Hidden Markov Models for Automated Protocol Learning

Sean Whalen[1,2], Matt Bishop[1], and James P. Crutchfield[1,2,3]

[1] Department of Computer Science
University of California, Davis
{whalen,bishop}@cs.ucdavis.edu
[2] Department of Physics
University of California, Davis
chaos@cse.ucdavis.edu
[3] Santa Fe Institute
1399 Hyde Park Road
Santa Fe, New Mexico 87501

Abstract. Hidden Markov Models (HMMs) have applications in several areas of computer security. One drawback of HMMs is the selection of appropriate model parameters, which is often ad hoc or requires domain-specific knowledge. While algorithms exist to find local optima for some parameters, the number of states must always be specified and directly impacts the accuracy and generality of the model. In addition, domain knowledge is not always available or may be based on assumptions that prove incorrect or sub-optimal.

We apply the ϵ-machine—a special type of HMM—to the task of constructing network protocol models solely from network traffic. Unlike previous approaches, ϵ-machine reconstruction infers the minimal HMM architecture directly from data and is well suited to applications such as anomaly detection. We draw distinctions between our approach and previous research, and discuss the benefits and challenges of ϵ-machines for protocol model inference.

Keywords: Statistical Inference, Reverse Engineering, Network Protocols, Markov Models, Computational Mechanics.

1 Introduction

Understanding the structure of a network protocol allows us to "speak" its language and converse with other systems on the network that use it. The structure of commonly used protocols, such as HTTP and FTP, are provided by their specification. In addition, these protocols use fragments of English as well as other ASCII text such as domain names. As a result, the presence of HTTP or FTP traffic can be identified by visual inspection of a network trace, assuming the channel is not encrypted. One can then use its specification, or one of many free or commercial tools, to understand the traffic present in the trace.

The task becomes more difficult when the protocol in question uses non-ASCII representations of state to establish connections and exchange data. Still, there are many approaches to identify the protocol such as using port numbers, unique payload signatures, or machine learning techniques [1]. Once the protocol is identified, the traffic can again be understood by using the specification.

In contrast to protocol identification, consider the scenario where we do not have access to the protocol's specification—it is either proprietary, undocumented, or otherwise obfuscated. We can treat the protocol as a black box, where a hidden state machine governs the transmission of packets on the network. To understand the structure of the packets, the task of protocol inference is to approximate this hidden state machine with only the observed packets as a guide.

Hidden Markov Models (HMMs) [2] are a common statistical model for systems with hidden internal states that can be measured only indirectly by observation. These models have numerous applications in computer science, including several in computer security. An HMM is specified by a state transition matrix and a symbol emission matrix. This means that, for an N-state HMM with a discrete alphabet of size M, there are $N(N-1)+N(M-1)$ free parameters to be specified. These parameters can be trained using the Baum-Welch algorithm [2], but training is often slow and gets stuck in local optima. In addition, the number of states must still be specified. A model with too many states tends to over-fit the data, while too few states may not fit the data at all. Worse, when dealing with an unknown protocol, there is little if any knowledge available for selecting appropriate model parameters.

To address this problem, we turn to the ϵ-machines of *computational mechanics* [3]. An ϵ-machine is the minimal deterministic HMM of a stochastic process. A *reconstruction algorithm* creates an ϵ-machine from a set of finite strings, and infers the parameters of the minimal HMM that generates those strings. Here, we treat a network protocol as a stochastic process and the traffic it generates as input strings for the reconstruction algorithm. With the ϵ-machine in hand, we can perform different tasks such as protocol mimicry, intelligent fuzzing, traffic generation, and anomaly detection.

We next discuss recent work done on protocol inference and the benefits of our approach. We follow this with background on HMMs and ϵ-machines, and demonstrate our reconstruction technique using several simple protocols. Finally, we discuss future applications and the limitations of our approach. Our discussion assumes familiarity with stochastic processes and information theory at the level of Cover and Thomas [4].

2 Related Work

Current approaches to protocol inference can be divided into two primary groups: those that infer partial or complete protocol formats [5, 6, 7, 8, 9, 10], and those that infer a state machine model [11, 12]. Both groups can be further divided into those that examine network traces [5, 11, 6, 7], and those that additionally examine how a protocol implementation processes those traces [8,9,10,12]. Each approach has different strengths and weaknesses, but both must identify the location and size of protocol headers.

Much of the recent work can be traced back to Protocol Informatics, which attempted to "determine the location and length of fields within protocol packets" using sequence alignment algorithms typically found in bioinformatics [5]. This approach was extended by RolePlayer, which used heuristics identify the locations of IP addresses and domain names in a packet, in order to "successfully replay one side of a [protocol] session" [6].

This work led to Discoverer, which focused on "reverse engineering the [complete] message format specification" [7]. In this work, Cui et al. found that selecting robust parameters for sequence alignment was difficult, and that alignment has trouble identifying variable length fields in messages of the same format. In response, they developed a type-based alignment algorithm that infers the semantics of different fields, and used these semantics to cluster messages of the same format. Inference of the state machine, which is the focus of our approach, was left to future work.

Prospex addressed this issue by inferring non-probabilistic state machines from execution traces of a protocol implementation [12]. Their state machine "reflects the sequences in which messages may be exchanged". They converted their machines into input specifications for the fuzzing tool Peach, and found several known and unknown flaws in open source software.

In contrast, our ϵ-machine approach infers the minimal HMM from passively observed network data without using execution traces. This strikes a middle ground between Discoverer and Prospex, with several unique contributions. These include using a probabilistic model that enables anomaly detection via model comparison techniques, and avoiding ad hoc specification of model parameters by inferring them from the data.

We continue with a brief overview of HMMs and ϵ-machines, and refer the reader to references [2] and [3] for further detail.

3 Background

A discrete stochastic process is a sequence $\ldots X_1, X_2, X_3 \ldots$ of random variables X_n indexed by time; realizations $\ldots x_1, x_2, x_3 \ldots$ are often referred to as *time series* data. Here, we use time series and *string* interchangeably. The set of strings a process generates forms a stochastic language in which each string occurs with some probability. We treat a network protocol as a stochastic process, transmitting packets in the protocol's language with varying probabilities. Several well known model classes, such as Markov Chains and Hidden Markov Models, are commonly used to represent finite-memory stochastic processes. We will consider these in turn, eventually introducing ϵ-machines as a useful alternative.

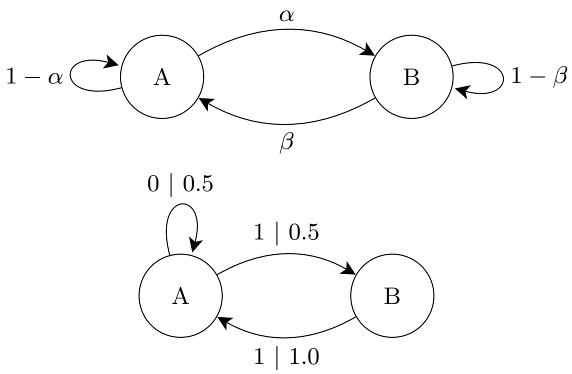

Fig. 1. *Top:* The general form of a two-state Markov Chain. *Bottom*: An HMM for the Even Process. A state transition occurs with some probability p and generates a symbol s, displayed as the edge label $s \mid p$. For example, the label $1 \mid 0.5$ indicates symbol 1 is generated with probability 0.5.

3.1 Markov Models

A Markov Chain is a representation of a stochastic process that assumes the conditional probability of a future state X_{n+1} depends only on the present state X_n [13]:

$$\begin{aligned}\Pr(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) \\ = \Pr(X_{n+1} = x | X_n = x_n) \ , \quad (1)\end{aligned}$$

where $X_1 \ldots X_n$ is a sequence of random variables representing process state over time. While this dependency can be extended to include a fixed number of past states, a finite state Markov Chain can only represent a stochastic process that has a limited dependence on history.

Transitions between states are random, occurring with probabilities specified in a row-normalized transition matrix T of size $|X| \times |X|$. Here, $|X|$ is the number of states. The probability of transitioning from state i to state j is denoted T_{ij}. As an example, a Markov Chain with two states $X = \{A, B\}$ has the form:

$$T = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} \ , \quad (2)$$

where $\alpha, \beta \in [0, 1]$ are parameters. This corresponds graphically to the state machine diagram shown in Fig. 1.

Due to their limited history dependence, Markov Chains represent only a subset of stochastic processes. Consider the Even Process which generates binary strings in which the number of consecutive 1s, bounded by 0s, is always even. For example, the strings 0110 and 011110 are in its language, but the string 010 is not. It turns out that this process is not equivalent to a Markov Chain of any finite order [14]. In such cases, one must employ a more sophisticated model class such as Hidden Markov Models (HMMs).

3.2 Hidden Markov Models

An HMM is a Markov Chain in which the states, now denoted $\boldsymbol{\mathcal{S}}$, are not observed directly but indirectly through measurement symbols X_n. Each observed symbol is generated by a transition between hidden states according to some distribution.

Since multiple transitions can generate the same observed symbol, the internal transitions and states of the system typically are not directly revealed by observation. Nonetheless, given an HMM, several quantities of interest can be calculated [2], including the probability of observing a particular string, the most likely hidden-state sequence for a given string, and the state transition and symbol emission probabilities that maximize the probability of a particular string.

The Even Process is exactly represented by the two-state HMM given in Fig. 1. By distinguishing internal states from observed symbols, HMMs can finitely represent a much wider class of stochastic processes than Markov Chains.

3.3 ϵ-Machines

In fact, Fig. 1 shows a special HMM representation for the Even Process—one with a minimal number of states. Moreover, the transitions are *deterministic*, meaning that the measurement symbols occur on at most one transition leaving a state. This property guarantees that, although there is not a one-to-one relationship between internal states and measurement symbols, there is a one-to-one relationship between sufficiently long measurement *words* and internal state *paths*. Thus, internal state information is present, if very indirectly, in the observed process.

An HMM with these properties is called an *ϵ-machine*. An ϵ-machine is the minimal, optimal predictor for a stochastic process and so captures all of the latter's causal structure [3]. For these reasons, this is the model class we will use.

Formally, an ϵ-machine consists of a set of *causal states* $\boldsymbol{\mathcal{S}}$, a measurement alphabet $\mathcal{A}$, and a set of transition matrices $\{T^{(s)} : s \in \mathcal{A}\}$. There are several *reconstruction algorithms* that one can use to infer an ϵ-machine from a time series. We use the state splitting algorithm of Shalizi et al., whose time complexity is $O(|\mathcal{A}|^{2L_{max}+1}) + O(N)$ [15]. Separate from the mathematical theory, different reconstruction algorithms make specific assumptions about the underlying process. The type of process being modeled thus affects the choice of algorithm.

Given an ϵ-machine, we can calculate certain important properties of a process. In particular, the determinism of the state transitions enables direct calculation of information-theoretic quantities, such as the process's rate of information production (source entropy rate) and the amount of historical information it stores (statistical complexity) [3,14]. Such properties cannot be calculated from an HMM representation that is not an ϵ-machine.

To estimate an HMM from time series, the number of states and transitions must be guessed a priori. In contrast, ϵ-machine reconstruction algorithms infer the minimal deterministic HMM architecture directly from time series data [16, 15]. This is a critical distinction if one wishes to discover the structure embedded in a process, as opposed to guessing it ahead of time. This is advantageous when reverse engineering protocols where prior information is unavailable.

4 Protocol Inference

4.1 Approach

We first define a network protocol as a set of *message types*. Each message type consists of a sequence of bits, and related bits are often grouped into *headers*. A particular message type may contain a set of headers, as well as a data *payload*. This payload may contain data provided by the user, or may *encapsulate* messages of a higher level protocol. Thus, we can think of a protocol message at several levels of abstraction: as a sequence of bits, bytes, or headers and payloads.

By changing the level of abstraction, we can adjust the order of the underlying Markov Chain as well as its alphabet size $|\mathcal{A}|$. In addition, we are interested in the structure of the protocol and not highly entropic user data such as images or movies, so we attempt to detect and ignore payloads. This further reduces the alphabet size, and is essential to the practical use of ϵ-machines.

Consider a minimal protocol having a single message type, consisting of a 2-byte length header and a payload. The length header specifies the number of bytes in the payload as an unsigned 16-bit integer. A four byte message sending the ASCII characters for "no" could then be viewed as a sequence of bits:

00000000 00000010 01101110 01101111

or of bytes:

0 2 110 111

or of headers and payloads:

2 "no"

The binary sequence has $|\mathcal{A}| = 2$, but requires a prohibitive order-16 model to capture the first header. At the byte level this becomes order-2, but $|\mathcal{A}|$ increases to 4. Finally, if we know where the separation between header and payload is, we can use an order-1 model with $|\mathcal{A}| = 2$. If more messages are observed, the alphabet size of the byte representation could increase to 256, so operating at the header level is desirable. Of course, knowing the location and size of message headers requires either the protocol specification or heuristics.

Protocols such as HTTP and FTP use ASCII tokens, so header boundaries are easily identified by inspection—typically tabs, spaces, newlines, and carriage returns. For more difficult binary protocols, Beddoe aligns bytes across different messages using bioinformatics algorithms and then uses simple statistics as a boundary detection heuristic [5]. Cui et al. discuss difficulties with sequence alignment and devise a significantly improved type-based alignment algorithm [7]. Both methods are compatible with our approach, though we use minimum entropy clustering [17] to group messages of the same type and then apply the simple statistics of Beddoe to identify likely header boundaries. This method is adequate for the protocols discussed here, but complex protocols may require additional sophistication.

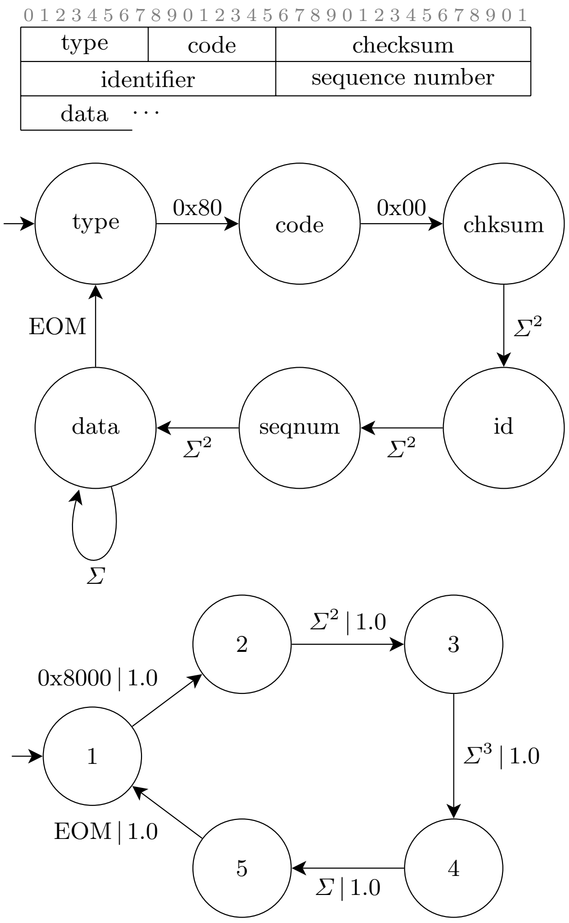

Fig. 2. *Top:* Specification of an ICMP echo request [18], *Middle:* HMM representation, *Bottom:* ϵ-Machine representation. The symbol Σ represents a random byte of data, with Σ^n denoting n consecutive random bytes.

Thus, our inference approach can be separated into three primary tasks: 1) grouping bytes into headers, 2) filtering highly entropic data, and 3) reconstructing the ϵ-machine. Tasks 1 and 2 exist primarily to reduce the alphabet size, and can be changed independently of task 3. For example, type-based alignment could be exchanged with minimum entropy clustering to transparently improve the accuracy of the inferred model.

4.2 Results

We introduce protocol ϵ-machines and their accompanying notation using two simple binary protocols, followed by two more complex protocols. The first of these, the Internet Control Messaging Protocol (ICMP) [18] defines several message types used for network diagnostics. One of these types, the echo request/reply, finds common use in the ping command line utility bundled with most operating systems. For this discussion we focus on echo requests, consisting of a 1-byte type set to `0x80`, 1-byte code set to `0x00`, 2-byte checksum whose contents are a function of the message, 2-byte identifier, 2-byte sequence number, and zero or more bytes of payload.

The message specification, a corresponding 6-state HMM, and the ϵ-machine are shown in Figure 2. A state is created in the HMM for each header, with transitions between states whose headers are adjacent in the specification. Transition are labeled with the symbols to be generated. The symbol Σ^n denotes n consecutive random bytes. The symbol `EOM` signals the message is complete and ready for transmission.

The ϵ-machine inferred from captured echo requests is shown below the HMM. Transitions are labeled with the symbol s generated by the transition and the probability p of the transition being taken, denoted $s \mid p$. This intentionally simple example has no branching between states, resulting in transition probabilities of 1. The type and code headers are constant values, causing their separate HMM states to be merged in the ϵ-machine.

Many protocols contain a sequence number header represented as a 16-bit integer. However, the first byte of this header changes very rarely compared to the second byte that is incremented with each message. In the requests captured for this example, the identifier header and the first byte of the sequence number remained constant, resulting in their grouping into a single value by the boundary detection heuristic (see the $\mathcal{A}^3$ transition between state 3 and 4). While this does not match the specification, it is a reasonable grouping to make based solely on the statistics of the observed messages. Given enough data, the bytes will be grouped into the correct fields.

We next examine Modbus, a protocol commonly used in supervisory control and data acquisition (SCADA) systems for managing industrial and infrastructure processes such as power generation and waste management. Designed in the late 70s to operate on simple programmable logic controllers, it has gained recent notoriety due to its lack of security. These issues have escalated due to the Modbus/TCP variant [19] connecting these systems to standard TCP/IP networks.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

trans id	prot id	
length	function	unit id
data ···		

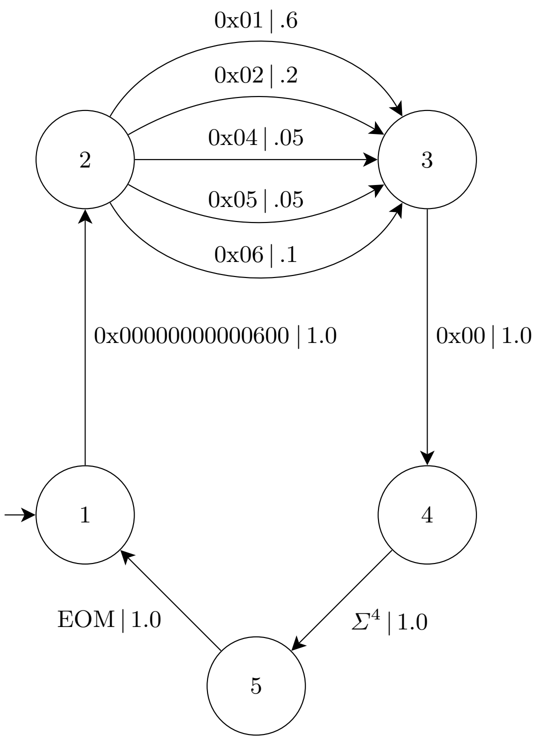

Fig. 3. *Top:* Specification of a Modbus/TCP request [19], *Bottom:* ϵ-Machine representation

The specification of Modbus/TCP requests and an inferred ϵ-machine are shown in Figure 3. A request consists of a 2-byte transaction id, 2-byte protocol id set to `0x00`, 2-byte length, 1-byte unit id, 1-byte function code, and variable length payload.

The captured traffic, generated by a protocol simulator, consists of 150 requests where all transaction ids and unit ids are set to `0x00`. Traffic generated by the simulator is actual Modbus traffic, and not a statistical approximation. Observed function codes include `0x01` for reading coils, `0x02` for reading discrete inputs, `0x04` for reading input registers, `0x05` for writing single coils, and `0x06` for writing single registers. Each payload contains 4 bytes specifying the range of coils or registers to read or write. Branching occurs between state 2 and 3, with each transition probability representing the maximum likelihood estimate of a different function code.

FTP

Sample Size	Recurrent States	Time (Seconds)
300	6	0.18
600	9	0.20
900	10	0.23
1200	10	0.11
1500	11	0.32

HTTP

Sample Size	Recurrent States	Time (Seconds)
14337	12	0.18
28674	14	0.35
43011	18	0.84
57348	20	1.16
71685	22	1.86

Fig. 4. Scaling of inferred states and inference time as a function of data length, for FTP (top) and HTTP (bottom). Time is not necessarily monotonically increasing due to finite sample effects. State counts are given for non-deterministic presentations of the ϵ-machine.

Reconstruction also works with more complex protocols such as FTP and HTTP. Model size prevents including the full ϵ-machines here, so we present summaries of their reconstruction in Figure 4. Shown is the scaling of inferred states and reconstruction time as a function of data length, performed on a single core of an Intel Core 2 Duo 2.4GHz CPU under OS X 10.6.3. A Python implementation of the state splitting reconstruction algorithm [15] was used, and will soon be available in the open source Computational Mechanics in Python (CMPy) library. Captured traffic was obtained from the UCDavis Honeynet Project.

An 18-state ϵ-machine can be inferred from 60,000 symbols in less than a second using an interpreted language. A random walk on the machine generates new packets that are accepted by a remote protocol implementation as valid, indicating the structure of the protocol is correctly captured. Together, these results show that probabilistic reconstruction of both binary and text-based protocols is possible when alphabet size is managed. Given this, we next discuss future applications of probabilistic models to protocol inference.

5 Future Work

Capturing probabilities enables an ϵ-machine to model normal behavior and detect anomalies using model comparison techniques. This is an advantage of ϵ-machines over non-probabilistic state machines such as minimized prefix tree acceptors [20].

We employ relative entropy [4] for model comparison between ϵ-machines, as well as measuring the model's fit. Relative entropy, also known as the Kullback-Liebler

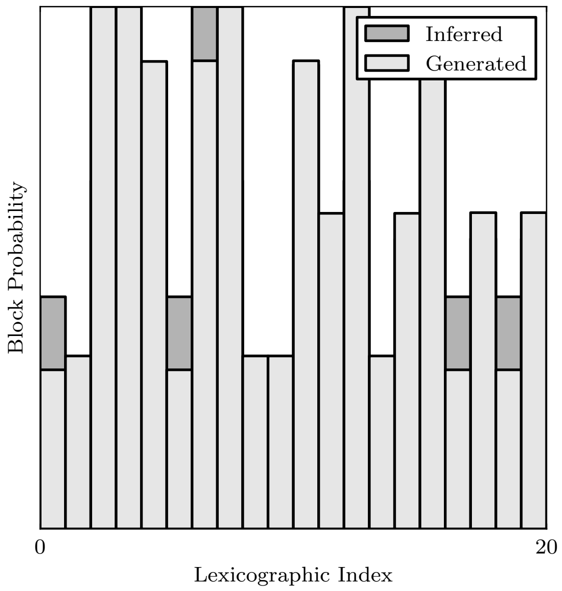

Fig. 5. Overlayed distributions of block length 4 symbols using an ϵ-machine inferred from Modbus traffic (dark gray) to generate new traffic (light gray). Relative entropy between the distributions is 0.09 bits, indicating the distributions are close.

divergence, measures the distance between two probability distributions P and Q and is defined as:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \ , \tag{3}$$

and $D_{KL}(P||Q) = 0$ when P and Q are equal. Thus, a large relative entropy between machines may indicate anomalous behavior. More work is needed to determine appropriate thresholds for flagging behavior as anomalous with an acceptable false positive rate.

A random walk on an ϵ-machine generates new traffic, useful for both protocol mimicry and traffic simulation. We measure the fit of the model by taking the relative entropy between the new and original traffic distributions as shown in Figure 5. An ϵ-machine inferred from 500 captured Modbus requests was used to generate new traffic with a distribution almost identical to the original, having 0.09 bits of relative entropy.

Random walks on the machine can also be used for intelligent fuzzing. Fuzzing tests the robustness of a program by feeding it invalid input values, often in the form of random inputs or mutated valid inputs. If the program does not correctly handle invalid input, it may crash or leave the system vulnerable to attack.

While generally considered effective for finding bugs, a substantial drawback to this approach is code coverage. For example, if the code's execution path depends on the value of a 32-bit integer, a random input has a 1 in 2^{32} chance of evaluating that code path [21]. Working with mutated valid inputs enables more targeted testing, but requires some knowledge of the specification. The inferred ϵ-machine enables such targeted fuzzing when no specification is available.

Consider a previously known flaw in Golden FTP Server 2.70 for Windows [22]. A CWD command sent from the client with certain large arguments crashes the server and enables remote code execution. Using an ϵ-machine inferred from FTP traffic and tuned to produce longer runs of random data, this flaw was reproduced by a random walk on the machine. The subgraph of the ϵ-machine relating to the crash is given in Figure 6. We plan to investigate if probabilistic models confer additional benefits to intelligent fuzzing.

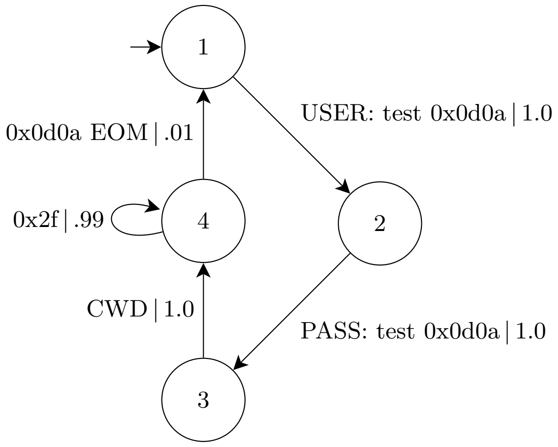

Fig. 6. Subgraph of the ϵ-machine used for fuzzing Golden FTP Server 2.70, crashing the server when a CWD command is followed by more than 150 bytes. Symbol probabilities in the inferred ϵ-machine were tuned to produce longer sequences of random data for guided fuzzing.

6 Conclusion

We presented a novel HMM-based approach for inferring the state machine of network protocols using only their traffic. While generally applicable to any non-encrypted protocol stream, our emphasis is on protocols without a publicly available specification.

Our approach uses ϵ-machine reconstruction [3] to infer the minimal deterministic HMM of a protocol. The parameters of the HMM are inferred directly from the data, which avoids the typical pitfalls involved in parameter selection. We demonstrated our approach using ICMP, Modbus, FTP, and HTTP, and use the inferred ϵ-machines for protocol mimicry, fuzzing, and traffic generation. We plan to use the probabilistic nature of our models for anomaly detection, and have early success doing so in high performance computing environments.

Due to the limitations of traffic-based approaches, as well as sensitivity to alphabet size, more work remains to adapt reconstruction to high complexity protocols. In some cases where domain knowledge is available, traditional HMMs may scale better than ϵ-machines, and we leave this investigation to future work. However, our approach is well suited to protocol inference where domain knowledge is lacking for manual construction of state machine models.

Acknowledgements

This work was partially supported by the DARPA Physical Intelligence Program, as well as the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC02-05CH11231.

References

1. Erman, J., Mahanti, A., Arlitt, M.: Internet traffic identification using machine learning. In: Proceedings of the 49th IEEE Global Telecommunications Conference, pp. 1–6 (2006)
2. Rabiner, L.: A tutorial on Hidden Markov Models and selected applications in speech recognition. Proceedings of the IEEE 77, 257–286 (1989)
3. Crutchfield, J.P., Young, K.: Inferring statistical complexity. Phys. Rev. Let. 63 (1989); Crutchfield, J.P.: Physica D 75 11–54 (1994); Crutchfield, J. P., Shalizi, C. R.: Phys. Rev. E 59(1), 275–283, 105–108 (1999)
4. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley Interscience, New York (2006)
5. Beddoe, M.: Network protocol analysis using bioinformatics algorithms. Technical report, McAfee Inc. (2005)
6. Cui, W., Paxson, V., Weaver, N., Katz, R.: Protocol-independent adaptive replay of application dialog. In: Proceedings of the 13th Annual Symposium on Network and Distributed System Security (2006)
7. Cui, W., Kannan, J., Wang, H.: Discoverer: Automatic protocol reverse engineering from network traces. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pp. 1–14 (2007)
8. Lin, Z., Jiang, X., Xu, D., Zhang, X.: Automatic protocol format reverse engineering through context-aware monitored execution. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (2008)
9. Wondracek, G., Milani Comparetti, P., Kruegel, C., Kirda, E.: Automatic network protocol analysis. In: Proceedings of the 15th Symposium on Network and Distributed System Security (2008)
10. Caballero, J., Poosankam, P., Kreibich, C., Song, D.: Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In: Proceedings of the 16th ACM conference on Computer and Communications Security, pp. 621–634 (2009)

11. Leita, C., Mermoud, K., Dacier, M.: Scriptgen: An automated script generation tool for honeyd. In: Proceedings of the 21st Annual Computer Security Applications Conference, pp. 203–214 (2005)
12. Milani Comparetti, P., Wondracek, G., Kruegel, C., Kirda, E.: Prospex: Protocol specification extraction. In: IEEE Symposium on Security and Privacy (2009)
13. Norris, J.R.: Markov Chains. Cambridge University Press, Cambridge (1997)
14. Crutchfield, J., Feldman, D.: Regularities unseen, randomness observed: Levels of entropy convergence. Chaos 15, 25–54 (2003)
15. Shalizi, C.R., Shalizi, K.L.: Blind construction of optimal nonlinear recursive predictors for discrete sequences. In: Proceedings of the 20th conference on Uncertainty in Artificial Intelligence, pp. 504–511 (2004)
16. Shalizi, C., Shalizi, K., Crutchfield, J.: Pattern discovery in time series, Part I: Theory, algorithm, analysis, and convergence, 2002 Santa Fe Institute Working Paper 02-10-060; arXiv.org/abs/cs.LG/0210025
17. Li, H., Zhang, K., Jiang, T.: Minimum entropy clustering and applications to gene expression analysis. In: Computational Systems Bioinformatics Conference, International IEEE Computer Society, pp. 142–151 (2004)
18. Postel, J.: Internet Control Message Protocol (1981), Updated by RFCs 950, 4884
19. Modbus Organization: Modbus Messaging Implementation Guide 1.0b (2006)
20. Bugalho, M., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recognition 38 (2005)
21. Godefroid, P.: Random testing for security: blackbox vs. whitebox fuzzing. In: Proceedings of the 2nd international workshop on Random testing, p. 1 (2007)
22. Infigo Information Security: Multiple FTP Servers vulnerabilities (2006) (accessed October 29, 2006)