Markov Chains, Classifiers, and Intrusion Detection

S. Jha* K. Tan† R.A. Maxion†

Abstract

This paper presents a statistical anomaly detection algorithm based on Markov chains. Our algorithm can be directly applied for intrusion detection by discovering anomalous activities. Our framework for constructing anomaly detectors is very general and can be used by other researchers for constructing Markov-chain-based anomaly detectors. We also present performance metrics for evaluating the effectiveness of anomaly detectors. Extensive experimental results clearly demonstrate the effectiveness of our algorithm. We discuss several future directions for research based on the framework presented in this paper.

1 Introduction

An *intrusion detection system* (IDS) is a system that identifies *intrusions*, where intrusion is defined as misuse or unauthorized use by authorized users or external adversaries [17, 19]. Surveys of intrusion detection systems can be found in [1, 14, 16, 20]. A classification of intrusion detection systems appears in [11, Section II]. In this paper, we consider intrusion detection systems that are based on anomaly detection. The objective of anomaly detection is to establish profiles of "normal" system activity. Traces of system activity that deviate from these profiles are considered anomalous and consequently an alarm is raised. There are two classes of anomaly-detection-based IDS.

Signature or *pattern* based IDS have an internal table of "anomalous patterns". If an ongoing activity matches a pattern in the table, an alarm is raised. The table of patterns represent system traces corresponding to common attacks. Examples of signature-based intrusion detection systems are Snort [22] and Bro [21]. The advantages of signature-based IDS are commonly known to be their potential for low false alarm rates, and the information they often impart to a system security officer about a detected attack. Such information is often encoded in the rules or patterns central to the functionality of such systems. This information is often invaluable when initiating preventive or corrective actions. However, signature-based IDS have several disadvantages. Since the set of anomalous patterns are based on known attacks, new attacks cannot be discovered by these systems. Therefore, whenever a new attack is discovered, patterns corresponding to the attack have to be manually constructed. Moreover, signature-based IDS can be easily fooled by a sophisticated attacker. For example, an attacker can "mix" normal activity with a real attack so that the trace does not match any of the pre-defined patterns. *Statistical* anomaly-detection-based IDS (henceforth referred to as statistical IDS), have been devised to address these shortcomings of signature-based IDS. Denning and Nuemann presented a detailed discussion of a statistical anomaly detection algorithm [5]. IDES [15] is a prototypical example of a statistical IDS. In a statistical IDS a model of the normal behavior of a user is constructed. The statistical model is then used to construct a classifier that can discriminate between normal and anomalous traces. The techniques presented in this paper fall into the second category. However, we also describe a procedure for generating anomalous patterns from our statistical model. Therefore, techniques presented in this paper can also be used to automatically generate patterns for signature-based systems. An important question in anomaly-detection based intrusion detection systems is: how is the trace of the system activity represented? We use the sequence of system calls corresponding to a process as the trace of its activity. To the best of our knowledge, this was first proposed in [9]. However, our approach is general and can be used for other types of traces of system activity, such as audit trails.

Any statistical approach to intrusion detection adheres to the general strategy described below. First, using a set of normal traces a statistical model is constructed. This statistical model is then used to construct a classifier that can discriminate between normal and abnormal traces. The key observation is that the statistical model is an accurate predictor of normal behavior, so if an on-going activity is not

*Computer Sciences Department, University of Wisconsin, Madison, WI 53706.

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

accurately predicted by the model, it is likely to be anomalous. This general strategy is depicted in Figure 1. Our approach follows this general road-map. Using a set of normal traces, we construct a Markov chain. The Markov chain is then used to construct a classifier. The main contributions of this paper are:

- We provide a formal framework for constructing classifiers based on Markov chains. We also investigate applications of these classifiers to intrusion detection.
- We provide several metrics for evaluating effectiveness of classifiers in the context of intrusion detection. These metrics can be also used by other researchers.
- Our framework is quite general and can be used to construct other classifiers based on Markov chains.

The outline of the paper is as follows: Section 2 provides a general outline of our algorithm. Detailed description of our algorithm is given in Section 3. Section 4 describes an algorithm for generating a set of anomalous patterns from Markov chains. This algorithm is suitable for generating anomalous patterns used by systems such as Snort [22] and Bro [21]. Experimental results are described in Section 5. Section 6 describes related work. Future work and concluding remarks are provided in Section 7 and 8 respectively.

2 Outline of the Methodology

In this section, we provide a step-wise description of our methodology. Technical details are given in Section 3. Assume that we are given two suites of traces T and T_{an}. Recall that in our case a trace is simply a sequence of system calls generated by a process. The suite T consists of traces of normal activity and T_{an} consists of traces of anomalous activity (presumably corresponding to some known attacks).

- **Step 1** ***(Construct the test suite)***
  In this step we split the suite T into two. The first suite T_{tr} is called the *training suite* and is used for constructing classifiers. The second suite T_{te} is called the *test suite* and is used for testing classifiers and tuning various parameters. First, we decide a ratio γ which we call the *testing ratio*. We use random sampling to construct T_{tr} and T_{te}. For each trace σ in T, we generate a random number u which is uniformly distributed over the range $[0, 1]$. If $u \leq \gamma$, the trace is added to T_{te}, otherwise it is added to T_{tr}. Roughly speaking, γ denotes the fraction of the traces that are in the test suite T_{te}.
- **Step 2** ***(Construct a classifier)***
  We use the training suite T_{tr} to construct a Markov chain, which is then turned into a classifier for traces. Since the classifier is constructed from a suite of normal traces, it is able to discriminate between normal and anomalous traces. Details of the construction can be found in Section 3.
- **Step 3** ***(Tuning Parameters)***
  There are various exogenous parameters used during the construction of the classifier. First, we define various performance metrics for a classifier. These metrics are computed using suites T_{te} and T_{an}. Various exogenous parameters are tuned using these performance metrics.

3 Detailed Description

Let Σ denote the set of *alphabets* or *symbols*. We will use alphabets and symbols synonymously throughout the paper. A *trace* over Σ is a finite sequence of alphabets. The set of finite *traces* over Σ is denoted by $\Sigma^\star$, the *empty trace* is denoted by ϵ, and the set of traces of length n is denoted by Σ_n. Given a trace $\sigma \in \Sigma$, $|\sigma|$ denotes the length of the trace. Given a trace σ and a positive integer $i \leq |\sigma|$, α_i and and $\alpha[i]$ denote the prefix consisting of the first i alphabets and the i-th symbol respectively. The concatenation of two traces σ_1 and σ_2 is denoted by $\sigma_1 \cdot \sigma_2$. $B = \{0, 1\}$ denotes the *binary* alphabet set.

Definition 3.1 A *classifier* over the alphabet set Σ is a total function $f : \Sigma^\star \to B$.

A *suite* over Σ is a subset of $\Sigma^\star$. In our experiments, we will use three types of suites, the *training suite* T_{tr} (used for training), the *test suite* T_{te} (used for testing), and the *anomalous suite* T_{an} (set of anomalous traces). The training suite, which is a set of normal traces, is used to construct a classifier. The test suite is also a set of normal traces and is used to test and tune the classifier. The anomalous suite is a set of anomalous or abnormal traces.

Note: The reader should interpret 1 as "bad" and 0 as "good". In the context of intrusion detection, if a classifier outputs a 1 after reading a trace, it should be interpreted as an alarm (something anomalous is happening). On the other hand, 0 indicates normal behavior. In the general *classification problem*, there are a finite number of classes $\{1, \cdots, M\}$. Let U be the universe of objects to be classified. A classifier f is a function from U to $\{1, \cdots, M\}$ [6]. In the context of intrusion detection we want to classify traces as normal or anomalous, so $M = 2$.

Intrusion detection is an on-line activity where alarms have to be raised in real-time. Therefore, for the purposes of intrusion detection it is unacceptable to watch the entire sequence of activities (or equivalently scan the entire trace) and then classify the sequence. Next, we formalize what it

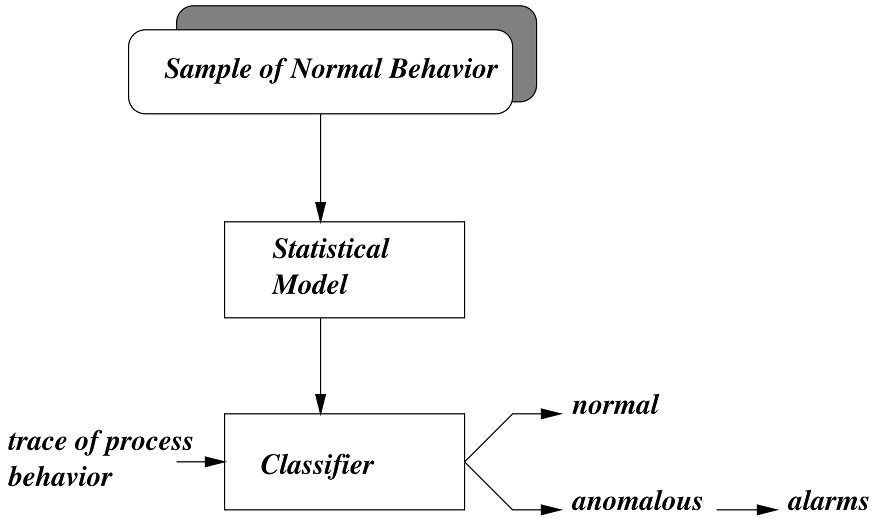

Figure 1. General Strategy for Intrusion Detection

means for a classifier to be *on-line*. Intuitively, an on-line classifier can efficiently classify a trace σ of length n based on a the history of the classifier on the prefix σ_{n-1} and the last two symbols $\sigma[n-1]$ and $\sigma[n]$.

Definition 3.2 A classifier $f : \Sigma^\star \rightarrow B$ is called *on-line* if and only if there exists "efficiently computable"[1]. functions H, T, and $\beta : \Sigma^\star \rightarrow \Re^k$ such that following equations hold:

$$\begin{aligned} \beta(\sigma) &= H(\beta(\sigma_{n-1}), \sigma[n-1], \sigma[n]) \\ f(\sigma) &= T(\beta(\sigma)) \end{aligned}$$

In the equations given above, the length of the trace σ is denoted by n. Types for functions H and T can be easily inferred from the equations. Notice that the value of the function β on the trace σ only depends on the value of the function for σ, and the last two symbols of the trace. In some sense, β only depends on the "immediate history" and can be efficiently computed (loosely speaking the function β is "Markov" because its value only depends on the immediate history and the current state).

Example 3.1 Assume that we are given a finite set T of traces. A classifier f outputs a 1 after scanning a trace σ if there is a suffix of σ that is in the set T; otherwise, the classifier f outputs a 0. In other words, the classifier f outputs an alarm as soon as it detects a pattern from the set T. For example, if $T = \{aaa, baa\}$, then the classifier outputs a 1 after scanning $acaaa$ and $acbaaa$. Hence classifiers can model most *signature-based* intrusion detection systems. A set of finite traces T can be compiled into a deterministic finite state automata A_T. Using this observation it can be easily seen that a signature based IDS corresponds to an on-line classifier.

Let T be a finite table of patterns. Consider the regular language L_T over the alphabet Σ such that a trace σ is in L_T iff there is a suffix of the trace that is in T. Let $A(L_T)$ be the deterministic finite state automata corresponding to L_T. Consider definition 3.2. Let δ be the next-state transition function for the automata A_T. The transition function δ can be extended to traces in a standard manner. Let $\beta(\sigma)$ be identifier of the state $\delta(s_0, \sigma)$, where s_0 is the initial state of the automata $A(D_T)$. The function H simply "mimics" the next-state transition function δ and uses the following equality:

$$\delta(s_0, \sigma) \quad = \quad \delta(\delta(s_0, \sigma_{n-1}), \sigma[n])$$

We assume that each state of the automate has an associated identifier that is a real number. The function T maps an identifier of a final state of the automata to 1 and the rest of the real numbers are mapped to 0. It is easy to verify the β, T, and H have the required properties and are efficiently computable.

3.1 Constructing Markov Chains

Assume that we are given a *training suite* $T_{\text{tr}} \subseteq \Sigma^\star$. We will use the suite T_{tr} to construct a Markov chain. In the next subsection we demonstrate how to construct a classifier from a Markov chain. Construction of the Markov chain is parametrized by the *window size* w. First, we augment the alphabet set Σ with a special *null symbol* ϕ. We will not provide background about Markov chains in this paper. Interested readers should consult a standard text on probability theory (such as [7]) for the required background.

[1]The precise definition of "efficiently computable" depends on the statistical model being used. In our case, efficiently computable means polynomial in the number of states in the Markov chain

Before we describe the algorithm to construct a Markov chain from the suite T_{tr}, we will define a few primitive operations. A *state* in the Markov chain is associated with a trace of length w over the alphabet $\Sigma \cup \{\phi\}$. A *transition* is a pair of states. The pair (s, s') denotes a transition from s to s'. Each state and transition is also associated with a counter. We also maintain a hash table H of visited states. The hash function used for the hash table is not crucial to the description of the algorithm. The primitive operation *shift*(σ, x) shifts the trace σ left and appends the alphabet x at the end, e.g., *shift*(aba, c) is equal to bac.

The initial state of the Markov chain (denoted by *initial-state*) is associated with trace of length w consisting of all null symbols ϕ, e.g., if $w = 3$, the initial state is associated with the trace $[\phi, \phi, \phi]$. The operation *next*(σ) returns the first symbol of the trace σ and left shifts σ by one position, e.g. *next*(abc) returns a and updates the trace to bc. For each trace $\sigma \in T_{\text{tr}}$, we execute the following steps until all the alphabets of σ are scanned.

1. Let $c =$ *next*(σ).
2. Set *next-state* to *shift*(*current-state*, c).
3. Increment the counter for the state *current-state* and transition (*current-state*, *next-state*).
4. Update *current-state* to be *next-state*.

After all the traces in the suite T_{tr} have been processed, each state and transition have a positive integer associated with them. The probability of transition (s, s') is $\frac{N(s,s')}{N(s)}$, where $N(s, s')$ and $N(s)$ are the counters associated with transition (s, s') and s respectively. In other words, probability of a transition is the *ratio of the frequency of the transition and the frequency of its source in the suite* T_{tr}.

Example 3.2 Assume that the training suite T_{tr} is

$$\{aabc, abcbc\}$$

The structure constructed after scanning all the traces in the training suite T_{tr} is shown in Figure 2. The figure also shows the counters associated with the states and the transitions.

3.2 Markov Chains to Classifiers

Assume that a Markov chain corresponding to a training suite T_{tr} with window size w has already been constructed. We will denote the Markov chain MC by a 3-tuple (S, P, s_0), where S is the set of states, $P : (S \times S) \to \Re_+$ denotes the transition probabilities, and $s_0 \in S$ is the initial state. The probability of a transition (s, s') is denoted by $P(s, s')$. In order for P to be a valid measure, the following equality should hold for all states s:

$$\sum_{s' \in succ(s)} P(s, s') = 1$$

where *succ*(s) denotes the set of successors of s. The trace of length w associated with the state s is denotes by $\sigma(s)$. Recall that the trace associated with the initial state s_0 is the trace consisting of all null alphabets ϕ.

Consider a trace $\alpha \in \Sigma^\star$. Recall that $\alpha[i]$ denotes the i-th alphabet of the trace α. Let the initial trace β_0 be $\sigma(s_0)$, i.e., the trace associated with the initial state s_0. The trace after scanning the first symbol $\alpha[1]$ is $\beta_1 =$ *shift*$(\beta_0, \alpha[1])$. The trace β_k obtained after scanning the k-th symbol is recursively defined as

$$\textit{shift}(\beta_{k-1}, \alpha[k]).$$

Hence, a trace α defines a sequence of traces $\beta_0, \cdots, \beta_m$ (where each trace β_i is of length w and $m = |\alpha|$). We define a metric $\mu(\alpha)$ corresponding to a trace α. This metric will be based on the Markov chain $MC = (S, P, s_0)$ and will be computed iteratively. Initially, let Y and X both be equal to 0.0, and $i = 0$. Until i is equal m we execute the following steps:

1. There are two cases.
   (Case A:) $\beta_i \to \beta_{i+1}$ is a valid transition in MC
   If there are two states s and s' in the Markov chain MC such that $\sigma(s) = \beta_i$ and $\sigma(s') = \beta_{i+1}$, then update Y and X according to the following equations:
   $$Y = Y + F(s, (s, s'))$$
   $$X = X + G(s, (s, s'))$$
   (Case B:) β_i or β_{i+1} is not a state in MC
   If $\beta_i \to \beta_{i+1}$ is not a valid transition in MC, then we update Y and X according to the following equations:
   $$Y = Y + Z$$
   $$X = X + 1$$
2. Increment i to $i + 1$.

The metric $\mu(\alpha)$ is defined as $\frac{Y}{X}$ at the end of the procedure just described. The procedure we just outlined, defines a function $\mu : \Sigma^\star \to \Re$. Intuitively, the metric $\mu(\alpha)$ measures how well does the Markov chain MC predicts the trace α, e.g., a lower $\mu(\alpha)$ indicates that the Markov chain predicts the trace α well. Notice that μ is parametrized by the functions F, G and the number Z. Different choices of F and G will result in different classifiers.

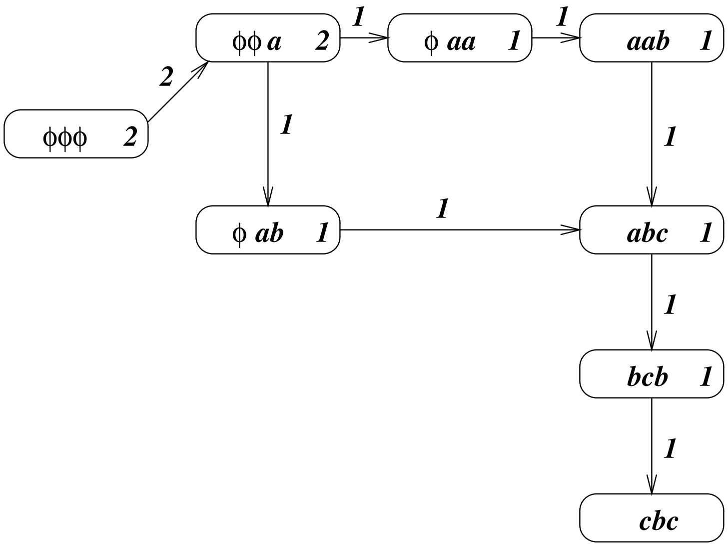

Figure 2. The Markov Structure

Assume that we are given a threshold $r \in \Re$. A classifier f can be constructed from the metric μ in the following manner:

$$f(\alpha) = \begin{cases} 1 & \mu(\alpha) \geq r \\ 0 & \text{otherwise} \end{cases}$$

In other words, the trace is classified as "bad" if the metric is above a certain threshold r.

Note: It can be easily seen that the classifier f is on-line (see definition 3.2). Define a function $\beta : \Sigma^\star \rightarrow \Re^2$ as

$$\beta(\sigma) = (Y, X)$$

where Y and X are variables defined by the procedure for computing μ. Function H in the definition 3.2 can be easily derived from the description of the procedure to compute μ. The function T is given by

$$T(Y, X) = (\frac{Y}{X} \geq r) .$$

Hence, the classifier f we construct is an on-line classifier. The complexity of functions β, H, and T is linear in the number of states of the Markov chain.

Discussion: For a trace α, the metric $\mu(\alpha)$ depends on the entire trace. Recall that the procedure maintains cumulative values Y and X and the metric is defined as $\frac{Y}{X}$. Therefore, we call the metric μ *global*. A classifier that is based on a global metric is called *global*. Next, we describe a *local classifier*. Recall that a trace α corresponds to a sequence of traces $\beta_0, \cdots, \beta_m$, where $m = |\alpha|$ and each trace β_i is of length w. Let z be a positive integer. After scanning the i-th symbol $\alpha[i]$ of the trace α, the history of size z is given by $(\beta_k, \beta_{k+1}, \cdots, \beta_i)$, where k is the minimum of 0 or $i - z + 1$. The history of size m corresponds to two vectors $(y_k, \cdots, y_i)$ and $(x_k, \cdots, x_i)$, where y_j and x_j are defined as:

- $\beta_{j-1} \rightarrow \beta_j$ is a valid transition in MC.
  Let s and s' be two states in the Markov chain MC such that $\sigma(s) = \beta_i$ and $\sigma(s') = \beta_{i+1}$. In this case y_j and x_j are defined as $F(s, (s, s')$ and $G(s, (s, s'))$ respectively.

- β_i or β_{i+1} is not a state in MC
  In this case y_j and x_j are defined as Z and 1 respectively.

The local metric $\gamma(\alpha)$ corresponding to the trace α is defined as

$$\frac{\sum_{i=m}^{k} y_i}{\sum_{i=m}^{k} z_i}$$

where $k = \min(0, m - z + 1)$. In other words, the *local metric is computed using the history of past z symbols*. A classifier that is based on a local metric is called a *local classifier*. We have also implemented local classifiers in our system. However, in our experiments, global classifiers consistently out-performed their local counterparts. Therefore, we have not presented the results for the local classifiers.

Next, we discuss some common functions F and G. Depending on the choice of these functions we obtain different classifiers.

- **The miss-probability metric**
  In this case the functions F and G are defined as follows:

$$F(s, (s, s')) = \sum_{s_1 \in succ(s) \wedge s' \neq s_1} P(s, s_1)$$

$$\begin{aligned} G(s,(s,s')) &= \sum_{s_1 \in succ(s)} P(s,s_1) \\ &= 1 \end{aligned}$$

The function $F(s,(s,'s))$ adds all the probabilities of the transitions from the state s that are not equal to (s,s'). In other words, if while scanning a trace a low probability transition is taken, then $F(s,(s,s'))$ has a higher value.

- **The miss-rate metric**

$$\begin{aligned} P_{\text{max}}(s) &= \max_{s' \in succ(s)} P(s,s') \\ F(s,(s,s')) &= (P(s,s') \neq P_{\text{max}}(s)) \\ G(s,(s,s')) &= 1 \end{aligned}$$

In this case every transition that is not equal to the "maximal" transition (the transition with the maximum probability) is penalized equally by 1.

- **The local-entropy-reduction metric**
Given a state s, the *local entropy* of the state (denoted by $LE(s)$) is

$$\sum_{s' \in succ(s)} -P(s,s') \log(P(s,s'))$$

The *entropy* of a Markov chain $MC = (S,P,s_0)$ is given by

$$\sum_{s \in S} \pi(s) LE(s)$$

where $\pi(s)$ is the steady state probability of being in state s. Descriptions of Markov chains and their entropy can be found in [7] and [4] respectively. Functions F and G in this case are defined as:

$$\begin{aligned} F(s,(s,s')) &= LE(s) + P(s,s') \log(P(s,s')) \\ G(s,(s,s')) &= LE(s) \end{aligned}$$

For a transition (s,s'), the expression

$$LE(s) + P(s,s') \log(P(s,s'))$$

evaluates to

$$\sum_{s_1 \in succ(s) \wedge s_1 \neq s'} -P(s,s_1) \log(P(s,s_1))$$

In other words, $LE(s)$ denotes the "residual" local entropy of the state s after deleting the transition (s,s'), or the local entropy reduction due to taking the transition (s,s').

3.3 Performance metrics

In this section, we discuss various performance metrics for evaluating effectiveness of a classifier $f : \Sigma^\star \rightarrow B$. Assume that we have constructed a classifier f using a training suite T_{tr}. Let T_{te} and T_{an} denote the test and anomalous suite respectively. Given a trace σ, the number of alarms that a classifier f generates (denoted by *alarm*(σ, f)) is given by the following expression:

$$\sum_{i=1}^{|\sigma|} f(\sigma_i)$$

In other words, the number of alarms is the number of 1's the classifier f generates while scanning the trace σ. The test suite T_{te} consists of normal traces. The *percentage of false alarms* corresponding to the test suite T_{te} and the classifier f is given by the expression

$$\frac{\sum_{\sigma \in T_{\text{te}}} alarm(\sigma, f)}{\sum_{\sigma \in T_{\text{te}}} |\sigma|}$$

The percentage of false alarms a classifier f generates on a test suite T_{te} is denoted by FA(T_{te}, f). A high percentage of false alarms is undesirable because every alarm in an IDS has to be attended to by a system administrator. In our experience, when a system administrator encounters a high percentage of false alarms, they turn off the IDS.

For anomalous traces, a good classifier f should generate an alarm quickly. Therefore, for anomalous traces we define *mean time to first alarm* or *MTFA*. Given an anomalous trace σ, classifier f will generate its first alarm on σ after scanning $\lceil MTFA \star |\sigma| \rceil$ symbols. Given a classifier f and a trace σ, let *alarm*$_1(\sigma, f)$ be defined as

$$\min\{i | f(\sigma_i) = 1\}.$$

In other words, while scanning the trace σ, the classifier f generates its first alarm after reading the *alarm*$_1(\sigma, f)$-th symbol. *MTFA* corresponding to an anomalous suite T_{an} and a classifier f (denoted by *MTFA*(T_{an}, f)) is given by the following expression:

$$\frac{\sum_{\sigma \in T_{\text{an}}} \frac{alarm_1(\sigma, f)}{|\sigma|}}{|T_{\text{an}}|}$$

In the expression given above, $|T_{\text{an}}|$ denotes the the number of traces in T_{an} or the size of the suite. Intuitively speaking, the metric *MTFA* measures how fast a classifier f detects an anomalous trace.

One can generalize the metric *MTFA* to *mean time to k-th alarm* or *MTkA*, i.e., the time at which the classifier generates the k-th alarm. However, in this paper we will only consider MTFA.

3.4 Tuning Parameters

Recall that in our method, there are five exogenous parameters in constructing a classifier from a training suite T_{tr}. These parameters are:

- the window size w,
- functions $F(s,(s,s'))$ and $G(s,(s,s'),$
- a real number Z,
- and a threshold r.

Assume that the functions F and G and the real number Z have already been determined. We discuss how to decide the values of the parameters w and r.

As the window size w increases, the Markov chain constructed is a better model of the training suite T_{tr}. However, for a very large w the Markov chain MC models the training suite "too well". This is the classical over-fitting problem in statistics. Let $\mu(\sigma)$ be the metric defined earlier. Intuitively, $\mu(\sigma)$ is a measure of the *discrepancy* between the Markov chain MC and the trace σ, i.e., lower $\mu(\sigma)$ denotes a better fit. The discrepancy $D(T_{\mathrm{tt}})$ over the entire test suite T_{tt} is

$$\frac{\sum_{\sigma\in T_{\mathrm{tt}}}\mu(\sigma)}{\sum_{\sigma\in T_{\mathrm{tt}}}|\sigma|}$$

Discrepancy $D(T_{\mathrm{an}})$ corresponding to the anomalous suite T_{an} is defined in a similar manner. We keep increasing the window size w until the separation $D(T_{\mathrm{an}})-D(T_{\mathrm{tt}})$ between the anomalous and test suites is above a certain threshold. This means that the classifier is able to "discriminate" between the anomalous and normal traces. We will provide a detailed account of this in the experimental section.

Assume that the window size w has been determined. Next we describe how to set the threshold r. If the threshold r is low, then on an anomalous trace the classifier f will generate an alarm very quickly. Therefore, a low r means a lower *MTFA*. However, a lower threshold also generates more false alarms on normal traces, which is inconvenient for the system administrator. Therefore, there is a tradeoff between setting the threshold r too low. We set the threshold r so that the percentage of false alarms corresponding to the test suite and the *MTFA* of the anomalous suite are both below an acceptable level. Details on tuning parameters w and r can be found in the experimental section.

4 Generating Anomalous Patterns from Classifiers

Most intrusion detection systems that are commercially available are *signature-based* systems. Recall that a signature-based IDS relies on a table T of anomalous patterns. Whenever a suffix of a trace of an activity matches a pattern in the table T, an *alarm* is raised. Patterns in the table T correspond to common attacks or simply patterns that do not represent "normal" behavior of the system. Classifiers that are constructed from Markov chains (such as the one discussed in Section 3) are much more general than simply a table of patterns. However, they do not fit the current architecture for systems that are currently available. In this section we address this mis-match. We describe an algorithm that constructs a table of anomalous patterns from a Markov chain constructed from a training suite T_{tr}.

Recall that a Markov chain MC is a 3-tuple (S,P,s_0), where S is the set of states, P denotes the transition probability, and s_0 is the initial state. The reader should refer to subsection 3.1 for an explanation of these terms.

The metric $\mu:\Sigma^\star\rightarrow\Re$ defines a total order on the traces, i.e., $\sigma_1\geq\sigma_2$ if and only if $\mu(\sigma_1)\geq\mu(\sigma_2)$. A procedure for computing this metric was given in subsection 3.2. In the procedure described in subsection 3.2, we started from the initial state s_0. If the procedure is started from a state s of the Markov chain, it defines a new metric $\mu_s:\Sigma^\star\rightarrow\Re$. Intuitively, μ_s defines a metric if the initial state of the Markov chain is s.

Recall that each state s is associated with a unique trace $\sigma(s)$ of length w. Therefore, state and trace will be used synonymously throughout this section. Moreover, let $\Sigma=\{x_1,\cdots,x_m\}$ be the alphabet set. In our context, Σ are the system calls that appear in the various suites. Our algorithm is based on the function *pattern*(s,L,k). This function returns a set of trace $W\subseteq\Sigma^\star$ that satisfy the following conditions:

- each trace in W is of length less than or equal to L,
- the number of traces in W is less than or equal to k,
- and for all traces $\sigma\in\Sigma^\star\setminus W$, if $|\sigma|\leq L$, then there exists a trace $\sigma'\in W$, such that $\mu_s(\sigma)\leq\mu_s(\sigma')$.

In other words, W represents a set of k traces of length less than or equal to L that have the "highest" μ_s-value, or they represent k "worse traces" of length less than or equal to L. We give a recursive definition for the function *pattern*$(\cdot,\cdot,\cdot)$. The recursion is on the second parameter L.

(Base case): $L=0$
In this case,

$$pattern(s,0,k)\;=\;\{\epsilon\}$$

In other words, for $L=0$ the set of traces/patterns only contains the empty trace ϵ.

(Recursion): $L>0$
Let s_i be the state such that the trace $\sigma(s_i)$ corresponding to

it is *shift*$(\sigma(s), x_i)$. In other words, the trace corresponding to s_i is constructed by shifting the trace associated with s and appending the i-th alphabet x_i. Assume that the set of traces *pattern*$(s_i, L-1, k)$ have been computed for all the states. We give a definition of the set *pattern*(s, L, k) in terms of the sets

$$\textit{pattern}(s_1, L-1, k), \cdots, \textit{pattern}(s_k, L-1, k) \ .$$

For the sake of brevity we will denote the set *pattern*$(s_i, L-1, k)$ by W_i. Suppose the transition $s \rightarrow s_i$ corresponds to the alphabet $x_i \in \Sigma$, i.e., the trace associated with s_i is *shift*$(\sigma(s), x_i)$, where $\sigma(s)$ is the trace associated with the state s. We construct the set $x_i \cdot W_i$ which is formally defined as

$$\{x_i \cdot \sigma | \sigma \in W_i\}$$

In other words, we add x_i in the front of every trace in $x_i \cdot W_i$. After that, we compute the metric μ_s for every trace in the sets $x_i \cdot W_i$ ($1 \leq i \leq k$). Next we construct a set of traces R given by the following expression:

$$\textit{pattern}(s, L-1, k) \cup (\bigcup_{i=1}^{m} x_i \cdot W_i)$$

Traces in R are sorted using their values according to the function μ_s. Intuitively, we sort the traces by the value that denotes how well does the Markov chain MC model the trace, where the traces with "bad fits" come first. The set *pattern*(s, L, k) is defined as the first k traces in the sorted order.

Correctness: Given a trace σ such that $|\sigma| \leq L$ that is not in *pattern*(s, L, k), we prove that there exists a trace $\sigma' \in$ *pattern*(s, L, k) such that $\mu_s(\sigma) \leq \mu_s(\sigma')$. We prove the result by induction on L. The result for $L = 0$ is obvious. After all, there is only trace of length 0, i.e., the empty trace ϵ. Assume that the result is true for $L-1$, where $L \geq 2$. If $|\sigma| \leq L-1$, then by the induction hypothesis there exists $\sigma' \in$ *pattern*$(s, L-1, k)$ such that $\mu_s(\sigma) \leq \mu_s(\sigma')$. Notice that we construct the set *pattern*(s, L, k) by sorting the following set according to the μ_s metric:

$$\textit{pattern}(s, L-1, k) \cup (\bigcup_{i=1}^{m} x_i \cdot W_i)$$

Therefore, there exists a trace $\sigma'' \in$ *pattern*(s, L, k) such that $\mu_s(\sigma') \leq \mu_s(\sigma'')$, and hence the result is proved. Suppose $|\sigma| = L$. Let the first alphabet of σ be x_i and s_i be the state corresponding to *shift*$(\sigma(s), x_i)$. As before, let W_i be the set *pattern*$(s_i, L-1, k)$. Let α be the suffix of σ starting at the second symbol. Since $|\alpha| = L-1$, by the induction hypothesis we have the following two cases:

Case: $\alpha \in$ *pattern*$(s_i, L-1, k)$

Trace α is in the following set:

$$\textit{pattern}(s, L-1, k) \cup (\bigcup_{i=1}^{m} x_i \cdot W_i)$$

Since this set is sorted using the metric μ_s, the result follows.

Case: $\alpha \notin$ *pattern*$(s_i, L-1, k)$

In this case there exists an $\alpha' \in$ *pattern*$(s_i, L-1, k)$ such that $\mu_{s_i}(\alpha) \leq \mu_{s_i}(\alpha')$. From the nature of the metric μ_s the following inequality easily follows:

$$\mu_s(x_i \cdot \alpha) \leq \mu_s(x_i \cdot \alpha')$$

Notice that $x_i \cdot \alpha'$ is in the set $x_i \cdot W_i$, and the result follows.

Improving efficiency: First, notice that $\mu_s(x \cdot \sigma)$ can be efficiently computed (for the metrics given in this paper, this can be done in $O(1)$ time) from the transition probabilities and $\mu_{s'}(\sigma)$, where the transition (s, s') corresponds to the alphabet x. Recall the procedure for computing the metric μ. Suppose we are given Y and X corresponding to the trace σ if the initial state is s'. Using this information, Y and X for $x \cdot \sigma$ can be computed in $O(1)$-time. Let Y and X values for the trace σ corresponding to the initial state s' be Y_1 and X_1. If $s \rightarrow s'$ is a valid transition in the Markov chain MC, then the Y and X values for the trace $x \cdot \sigma$ are

$$\begin{aligned} Y &= Y_1 + F(s, (s, s')) \\ X &= X_1 + G(s, (s, s')). \end{aligned}$$

Otherwise, Y and X are given by

$$\begin{aligned} Y &= Y_1 + Z \\ X &= X_1 + 1 \end{aligned}$$

The value of the metric μ_s on the trace $x \cdot \sigma$ is simply $\frac{Y}{X}$. Therefore, if for each trace σ in the set *pattern*(s, L, k) we keep the Y and X values of the traces, the recursive step becomes very efficient. Moreover, once we compute *pattern*(s, L, k), we can store that value and re-use (this is called *memeoazation* [3]). The number of traces (denoted by N) of length $\leq k$ over the alphabet Σ is $\frac{|\Sigma|^{k+1}-1}{|\Sigma|-1}$. It can be easily checked that the algorithm with these modifications runs in $O(NLk)$ steps, where each step can take $O(|\Sigma| k \log(|\Sigma| k))$ time. Notice that the worst case complexity of the algorithm is exponential in the size of the alphabet. We suspect that in practice the algorithm will run much faster. The computational complexity of the problem and heuristics for improving efficiency are left for future work.

The algorithm given above generates a set of anomalous traces from a Markov chain trained on a suite of normal traces. Suppose we train a Markov chain MC_{an} using a suite of anomalous traces. In this case we want to generate a set of patterns or traces that "fit" the Markov model MC_{an} well. This can be easily done using a variant of the algorithm given above. Instead of sorting in the descending order (according the value of the metric μ_s), we sort in the ascending order. Therefore, we obtain traces that have a low μ-value or fit the Markov model MC_{an} the best.

5 Experimental Results

The suites in this study were obtained from a large body of work performed by the researchers in the Computer Science Department, University of New Mexico, towards evaluating anomaly-detection algorithms. The original experiments that employed these suites and their results were published in [26]. The data consists of traces of eight privileged UNIX programs and a trace of an operating anomaly-detector called STIDE. A trace is a sequence of system calls issued by a single process during its execution. Privileged programs were targeted for monitoring since their misuse can cause greatest harm to the host. It should be noted that only the system calls were recorded in these traces, parameters passed to the system calls were ignored. Nevertheless the datasets comprises of data obtained from a wide variety of circumstances, e.g., data obtained from monitoring the normal usage of a program in the field, data obtained from monitoring programs that run as daemons and those that do not, programs that vary widely in their size and complexity, and different kinds of intrusions. Importantly, to provide some insight as to how an algorithm would perform in the field, the body of data consists of a combination of *live* and *synthetic* traces, where

- live is defined to be *traces of programs collected during normal usage of a production computer system*, and
- synthetic is defined to be traces *collected in production environments by running a prepared script; the program options are chosen solely for the purpose of exercising the program, and not to meet any real user's requests.*

Anomalous data was obtained by recording the system calls used by the monitored programs while an intrusion was present. The intrusions were obtained from public advisories posted on the Internet. More detailed information about the kinds of intrusions used or the dataset can be found in [26]. The table shown in Figure 3 lists the programs monitored for normal behavior and the names of the corresponding intrusions deployed to obtain intrusive data.

Figures 4, 5, and 6 show the difference between the discrepancy of the anomalous and test suite corresponding to the three classifiers. The first column of the table corresponds to the number of the suite. Subsequent columns show the results for various window sizes used to construct the Markov chain. For example, the second column shows the results when window size of 10 was used to construct the Markov chain. The first component in each entry corresponds to the test suite and the second component shows the results for the anomalous suite. For example, for set 2, window size 10, and the first classifier the discrepancy for the test and anomalous suites are 0.0448 and 3.5358 respectively. For suites where there was only one normal trace, we could not leave out traces to construct a test suite. Therefore, in these cases we mark the component corresponding to the test suite by a "*". Let D_{an} and D_{tt} be the discrepancy corresponding to the test and anomalous suite respectively. Ideally, we want $D_{\mathrm{an}} - D_{\mathrm{tt}}$ to be large. This means that the classifier does well on the test suite and worse on the anomalous suite. Observing the data, following points can be made:

- In all the suites except 5 and 8 we start observing the difference in the discrepancies right away.
- For suite 5 the difference in the discrepancies become pronounced after the window size of 40.
- For suite 8 the difference becomes significant after the window size of 45.
- Classifiers based on the miss-probability and miss-rate metric outperform the one based on the local-entropy metric. The results for the classifiers based on miss-probability and miss-rate are comparable.

For each suite we set the window size such that difference between the discrepancies D_{an} and D_{tt} is above an acceptable level. Therefore, we set the window size for suite 8 to 45. For all other suites, the window size is set to 40.

Initially the threshold r_0 was set to $\frac{D_{\mathrm{an}}+D_{\mathrm{tt}}}{2}$, or the average of the discrepancies for the anomalous and test suites respectively. Suites 1 and 6 only have one normal trace, so in this case D_{tt} is not available. For these cases we set the initial threshold r_0 to $0.8D_{\mathrm{an}}$. After that, we conducted several experiments with thresholds around the initial threshold. Lower threshold means that *MTFA* for the anomalous suite will be low, i.e., we will generate an alarm earlier on the anomalous trace. However, a lower threshold also means that the percentage of false alarms for the test suite will be large. Therefore, there is a tradeoff in setting the value of the threshold r. We conducted extensive experiments with several thresholds. However, for the sake of brevity we only report results for three threshold values and the classifier based on the miss-probability metric. Figure 7 shows experimental results for $0.8r_0$, r_0, and $1.2r_0$, where r_0 is the initial threshold. Following observations can be made from the data:

- The percentage of false alarms decreases as the threshold increases.
- The mean time to first alarm (or MTFA) increases as the threshold increases.
- There are few suites that do not show the above mentioned behavior (see results for suite 8).

dataset	System	Normal	Anomalous	Intrusion
1: Synthetic xlock	Linux	71 traces 339,177 calls	2 traces 949 calls	buffer overflow
2: Live xlock	Linux	1 trace 16,598,639 calls	2 traces 949 calls	buffer overflow
3: Live ”named”	Linux	27 traces 9,230,572 calls	2 traces 1800 calls	buffer overflow
4: Live login	Linux	12 traces 8,894 calls	9 traces 4,853 calls	trojanized login program
5: Live ps	Linux	12 traces 6,144 calls	26 traces 6,968 calls	trojanized ps program
6: Live inetd	Linux	3 traces 541 calls	31 traces 8,371 calls	denial of service
7: Live lpr (MIT)	SunOS 4.4.1	2,703 traces 2,926,304 calls	1002 traces 169,252 calls	symbolic link attack
8: Live lpr (UNM)	SunOS 4.4.1	4,298 traces 2,027,468 calls	1001 traces 168,236 calls	symbolic attack
9: Live STIDE	Linux	71,760 traces 44,500,219 calls	105 traces 205,935 calls	denial of service

Figure 3. explanation of data sets

- False alarm rates for all suites except 5 and 8 are very low. Recall that for suite 5 and 8 the difference between the discrepancies was not very large.

Remark: If a classifier generates an alarm while scanning an anomalous trace, then that trace is classified as anomalous. So the *hit rate* for a classifier can be defined as the percentage of traces in the anomalous suite $T_{\rm an}$ that are classified as anomalous. In our experiments, we discovered that the hit rate was always close to 100%. Therefore, we do not consider hit rate as a good metric for the classifiers considered in this paper. Percentage of false alarms and mean time to first alarm are better measures of effectiveness. However, we believe that devising other metrics for quantifying classifiers has received scarce attention in the intrusion detection literature.

6 Related Work

We only compare our work to intrusion detection schemes that are based on statistical anomaly detection. As already pointed out the first example of such a intrusion detection system was IDES [15]. IDES was a large scale system and considered several classes of events. We only considered traces of system calls. Therefore, a direct comparison of IDES and our technique is not possible. We compare our technique with some recent statistical anomaly detection schemes.

Nassehi [18] describes an anomaly detection scheme based on Markov chains. However, his anomaly detection algorithm is different from the one presented in this paper. Nassehi constructs a Markov chain by using a window of size one. Let n_S the vector of normalized frequencies of transitions in the Markov chain based on the sample used for training. An ongoing activity is traced through the Markov chain by maintaining a history of a certain window. Let n_h be the vector of normalized frequencies of transitions for the history. An alarm is raised if the following condition holds:

$$(n_h - n_S)\ D\ (n_h - n_S)^T\ >\ r$$

The matrix D and threshold r are described in detail in [18]. If R and S are the number of transitions and states in the Markov chain, then each time a symbol is scanned the time required is $5(R - S) + 1$. This seems prohibitively expensive for large intrusion detection systems. In contrast, our anomaly detection scheme requires constant time for each symbol that is scanned. Warrender et al. [26] used Hidden Markov Models [8] (or HMMs) as the underlying model. The results reported in [26] seem comparable to the one presented in this paper. However, the training algorithm for HMMs is very expensive, i.e., it runs in time $O(nm^2)$, where n is the number of states in the HMM and m is the size of the trace. In contrast, the training time for Markov chains is $O(m)$. Moreover, for each symbol (or in this case system call) their method takes $O(n^2)$ time. Our anomaly detector takes constant time to process each symbol, and therefore can generate alarms faster. We have also undertaken a rigorous study of HMMs for anomaly detection. This work is ongoing and we will report on the results at a future date. An *instance learning based* algorithm for anomaly detection is described in [12]. Their algorithm is used for *masquerade detection*, i.e., a user acting as another user. We have started investigating techniques for extending our algorithm for masquerade detection. Applications of anomaly detection for uncovering stealthy portscans is described in [23]. We are currently devising an algorithm similar to the one presented in this paper for detecting stealthy port scans.

Section 4 describes an algorithm for generating anomalous patterns from Markov models. We have not implemented this algorithm. Therefore, we cannot provide a detailed comparison of our algorithm with other such algorithms. A data mining based technique for generating anomalous patterns or rules is described in [13]. An intrusion detection system based on information retrieval techniques is described in [2]. An approach for discovering anomalous patterns or rules based on inductive learning is presented in [24]. Once the implementation of our algorithm is finished, we will perform a detailed comparison of our approach with other techniques.

There are obvious connections to hypothesis testing [10] and intrusion detection. Let M be the Markov model constructed from the test suite. Given a trace σ, we are testing the hypothesis that σ is generated from the distribution implied by the Markov model M. If we reject the hypothesis, then the trace σ is an anomalous trace. An algorithm for hypothesis testing that does not have apriori bound on the number of trials is the *sequential probability ratio test* developed by Abraham Wald [25]. We plan to investigate the literature on hypothesis testing in more detail in order to find connections with our algorithm.

7 Future work

There are several directions for future work. We want to improve the space and time efficiency of the algorithms used in our methodology. We want to investigate the computation complexity of the algorithm for generating anomalous patterns described in Section 4. Moreover, we want to devise heuristics for improving the efficiency of this algorithm. Currently, we only generate models from traces of system calls. We want to investigate other events related to system activity, such as audit trail data and network traffic. The Markov-chain based classifier described in this paper has been implemented as a part of a library being developed at the University of Wisconsin. We want to in-

Number	w=10	w=20	w=30	w=40	w=45
1	(*,3.6392)	(*,3.9473)	(*,4.0402)	(*,4.0402)	(*,4.0402)
2	(0.0448,3.5358)	(0.0842,3.7385)	(0.1857,3.7575)	(0.3128,3.7575)	(0.38139,3.7575)
3	(0.0039,1.8943)	(0.0039,2.3181)	(0.0038,2.5379)	(0.0038,2.6868)	(0.0038,2.7194)
4	(0.2637,0.7097)	(0.2543,0.9034)	(0.2522,0.9774)	(0.2522,1.0361)	(0.2522,1.0643)
5	(0.8537,0.5911)	(1.1473,1.0536)	(1.3531,1.4662)	(1.3504,1.8536)	(1.3506,2.0343)
6	(*,0.1100)	(*,0.1212)	(*,0.1283)	(*,0.1346)	(*,0.1377)
7	(0.0408,0.1984)	(0.0338,0.7268)	(0.0334,0.8092)	(0.0375,0.8132)	(0.0427,0.8205)
8	(0.0301,0.0532)	(0.0297,0.0467)	(0.0293,0.0380)	(0.0335,0.0452)	(0.0352,0.1045)
9	(0.0439,0.2501)	(0.0191,0.5673)	(0.0176,0.7684)	(0.0168,0.9030)	(0.0171,0.9524)

Figure 4. Results for miss-probability metric

Number	w=10	w=20	w=30	w=40	w=45
1	(*,3.6392)	(*,3.9473)	(*,4.0402)	(*,4.0402)	(*,4.0402)
2	(0.0310,3.5318)	(0.0771,3.7364)	(0.1808,3.7575)	(0.3087,3.7575)	(0.3766,3.7575)
3	(0.0016,1.8978)	(0.0016,2.3148)	(0.0016,2.5364)	(0.0016,2.6852)	(0.0016,2.7178)
4	(0.0073,0.5510)	(0.0006,0.7468)	(0.0006,0.8218)	(0.0006,0.8806)	(0.0006,0.9087)
5	(0.8475,0.5765)	(1.1411,1.0412)	(1.3457,1.4553)	1.3439,1.8453)	(1.3439,2.0260)
6	(*,0.1034)	(*,0.1146)	(*,0.1218)	(*,0.1281)	(*,0.1312)
7	(0.0273,0.1600)	(0.0257,0.7102)	(0.0269,0.8017)	(0.0312,0.8067)	(0.036,0.8076)
8	(0.0215,0.0485)	(0.0212,0.0432)	(0.0232,0.0333)	(0.0270,0.0381)	(0.0293,0.0971)
9	(0.0248,0.2434)	(0.0120,0.5637)	(0.0112,0.7657)	(0.0110,0.9004)	(0.0114,0.9499)

Figure 5. Results for miss-rate metric

Number	w=10	w=20	w=30	w=40	w=45
1	(*,3.6392)	(*,3.9473)	(*,4.0402)	(*,4.0402)	(*,4.0402)
2	(0.0572,3.5311)	(0.0907,3.7360)	(0.1920,3.7554)	(0.3191,3.7554)	(0.3872,3.7554)
3	(0.0127,1.8978)	(0.0127,2.3145)	(0.0125,2.5386)	(0.0125,2.6881)	(0.0122,2.7209)
4	(0.1891,0.6668)	(0.1796,0.8605)	(0.1780,0.9348)	(0.1779,0.9935)	(0.1779,1.0216)
5	(0.8524,0.6036)	(1.1442,1.0627)	(1.3470,1.4730)	(1.3436,1.8582)	(1.3439,2.0385)
6	(*,0.1226)	(*,0.1336)	(*,0.1406)	(*,0.1466)	(*,0.1496)
7	(0.0384,0.1811)	(0.0332,0.7177)	(0.0324,0.8012)	(0.038,0.8041)	(0.0435,0.8124)
8	(0.0317,0.0351)	(0.0298,0.0287)	(0.0292.0.0252)	(0.0339,0.0360)	(0.0362,0.0988)
9	(0.0646,0.2565)	(0.0323,0.5747)	(0.0274,0.7726)	(0.0264,0.9069)	(0.02630.9561)

Figure 6. Results for local-entropy-reduction metric

Number	$0.8r_0$		r_0		$1.2r_0$	
	FA%	MTFA%	FA%	MTFA%	FA%	MTFA%
1	*	39.46	*	53.38	*	82.92
2	0.0	39.35	0.0	42.09	0.0	48.84
3	0.0	6.68	0.0	7.21	0.0	7.75
4	0.0	15.77	0.0	8.40	0.0	8.54
5	10.67	16.53	9.11	18.06	7.03	19.81
6	*	0.45	*	0.45	*	0.45
7	0.09	90.38	0.08	91.56	0.07	93.33
8	3.87	50.04	10.59	84.63	6.13	84.63
9	0.07	14.29	0.04	14.49	0.025	14.87

Figure 7. Results for various thresholds

corporate other statistical models, such as Hidden Markov Models [8], into this library. Our goal is to incorporate a wide variety of classification techniques and test their effectiveness in the context of intrusion detection. We also plan to implement our algorithms in existing intrusion detection systems, such as Snort [22] and Bro [21]. We are also interested in investigating specialized applications of our statistical anomaly detection algorithm, such as detecting stealthy port scans [23].

8 Conclusion

In this paper, we presented an anomaly detection algorithm based on Markov chains. We presented a general framework for constructing classifiers from Markov chains and presented three specific classifiers based on this framework. Performance metrics to test these classifiers were also defined. Experimental results clearly demonstrated the effectiveness of our approach. Moreover, these classifiers can be easily incorporated into an intrusion detection system. This paper creates several avenues for future work as described in the previous section. We are currently pursuing these directions.

Acknowledgement

We thank the referees of *CSFW 14* for helpful comments. We are particularly thankful for referee three who pointed out the connection to hypothesis testing.

References

[1] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnegie Mellon, January 2000.

[2] R. Anderson and A. Khattak. The use of information retrieval techniques for intrusion detection. In *Proceedings of First International Workshop on the Recent Advances in Intrusion Detection (RAID)*, September 1998.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1991.

[4] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.

[5] D. Denning and P. Nuemann. Requirements and model for IDES - a real-time intrusion detection expert system. Technical Report Technical Report, CSL, SRI International, August 1985.

[6] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer Verlag, 1996.

[7] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, 2nd edition, 1995.

[8] R. Elliott, L. Aggoun, and J. Moore. *Hidden Markov Models: Estimation and Control*. Springer Verlag, 1995.

[9] S. Forrest, S. Hofmeyr, S. Somayaji, and T. Longstaff. A sense of self for UNIX processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

[10] R. Hogg and E. Tanis. *Probability and Statistical Inference*. Prentice Hall, 2001.

[11] K. Ilgun, R. Kemmerer, and P. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[12] T. Lane and C. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, August 1999.

[13] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, 1999.

[14] T. Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11-th National Computer Security Conference, Baltimore, MD*, pages 65–73, October 1988.

[15] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. garvey. A real-time intrusion detection expert system (IDES)-final technical report. Technical Report Technical report, Computer Science Laboratory, SRI international, Menlo Park, California, February 1992.

[16] N. McAuliffe, L. Schaefer, D. Wolcott, T. Haley, N. Kalem, and B. Hubbard. Is your computer being misused? In *Proceedings of the Sixth Computer Security Applications Conference*, pages 260–272, December 1990.

[17] B. Mukherjee, L. T. Heberlein, and K. Levitt. Network intrusion detection. *IEEE Network*, May/June 1994.

[18] M. Nassehi. Anomaly detection for Markov models. Technical Report Tech report RZ 3011 (#93057), IBM Research Division, Zurich Research Laboratory, March 1998.

[19] S. Northcutt. *Network Intrusion Detection: An Analyst's Handbook*. New Riders, 1999.

[20] P. Nuemann. A compartive anatomy of computer system/network anomaly detection. Technical report, CSL, SRI BN-168, Menlo Park, CA, May 1990.

[21] V. Paxon. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7-th USENIX Security Symposium, San Antonio, Texas*, 1998.

[22] M. Roesch. Snort- lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA conference*, November 1999.

[23] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. In *Proceedings of the ACM CCS IDS Workshop*, November 2000.

[24] H. Teng, K. Chen, and S. C.-Y. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *IEEE Symposium on Security and Privacy*, pages 278–284, 1999.

[25] A. Wald. *Sequential Analysis*. John Wiley and Sons, 1947.

[26] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.