Learning Stateful Models for Network Honeypots

Tammo Krueger
Technische Universität Berlin
Berlin, Germany

Hugo Gascon
Technische Universität Berlin
Berlin, Germany

Nicole Krämer
Technische Universität München
Munich, Germany

Konrad Rieck
University of Göttingen
Göttingen, Germany

ABSTRACT

Attacks like call fraud and identity theft often involve sophisticated stateful attack patterns which, on top of normal communication, try to harm systems on a higher semantic level than usual attack scenarios. To detect these kind of threats via specially deployed honeypots, at least a minimal understanding of the inherent state machine of a specific service is needed to lure potential attackers and to keep a communication for a sufficiently large number of steps. To this end we propose PRISMA, a method for *protocol inspection and state machine analysis*, which infers a functional state machine and message format of a protocol from network traffic alone. We apply our method to three real-life network traces ranging from 10,000 up to 2 million messages of both binary and textual protocols. We show that PRISMA is capable of simulating complete and correct sessions based on the learned models. A case study on malware traffic reveals the different states of the execution, rendering PRISMA a valuable tool for malware analysis.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; I.5.1 [**Pattern Recognition**]: Models—*Statistical*

Keywords

Markov Models, Clustering, Non-negative Matrix Factorization, Honeypots, State Machine Inference

1. INTRODUCTION

In today's fast changing area of network technology, new services for communication emerge almost every day, such as Internet telephony or television. While classic attack vectors like server exploits are still a relevant threat, commercially motivated attacks like call fraud and identity theft are becoming increasingly widespread. These kinds of attacks are often just little aberrations from normal communication patterns (see for instance [33, 19]) and are therefore hard to detect using conventional honeypots with their exploit-centric and often stateless view of the services.

Most of these services rely on already specified protocols, of which usually only a selected subset is used. Other apply non-standard extensions or use a mixture of protocols to implement communication. For example, web applications, such as Twitter and Facebook, can be seen as extensions of the HTTP protocol that implement a certain workflow on top of the actual protocol. As a consequence, it is not sufficient anymore to infer the underlying protocol specification or to learn specific attack patterns, but special tailored models for the actual network service at hand are needed.

Previous work on automatically constructing such models has mainly followed two contrasting directions: One strain of research has focused on extracting the complete protocol specification from the implementation of services using taint analysis [5, 35, 9, 24, 6]. Although very effective, these approaches require access to an implementation and cannot be applied if only network traffic is available. A different direction has concentrated on learning and simulating network vulnerabilities, most notably here is the honeypot ScriptGen [23, 22]. While such honeypots can automatically infer parts of a protocol from network traffic, they have not been designed for tracking involved attacks that require a longer sequence of stateful communication.

In this paper, we present a probabilistic approach to model both the message content and the underlying state machine from network traffic. By inspecting the message interplay between client and server based on a preceding event identification, our model is capable of learning an approximation of the state machine of the service, which not only captures the behavior but can also be used in a generative fashion for simulating long communication sessions.

The main contributions of this *protocol inspection and state machine analysis (PRISMA)* are as follows:

1. Our method is able to learn a stateful model from the network traffic of a service that can be used for simulating valid communication.

2. To construct this model, our method infers the message format, the state machine as well as rules for propagating information between states using machine learning techniques.

3. Our method builds on special tailored embedding and clustering techniques that allow for large-scale applications with millions of network messages.

AISec'12, October 19, 2012, Raleigh, North Carolina, USA.

The remainder of the paper is structured as follows: Section 2 describes the individual steps of PRISMA. After evaluating PRISMA on different data sets in Section 3, we give a detailed account of the related work in Section 4. Further directions and application domains are outlined in Section 5, which concludes the paper.

2. THE PRISMA METHOD

Given a collection of recorded traffic of a specific network service, the goal of PRISMA is to extract a state machine with associated templates and rules, which describe the information flow from message to message. After a first preprocessing stage, where the raw network traffic is converted to sessions containing messages, our method proceeds in the following steps (see Figure 1):

1. To find common structures in the data, we first define a similarity measure between messages. This is done by *embedding* the messages in special vector spaces which are reduced via statistical tests to focus on discriminative features (see Section 2.2).

2. We proceed by modeling each session of messages as a sequence of *events*. By leveraging the embedding of the previous step we apply *part-based* or *position-based clustering* which groups individual messages into events (see Section 2.3).

3. Each of the extracted sequences of events can be seen as a path through the protocol's state machine. To infer an approximation of this state machine, we use the probabilistic concept of *Markov models*, where transitions are linked with probabilities (see Section 2.4).

4. Finally, we automatically generate *templates* for the messages associated with each state of the Markov model and derive *rules* that describe the information flow between the different states during a communication (see Section 2.5).

Throughout the paper we use the term *message* as an atomic exchange of a byte sequence between a client and server. An *event* describes a certain action on the client or server side which is directly connected with the state machine of the modeled network service. A *template* is a message structure consisting of a sequence of tokens and fields. *Rules* describe the message flow between the fields of consecutive templates instantiated for a concrete session.

2.1 Preprocessing of Network Data

To learn the inner structure and behavior of a specific network service we first have to collect sufficient data for the inference. Normally, this can be done at a dedicated sensor, which collects the raw packet data in a binary format for instance via the tool `tcpdump`. Apart from the payload, each packet contains a source and destination address. To actually reconstruct the information flow between a client and a server, these packets have to be re-assembled to eliminate artifacts from the network and transport layer. For this task we have devised a network recorder which uses the mature library `Libnids` for re-assembling TCP and UDP communication streams.

These streams are the input for a session extractor, which generates for each re-assembled packet payload a specific session identifier according to the source and destination of the packet. If two packets occur with a very small delay of τ_{msg} milliseconds, the payloads will be merged. If a specific session identified by its source and destination does not have any more communication within τ_{session} milliseconds, the corresponding session will be flagged as terminated, such that any other message arriving with this specific source/destination combination will open a new session.

This network recorder and session extractor preprocess the raw network traces into sessions containing messages. In the following we will use this preprocessed data in the subsequent steps of the analysis.

2.2 Embedding of Messages

After the preprocessing, a message x can be modeled as sequence of bytes, that is, $x \in B^\star$, with $B = \{0, \dots, 255\}$. To infer common structures from a pool of messages we need a similarity measure which is capable of focusing the analysis on discriminative features. To account for different styles like binary versus textual protocols we introduce two different embeddings both of which can be compressed via statistical tests to enable a more focused analysis.

2.2.1 Embedding with n-grams

One common approach from the domain of natural language processing is to map byte sequences into a finite-dimensional feature space whose dimensions are associated with *n-grams*, substrings of fixed length n. Formally, we can describe these substrings as $W = B^n$ and define an embedding function $\phi : B^\star \mapsto \mathbb{R}^{|W|}$ as follows

$$\phi(x) := (\phi_w(x))_{w \in W} \quad \text{with} \quad \phi_w(x) := \text{occ}_w(x)$$

which simply records, whether a specific n-gram w occurs in a given string. For instance,

$$\phi(\text{”Hello”}) = (0, \dots, \overset{\text{Hel}}{1}, \overset{\text{ell}}{1}, \overset{\text{llo}}{1}, \dots, 0)^T \in \mathbb{R}^{16777216}$$

for $n = 3$. From this example we can see that the corresponding feature space has a finite but high dimensionality. However, this space is generally sparsely populated, which allows for efficient data representation [31].

2.2.2 Embedding with tokens

Another well-known concept from the domain of natural language processing is the tokenization of a byte sequence via pre-defined separator characters S. This embedding $\phi : B^\star \mapsto \mathbb{R}^{|W|}$ maps the byte sequence to a feature vector, which records the occurrences of all possible words W according to the separators, that is, $\phi(x) := (\phi_w(x))_{w \in W}$. For example, if we consider the set of separators $S = \{\text{␣}\}$, we get the following embedding

$$\phi(\text{”Hey ho, let's go”}) = (0, \dots, \overset{\text{Hey}}{1}, \overset{\text{ho,}}{1}, \overset{\text{let's}}{1}, \overset{\text{go}}{1} \dots, 0)^T \in \mathbb{R}^{|W|}.$$

Similarly to the n-gram embedding, the dimension of the resulting feature space is large but sparsely populated, therefore efficient storage models are also available [31].

2.2.3 Dimensionality Reduction

For finding structure in network communication, the analysis has to focus on features which discriminate the messages in the data pool. Volatile features, like randomly generated nonces or cookies, will only occur once and lead to an unnecessary bloated vector space. The same holds true

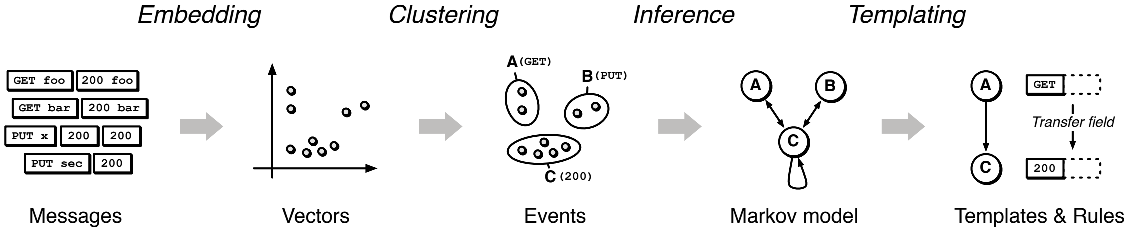


Figure 1: Overview of the Protocol Inspection and State Machine Analysis (PRISMA).

for constant tokens of the protocol, since their occurrence in a message will be almost certain.

Consequently, we use a statistical test-driven dimension reduction [20], which allows us to split the feature space as follows: $F = F_{constant} \cup F_{variable} \cup F_{volatile}$. To this end, we apply a binomial test to each feature, whether it is distributed with a frequency of approximately 1 (corresponding to a constant feature) or 0 (a volatile feature, respectively). After application of a multiple testing correction [15] we keep only those features, which are not constant and not volatile given a statistical significance level of $\alpha = 0.05$. To further simplify the feature space, we group together features which exhibit a correlation near to one.

Given these embeddings and dimension reduction technique we are now able to define a data-driven feature space for messages, which allows us to introduce geometrical concepts like metrics. This opens up the whole field of machine learning tools to be applied for network communication.

2.3 Clustering for Event Inference

Messages which occur at specific events in the flow of communication often exhibit similar structural features. Thus, to extract event information we can exploit this structural dependency. Inside a vector space we can define a metric to capture our notion of the similarity between two messages. For instance the Euclidean metric

$$d_e(x, y) := \sqrt{\sum_{w \in W} (\phi_w(x) - \phi_w(y))^2}$$

calculates the distance between two points based on the occurrence of the $|W|$ words contained in the whole corpus. Using these metrics clustering algorithms can be applied to extract common structures in a data pool and thereby indirectly recover the underlying event information.

For inferring structure from network protocol traces we suggest two possible clustering techniques: One for protocols, which are assembled of parts and one for monolithic communication, where tokens are weighted according to their absolute position in the message. Obviously, the experimenter is free to choose an appropriate clustering technique for the data at hand, but we found these two methods to work best with protocol data of the described kind.

2.3.1 Part-based Clustering

Non-negative matrix factorization (NMF) describes the data by an approximation of the whole embedding matrix $A \in \mathbb{R}^{\tilde{f},N}$ containing N data points with $\tilde{f}$ reduced features by two strictly positive matrices $B \in \mathbb{R}^{\tilde{f},e}, C \in \mathbb{R}^{e,N}$:

$$A \approx BC \quad \text{with } (B, C) = \underset{B,C}{\arg\min} \|A - BC\| \qquad (1)$$
$$\text{s.t. } b_{ij} \geqslant 0,\ c_{jn} \geqslant 0\,.$$

The inner dimension e of the matrix product BC is chosen, such that $e \ll \tilde{f}$ leads to an even more compact representation. Due to the positivity constraint, the matrix B can be interpreted as a new basis (the *parts* of a message), while the matrix C contains the coordinates in this newly spanned space (the *weights* of the different parts). These coordinates are used to ultimately assign a message to a cluster by finding the position with the maximal weight.

There are several possible solutions for solving Equation 1 (see for instance [28, 21, 16, 11]). Here we stick to a practical implementation as introduced in [1]: Based on the *Alternating Least Squares* approach we alternately solve the following constraint least square problems given the regularization constants λ_B, λ_C

$$\min_{C} \quad \|A - BC\|^2 + \lambda_C \|C\|^2 \qquad (2)$$

$$\min_{B} \quad \|A^\top - C^\top B\|^2 + \lambda_B \|B\|^2 \qquad (3)$$

with the corresponding solutions

$$C = \left(B^\top B + \lambda_B I\right) B^\top A \qquad (4)$$

$$B = \left(CC^\top + \lambda_C I\right) CA^\top\,. \qquad (5)$$

The regularization constants can be treated as a meta-parameter of the procedure which we choose by cross-validation. Since both the number of features and the number of samples in the matrix A can get quite large (for instance the FTP data set introduced later contains roughly 1.8 million samples and 90,000 features), direct calculation of Equations (4) and (5) often is infeasible.

Therefore, we devise a reduced, equivalent problem, taking into account that after the dimension reduction step of Section 2.2 we have duplicates in our data matrix $A = [a_1, \ldots, a_N]$, i.e. denote by $\widetilde{A} \in \mathbb{R}^{\tilde{f} \times \widetilde{N}}$ the matrix without duplicate columns. For the simplification of Equation (2) note, that

$$c_i = \left(B^\top B + \lambda_C I\right) B^\top a_i.$$

Hence, we can replace A by $\widetilde{A}$ in Equation (4) to obtain $\widetilde{C}$ and then duplicate the resulting $\widetilde{c}_i$ accordingly to retrieve C. For the simplification of Equation (3) note, that

$$\|A^\top - C^\top B\|^2 = \|\widetilde{A}^\top - \widetilde{C}^\top B\|_W^2 \qquad (6)$$

with W the $\widetilde{N} \times \widetilde{N}$ diagonal matrix consisting of the number of duplicates of the corresponding column in $\widetilde{A}$. As shown in [14], the optimization problem of Equation (3) with the right side of Equation (6) as new objective can be solved by

$$B \;=\; \left(\widetilde{C} W \widetilde{C}^{\top} + \lambda I\right) \widetilde{C} W \widetilde{A}^{\top} .$$

These two simplifications allow us to apply NMF to even large data sets with no reduction in accuracy. The inner dimension e can be chosen according to an argument in [32]: The ordered eigenvalues of the data matrix can be split into a part which is actually contributing to the real signal and a noise part. If we estimate the eigenvalues λ_i on the original data matrix A and the eigenvalues $\widehat{\lambda}_i$ on a scrambled version $\widehat{A}$, where we randomize the features for each message and add confidence intervals to the eigenvalues $\lambda_i, \widehat{\lambda}_i$ according to [18], we can pick the last index as inner dimension e, in which the confidence intervals $\lambda_e, \widehat{\lambda}_e$ do not overlap.

2.3.2 *Position-based Clustering*

While NMF is a good choice for protocols, where a message is constructed out of parts, some protocols show very position-dependent features. Since the clustering step in PRISMA is totally independent from the concrete algorithm used, as long as the procedure assigns a cluster label to each message, the experimenter is not fixed to NMF but is free to choose an appropriate method. To take position dependent features into account, we propose a weighted distance measure

$$d_w(x, y) := \sqrt{\sum_{w \in W} (10^{1-p(w,x)} \phi_w(x) - 10^{1-p(w,y)} \phi_w(y))^2},$$

where $p(w, x)$ returns the position of token w in string x. This distance measure can be used to calculate the distance matrix D, which subsequently forms the input to single linkage hierarchical clustering. Note that we can also restrict the calculation of the distance matrix to the reduced data matrix $\widetilde{A}$. This not only saves computing time but also keeps the size of D in a reasonable range.

2.4 Inference of the State Machine

Network communication is driven by an underlying state machine, in which certain events trigger certain responses and switches to proceeding states. Sometimes, these switches are probabilistic by nature (for instance a service is temporarily unavailable) or can be modeled as such (for instance in a login procedure 90% of the attempts are successful).

One possible way to model the state machine of a network service in a probabilistic way is by a *hidden Markov model*: The unobserved states correspond to the internal logical states of the service and the messages sent over the network correspond to emitted symbols. Using the Baum-Welch algorithm [2] and enough data of service communication it would be possible to estimate an underlying hidden Markov model to describe the observed data. However, the Baum-Welch algorithm does not guarantee that the found model has the highest likelihood in a global sense.

2.4.1 *Learning the Markov Model*

Instead of directly trying to infer the underlying hidden Markov model, we start with a regular Markov model which we will later on simplify to a minimal hidden variant. The whole learning process is therefore deterministic and has no inherent randomization like the initial model matrices in the Baum-Welch algorithm. With this approach we circumvent the problem of finding a potential non-optimal model. This determinism comes at a price: it is a well known fact that hidden Markov models are more powerful than regular ones [10]. In summary we trade the potential uncertainty with a decrease in model complexity, therefore regularizing the hypotheses space.

Given the session information of the preprocessing step and the label information for each message of the event clustering we could directly learn a regular Markov model of event chains by estimating the initial and transition probabilities by their maximum likelihood estimates. However, in this simple Markov model we would drop the direction of the event (i.e. was an event triggered by the client or the server) and limit the history to one message due to the Markov assumption (i.e. the generation of the next event depends just on the previous one). Especially the last limitation would be too strict for network communication, since we would loose the context in which a message would have been generated.

2.4.2 *Convoluting the State Space*

To circumvent the limitation of the regular Markov model, we use a convoluted and communication-annotated version of the event sequence of a session as follows:

1. Each event will be annotated to reflect, whether it was generated from the client or the server side.
2. For a horizon of k we convolute the annotated and padded event sequence by sliding a window of size k over it and recording the occurring k-tuples.

As an example assume we have observed the event sequence $[abcd]$ where the messages were generated alternatingly from the client and server. With a horizon of $k = 2$ we would convolute this event sequence to

$$[(\varnothing, \varnothing), (\varnothing, a_C), (a_C, b_S), (b_S, c_C), (c_C, d_S)].$$

So the new, convoluted event space $\widetilde{E}$ will contain $(2|E| + 1)^k$ potential events, with $(\varnothing, \varnothing, \ldots, \varnothing)$ being the starting state. By calculating the transition probabilities in this new convoluted event space $\widetilde{E}$ by their maximum likelihood estimates we specify a regular Markov model with an annotated event horizon of k.

2.4.3 *Minimizing the Markov Model*

For client server communication a horizon of at least $k = 2$ is necessary, to keep the communication context. For more involved processes an even higher horizon might be necessary, which leads to an exponential growth of possible states. We will see in the evaluation section, that for real network communication this convoluted state space is often very sparsely populated, yet the resulting networks can be large, making the introspection by a human user difficult.

As a remedy we propose the following minimization algorithm to boil down the size of the Markov model while preserving its overall capabilities:

1. Transform the Markov model M into a deterministic finite automaton (DFA) $\widehat{M}$:
    (a) Keep transitions which have a probability bigger than zero and their associated states.

	State A_S	State B_C	State C_S
Session 1	`ftp 3.14`	`USER anon`	`331 User anon ok`
Session 2	`ftp 3.12`	`USER ren`	`331 User ren ok`
	⋮	⋮	⋮
Session n	`ftp 2.0`	`USER liz`	`331 User liz ok`
Template	`ftp` □	`USER` □	`331 User` □ `ok`

Figure 2: Example of template generation for a simplified FTP communication.

(b) At each transition the DFA $\widehat{M}$ accepts the new event of the second state (for example the transition connecting state (a_C, b_S) with state (b_S, c_C) would consume event c_C).

2. Apply the DFA minimization algorithm as introduced in [26] to get the equivalent DFA $\widetilde{M}$ with the minimal number of states but accepting the same language.

3. As a side effect, this algorithm returns an assignment $A_{\widetilde{E},\widetilde{M}}$ of the original states of the convoluted event space $\widetilde{E}$ to the compressed states of the DFA $\widetilde{M}$.

The resulting DFA $\widetilde{M}$ can be used for the inspection of the underlying state model and can be interpreted as a special hidden Markov model: Instead of observing the convoluted events $\widetilde{E}$ we now observe the states of $\widetilde{M}$ according to the assignment $A_{\widetilde{E},\widetilde{M}}$ found by the minimization algorithm. These meta-states subsume equivalent states, and will therefore lead to the acceptance of the same event sequences as the original model. We will show in the evaluation section, that these simplified models drastically decrease the model size and are therefore good candidates for the analysis of the state machine by a human administrator.

2.5 Learning Templates and Rules

Each session can be seen as a sequence of events which trigger specific state switches of the state machine. To learn the general information flow during this process, we generalize the messages associated with a state to a template that consists of fixed and variable parts, which often are filled by contents of previous messages. Exploiting the extracted Markov model we are now ready to give a procedure to extract templates and rules for the network service at hand.

2.5.1 Inference of Templates

In the event clustering step, we focused on variable, yet neither constant nor volatile features to identify common patterns in the exchanged messages. While this focus makes sense for the identification of underlying events, it is essential to have all features back for the generation of valid, protocol-conformant messages.

An additional aspect for the extraction of generic message templates is the underlying state machine of the analyzed service: it is very likely, that the exchanged messages correlate with the current state of the service. Thus, a valid assumption is to assign the messages of each session to its according state in the previously extracted state machine as shown for an artificial example in Figure 2: By looking for recurring tokens in each state, generic templates can be constructed containing fixed passages and variable fields according to the distribution in the learning pool.

In more detail, the template inference procedure is structured as follows:

1. Tokenize each message according to the previously chosen embedding.

2. Assign the message of each session to the state of the inferred Markov model.

3. For each state of the Markov model:

   (a) Group all assigned messages with the same number of tokens and process each of these groups.

   (b) If all messages in a group contain the same token at a specific position, a fixed token is recorded at the resulting template, otherwise a variable field is saved.

At the end of this procedure we will have templates for each state of the Markov model representing the generic messages that might occur. Note that each state might have several different templates assigned according to the observed length distribution: I.e., we simplify the multiple alignment procedure for the extraction of generic templates by focusing the alignment to messages of the same length.

2.5.2 Inference of Rules

Finding rules for filling specific fields in these templates according to previously seen messages now amounts to a simple, yet powerful combination of the Markov model, extracted templates, and session information. For each possible combination of template occurrences of the horizon length k, i.e., $(t_1, t_2, \ldots t_k)$:

1. Find all messages which are assigned to these k templates and occur in a session in this exact order.

2. For each field f in the template t_k:

   (a) Look for a rule to fill f with field content $\hat{f} \mp f$ of templates $(t_1, t_2, \ldots t_k)$ in $F\%$ of the sessions.

   (b) If no rule matches, just record the tokens, that occur in the training pool (*Data* rule).

The checked rules are described in Table 1. This procedure ensures that information that is found in preceding messages which can systematically reproduce contents in a following message in $F\%$ of the cases will get copied over. For instance in the example shown in Figure 2 we can observe that in all cases the field of the template associated with state C can be filled with the field of the previous message. The *Data* rule acts as a fallback solution if no match could be found and as a pump-priming rule for the first messages of a session.

2.6 Simulation of Network Communication

The inferred PRISMA model now contains three parts: the actual Markov model, the inferred templates and the rule sets associated with these templates. To use these parts of a PRISMA model for simulation of a communication we devised the Lively Essence Network Sensor (LENS) depicted in Algorithm 1. In addition to the inferred model parts, this module is initialized with the role (client or server) which should be simulated. Note that the PRISMA model itself is role agnostic and therefore can be used to simulate both sides of a communication. This allows us to even let the

Rule	Description
Copy	Exact copy of the content of one field to another.
Seq.	Copy of a numerical field incremented by d.
Add	Copy the content of a field and add data d to the front or back.
Part	Copy the front or back part of a field splitted by separator s
Data	Fill the field by randomly picking data d which we have seen before.

Table 1: Rules which are checked during model building. Parameters like d and s are automatically inferred from the training data.

Algorithm 1 The Lively Essence Network Sensor (LENS)

```
function LENS(markovModel, templates, rules, role)
    while communication is active do
        Wait for message with timeout $t$
        if message $M$ received then
            Find matching template $T$ according to the current state
            Split the message $M$ according to $T$ into fields
            Switch the state to the state associated to $T$
        Randomly choose the state $S$ according to the transition probabilities of markovModel
        if $S$ is in accordance with role then
            Find rule set according to the previous $k$ (horizon) templates
            Apply rules to fill the new template to form the message
            Send out message
            Set current state to $S$
```

model talk to itself by running two instances of LENS with different roles and passing the messages generated from one instance to the other and vice versa.

Appendix A gives a complete example of a PRISMA model based on a simple toy problem: Given network traces of a robot communicating with its environment a behavioral model is learned via PRISMA.

3. EVALUATION

In this section we show, that the PRISMA method is capable of learning and simulating network communication from real network traces. To this end we use several network traces recorded via `tcpdump` and plug one part of the data into our processing pipeline and check the quality of the model both according to the remaining data and syntactical and semantical features of the simulated sessions. By this we ensure an evaluation of PRISMA under real-life conditions:

1. Comparison against the held-out sessions assures *completeness* of the models, meaning that the learned models are capable of replaying real sessions as observed in the data pool.

2. Checking syntactical and semantical features of the simulated sessions guarantees the *correctness* of the models from a communication perspective.

	Size	Dimension	% kept	% unique
SIP	34,958	72,937	0.39%	2.58%
DNS	5,539	6,625	13.15%	35.64%
FTP	1,760,824	87,140	2.17%	0.24%

Table 2: Properties of data sets: *size* gives the total number of messages in the data set and *dimension* the number of features before the dimension reduction step. *% kept* and *% unique* gives the percentage of features and messages, which are kept after the dimension reduction step.

In Section 3.1 we introduce the data sets and discuss the resulting feature spaces after dimension reduction. Then, we look at the general properties of the learned PRISMA models in Section 3.2 and the completeness and correctness of these models in Section 3.3. We conclude the evaluation with a case study on malware analysis, showing that PRISMA can be useful in application domains beyond honeypots.

3.1 Data sets and Dimension Reduction

For the evaluation of the PRISMA framework we have chosen three representative data sets, of which two are text-based and one is purely binary (see Table 2):

- *SIP:* A data set recorded in a real, medium sized telephony infrastructure containing roughly 7 days of communication of 20 participants with different Session Initiation Protocol (SIP) clients.

- *DNS:* Domain Name System (DNS) requests of a home network with 7 different clients collected during one day of heavy use.

- *FTP:* File Transfer Protocol (FTP) data set from the Lawrence Berkeley National Laboratory [29] containing 10 days of communication.

Naturally, these data sets vary in size: while the SIP data set is a medium-sized pool of roughly 35,000 messages, the DNS data set contains just 6,000 messages. The FTP data comprise of nearly 1.8 million messages rendering it the biggest data set of the evaluation. To accommodate the different properties of the data sets, we apply different embeddings: Since SIP and FTP consist of human-readable text, both can be tokenized with the usual white space characters. Due to the binary layout of the DNS data, this tokenization approach would not be feasible, therefore we have chosen a 2-gram embedding for DNS. For all data sets we randomly select 90% of the data to learn the PRISMA model and keep the rest for the evaluation carried out in Section 3.3.

The resulting feature dimensionality reductions and unique messages are shown in Table 2. The first thing to note is the power of the dimension reduction step: the relative number of kept features ranges from 0.4% for SIP, 13.2% for DNS and 2.2% for the FTP data set showing the extreme focus, which emanates from the dimensionality reduction. A direct consequence of this is the relative number of unique messages for each data set, ranging from 2.6% for SIP, 35.6% for DNS and 0.2% for FTP. The striking difference between DNS and SIP/FTP in terms of reduction can clearly be explained by the different conceptual layouts of the languages: the highly compressed, binary format of the DNS protocol leaves less room for optimization of the feature space, therefore also the number of unique messages after the dimension reduction is higher compared to the other text-based protocols.

	# nodes	Coverage	Min. DFA	Coverage
SIP	148	14.5%	100	9.8%
DNS	381	0.8%	153	0.3%
FTP	1,305	0.8%	653	0.4%

Table 3: Number of nodes for the PRISMA models both for the unoptimized Markov model and the minimal DFA. *Coverage* relates these numbers to the potential number of nodes possible.

	Copy	*Seq.*	*Add*	*Part*	*Data*	Total
SIP	1,916	77	135	52	1,793	3,972
DNS	3,142	4	0	0	3,527	6,673
FTP	532	18	253	35	4,671	5,509

Table 4: Number of different rules of the PRISMA models extracted for the different data sets.

Overall, we see that the dimension reduction is highly effective even for binary protocols. By focusing only on the varying parts of the messages and unique messages in this reduced feature space, valuable computation time can be saved and renders the PRISMA approach capable of modeling even big data collections.

3.2 Properties of Learned Models

Following the embedding and dimensionality reduction step we apply the event clustering step as described in Section 2.3: Both for the SIP and DNS data set we apply the NMF clustering algorithm, since a quick inspection of the data shows, that the part-whole-relationship underlying the NMF algorithm holds for these two data sets. The relative short FTP messages follow a more or less fixed setup, rendering the position-dependent clustering approach better suited for this kind of data.

Table 3 summarizes the number of nodes of the extracted Markov model for each data set and relates this number to the potential number of nodes which are attainable as described in Section 2.4. We see that the total number of nodes for the SIP data set is smallest, yet the relative coverage is highest. For DNS and FTP the absolute number of nodes is higher, but the relative coverage of the potential node space is very sparse, indicating that there is a inherent dependency of relative coverage and estimated number of clusters. Application of the DFA minimization algorithm to the Markov model significantly reduces the number of nodes for the models converting the resulting networks into dimensions manageable by human users.

The corresponding number of rules for each model is shown in Table 4. Note that for the n-gram embedding the *Add* and *Part* rules are deactivated, since they are already handled by the *Copy* rule. We see, that all rules are represented. The SIP data set exhibits a higher number of more involved rules compared to all other data sets reflecting the highly redundant structure of this protocol. Both DNS and FTP have an inherent variable part (the server name for DNS and the file names for FTP) which results in a higher number of *Data* rules compared to the SIP data.

Figure 3 gives a visual impression of the learned model for the FTP data set. To generate this session we simulated both sides of the communication with our PRISMA model learned on the FTP data set: one model was executed to act as the client and the other one acted as the server. We see in the resulting log, that the session that was generated is valid FTP: Starting with the initial login procedure, the client sets the `TYPE` of the communication to binary data, then enters passive mode and gets a file from the server. Note, that the name of the file from the client request is copied over to the corresponding reply of the server, showing the power of the inferred rules. Obviously, the byte size of 56 is not the proper size of the requested file, since it was chosen randomly from the *Data* rule, but the message itself is a valid FTP reply showing the ability of PRISMA to even generate new messages not seen in the training pool before.

3.3 Completeness and Correctness

While the previous figures and examples show that the PRISMA method produces relatively condensed models of both the embedding space and the state machine, questions regarding the completeness and correctness of these models are treated in this section.

3.3.1 Completeness

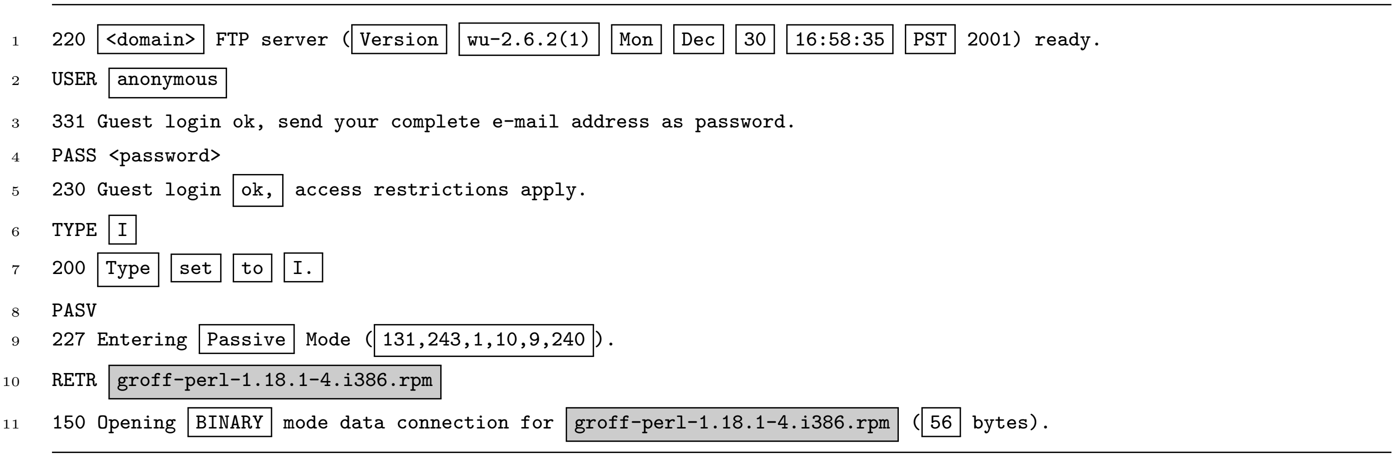
To judge the *completeness* of the models we take the 10% of the held-out data and simulate either the client or the server side to evaluate whether our learned model contains a path, which could generate a session which resembles the data most. Since the transitions in the model are probabilistic, we cannot ensure that the path we choose during the simulation is synchronized with the actual content of the session. For instance, a session might contain a specific branch of the state machine, which occurs just 5% of the time like a server overload error reply or the like. To alleviate this probabilistic effect we repeat each simulation 100 times and introduce a determinism by feeding the first two messages of a session to the model such that the states for the first two messages which are exchanged are aligned.

The results of these simulations are reported in Figure 4. We use the normalized Levenshtein distance as similarity (1 meaning equality) which counts the number of insertions, deletions, or substitutions necessary to transform one string into another. At each position of a session we take the maximum attained similarity over all repetitions to take account of the probabilistic effect as described before.

For the SIP data set we observe that the number of equal messages ranges between 80% and 60%. The similarity score is almost never below 0.9 showing that the learned models can correctly re-model the hold-out session. For DNS this behavior is similar but shows more variance due to the relative low number of sessions having more than 6 messages. The FTP data set shows an even better performance of the PRISMA model with nearly all messages showing equality up to position six. The frequency of exact resemblance then stays always above 70% showing that even complex protocols can be accurately simulated for more than four steps.

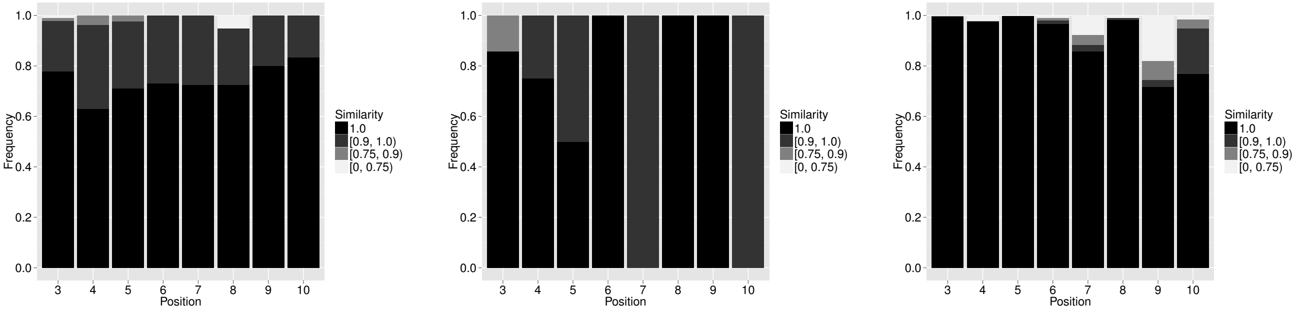
3.3.2 Correctness

Next, we focus on the syntactical and semantical *correctness* of the generated messages. For the syntactical correctness we utilize the protocol filters of the network protocol analyzer `Wireshark`. Only for the FTP protocol we had to check the validity of the commands manually according to the RFCs [30, 12, 25, 13]. For the check of semantical correctness we apply the following rules:



```
220 <domain> FTP server ( Version  wu-2.6.2(1)  Mon  Dec  30  16:58:35  PST  2001) ready.
USER anonymous
331 Guest login ok, send your complete e-mail address as password.
PASS <password>
230 Guest login  ok,  access restrictions apply.
TYPE I
200 Type  set  to  I.
PASV
227 Entering Passive Mode ( 131,243,1,10,9,240 ).
RETR groff-perl-1.18.1-4.i386.rpm
150 Opening BINARY mode data connection for groff-perl-1.18.1-4.i386.rpm ( 56 bytes).
```

Figure 3: Sample FTP session generated by executing two PRISMA models against each other (one as client, one as server). Data fields are marked by boxes, exact copy rules are filled in gray.



(a) SIP data set.

(b) DNS data set.

(c) FTP data set.

Figure 4: Distribution of maximal similarities by message position in a session replay. Recorded is the normalized edit distance giving a 1.0 for equal messages. The size of the black bar corresponds to the frequency of equal messages, the size of the dark gray bar for similarities ranging between 0.9 and 1.0, the light gray bars for similarities ranging between 0.75 and 0.9.

- *SIP:* For each message of a session we check, whether the CallID, from- and to-tag are preserved, since this triple of values identifies a SIP-session.
- *DNS:* If the message of a session is a reply, we check whether it was queried before in this session and has the same query id.
- *FTP:* For each FTP request we check, whether both the request and the returned reply code is a valid one according to the RFCs [30, 12, 25, 13].

For each session we count the number of syntactically and semantically correct messages and report the relative frequency of correct messages for the complete session. In addition to the session generated for the completeness evaluation (denoted as *unidirectional*) we also simulate 100,000 sessions, in which both sides are generated by our model (denoted as *bidirectional*).

The results are shown in Table 5: The syntactical correctness of the sessions is almost always perfect. Only the bidirectional simulations for the FTP data set shows a relative decline of having just 82% percent of the sessions which are totally correct. Regarding the semantics, DNS shows also a nearly perfect behavior. The performance of the SIP model is with 98% and 94% of the sessions totally correct for the uni- and bidirectional simulation, respectively, also in a very good range. While the semantics for the FTP in the unidirectional case show good behavior, the performance declines for the bidirectional simulations: just 57% of the sessions are totally correct. Since FTP sessions tend to be very long, we investigate the correctness in more detail in Table 6. By splitting the frequency bins we observe that the bulk of the sessions have more than 80% correct messages. In combination with the higher length of a FTP session this

	Syntax		Semantic	
	Unidir.	Bidir.	Unidir.	Bidir.
SIP	1.000	1.000	0.988	0.945
DNS	1.000	1.000	1.000	0.994
FTP	0.999	0.821	0.934	0.576

Table 5: Frequency of sessions having 100% syntactical and semantical correct messages for the different simulation paradigms (uni- and bidirectional).

Msgs.	Syntax		Semantic	
Correct	Unidir.	Bidir.	Unidir.	Bidir.
100%	0.999	0.821	0.934	0.576
$> 90\%$	1.000	0.953	0.988	0.878
$> 80\%$	1.000	0.996	1.000	0.982

Table 6: Breakdown of cumulative syntactical and semantical correctness of sessions for the FTP data.

shows that even for difficult, potentially vast communication patterns the PRISMA model is able to capture both the syntax and the semantics of the communication.

In summary, the evaluation shows that the inferred PRISMA models are very compact and show a very high degree of completeness as well as syntactical and semantical correctness. This renders these models ready for the deployment in real-life network infrastructures to act as a honeypot specifically designed for the occurring traffic in this network. Contacts to this honeypots can be held up for a high number of steps to gather in-depth information of the behavior and intentions of the potential intruder. This information cannot only be used to estimate the threat potential in an infrastructure at a given time point but also to learn more about the attacks or mischief being conducted.

3.4 Case Study: Koobface

In this section we apply PRISMA to network traffic collected from malicious software by Jacob et al. [17]. We picked one specific class of malware instances and used the token embedding and part-based clustering. We had a total of 147 sessions with 6,674 messages. A detail of the resulting model is depicted in Figure 5.

In the upper part we see a scanning loop, in which the malware tries to find a command-and-control server: as long as the server does not answer in a specific format, the scan is continued. After the malware has received a correct reply in state F_S a handshaking procedure between malware and server takes place, which is followed by a download cycle. In state I_C, the malware starts to download the first file from the list (`go.exe`), while in the following states all the other files are downloaded. This can be nicely seen by the *Data* rule associated to the template of state K_C, which contains several instances of the following paths:

```
/.sys/?getexe=tg.14.exe, /.sys/?getexe=ms.26.exe,
/.sys/?getexe=hi.15.exe, /.sys/?getexe=be.18.exe,
/.sys/?getexe=tw.07.exe, /.sys/?getexe=v2captcha.exe,
/.sys/?getexe=v2googlecheck.exe
```

By inspecting the extracted state machine and the associated templates and rules a malware analyst can gain insights into the inner workings of a malware instance from the collected network traces alone. This renders PRISMA a valuable tool beyond the realms of honeypot applications.

4. RELATED WORK

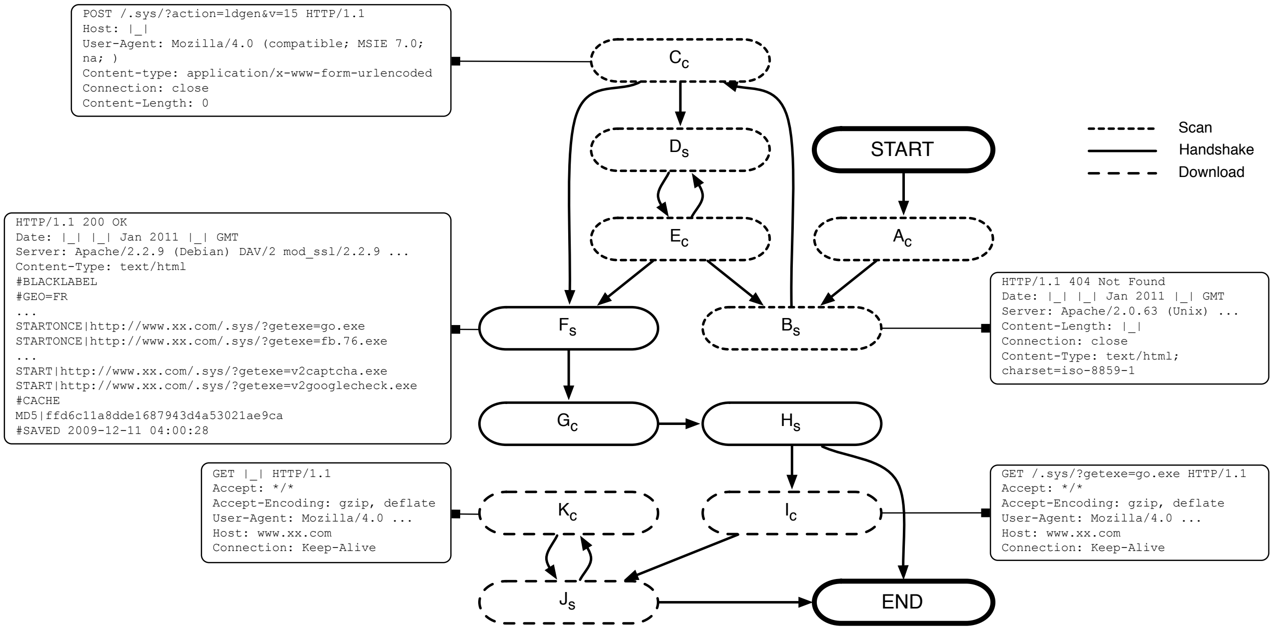
The generation of valid models from communication network protocols is, undoubtedly, a problem that has received much attention in recent years. From the pure reverse engineering perspective, the open source community has tried to fully understand the inner workings of proprietary protocols in order to develop open implementations of several network services (e.g. SMB, ICQ, Skype). Most of this work has been done in a manual fashion, but the special relevance of network protocol analysis for the security field has led to many research efforts on automatic learning of the protocol state machine and the format of messages involved in a valid communication session.

The work by Beddoe [3] constitutes a first attempt to extract the fields from protocol messages by drawing upon advanced computational techniques. This approach proposes the clustering of complete messages and the construction of a phylogenetic tree in order to guide the process of global sequence alignment through the Needleman-Wunsch algorithm. With RolePlayer [8], the authors build on these ideas to tackle the problem of automatically replaying valid messages from a protocol. Although they present a limited approach that requires the other side of the communication to follow the script that has been used to configure the system, it already considers the problem of simulating a state dependent communication. Within the same scope is Replayer [27]. The system presented by Newsome et al. proposes an enhanced solution beyond heuristics, introducing the concepts of theorem proving and weakest pre-condition verification as means to handle protocol dependencies.

A similar approach with a specific security application and also focused on replaying valid messages is introduced in the realm of honeypots by ScriptGen [23, 22]. This low interaction honeypot learns and simulates communication patterns of vulnerabilities. The objective of ScriptGen is not to infer an accurate specification of the protocol but to obtain the maximum information on the exploitation attempt of a service. Although closely related to our approach, ScriptGen is designed for monitoring low-level attacks against implementations, whereas PRISMA enables collecting and tracking semantic attacks on top of these implementations. In a similar strain of research, Cui et al. [7] have studied the use of tokenization and clustering of individual messages to find fields in message structure. However, this work does not infer the state machine of a protocol and thus can not be used for simulating network communication.

Different approaches based on dynamic taint analysis have been proposed to infer protocol specifications [5, 24, 35, 9]. In order to overcome the lack of semantics of clustering techniques, they rely on dynamic binary analysis of the network service that handles the protocol messages. This eases finding keywords and delimiters but unfortunately all these works defer the task of learning the protocol state machine. An extension to this work with a practical focus on security is carried out in [4]. Caballero et al. devise Dispatcher, a system that is capable of infiltrating botnets (whose operation may be based on customized or proprietary protocols), by being able to model, as in our work, messages from both sides of the communication. Also at the host level, it is worth mentioning the work of Wang et al. [34], which uses binary analysis to extract the properties of messages at memory buffers once they have already been decrypted.

Finally, Comparetti et al. [6] build on these ideas in order to construct the state machine of a protocol, again from the dynamic behavior of the application that implements such protocol. The extent of their work certainly resembles ours, nonetheless, our approach is free of the additional burden of binary taint analysis since it is fully network based. The gathering of large amounts of input traces for our system is thereby a straightforward task.



```
POST /.sys/?action=ldgen&v=15 HTTP/1.1
Host: |_|
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0;
na; )
Content-type: application/x-www-form-urlencoded
Connection: close
Content-Length: 0
```

```
HTTP/1.1 200 OK
Date: |_| |_| Jan 2011 |_| GMT
Server: Apache/2.2.9 (Debian) DAV/2 mod_ssl/2.2.9 ...
Content-Type: text/html
#BLACKLABEL
#GEO=FR
...
STARTONCE|http://www.xx.com/.sys/?getexe=go.exe
STARTONCE|http://www.xx.com/.sys/?getexe=fb.76.exe
...
START|http://www.xx.com/.sys/?getexe=v2captcha.exe
START|http://www.xx.com/.sys/?getexe=v2googlecheck.exe
#CACHE
MD5|ffd6c11a8dde1687943d4a53021ae9ca
#SAVED 2009-12-11 04:00:28
```

```
HTTP/1.1 404 Not Found
Date: |_| |_| Jan 2011 |_| GMT
Server: Apache/2.0.63 (Unix) ...
Content-Length: |_|
Connection: close
Content-Type: text/html;
charset=iso-8859-1
```

```
GET |_| HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 ...
Host: www.xx.com
Connection: Keep-Alive
```

```
GET /.sys/?getexe=go.exe HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 ...
Host: www.xx.com
Connection: Keep-Alive
```

Figure 5: Extracted state model for Koobface traffic: The upper part of the model corresponds to a scanning phase of the malware. The middle part is a handshaking procedure with an infected machine, where the malware gets a new list of malware which is finally downloaded in the lower part of the state machine.

5. CONCLUSION AND FUTURE WORK

With PRISMA we have presented a tool capable of learning and simulating communication of a given service from network traffic alone. By representing the internal state machine of the service with a Markov model and extracting templates and rules via aligning the collected session to this state machine, PRISMA is able to extract information necessary for an efficient simulation of the data pool. The evaluation shows that both from the viewpoint of completeness and syntactical and semantical correctness PRISMA is capable to emulate real-life network traffic.

Our next goal is to deploy PRISMA as a honeypot in a dedicated network infrastructure. While our evaluation shows that the PRISMA models are solid we expect valuable input of this real-life application to further robustify our approach. Additionally, stateful fuzzing is an interesting further application of PRISMA; for instance, one can use the extracted Markov model to find communication paths inside the state machine, which occur very seldom and therefore should tend to be rather untested and error-prone. We believe that the template structure and the rules can give valuable clues which fields should be fuzzed with what content to maximize the probability of an enforced error. The analysis of Koobface network traces with PRISMA shows that the method can be readily applied in the domain of malware analysis. Still, further refinements, for instance finding the most interesting path in the state machine of the malware, can enhance the usability of PRISMA for this scenario.

Acknowledgments This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project PROSEC (FKZ 01BY1145) and ALICE (FKZ 01IB10003B). We thank Franziska Biegler and Klaus-Robert Müller for fruitful discussions and suggestions and Ralf Hund for supplying us with the Koobface network traces.

6. REFERENCES

[1] R. Albright, J. Cox, D. Duling, A. Langville, and C. Meyer. Algorithms, initializations, and convergence for the nonnegative matrix factorization. Technical Report 81706, North Carolina State University, 2006.

[2] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363, 1967.

[3] M. A. Beddoe. Network Protocol Analysis using Bioinformatics Algorithms. Technical report, McAfee Inc., 2005.

[4] J. Caballero, P. Poosankam, and C. Kreibich. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS)*, 2009.

[5] J. Caballero, H. Yin, and Z. Liang. Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CSS)*, 2007.

[6] P. Comparetti and G. Wondracek. Prospex: Protocol Specification Extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[7] W. Cui and J. Kannan. Discoverer: Automatic Protocol Reverse Engineering From Network Traces. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[8] W. Cui, V. Paxson, N. C. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)*, 2006.

[9] W. Cui, M. Peinado, K. Chen, and H. Wang. Tupni: Automatic Reverse Engineering of Input Formats. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

[10] A. M. Fraser. *Hidden Markov Models and Dynamical Systems.* Society for Industrial and Applied Mathematics, 2008.

[11] M. Heiler and C. Schnörr. Learning sparse representations by non-negative matrix factorization and sequential cone programming. *Journal of Machine Learning Research*, 7:1385–1407, 2006.

[12] P. Hethmon. Extensions to FTP. RFC 3659 (Proposed Standard), Mar. 2007.

[13] P. Hethmon and R. Elz. Feature negotiation mechanism for the File Transfer Protocol. RFC 2389 (Proposed Standard), Aug. 1998.

[14] P. Holland. Weighted ridge regression: Combining ridge and robust regression methods. Technical Report 11, National Bureau of Econ. Research, 1973.

[15] S. Holm. A simple sequentially rejective multiple test procedure. *Scand. Journal of Statistics*, 6:65–70, 1979.

[16] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[17] G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking command and control connections from bot traffic. *Proceedings of the 20th USENIX Security Symposium*, 2011.

[18] I. T. Jolliffe. *Principal Component Analysis.* Springer, 1986.

[19] H. Kaplan and D. Wing. The SIP identity baiting attack. Internet-draft, Internet Engineering Task Force, 2008.

[20] T. Krueger, N. Krämer, and K. Rieck. ASAP: automatic semantics-aware analysis of network payloads. *Proceedings of the ECML/PKDD conference on Privacy and security issues in data mining and machine learning*, 2011.

[21] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[22] C. Leita and M. Dacier. Automatic Handling of Protocol Dependencies and Reaction to 0-Day Attacks with ScriptGen Based Honeypots. In *Proceedings of the 9th international conference on Recent Advances in Intrusion Detection (RAID)*, 2006.

[23] C. Leita and K. Mermoud. Scriptgen: An Automated Script Generation Tool For honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[24] Z. Lin, X. Jiang, and D. Xu. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.

[25] D. Mankins, D. Franklin, and A. Owen. Directory oriented FTP commands. RFC 775, Dec. 1980.

[26] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.

[27] J. Newsome, D. Brumley, and J. Franklin. Replayer Automatic Protocol Replay by Binary Analysis. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS)*, 2006.

[28] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[29] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2003.

[30] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659.

[31] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9:23–48, 2008.

[32] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276 – 280, 1986.

[33] R. State, O. Festor, H. Abdelnur, V. Pascual, J. Kuthan, R. Coeffic, J. Janak, and J. Floroiu. SIP digest authentication relay attack. Internet-draft, Internet Engineering Task Force, 2008.

[34] Z. Wang, X. Jiang, W. Cui, and X. Wang. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *European Symposium on Research in Computer Security (ESORICS)*, 2009.

[35] G. Wondracek and P. Comparetti. Automatic Network Protocol Analysis. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.

APPENDIX

A. THE ROBOT PROTOCOL

In order to illustrate the message building algorithm used by the PRISMA method, we have developed a closed game-based experiment that eases to understand how the elements learned are integrated with the Markov model.

The Robot game is a simple setup where a player is randomly placed in a room filled with contaminated objects. Its goal is to find these objects and carry them to the base location where they will be eliminated. Unfortunately, it has no view of the position of the objects or itself within the room. Thus, in order to move around and notice if it has found an object or reached the limits of the room, it exchanges messages with a room controller.

In our experiment, the player role is performed by a robot with a fixed algorithm. At first, it randomly walks through the rectangular room until it finds an object. Then, it goes straight up to the top of the room carrying the object and to the left to reach the base, which is situated at the upper left corner. The object is then destroyed and the robot is dropped again randomly inside the room. The process continues until the robot has found and removed all the objects.

The complete setup has a client-server architecture where the robot communicates with the controller by a simple protocol over the network. These network traces are then analyzed by PRISMA, yielding a robot-honeyclient that is capable of mimicking the behavior of the original robot. The robot client sends its instructions to the controller using the

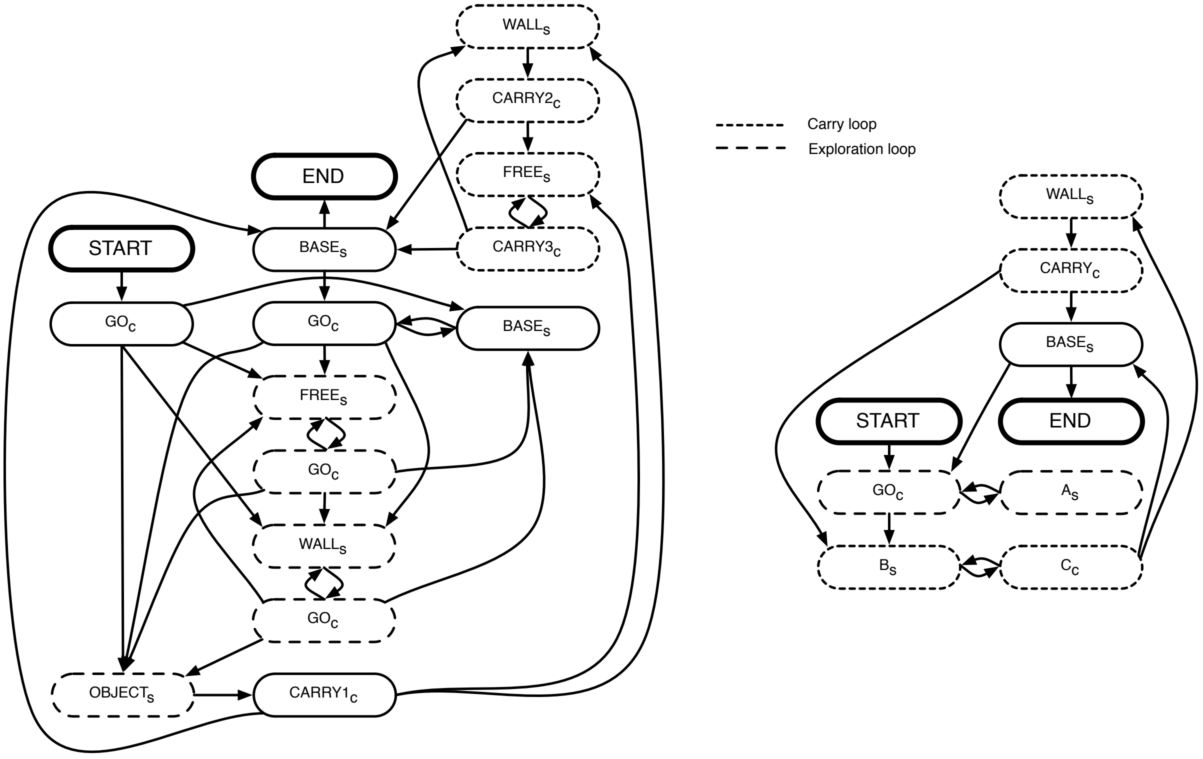


Figure 6: The Markov model of the robot protocol and the minimized version of the state model.

messages `GO <dir>` and `CARRY <object> <dir>`, where possible directions are `UP`, `RIGHT`, `DOWN` and `LEFT`. The controller server responds with the following status messages after each action of the robot: `WALL`, `FREE`, `BASE` and `OBJECT <object>`, where `<object>` denotes the id of the object.

A.1 Markov Model

Figure 6 presents the Markov model of the robot protocol, which has been extracted from the traces. As intended for this example and for completeness, every possible message is present in the pool of simulated communication data. Therefore, the model represents the complete robot protocol. The lower part of the Markov chain models the communication between client and server during the exploration phase of the robot, while the upper part models the loop where the robot is carrying the object to the base. The subindex indicates the active side of the communication, client or server, in each state. The right side of Figure 6 depicts the simplified model obtained after the minimization algorithm has been applied to the original model. States labeled A, B and C are meta-states resulting from the abstraction of several states in the original model and are involved in each of the described phases. Meta-state A represents the behavior of the robot client during the exploration phase while meta-states B and C are part of the carrying loop.

A.2 Templates and Rules

The different templates associated with each state of the model are inferred from the traces obtained during the simulation. The number of runs is specified as an input parameter. Each run requires as many sessions to complete as objects are placed in the room and each session is formed by an arbitrary number of messages as a result of the random direction of the movement. The following examples of templates and rules show how the search algorithm is integrated with the model to build the different protocol messages:

State	Template	Format
OBJECT$_\text{S}$	13	`OBJECT` □
CARRY1$_\text{C}$	11	`CARRY` □ `UP`
CARRY2$_\text{C}$	10	`CARRY` □ `LEFT`
CARRY3$_\text{C}$	2	`CARRY` □ □
FREE$_\text{S}$	5	`FREE`

Transition	Type	Src ID	Src Field	Dst Field
`3;13;11`	*Copy*	13	0	0
`11;0;10`	*Copy*	11	0	0
`2;5;2`	*Copy*	2	0	0
`2;5;2`	*Copy*	2	1	1

When an object is found by the robot, the room controller builds a message with the format shown in template 13 in state OBJECT$_\text{S}$. Following the only possible transition in the model, the next state is CARRY1$_\text{C}$, where the client constructs the message using the format of template 11 and the rule `3; 13; 11`. This rule indicates that the data in the field 0 must be copied to the field 0 of the current template. This results in the message: `CARRY <object> UP`. When the robot has hit the upper wall, it builds a message in state CARRY2$_\text{C}$ according to template 10 and rule `11; 0; 10`. Now the robot is carrying the object to the `LEFT` in state CARRY3$_\text{C}$ until it finds the base. The object and the direction that must be followed are introduced in the message format of template 2 associated to this state by using the rules with transitions `2; 5; 2`.