Polymorphic Code Detection with GA optimized Markov Models

Udo Payer[1] and Stefan Kraxberger[2]

[1] Institute for Applied Information Processing and Communications (IAIK), University of Technology Graz
[2] Stiftung - Secure Information and Communication Technologies (SIC), Graz, Austria

Abstract. This paper presents our progression in the search for reliable anomaly-based intrusion detection mechanisms. We investigated different options of stochastic techniques. We started our investigations with Markov chains to detect abnormal traffic. The main aspect in our prior work was the optimization of transition matrices to obtain better detection accuracy. First, we tried to automatically train the transition matrix with *normal* traffic. Then, this transition matrix was used to calculate the probabilities of a dedicated Markov sequence. This transition matrix was used to find differences between the trained normal traffic and characteristic parts of a polymorphic shellcode. To improve the efficiency of this automatically trained transition matrix, we modified some entries in a way that byte-sequences of typical shellcodes substantially differs from normal network behavior. But this approach did not meet our requirements concerning generalization. Therefore we searched for automatic methods to improve the matrix. Genetic algorithms are adequate tools if just little knowledge about the search space is available and the complexity of the problem is very hard (NP-complete).

Keywords: intrusion detection, polymorphic shellcode detection, markov models, genetic algorithms, optimization.

1 Introduction

During the past years, different polymorphic shellcode engines have shown up in the internet. The concept of polymorphism is not new in the field of viruses, but it took about 10 years that these polymorphic mechanisms were also used in the field of polymorphic shellcodes. The most popular representatives are CLET and ADMmutate. Especially on the example of CLET, the authors of [CLET03] used a spectrum analysis mechanism to defeat data mining methods. The problem was to develop an engine which is capable to generate shellcodes which will be considered as normal by NIDSs. The basic idea of this approach was to analyze it usual traffic generated by the *usual use* of network services. This mechanism is described in [ADM03] and [CLET03] in more detail. The knowledge about bytes and byte-sequences can then be used to generate shellcode-sequences, depending

on the probability of each occurring byte in the it normal traffic.
CLET and some other polymorphic engines try to be as similar to the *normal* traffic as possible. The *problem* with known shellcode-engines is that just single parts of the generated codes are used to adjust the generated byte-spectrum to the byte-spectrum of the overall network traffic. All other parts remains unchanged and can therefore be detected by using statistical methods. As shown in [Yn01],[YEZ02],[Yn00],[JTM01],[JV99] Markov models and HMMs are stochastic methods which can be used if an statistical relation between events and intrusion is given. Therefore it also must be able to use these methods to make decisions directly upon network traffic.

2 Markov models

2.1 Overview

A Markov chain is a sequence of random values whose probabilities at a given time depends upon conditional probabilities of the recent past. The controlling factor in a Markov chain is the transition matrix which is used to calculate the conditional probabilities of dedicated state sequences and lengths.

2.2 Definition

There are three items involved to specify a general markov chain:

- State space S.

  S is a finite set of states. Let us label the states as $S = \{1, 2, 3, ..., N\}$ for some finite N.
- Initial distribution a_0.

  This is the probability distribution of the Markov chain at time 0. For each state $i \in S$, we denote by $a_0(i)$ the probability $P = \{X_0 = i\}$ that the Markov chain starts in state i. Formally, a_0 is a function taking S into the interval [0,1] such that

  $$a_0(i) \geq 0 \text{ for all } i \in S \tag{1}$$

  and

  $$\sum_{i \in S} a(i) = 1. \tag{2}$$
- Probability transition matrix P.

  If S is the finite set $\{1, 2, ..., N\}$, then P is an $N \times N$ Matrix. The interpretation of the number p_{ij} is the conditional probability, given that the chain is in state i at time n, and that the chain jumps to the state j at time $n + 1$. That is,

  $$p_{ij} = P\{q_{n+1} = j | q_n = i\}. \tag{3}$$

We can also express the probability of a certain sequence $\{q_1, q_2, \ldots, q_n\}$ (the joint probability of the recent past and current observations) using the Markov assumption:

$$P(q_1, ..., q_n) = \prod_{i=1}^{n} P(q_i|q_{i-1}) \tag{4}$$

3 Transition matrix

In our first approach we trained the transition matrix automatically from a given traffic data. This given network traffic was real network traffic captured, and stored in a file. Thereafter, this file was used to train a Markov model. But before training, this Markov model has to be initialized. Therefore, we have to specify and initialize the following items:

- State space S
  Because every character is coded as 1 byte, the possible state space for network traffic would be $S = \{0, 1, 2, ..., 255\}$. Due to the fact that Markov models are not intended to use zero as a state, we shifted the state space by 1. Therefore we used $S = \{1, 2, 3, ..., 256\}$ as state vector.
- Initial distribution a_0.
  For the first version we used an initial distribution of equal probability for every state to be the first state.

$$a_0(i) = 0,00390625 \text{ for all } i \in S \tag{5}$$

- Probability transition matrix P.
  As stated in section 2.1 it is crucial to determine an appropriate transition matrix to get good results. This is the main part of our work. In our first approach we learned the transition matrix automatically from a given traffic data. Therefore we collected the probabilities for all transitions from one character to another. By using this information it is possible to detect something which is not *normal* in relation to the learned traffic. The training data is represented as an array **b** with elements from the state space S. Then we counted all transitions from one particular character to another (e.g. 129 to 192) and divided the sum by the whole number of transitions for this character. The transition probability p_{ij} for one possible transition $i \rightarrow j$ requires the computation of

$$r_{ij} = \# \{k \in \{1, \ldots, N-1\} : \mathbf{b}_k = i \wedge \mathbf{b}_{k+1} = j\} \tag{6}$$

  where N specifies the length of the data **b** from which we learn the transition matrix. Then, p_{ij} can be written as

$$p_{ij} = \frac{r_{ij}}{\sum_{j=1}^{N} r_{ij}}. \tag{7}$$

Figure 1 shows the probability distribution of a CLET polymorphic shellcode calculated with a Markov sequence length of 30 using the Markov assumption. The x-axis shows the byte length (position) and the y-axis represents the probability value for the sequence. To proof if a sequence is a polymorphic shellcode we used a threshold which we got from probability observations of many real shellcodes. With the learned transition matrix it was possible to detect the decipher engine and the enciphered code.

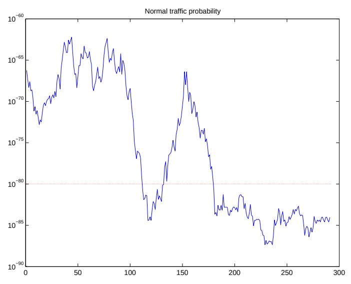

Fig. 1. Learned transition matrix

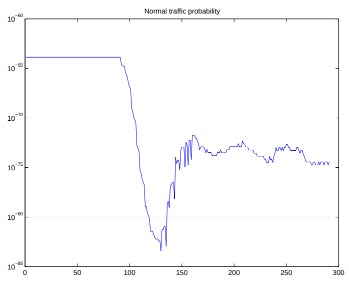

Fig. 2. Designed transition matrix

Since we did not get just *one* significant pike, causing more false positives, we decided to *improve* the distribution matrix. We used the knowledge that some byte-sequences are more likely in shellcodes than in normal traffic and modified the corresponding values manually. This modified matrix applied to shellcodes resulted in a substantial difference compared with normal network traffic. With the created transition matrix we are able to obtain the requested probability distribution (Figure 2). The First 100 Bytes show the probability of the NOP zone and then the significant pike represents the decipher engine. Afterwards the ciphered shellcode itself is displayed.

4 Optimizing the transition matrix

Since it is very hard to manually modify a transition matrix and take all important parameters into account, we searched for new solutions to this problem.
Possible modifications are manifold and we just know very little about the values (instructions) and their influence on the result. Therefore, we decided to give an automated (optimization) search algorithm a chance. There are many methods which can be used to find a suitable solution, but all these methods do not necessarily show the best solution [RN95],[PJ84]. The solutions found by these methods are often considered as *good solutions*. One reason is that it is often very hard to prove the correctness of possible optimal solutions. Therefore, we decided to give the genetic algorithm a try.

5 Genetic algorithms

Genetic algorithms are inspired by Darwin's theory of evolution. Solution to a problem solved by genetic algorithms uses an evolutionary process (it is evolved). The algorithm starts with a set of solutions (represented by chromosomes) called population. Solutions from one population are selected and used to form a new population through mutation and cross-over. This is motivated by a hope, that the new population will be better (yield better) than the old one. Solutions which are then selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to be reproduced.

5.1 Definitions

GAs always deal with solutions, goals, criteria, and fitness functions. These general terms are described in more details within this section.

- Solution

  In our case a solution is a representation of a specific transition matrix representing a set of possible offsprings. Due to the fact that our GA-implementation could not handle matrices, we converted the transition matrix into a vector where the rows of the matrix are appended consecutively. So we got a vector with 65536 values.
- Goal

  To measure the fitness of the solutions we have to compare the calculated probability distribution of an distinct shellcode with a desired probability distribution. This desired distribution is called **goal**.
- Evaluation criteria

  Since a genetic algorithm demands a single value as a measure for the fitness, we subtract the probability distribution of the solution from the goal, squared them, summed it up and divided by the length of the goal vector.

  $$fitnessvalue = -\frac{\sum_{i=1}^{N}(R_i - G_i)^2}{N} \tag{8}$$

  where R is the result and G the goal vector for the probability distribution of a single solution. N specifies the length of the distribution vector.
- GA parameters

  We used the GAOT Matlab package from the North Carolina State University with their default settings for *floatGA* [HJK95]. The options we used where *[1e-2 1 1 0.1]*.

5.2 Optimizing the detection for one specific shellcode

First we tried to use just one shellcode for our evaluation function to see if it is possible to optimize the transition matrix. By using just a single shellcode, it was possible to generate a transition matrix nearly reaching the preferred goal (figure 3). The preferred goal in this case shows a very significant peak at the position where the decipher engine appears in the evaluation function.

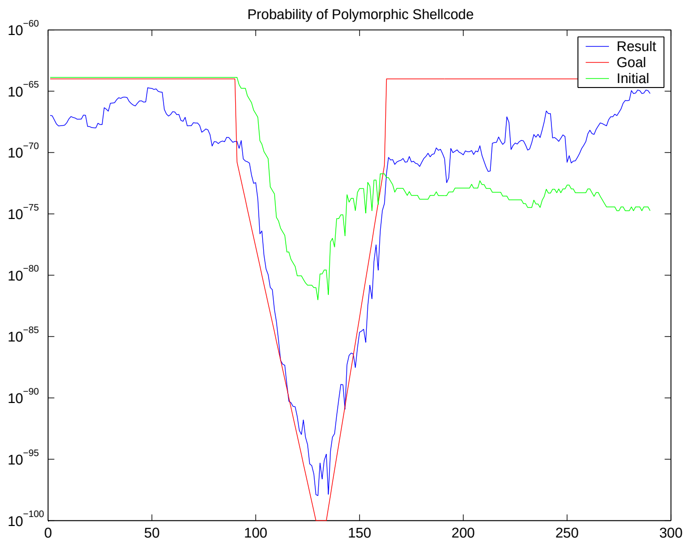

Fig. 3. Optimization result for an arbitrary shellcode

But due to the fact that we just used a single shellcode, the obtained transition matrix is very specialized and just qualified to find this single shellcode. All other shellcodes (generated by the same engine) seems to be to different and cannot be detected by this "improved" transition matrix. In figure 4 you can see five deciphering engines with a sequence-probability of 10^{-70}, whereas the second one shows a a small peak in the middle of the shellcode where we expect the deciphering engine. All other shellcodes just show that shellcodes are more unlikely that normal network traffic, but the significant peak in the middle of the shellcode is missing. The strong peaks at the top of the figure came from 0 bytes in the network traffic. Since no 0's are allowed in a polymorphic shellcode we do not calculate the sequence probability of such a sequence and assign a value of 1 instead. This very much depends on the selection of the boundaries. So we need another approach.

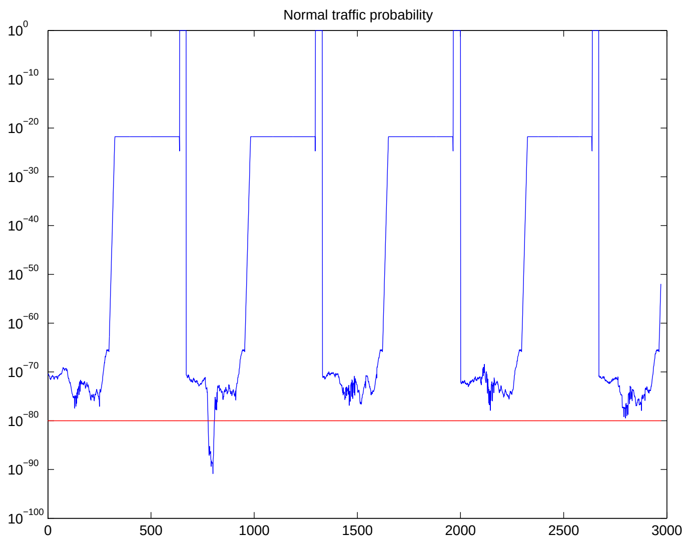

Fig. 4. Detection result for 5 shellcodes

5.3 Applying the evaluation function to 10 shellcode instances

Since always a single fitness value decides about success or failure of a dedicated solution it is quite obvious that more samples would lead to better results. We started with a sequence of just 10 shellcodes packed-up back-to-back to evaluate our solutions. All used shellcodes were generated randomly and can even be a mix of different shellcode generators (Clet, ADMmutate, JempiScode).
After this preparation phase, the evaluation function is applied to the result of a dedicated solution (solution-vector). In doing so, the fitness-values are calculated for each single shellcode (according to 8) and the arithmetic median on all fitness values is calculated and returned to the genetic algorithm.

In figure 5 the sequence of 10 different shellcodes, the best initial solution, the goal, and the best found solution is shown. Here we see that calculating the arithmetic median on several shellcode instances and the use of this value as fitness value to train the GA yields much better than 5.2. In figure 5, you still can see the difference between normal traffic and shellcodes. But the most interesting point is the existence of the conspicuous peak in the middle of all shellcodes.

5.4 40 decipher engines and similar traffic as evaluation function

The next idea was to use shellcode-data from different decipher engines. No longer we are dealing with the whole shellcode. From now we just look at the most interesting part of the shellcode - the decipher engine (the *peaks*). To get a goal-function, we had to add some other traffic at the beginning and at the

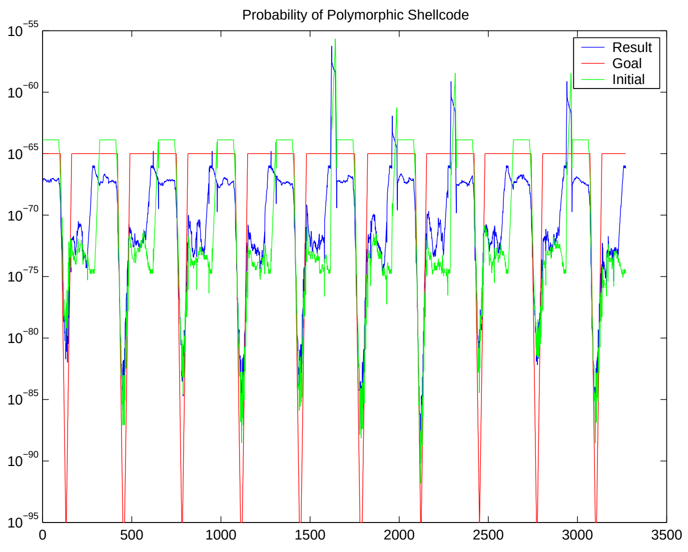

Fig. 5. Optimization result for 10 different shellcodes

end of our evaluation example. To get better results, we used normal traffic data detected as false positives, by the captured method described in 5.3. And once again - in figure 6 the best initial solution the goal and the best found solution is displayed.
The green line represents the untrained, initial behavior. Since the shown diagram is an enlarged picture of the interesting part of the diagram, we can se no difference between the deciphering engine and the rest of the shellcode.
The blue line is the trained result, showing significant differences 10^{10} between the deciphering engine and byte sequences looking very similar to decipher engines. By careful threshold-selection, we are now able to distinguish between real shellcode and false positives detected by 5.3, since peaks detected by 5.3 at 400 and 600 are eliminated by the improved transition matrix.

6 Experimental results

At least we present the detection results for all transition matrices we produced. We have tested our implementation with real data from the hard disk. The amount of data we used was 126 MB, which we collected from different locations to get fair distributed data. The data itself contains no shellcodes but decipher engine similar code. We used a threshold for the detection with which could detect all the shellcodes from our test set. We are using only sequence calculation without any additional improvements like:

- NOP-zone detection
- Prefilter- or preprocessing-techniques

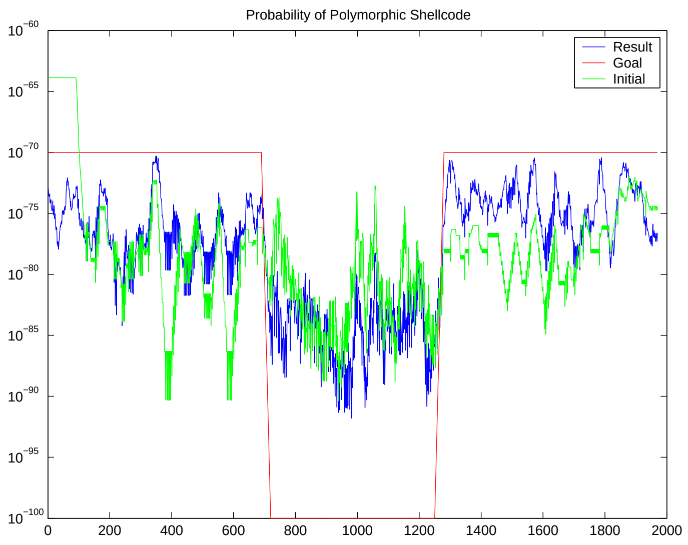

Fig. 6. Optimization result for 40 decipher engines and similar traffic data

- Abstract Payload Execution
- Assembler command improvement

We calculated the probability of Markov sequences with a length of 30 for the whole data by using a sliding window. The sliding window was shifted by one for every new sequence.

	P1	P2	P3	P4
False negatives	0	0	0	0
False positives	33540	2540	652	13

Table 1. Markov model detection performance with different transition matrices

- P1 - Learned transition matrix from normal traffic.
- P2 - Manually created transition matrix.
- P3 - Transition matrix obtained as solution from approach 5.3
- P4 - Transition matrix obtained as solution from approach 5.4

7 Conclusions

Since we did not know a good algorithms to modify a Markov model, trained with normal network traffic (to be able to detect any deviations from normal

traffic) we used GAs to solve this problem. Starting with the evaluation of a single shellcode-instance, we proofed the concept of MM-adaptation by GAs to make MMs more significant in the special case of polymorphic shellcode.
We learned quick that a single shellcode-sample was insufficient to be used in our GA-fitness function. Thus, we increased the number of shellcode-probes and we used 10 instances to train the MM. We know that 10 instances are still insufficient to be able to detect a broader spectrum of polymorphic code (generated by different polymorphic generators). But the main idea of this paper was just to give a proof of concept.

References

[Cj04] Chang, J.: *Stochastic Processes.* (http://pantheon.yale.edu/ jtc5/251/) (Accessed 2004/11/17).

[CLET03] CLET Team: Polymorphic shellcode engine. *Phrack Magazine* 49(14).

[ADM03] ADMmutate: ADMmutate shellcode engine. (http://www.ktwo.ca) (Accessed 2004/11/24).

[Mm03] Mahoney, M.: *Network traffic anomaly detection based on packet bytes.* In Proc. ACM-SAC. 2003.

[KL04] Kolesnikov, O.; Lee, W.: *Advanced Polymorphic Worms: Evading IDS by blending in with normal traffic.* (http://www.cc.gatech.edu/ ok/) (Accessed 2004/11/16)

[Oe02] Oswald, E.: *Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems.* (CHES 2002). 4th International Workshop on Cryptographic Hardware and Embedded Systems. Redwood Shores. CA. USA.

[DHS00] Duda O., Richard; Hart E., Peter; Stork G., David: *Pattern Classification.* Wiley Intersience. New York.

[RN95] Russell A.; Norvig, P.: *Artificial Intelligence: A Modern Approach.* Prentice-Hall.

[PJ84] Pearl, J.: *Heuristics: Intelligent search strategies for computer problem solving.* Addison-Wesley.

[HJK95] Houck, Chris; Joines, Chris; Kay, Mike: *A Genetic Algorithm for Function Optimization - A Matlab Implementation.* NCSU-IE TR 95-09.

[Yn01] Nong Ye et al.: *Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data.* IEEE Transactions on Systems, man and cybernetics - Part A: Systems and Humans. Vol. 31. No. 4. July 2001.

[YEZ02] Ye, N.; Ehiabor, T.; Zhang, Y.: *First-order versus high-order stochastic models for computer intrusion detection.* Quality and realiability engineering international. 2002.

[Yn00] e, N.: *A Markov chain model of temporal behavior for anomaly detection.* In: Proc. of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop. New York

[JTM01] Jha, S.; Tan, K.; Maxion, R.A.: *Markov Chains, Classifiers, and Intrusion Detection.*(CSFW 01) 14th IEEE Computer Security Foundations Workshop. Cape Breton. Novia Scotia. Canada.

[JV99] u, W.H.; Vardi, Y.: *A hybrid high-order Markov chain model for computer intrusion detection.* Technical Report. TR92. National Institute of Statistical Sciences. 1999.