal{A}_M$, PDRTA learned from a given blacklisted entry M; q is the state reached in $\mathcal{A}_M$ after receiving t;
- $t = \langle s, \delta \rangle$, is a timed event with $s \in \Sigma_M$, the set of event types for $\mathcal{A}_M$, and $\delta \in \mathbb{N}$, produced by a given monitored host C;

Every infection symptom is a behavioral fingerprint for a monitored host, given the infection represented by a blacklisted entry. Such fingerprints have a cross-network component (q) obtained from M, and a network-specific component (t), generated from Netflows captured within the NUO. Indeed one might imagine two or more networks to share the same behavioral model M for a specific infection, and still to be able to generate infection symptoms which are specific for each of them. That is why we refer to PDRTA models as *fingerprint generators*.

It is important to underline once more that infection symptoms are generated partially by using a PDRTA learned on Netflows coming from a malicious source (training Netflows data), and partially from data gathered on the monitored network (evaluation Netflows data). For instance, by considering the PDRTA shown in Figure 1 with the following frame, gathered in our network, as input: {(Q-UDP, 171) (Q-TCP, 8) (Q-TCP, 0) (Q-TCP, 0)(Q-TCP, 0) (Q-TCP, 0)}. The symptoms are: $\langle$STATE-5, (Q-UDP, 171)$\rangle$ and $\langle$STATE-2, (Q-TCP, 8)$\rangle$ with 1 occurrence and $\langle$STATE-2, (Q-TCP, 0)$\rangle$ with 4 occurrences.

Once we have learned a PDRTA as a fingerprint for a given malicious host M, we use two different strategies for finding the same infection in a new host C in newly observed data. Both strategies rely on infection symptoms.

Our first strategy, called *error based*, compares the infection symptoms with occurrence counts of the same fingerprints in new data. We thus compare whether a new candidate host C shows the same symptoms as a known malicious host M. Let $Counts_i^M$ and $Counts_i^C$ be counts of symptom i in M and C, respectively. Host C is classified as infected if the absolute error $S = \sum_i |Counts_i^M - Counts_i^C| < \tau$, i.e., if absolute differences between the expected and observed symptom counts fall below a a pre-computed threshold. This threshold is obtained using a configuration dataset of known benign hosts, see Section V. The absolute error S can be used as a score function. Using it, we can rank i different hosts C_i according to suspiciousness C_i. The second strategy, called *fingerprint based*, uses this configuration dataset to find distinguishing symptoms that occur when a host is malicious, but never when it is benign. $Counts_i^F$ denotes the sums of all symptom counts in the configuration set.

Host C is then classified as malicious if there exists a symptom i such that $Counts_i^F = 0$, $Counts_i^M > 0$, and $Counts_i^C > 0$.

E. Circumventing BASTA

At present, BASTA should not be considered as a complete system for the detection of botnets. It has rather been designed as a machine learning engine of a more complex detection device (e.g., random forest module in [18]). Having said that, it is good to clarify that BASTA is a botnet fingerprinting system. This means you need traffic from an infected host known as such, to get a PDRTA representing its behavior and the fingerprint generator. It is also important to point out that thanks to these generators, fingerprints have the desired property of being specific for the network. In particular, to circumvent the detection of BASTA, a bot master should have access to the traffic of the network he intends to attack in order to craft elusive flows and bypass its fingerprints.

V. Evaluation

BASTA is evaluated on a dataset released by Garcia et al [17]. The dataset is organized in 13 different scenarios,

TABLE II
NETFLOW EVENTS FROM SCENARIO 9.

prot	dir	time	duration	packet	byte	event	prot	dir	time	duration	packet	byte	event
udp	→	0	0.000304	1	68	(1,0)	tcp	↔	24	0.120181	6	212	(150,2)
udp	↔	5	0.000442	5	590	(52,5)	udp	→	22	0.17121	10	7701	(44,5)
tcp	↔	17	0.000527	3	479	(150,12)	...	...	...	...	...	...	...

each of them containing Netflows from a network infected by a different type of malware. The scenarios are numbered from 1 to 13, and referred to by their number. The goal of BASTA is identification of other infected hosts knowing at least one, e.g. from a blacklist. Our experiments focus on the four scenarios containing multiple bots running the same malware. Using the same scenario numeration as in [17], Scenario 9 contains a network infected by Neris, a spamming botnet that operates though IRC (Internet Relay Chat) and is capable of performing port scanning and click frauds. Scenario 10 and 11 contain a network infected by Rbot, a botnet capable of leading distributed denial of service attacks (DDoS). The UDP and ICMP protocol are used respectively. Scenario 12 is a network infected by NSIS.ay, a trojan capable of coordinating DDoS attacks.

In [17] several existing botnet detection methods are compared on this dataset. The different methods are trained on samples from Scenarios 3, 4, 5, 7, 10, 11, 12, and 13, and evaluated on Scenarios 1, 2, 6, 8, and 9. The purpose of this setup is to test whether the methods are able to generalize from one botnet to another. In addition, the methods are evaluated on how quickly they can detect new threats. Understandably, the methods perform poorly on this task, sometimes even worse than a baseline that labels all flows as malicious.

For a network administrator it is much more useful to have labels assigned to hosts instead of individual Netflow records because hosts can be investigated and reset to a trusted state. We therefore focus on labeling and ranking hosts.

Our initial goal is to detect new infections of known threats. A network administrator can learn models for such threats using Netflows that connect to known blacklisted hosts. Afterwards, we consider the setting used in [17] of trying to detect infections by unknown threats.

We preprocessed the data, removing null values if present, and accounted for discrepancies in date formats across multiple scenarios. We also removed all Netflows labeled as background as our use-case is a binary problem: discovering whether a given host has or not has been infected by a malware. For each scenario we constructed three disjoint datasets. Table III summarizes the sets.

- The configuration dataset, containing 30% of the sequences, randomly selected. It used for calculating percentiles in features with numeric domain. It is also used for estimating the selectivity threshold for the error based strategy and for identifying the distinguishing symptoms used in the fingerprint based strategy. It does not contain Netflow sequences from botnet hosts.
- The training dataset, containing all Netflows coming from one of the infected hosts.
- The evaluation dataset, containing the remaining sequences from the scenario: both infected and benign Netflows from all other hosts. The evaluation dataset sometimes is much bigger than the training set. The reason for this is in how we learn: model is learned from just one host while the objective is to detect other infected machines in a big network.

TABLE III
THE NUMBER OF FLOWS (HOSTS) OF EACH OF THE THREE SET PER SCENARIO.

Scenario	Configuration Set	Training Set	Evaluation Set	Infected Hosts
9	91386 (185)	29712 (1)	648627 (1077)	10
10	159995 (141)	19889 (1)	465462 (380)	10
11	1004 (32)	138077 (1)	149821 (87)	3
12	6506 (4)	807 (1)	2483 (18)	3

In all the experiments, the attribute mapper has been initialized using the configuration dataset with 20%, 40%, 60%, 80% as percentiles for all scenarios except for Scenario 11, where only the 50% percentile was used in order to avoid overfitting on a small training dataset. Regarding the selectivity threshold used in the error based strategy, it has been estimated by collecting the sum of errors for each host in the configuration set and computing the average error μ along with the standard deviation σ. The threshold has then been set to $\tau = \mu - 2\sigma$.

The following subsections discuss two different types of experiments. Section V-A shows performances by taking each scenario individually. These experiments illustrate the capabilities of the system in the task of detecting a host infected by a known threat. Section V-B shows an experiment involving all scenarios together. This experiment aims to assess BASTA performance with threats it does not know anything about, i.e. it cannot rely on a model learned on purpose for such a menace.

A. Single Scenario Simulation

All experiments in this section share the aim of evaluating the system in detecting an already known infection. A known infection is a host known to be infected by prior knowledge, e.g. through a blacklist. In each experiment we compare results with a BIGRAMS baseline.

BIGRAMS are essentially sequences of two consecutive events. For instance, if we consider the stream in Table II, we can get the following: $(1, 52), (52, 150), (150, 150), (150, 44)$. This baseline does not use any serialization, i.e. no sliding frames, no time information, just the labels. With BIGRAMS it is possible to estimate the conditional probability of the events. If we have the bigram (E_1, E_2) where E_1 and E_2 are consecutive events, the conditional probability $P(E_2|E_1)$ can be estimated by computing the join probability of $(E1, E2)$ and the marginal probability of $E1$. Performance is presented in terms of true/false positives/-negatives, where a true positive (TP in the tables) means a host correctly classified as malicious.

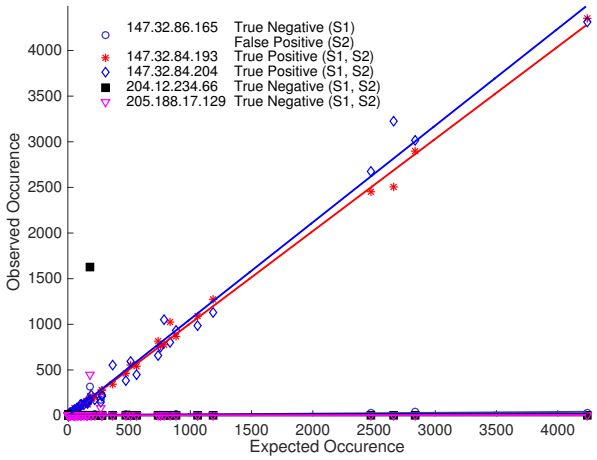

Fig. 2. Expected and observed frequencies of infection symptoms in Scenario 9. Each $x-y$ pair indicates the expected frequency count versus the observed frequency count of a specific infection fingerprint for one of five given hosts. Regression lines are also drawn for each host. For true negatives, we observe low counts while a fingerprint of an infection expects high counts. The regression line is along the x axis. For true positives, the observed and expected counts match, and the regression line is $y = x$.

Noiseless sources, noiseless targets: Table IV refers to a first setting where both source and targets are noiseless. In this context the word "noiseless" is used to express that every Netflow coming from any evaluation machine is legitimate if and only if such a machine is legitimate, and every Netflow coming from any evaluation and infected machine is actually malicious as well. The results on three out of four scenarios are impressive: 100% accuracy for the error based strategy and few false positives for the fingerprint based strategy. Scenarios 11 and 12 show somewhat worse results. One host in Scenario 11 is falsely identified as benign (one false negative) out of the two malicious hosts in the testing data. There is also an improvement in comparison with BIGRAMS on Scenarios 11 and 12, where the baseline is not able to detect any infected host. BIGRAMS seem to work better with the fingerprint based strategy, introducing less false positives and hitting all the malicious machines. Figure 2 illustrates the expected and observed frequency distribution in each testing hosts. $S1$ and $S2$ are the error based and fingerprint based strategies, respectively. Two hosts are detected correctly as infected (TP) by $S1$ as their regression plot is very close to $y = x$. The other three hosts are correctly detected as safe (TN). Interestingly, one host is incorrectly detected as infected by $S2$. Although this host is behaviorally very different from what the PDRTA model expect, it shows malicious behavior that never occurred in the configuration dataset, in this case the symptom $\langle s = 1, t = \langle 3, 4 \rangle \rangle$ (see Figure 1).

Noiseless sources, noised targets: Results in Table V are about a different setting where the sources for each scenario are still noiseless, but the infected evaluation targets are not. This is a more realistic situation where a fingerprint generator, namely the PDRTA, of an infection has been provided by some partners (e.g. security companies) able to run the infection

TABLE IV
TOP: ERROR BASED STRATEGY PERFORMANCES ON NOISELESS DATA. BOTTOM: FINGERPRINT BASED STRATEGY PERFORMANCES. THE TABLE REPORTS HOSTS CORRECTLY AND INCORRECTLY IDENTIFIED IN ABSOLUTE NUMBERS.

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	1068	0	0	9	1068	0	0
10	9	371	0	0	9	371	0	0
11	1	85	0	1	0	85	0	2
12	2	16	0	0	0	16	0	2

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	1038	30	0	9	1066	2	0
10	9	370	1	0	9	369	2	0
11	1	80	5	1	1	83	0	2
12	2	4	12	0	2	15	1	0

TABLE V
TOP: ERROR BASED STRATEGY PERFORMANCES WITH NOISELESS SOURCES AND NOISED TARGETS. BOTTOM: FINGERPRINT BASED STRATEGY PERFORMANCES. THE TABLE REPORTS HOSTS CORRECTLY AND INCORRECTLY IDENTIFIED IN ABSOLUTE NUMBERS.

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	4	1244	0	5	1	1244	0	8
10	0	503	0	9	0	503	0	9
11	1	115	0	1	1	115	0	1
12	0	18	0	2	0	18	0	2

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	1219	25	0	9	1244	0	0
10	9	503	0	0	9	503	0	0
11	1	111	4	1	1	114	1	1
12	2	6	12	0	2	17	1	0

in a safe environment. However, the target infected machines present mixed traffic made of both Netflows labeled as malicious and legitimate. We have devised this setting coupling each malicious host with the most verbose (in terms of number of NetFlows), legitimate, and available one. Then we have merged Netflows of the coupled host so creating new targets with mixed behavior. Even in this case we observe no false positive with the error based strategy, and an improvement in Scenario 9 on the baseline. BIGRAMS with fingerprint based strategy show better performance than the alternative system, confirming the trend of the previous setting.

TABLE VI
TOP: ERROR BASED STRATEGY PERFORMANCES WITH NOISED SOURCES AND NOISED TARGETS. BOTTOM: FINGERPRINT BASED STRATEGY PERFORMANCES. THE TABLE REPORTS HOSTS CORRECTLY AND INCORRECTLY IDENTIFIED IN ABSOLUTE NUMBERS.

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	2	1244	0	7	1	1244	0	8
10	2	503	0	7	2	503	0	7
11	2	101	14	0	1	98	17	1
12	0	17	1	2	0	18	0	2

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	1228	16	0	1	1244	0	8
10	9	503	0	0	9	503	0	0
11	1	98	17	1	1	89	26	1
12	2	5	13	0	2	17	1	0

TABLE VII

TOP: ERROR BASED STRATEGY PERFORMANCES ON NOISELESS DATA INCLUDING BACKGROUND FLOWS. BOTTOM: FINGERPRINT BASED STRATEGY PERFORMANCES. THE TABLE REPORTS HOSTS CORRECTLY AND INCORRECTLY IDENTIFIED IN ABSOLUTE NUMBERS.

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	237381	0	0	9	237381	0	0
10	9	129596	0	0	4	129596	0	5
11	1	20609	0	1	0	20609	0	2
12	0	41687	0	2	0	41687	0	2

	BASTA				BIGRAMS			
	TP	TN	FP	FN	TP	TN	FP	FN
9	9	237340	41	0	9	237320	61	0
10	9	129594	2	0	9	129592	4	0
11	1	20607	2	1	0	20607	2	2
12	2	41667	20	0	2	41668	19	0

Noised sources, noised targets: The last experiment, whose results are reported in Table VI, combines noised sources (obtained with the same previously described coupling procedure) and noised targets. Even with a comprehensible drop in performance compared with already described experiments, the error based strategy still shows better performance on all scenarios but 12. In this case both systems are unable to detect any infected machine, but only the alternative system returning a false positive. Hence, even with the fingerprint based strategy,the trend of the previous setting is unconfirmed and BASTA achieves better results on Scenario 9 and 11 making the comparison with the baseline more uncertain.

Including Background Traffic: In Section V we explained why, as a pre-processing stage, we decided to filter out the background traffic from the dataset at our disposal. On one hand we don't know the actual label of every background flow, i.e. whether it is malicious or not, posing more than few questions (e.g. how to evaluate a legitimate host which includes some background flows). On the other hand, by including background traffic, we reproduce a more realistic setting in which we are interested to test BASTA performance. Finally we decided to plan additional experiments on single scenarios, with noiseless data, including background traffic. We treated background flows in a neutral way: if a legitimate host contains background flows it remains legitimate, and hosts containing background flows only are considered not malicious. Results of our experiments are showed in Table VII, where the trends of the same experiment without background traffic (Table IV) have been confirmed.

B. Multi Scenario Simulation

We also test our system on the setup described in the paper [17] published with the Netflow dataset. The training set consists of samples from scenarios 3, 4, 5, 7, 10, 11, 12, 13, and the evaluation sets contains hosts from scenarios 1, 2, 6, 8, 9. It is important to mention that we selected training and evaluation scenarios as in the corresponding experiment in [17]. The scenarios in the training set contain different botnet families than the evaluation set and can therefore exhibit very different behavior. We learned PDRTAs for all malicious hosts in the training samples, and tested whether any of them mark any of the hosts in the testing scenarios as malicious.

TABLE VIII

PERFORMANCE ON THE SETUP IN [17] WITH ERROR BASED STRATEGY (TOP) AND FINGERPRINT BASED STRATEGY (BOTTOM). SCENARIOS 1,2,6, AND 8 ONLY CONTAIN A SINGLE MALICIOUS HOST, SCENARIO 9 CONTAINS 10 MALICIOUS HOSTS. OVERALL, THE EVALUATION SET CONTAINS 3087 HOSTS.

Scenario	1	2	6	8	9
TP	1	1	1	1	10
FP	3	73	55	84	109
TP	1	1	1	1	10
FP	4	26	12	52	72

In spite of our method focusing on behavioral detection, the fingerprint based method produces encouraging initial results, see bottom half of Table VIII. Overall our system detects all the infected hosts and produces 166 false positives out of 3072 benign hosts. The error based strategy is able to detect all 14 infected hosts, but the performance are deteriorated by an almost doubled amount of false positives (i.e. 324, see Table VIII, top half).

VI. CONCLUSIONS

We presented BASTA, a system using the RTI+ algorithm for learning PDRTA to learn behavioral models from Netflow traces. It has a simple but effective technique to obtain timed strings from Netflow data using attribute mappings and sliding windows over the flows. We presented two strategies for detecting new infections: error based and fingerprint based. Both strategy can effectively detect known infections, as proved in our experiments on single scenarios from the public dataset from [17]. In fact, In our experiments with detecting known infections, BASTA detected nearly all infected hosts with very few false positives. To increase the difficulty, we mixed traffic from and to different hosts. In this setting, BASTA is capable of detecting threats, too. Finally, in the very difficult setting from [17] (learning from one type botnet to detect another type), BASTA also detects threats albeit with some false positives. Administrators can adjust and reduce this number by using different thresholds, although this costs some true positives. In practice, choosing fewer false positives over maximizing true positives is seen as desirable. BASTA can also replace the other machine learning tools used by existing detection frameworks. In this way, BASTA can make use of the different methods and filters used by these frameworks that reduce the false alarms. Interestingly, the error-based approach gives better results on noiseless data while using simple fingerprints works better in noised settings. This is likely due to incorrectly estimated expected counts, and suggests that error-based is better in noise-free settings where the network manager is able to isolate the traffic (using for instance IP destinations). When this results in too few data, it is better to use the fingerprinting approach on the noisy intertwined traffic.

In the near future, we envision a new symptom generation algorithm specifically for dealing with noisy input sequences, i.e. both non-malicious and legitimate Netflow flows, using dynamic programming to filter flows. In future work, we will also confirm our results by testing the system in our campus network.

ACKNOWLEDGMENTS

This work was partially supported by Technologiestichting STW VENI project 13136 (MANTA) and (LEMMA).

REFERENCES

[1] G. Gu, R. Perdisci, J. Zhang, W. Lee, *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *USENIX Security*, pp. 139–154, 2008.

[2] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, "Botnet research survey," in *IEEE COMPSAC*, pp. 967–972, 2008.

[3] B. Li, J. Springer, G. Bebis, and M. Hadi Gunes, "Review: A survey of network flow applications," *J. Netw. Comput. Appl.*, vol. 36, pp. 567–581, Mar. 2013.

[4] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *IEEE SECURWARE*, pp. 268–273, 2009.

[5] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach," in *IEEE INFOCOM*, vol. 2, pp. 1384–1394, 2005.

[6] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *USENIX SRUTI worskhop*, pp. 43–48, 2006.

[7] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," *Computer Networks*, vol. 56, no. 1, pp. 20–33, 2012.

[8] O. Pomorova, O. Savenko, and S. Lysenko, "A technique for the botnet detection based on DNS - traffic analysis," in *Computer Networks: 22nd International Conference, CN 2015, Brunów, Poland, June 16-19, 2015. Proceedings*, vol. 522, p. 127, Springer, 2015.

[9] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *DSN*, pp. 177–186, IEEE, 2008.

[10] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA*, pp. 108–125, Springer, 2008.

[11] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, p. 6, 2012.

[12] G. Jacob, H. Debar, and E. Filiol, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in computer Virology*, vol. 4, no. 3, pp. 251–266, 2008.

[13] C. Y. Cho, E. C. R. Shin, D. Song, *et al.*, "Inference and analysis of formal models of botnet command and control protocols," in *ACM CCS*, pp. 426–439, 2010.

[14] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet Detection*, pp. 1–24, Springer, 2008.

[15] M. Jaber, R. G. Cascella, and C. Barakat, "Can we trust the inter-packet time for traffic classification?," in *IEEE ICC*, pp. 1–5, 2011.

[16] S. Verwer, M. de Weerdt, and C. Witteveen, "A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data," in *ICGI*, pp. 203–216, 2010.

[17] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[18] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pp. 129–138, ACM, 2012.

[19] T. Barabosch, A. Dombeck, K. Yakdan, and E. Gerhards-Padilla, "Botwatcher: Transparent and generic botnet tracking," in *Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan,November 2-4, 2015. Proceedings*, pp. 565–587, 2015.

[20] D. Babić, D. Reynaud, and D. Song, "Recognizing malicious software behaviors with tree automata inference," *Form. Methods Syst. Des.*, vol. 41, pp. 107–128, Aug. 2012.

[21] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *IEEE S&P*, pp. 110–125, 2009.

[22] C. Leita, K. Mermoud, and M. Dacier, "Scriptgen: an automated script generation tool for honeyd," in *IEEE ACSAC*, pp. 203–214, 2005.

[23] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, "Inferring protocol state machine from network traces: a probabilistic approach," in *Applied Cryptography and Network Security*, pp. 1–18, Springer, 2011.

[24] T. Krueger, H. Gascon, N. Krämer, and K. Rieck, "Learning stateful models for network honeypots," in *ACM AISEC*, pp. 37–48, 2012.

[25] W. Bongiovanni, A. E. Guelfi, E. Pontes, A. Silva, F. Zhou, and S. T. Kofuji, "Viterbi algorithm for detecting ddos attacks," in *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*, pp. 209–212, IEEE, 2015.

[26] S. Nagaraja, *Computer Security - ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, ch. Botyacc: Unified P2P Botnet Detection Using Behavioural Analysis and Graph Analysis, pp. 439–456. Cham: Springer International Publishing, 2014.

[27] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 97–111, May 2013.

[28] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[29] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM TOSEM*, vol. 7, no. 3, pp. 215–249, 1998.

[30] E. M. Gold, "Complexity of automaton identification from given data," *Information and Control*, vol. 37, no. 3, pp. 302–320, 1978.

[31] A. W. Biermann and J. A. Feldman, "On the synthesis of finite-state machines from samples of their behavior," *IEEE Trans. Comput.*, vol. 21, no. 6, pp. 592–597, 1972.

[32] D. Angluin, "On the complexity of minimum inference of regular sets," *Information and Control*, vol. 39, no. 3, pp. 337–350, 1978.

[33] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[34] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering," in *ACM CCS*, pp. 621–634, 2009.

[35] C. Dima, "Real-time automata," *Journal of Automata, Languages and Combinatorics*, vol. 6, no. 1, pp. 3–23, 2001.

[36] S. E. Verwer, *Efficient identification of timed automata: theory and practice*. PhD thesis, TU Delft, Delft University of Technology, 2010.

[37] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[38] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm," in *ICGI*, pp. 1–12, Springer, 1998.

[39] C. Goutte, P. Toft, E. Rostrup, F. Å. Nielsen, and L. K. Hansen, "On clustering fMRI time series," *NeuroImage*, vol. 9, no. 3, pp. 298–310, 1999.