Anomaly intrusion detection using one class SVM

Yanxin Wang, Johnny Wong, Andrew Miner
Department of Computer Science
Iowa State University
Ames,IA,50011
Email:{wangyx,wong,asminer}@cs.iastate.edu

Abstract— **Kernel methods are widely used in statistical learning for many fields, such as protein classification and image processing. We recently extend kernel methods to intrusion detection domain by introducing a new family of kernels suitable for intrusion detection. These kernels, combined with an unsupervised learning method - one-class Support Vector Machine, are used for anomaly detection. Our experiments show that the new anomaly detection methods are able to achieve better accuracy rates than the conventional anomaly detectors.**

I. Introduction

Intrusion detection refers to a broad range of approaches that detect malicious attacks on computers and networks. Generally speaking, these approaches can be categorized into misuse detection and anomaly detection.

Misuse detection works by comparing network traffic, system call sequences, or other measurable quantities to known attack patterns called *signatures*. While misuse detectors can be quite successful in preventing attacks with known signatures, they cannot detect previously unseen attacks, and sometimes fail to detect slight modifications to existing attacks. When a new attack surfaces, an attack signature must be constructed (usually by an expert), and the misuse detector must be reconfigured to include the new signature [1], [2], [3].

Anomaly detection systems "learn", or are told, what constitutes *normal* behavior, developing sets of normal models that are continually updated. An anomaly detector looks for deviations from the known normal behavior; when the deviation exceeds some threshold, an alarm is raised. Unlike misuse detectors, anomaly detectors are able to detect previously unseen attacks. However, anomaly detectors can suffer from a high false alarm rate; this can occur due to rare but legitimate behaviors (e.g., actions taken to respond for events of dropped packets or failed servers) that are not covered by the definition of normal system behaviors.

STIDE developed at University of New Mexico and Markov Chain anomaly detector introduced by Carnegie Mellon University are two well known anomaly detectors for UNIX system processes [4], [5]. The behavior of a process can be observed by tracing the system calls made by the process. These two anomaly detectors build normal models from a set of normal system call sequences made by the UNIX system process (called a training data set for an anomaly detector). The STIDE and Markov Chain anomaly detectors work very well when the testing data set does not include many normal patterns (sequences) that are not covered in the training data set; when the testing data set include many new normal patterns, the false alarm rate increases.

Machine learning algorithms offer one of the most cost-effective approaches to automated knowledge acquisition from data [6]. A variety of machine learning algorithms have been used to detect intrusions, including:

- Rules learning algorithms (e.g., RIPPER) [7], [8], [9], [10].
- Neural networks [11], [12], [13], [14].
- Genetic algorithm [15].
- Association rules and frequency episodes [16].
- Support vector machines (SVMs) [17], [18], [19].

This paper describes two anomaly detectors which are based on kernel methods and one-class SVM learning algorithm, and presents experimental results which show that the new anomaly detection methods have better accuracy rates (correctly classify the normal and abnormal instances) than STIDE and Markov Chain anomaly detectors.

Section II gives an overview of machine learning methods for intrusion detection and introduces SVM and kernel methods. In Section III, we present the theoretical analysis of STIDE anomaly detector using kernel theory and introduce STIDE kernel. Section IV gives an analysis of Markov Chain anomaly detector and introduces Markov Chain kernel. Section V demonstrates the experimental results and presents the discussion of our approach. Section VI concludes the paper with future research direction.

II. Related work

Misuse Intrusion Detection Systems use signatures to detect known attacks. Machine learning algorithms can help to learn these signatures. Also, in anomaly intrusion detection, machine learning method can address the challenge of building normal profile from the training data.

A. machine learning methods for intrusion detection

Decision trees are structures used to categorize data according to some common attributes, which are also called features [20]. Decision tree techniques are often used to automatically learn intrusion signatures for Intrusion Detection System (IDS) [21]. JAM project [16], a fraud and intrusion detection system developed for financial information systems in Columbia University, utilizes RIPPER [22], which is a decision tree learning algorithm developed by William Cohen of AT&T Laboratories. This approach first uses a set of data,

0-7803-8572-1/04/$20.00 ©2004 IEEE

including the 'good' and 'bad' labels, to train the learning machine, and then use the machine to predict if a record in the testing data set indicates an attack. In JAM project, the statistical attributes computed using simple *tcpdump* output are used as attributes for decision tree algorithm, and the rules derived from decision tree learning are coded and used to detect intrusions. Decision tree method is not easy to scale to large system.

Much research has been done on neural network for intrusion detection [23], [24], [25]. Neural network is often proposed as the statistical analysis component of an anomaly detection system. In [26], a model for recognizing abnormal behavior of a user on a computer system using a neural network is presented. An neural network based IDS will first use the training data including sequences of normal activities to train the neural network, and then run the neural network to predict normal activity. The neural networks learned can be much more complicated than the rules learned by decision tree methods. Neural network method has to take a long time to train and is difficult to scale.

Genetic algorithm is used to reduce features by Helmer et al. in Iowa State University [15]. This genetic algorithm uses standard mutation, crossover operators with certain probability of mutation and with certain probability of selecting the best individual to generate the best feature subset for intrusion detection.

Columbia University has done some research work on using association rules and frequency episodes to generate new features for intrusion detection classifiers [16]. The key ideas are to use data mining techniques to discover consistent and useful patterns of system features that describe program and user behaviors, and then use the set of relevant system features to compute classifiers that can recognize anomalies. Experiments show the discovered features can improve the classification accuracy rate.

Support Vector Machine (SVM) is a newly emerged machine learning algorithm that is successfully used in many machine learning applications such as image recognition, natural language processing, and computational biology [27]. It has strong theoretical motivation in statistical learning theory and has exhibited excellent accuracy on testing sets in practice. One class SVM, which is a variation of the standard SVM, is very promising for anomaly intrusion detection since it can overcome over-fitting problem [28].

B. Support Vector Machine

SVM is first introduced by Vapnik [29]. SVMs have viewed the classification problem as a quadratic optimization problem. SVM algorithm tries to find out the best machine for a data set in the sense that it maximizes the correctness of the machine regarding the training data set and at the same time, also maximizes the capability of the machine so it also precisely classifies future testing data sets. Mathematical optimization method is used to find out the best machine. To improve generalization of the machine, the bounds on error of classification suggest to maximize the margin. The risk of overfitting can also be minimized by choosing the maximal margin hyperplane in the feature space. More details about SVM can be found in [30].

New Mexico Technology University has used SVM for intrusion detection and the initial result seems promising [31]. They have also compared the classification ability of SVM and neural network, and found SVM perform better than neural network. New Mexico Technology University performed experiment on network intrusion data using polynomial kernels and radial basis kernels.

New Mexico Technology University also experimented on using SVM to select important features for intrusion detection [32]. They applied the technique of deleting a feature at a time to perform experiments on SVM to rank the importance of input features. The results showed that SVM based IDSs using a reduced number of features can deliver enhanced or comparable performance.

Learning problems can be categorized into supervised learning and unsupervised learning. In supervised learning, learning is based on labeled training data; while in unsupervised learning, the training data is unlabeled. Unsupervised learning is very beneficial for intrusion detection domain, since the labeled data is expensive while unlabeled data can be obtained very easily from log files and audit files. SVM is originally two-class based and used in supervised learning, whle it is adapted to one-class SVM by method introduced in [33].

In one class SVM, the noises in positive data, also called "outliers", is used as negative examples. The one class SVM problem can be formulated as follows [33]:

$$f(x) = \begin{cases} +1, & if\ x \in S \\ -1, & if\ x \in \bar{S} \end{cases}$$

The main idea is that the algorithm maps the data into a feature space H using an appropriate kernel function, and then attempts to find the hyperplane that separate the mapped vectors from the origin with maximum margin.

Give a training dataset $(x_1, y_1), \ldots, (x_l, y_l) \in R^n \times \{\pm 1\}$, let $\Phi : R^n \rightarrow H$ be a kernel map which transforms the training examples into the feature space H. Then, to separate the dataset from the origin, we need to solve the following quadratic programming problem:

$$\min \left(\frac{1}{2}\|\omega\|^2 + \frac{1}{\nu l}\sum_{i=1}^{l} \xi_i - \rho \right)$$

subject to

$$y_i(\omega \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0, i = 1, \ldots, l$$

where $\nu \in (0, 1)$ is a parameter that controls the trade off between maximizing the distance from the origin and containing most of the data in the region created by the hyperplane and corresponds to the ratio of "outliers" in the training dataset.

Then the decision function

$$f(x) = sign((\omega \cdot \Phi(x)) - \rho)$$

will be positive for most examples x_i contained in the training set.

Hyperplane is used to do a linear division. However, some data sets are not linearly separable. Thus, kernel functions, such as polynomial kernel and radial basis kernel, are widely used in SVMs to map from original input data sets to high dimension feature space in order to make the data sets linearly separable.

C. Kernel methods

In some optimization functions, e.g, dual presentation, the data points appear only as dot products. Thus, a kernel function is defined to return the value of the dot product between the two vectors in high dimension feature space: $K(x1, x2) = \Phi(x1) \cdot \Phi(x2)$ where $\Phi(x)$ is a function mapping from original input data space to another high dimensional feature space. Many machine learning algorithms, such as SVM [27], use kernel function.

Recently, there has been considerable interests in the development of kernels for applications including natural language processing, speech recognition and computational biology [34]. Most research effort about application based kernels has focused on string kernels. A family of string kernels for protein classification are introduced, such as spectrum and mismatch kernels, restricted gappy kernels, substitution kernels and wildcard kernels [35]. In each case, the kernel is defined via an explicit mapping from the space of all finite sequences of an alphabet Z to a vector space indexed by the set of k-length subsequences from Z. In the case of wildcard kernel, Z augmented by a wildcard character.

Kernels on trees can be defined by kernels on matching subset trees as [35]. A fast method is also introduced to calculate the kernel efficiently.

Recent research by Cortes et al., also introduces rational kernels, which are a general family of kernels based on weighted transducers or rational relations, that can be used for analysis of variable-length sequences or more generally weighted automata, in applications such as spoken-dialog applications [36]. There are two benefits of rational kernels: variable-length sequences can be handled and rational kernels can be computed efficiently using a general algorithm of composition of weighted transducers and a general single-source shortest-distance algorithm.

For kernel based learning algorithm such as SVM, only dot products of feature vectors need to be calculated, and the high dimensional feature vectors need not to be calculated, which leads to one of the advantages of kernel method - computational efficiency.

III. Design and Computation of STIDE Kernel

We define a kernel based on STIDE anomaly detector and also present an approach of combining the new kernel with one-class SVM for anomaly detection.

A. STIDE anomaly detector

In anomaly detection for system call sequences, an alphabet is used to represent system calls; a sequence of alphabet corresponds to a sequence of system calls, and sequences are collected using trace utility in UNIX. Normal sequences are obtained from normal execution of a program and abnormal sequences are generated from abnormal execution of a program.

STIDE anomaly detector uses a fixed-size sliding window of size k to generate all the subsequences with size k from the training data to form a normal k-size subsequences database [37]. Given a sequence of testing data, anomalies are detected by sliding a window of size k along the testing data: if the number of subsequences of size k not existing in the normal database is bigger than a certain predetermined anomaly threshold, δ, then the detector declares that an anomaly has occurred, and the current position in the testing data can be used to provide information about *where* the anomaly occurred.

Earlier research results proved that this STIDE anomaly detector has limited detection coverage [5]. Following the definitions in [5], a *foreign sequence* is a sequence whose individual symbols appear in the training sequence, but the foreign sequence does not itself appear in the training sequence. We say a sequence is *normal* if it is not foreign. A *minimal* foreign sequence is a foreign sequence whose proper subsequences are all normal. The STIDE anomaly detector has the following detection coverage [5]:

1) If the largest minimal foreign sequence in the anomaly space has length M, then a STIDE anomaly detector with window size $N \geq M$ can correctly classify any sequence.
2) If the anomaly space contains a minimal foreign sequence of length M, then a STIDE anomaly detector with window size $N < M$ cannot correctly classify all sequences.

B. STIDE kernel

Let alphabet set Z represents all the possible system calls that appear in the UNIX system program and k be the sliding window size used by the STIDE model. The input space Z^* is a space of sequences of characters from the alphabet Z. The k-STIDE kernel maps from Z^* to a $|Z|^k$-dimensional feature vector space.

The feature map from Z^* to a $|Z|^k$-dimensional feature vector is defined as follows:

$$\Phi_k(\mathbf{s}) = (e(\mathbf{s}_k))_{\mathbf{s}_k \in A}$$

where $\mathbf{s} \in Z^*$; $A \subseteq Z^k$; $e(\mathbf{s}_k)$ is 1 when $\mathbf{s}_k$ is a subsequence of s or 0 otherwise. Thus, $e(\mathbf{s}_k)$ is a vector of 0 and 1 with dimension $|A|$. A represents the feature set, which is a subset of all subsequences with size k.

The k-STIDE kernel is then: $K_k(\mathbf{s}_1, \mathbf{s}_2) = \Phi_k(\mathbf{s}_1) \cdot \Phi_k(\mathbf{s}_2)$

From kernel method point of view, the original STIDE anomaly detector [37] can be expressed based on the same kind of feature mapping. In the training phase of STIDE, a vector $\mathbf{v}_0$ is calculated: $\mathbf{v}_0 = (e(\mathbf{s}_k))_{\mathbf{s}_k \in A}$ where $e(\mathbf{s}_k)$ is 1

when s_k is a k-length subsequence in sequences in the training data set or 0 otherwise. In the testing phase, for each sequence s in testing data, a vector $\mathbf{v}$ is calculated as $\mathbf{v} = \Phi_k(\mathbf{s})$.

Then, the dot product of $\mathbf{v}$ and the vector $\bar{\mathbf{v}_0}$ is compared against an anomaly threshold δ. When the dot product is bigger than or equal to the anomaly threshold, an anomaly is raised, i.e., the linear threshold function for anomaly detection is: $\mathbf{v} \cdot \bar{\mathbf{v}_0} \geq \delta, anomaly; \mathbf{v} \cdot \bar{\mathbf{v}_0} < \delta, normal.$

Using STIDE anomaly detector, consider following example:

alphabet=$\{a, b, c\}$, window size k=2

New feature space Z^k: $\{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$

Training phase:

Training data of normal processes sequence: *abbabb, bcccc*

Thus, N={ab,ba,bb,bc,cc} is the normal sequences database.

According to the definition above, we get, $\mathbf{v}_0 = (0, 1, 0, 1, 1, 1, 0, 0, 1)$ and $\bar{\mathbf{v}_0} = (1, 0, 1, 0, 0, 0, 1, 1, 0)$.

So the linear threshold function should be: $\mathbf{v}' \cdot \bar{\mathbf{v}} \geq 2$, anomaly; otherwise, normal (assume anomaly threshold to be 2).

Testing phase:

Is sequence *abcca* normal?

The corresponding vector to this sequence is: $\mathbf{v} = (0, 1, 0, 0, 0, 1, 1, 0, 1)$.

Using the linear threshold function we get in training phase, $\mathbf{v} \cdot \bar{\mathbf{v}_0} = 1$, it is normal.

Is sequence *aacca* normal?

The corresponding vector to this sequence is: $\mathbf{v} = (1, 0, 1, 0, 0, 0, 1, 0, 1)$.

Using the linear threshold function we get in training phase, $\mathbf{v} \cdot \bar{\mathbf{v}_0} = 3$, it is abnormal.

Thus, STIDE anomaly detector compares every new instance with a standard abnormal vector $\bar{\mathbf{v}_0}$, and when the number of matches is bigger than a predetermined threshold, δ, an alarm is raised.

Different from STIDE anomaly detector, STIDE kernel based SVM anomaly detector compares every new instance with a group of support vectors and decides if it is normal or abnormal. The standard abnormal vector, $\bar{\mathbf{v}_0}$, is much simpler than the support vectors. Since the standard abnormal vector is very simple and catch much less knowledge than the support vectors, STIDE anomaly detector is less precise than SVM based anomaly detector using STIDE kernel.

C. Efficient computation of STIDE kernel

Since the system call sequence length is k and the alphabet is Z, there can be $|Z|^k$ permutation of all possible system call sequences, which means, the vector length is $|Z|^k$. To calculate STIDE kernel, the whole vector needs to be calculated. So the algorithm complexity is $O(|Z|^k)$, which increases exponentially with sliding window size k and can be impractical to be calculated when k get higher than 4 with alphabet size above 100.

To calculate k-STIDE kernel efficiently, all the subsequences of size k that do not exist in the log file of the privileged system program can be neglected. This saves tremendous space and computation time. Thus, the complexity of the k-STIDE kernel method is reduced to $O(|N|)$ where N is the set of all the subsequences with size k that exist in the traces of the privileged system program.

IV. Design and Computation of Markov Chain Kernel

Markov Chain models have been used for anomaly detection for system call sequences of privileged system programs in UNIX [5]. We define a Markov Chain kernel in this section.

A. Markov Chain anomaly detector

The Markov-based detector described in [5] uses a fixed-size detector window of size k to build a discrete-time Markov chain, as follows. Each state of the Markov chain is a sequence with size k appearing in the training data. The probability of transition from state $\mathbf{s} = (a_1, a_2, \ldots, a_k)$ to state $\mathbf{s}' = (a_2, \ldots, a_k, a_{k+1})$ is computed as $p(\mathbf{s}, \mathbf{s}') = F(\mathbf{s}, \mathbf{s}')/F(\mathbf{s})$, where $F(\mathbf{s}, \mathbf{s}')$ is the number of times the sequence $(a_1, \ldots, a_{k+1})$ appears in the training data, and $F(\mathbf{s})$ is the number of times the sequence $(a_1, \ldots, a_k)$ appears in the training data. Given a sequence of testing data, anomalies are detected by sliding a window of size k along the testing data: the current state s is given by the current k symbols, the next state s' is given by the k symbols after sliding the window by one position, and a "surprise factor" is calculated as $1 - p(\mathbf{s}, \mathbf{s}')$. If the surprise factor is above a selected *anomaly threshold*, then the detector declares that an anomaly has occurred, and the current position in the testing data can be used to provide information about *where* the anomaly occurred.

Early works have proved this Markov Chain anomaly detector can achieve better detection accuracy than the STIDE model [5]. The reason is Markov Chain model builds the normality by recording not only which subsequences are normal, but also the probability of occurrences of the subsequences. We have proved this Markov Chain anomaly detector has the following detection coverage [38]:

1) If the largest minimal foreign sequence in the anomaly space has length M, then a Markov-based detector with window size $N \geq M - 1$ can correctly classify any sequence.
2) If the anomaly space contains a minimal foreign sequence of length M, then a Markov-based detector with window size $N < M - 1$ cannot correctly classify all sequences.

B. Markov Chain kernel

Let alphabet set Z represent all the possible system calls that appear in the privileged system program and k be the sliding window size used by the Markov Chain model. The input space Z^* is a space of sequences of characters from the

alphabet Z. The k-Markov Chain kernel maps from Z^* to a $(|Z|^k)^2$-dimensional feature vector space.

The feature map is defined as follows: $\Phi_k(\mathrm{s}) = (p(\mathrm{s}_k, \mathrm{s}'_k)_{\mathrm{s}_k, \mathrm{s}'_k \in N})$, where $\mathrm{s} \in Z^*$; s_k and s'_k are two states in Markov Chain, each of them represents a k-size sequence from alphabet Z, and N are the state set of the Markov Chain; $p(\mathrm{s}_k, \mathrm{s}'_k)$ is the probability of transition from state s_k to state s'_k in sequence s.

The k-Markov Chain kernel is:

$$K_k(\mathrm{s}_1, \mathrm{s}_2) = \Phi_k(\mathrm{s}_1) \cdot \Phi_k(\mathrm{s}_2)$$

The original Markov Chain anomaly detector can be described using the feature mapping we defined above to calculate a probability vector $(p(\mathrm{s}, a'))_{a' \in A}$ where s is a state that represents a k-size sequence from alphabet Z that exists in the privileged daemons and $p(\mathrm{s}, a')$ is the probability of transition from the state $\mathrm{s} = (a_1, a_2, \ldots, a_k)$ to the state $\mathrm{s}' = (a_2, \ldots, a_k, a')$.

The Markov Chain anomaly detector uses a simple linear threshold function. In the training phase, a probability vector is calculated: $\mathbf{v} = (p(\mathrm{s}, a'))_{a' \in Z}$, where s is a state that represents a k-size sequence from alphabet Z that exists in the program execution; $p(\mathrm{s}, a')$ is the probability of transition from state $\mathrm{s} = (a_1, a_2, \ldots, a_k)$ to state $\mathrm{s}' = (a_2, \ldots, a_k, a')$. Then, this probability vector is represented by a family of probability vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, with $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2 + \ldots + \mathbf{v}_n$. Each vector in the family of vectors has one and only one feature with value 1. For example, the vector $\mathbf{v} = (1,0,1,0,0,1)$ will be represented by $(1,0,0,0,0,0)$, $(0,0,1,0,0,0)$ and $(0,0,0,0,0,1)$.

In the testing phase, for each system sequence s in the testing data, a probability vector $\mathbf{v}$, which represents the features of the subsequence transition probability in s, is calculated using the feature mapping: $\mathbf{v} = \Phi_k(\mathrm{s})$ as described above.

The dot product of $\mathbf{v}$ and each vector $\mathbf{v}'$ in the family of the probability vectors is compared against a selected anomaly threshold δ. When there is a dot product value bigger than or equal to the anomaly threshold, an anomaly is reported, i.e., the linear threshold function for anomaly detection is: $\min\{\mathbf{v} \cdot \bar{\mathbf{v}'}\} \leq \delta, anomaly; \min\{\mathbf{v} \cdot \bar{\mathbf{v}'}\} > \delta, normal.$

C. Efficient computation of Markov Chain kernel

For Markov Chain method, since the system call sequence length (same as the system call sequence) is k and the alphabet size is $|Z|$, there can be $|Z|^k$ permutation of all possible system call sequences. The Markov Chain method needs to calculate all the possible transitions between each system call sequence, which means, the vector length is $O((|Z|^k)^2)$, so the algorithm complexity is $O((|Z|^k)^2)$. This computation soon becomes infeasible with the increase of the sliding window size.

To calculate k-Markov Chain kernel efficiently, all the subsequences of size k that do not exist in the traces of privileged system program in training data sets can be neglected as with the STIDE kernel. For state $\mathrm{s}_k = (a_1, a_2, \ldots, a_k)$, only states $\mathrm{s}' = (a_2, \ldots, a_k, a')$ where $a' \in Z$ can be its subsequent state. So $p(\mathrm{s}_k, \mathrm{s}'_k)$ can be represented by $p(\mathrm{s}_k, a')$, which can be computed as $F(\mathrm{s}_k, a')/F(\mathrm{s}_k)$, where $F(\mathrm{s}_k, a')$ is the number of times the sequence $(a_1, a_2, \ldots, a_k, a')$ appears in the training data, and $F(\mathrm{s}_k)$ is the number of times the sequence $(a_1, a_2, \ldots, a_k)$ appears in the training data. Thus, the complexity of the k-Markov Chain kernel method is reduced to $O(|A| * |Z|)$ where A is the set of all the subsequences with size-k that exist in the privileged system program.

V. Experiment on one-class SVM based anomaly detectors

We experimented using the STIDE and Markov Chain kernels with SVMs, and compared the SVM-based detectors with the STIDE and Markov detectors as described in [4], [5]. We use a publicly available one-class SVM software libSVM, which is an implementation of Vanik's one-class SVM [39]. We use the synthetic and live sendmail data set of the University of New Mexico [40], which includes hundreds of normal executions of `sendmail` daemons and several traces of `sendmail` daemons that are attacked. Preliminary results of our comparisons are shown in Table I, using a ratio of 0.0002 for the one-class SVM case and a window size of $n = 3$.

The performance measures are calculated from TP, TN, FP, and FN, which respectively denote the numbers of true positives (system traces predicted to be intrusions that are in fact intrusions), true negatives, false positives, and false negatives. The measures shown in Table I are based on the formulae

$$\begin{aligned}
\textit{Accuracy} &= \frac{TP+TN}{TP+TN+FP+FN} \\
\textit{Detection rate} &= \frac{TP}{TP+FP} \\
\textit{False alarm} &= \frac{FP}{FP+TN} \\
\textit{Correlation} &= \frac{(TP \cdot TN)-(FP \cdot FN)}{\sqrt{(TP+FN) \cdot (TP+FP) \cdot (TN+FP) \cdot (TN+FN)}}
\end{aligned}$$

where the correlation coefficient is a measure of how predictions correlate with actual data. This ranges from -1 to 1 where a correlation coefficient of 1 corresponds to predictions that perfectly match class labels, and a coefficient of 0 corresponds to random guessing. While the measure of accuracy is commonly-used (and shown in Table I), it is not a particularly useful measure for evaluating the effectiveness of a classifier when the distribution of samples over different classes is unbalanced, which is usually the case in intrusion detection applications. For instance, if there are 10 traces of intrusive activity and 90 normal traces, a predictor that always predicts no intrusion will result in $TP = 0$, $TN = 90$, $FP = 0$, $FN = 10$, which corresponds to an accuracy of 90%. However, such a predictor is useless for correct identification of intrusions. Note that the correlation coefficient for this case is 0.

We also test the combination of one-class SVM with polynomial kernel, radial basis kernel. Table II shows the comparison between STIDE kernel based one-class SVM, Markov kernel based one-class SVM, polynomial kernel based

Detector	Accuracy	Detection rate	False alarm	Correlation
STIDE	86.1%	85.6%	13.4%	0.74
one-class SVM (STIDE kernel)	93.2%	90.9%	4.5%	0.82
Markov Chain	91.0%	91.7%	9.7%	0.80
one-class SVM (Markov kernel)	95.5%	93.3%	2.3%	0.85

TABLE I

IMPROVING INTRUSION DETECTION WITH SVM

One class SVM Detector	Accuracy	Detection rate	False alarm	Correlation
Polynomial kernel	90.7%	91.0%	11.4%	0.77
Radial basis kernel	92.1%	91.9%	4.5%	0.84
STIDE kernel	93.2%	92.8%	4.5%	0.82
Markov kernel	95.5%	93.3%	2.3%	0.85

TABLE II

COMPARISON OF POLYNOMIAL KERNEL, RADIAL BASIS KERNEL, STIDE KERNEL AND MARKOV CHAIN KERNEL BASED ANOMALY DETECTOR (RATIO THRESHOLD = 0.0002)

one-class SVM and radial basis kernel based one-class SVM using the same UNM dataset.

A. Discussion of the experiment results

One disadvantage of STIDE and Markov Chain anomaly detectors is their over-simplicity. They completely depend on the training data, thus they have the over-fitting problem. By substituting the simple linear threshold function with one-class SVM, which is proved to be very effective by statistical learning theory, improvement can be achieved over the STIDE and Markov Chain anomaly detectors. Using one-class SVM method, the generalization capability is maximized, so the overfitting problem is overcomed.

Also, the training data for STIDE and Markov Chain anomaly detectors must be pure normal data. If the training data include some intrusions, the STIDE and Markov Chain anomaly detectors built based on it cannot detect these intrusions. One-class SVM does not need pure normal data, it can be a mixture of normal data with some intrusion data.

Another disadvantage of STIDE and Markov Chain anomaly detectors is the selection of a threshold. The threshold is critical in these two approaches, however, there is no efficient and effective method for selection of a proper threshold. One-class SVM need a threshold too, and the threshold is the biggest ratio of the noise in the normal data. The threshold for one-class SVM is not critical; as long as it is in some reasonable range, the result won't change much.

One-class SVM overcomes some limitations of STIDE and Markov Chain anomaly detectors - over-simplicity, over-fitting, requirement of pure normal data and reliance on threshold.

VI. CONCLUSION AND FUTURE WORK

This research investigates application based kernels for intrusion detection. We identify two kernels that can represent the similarity of system call sequences. We experiment on combining STIDE kernel and Markov Chain kernel with one-class SVM to improve the classification result. The experiments provide strong evidence that STIDE kernels and Markov Chain kernels, in conjunction with one-class SVMs, could offer a more accurate, effective and efficient alternative to conventional anomaly detection algorithm (STIDE and Markov Chain methods) for detecting anomalies in system call sequences.

A threshold need to be decided for one-class SVM experiments. The automatic generation of the threshold can be an interesting research direction in the future.

REFERENCES

[1] Christopher Kruegel and Thomas Toth. Distributed pattern detection for intrusion detection. In *Network and Distributed System Security Symposium Conference Proceedings: 2002*, 1775 Wiehle Ave., Suite 102, Reston, Virginia 20190, U.S.A., 2002. Internet Society.

[2] T. Lunt. A real-time intrusion detection expert system (IDES). In *Final Report SRI-CSL-92-05*, 1992.

[3] T. Lunt and R. Jagannathan. A prototype real-time intrusion detection system. In *IEEE Symposium on Security and Privacy*, 1988.

[4] S. Forrest, S. A. Hofmeyer, and A. Somayaji. Computer immunology. *Comm. ACM*, 40(10):88–96, Oct. 1997.

[5] Roy A. Maxion and Kymie M. C. Tan. Anomaly detection in embedded systems. *IEEE Trans. Computers*, 51(2):108–120, Feb. 2002.

[6] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[7] William W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[8] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[9] Wenke Lee and Sal Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4), Nov 2000.

[10] W. Lee, S. Stolfo, and K. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[11] James Cannady. Artificial neural networks for misuse detection. http://citeseer.nj.nec.com/cannady98artificial.html, 1998.

[12] H. Debar, M. Becke, and D. Siboni. A neural network component for an intrusion detection system. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1992.

[13] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[14] Kymie Tan. The application of neural networks to UNIX computer security. http://citeseer.nj.nec.com/tan95application.html, 1995.

[15] V. Honavar L. Miller G. Helmer, J. S. K. Wong. Feature selection using a genetic algorithm for intrusion detection. In *Proceedings on Genetic and Evolutionary Computation Conference*, Orlando, FL., 1999.

[16] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *7th USENIX Security Symposium*, San Antonio, TX, 1998.

[17] S. Mukkamaka and A. H. Sung. Learning machines for intrusion detection: Support vector machines and neural networks. In *Proceedings of the International Conference on Security and Management*, pages 525–531, 2002.

[18] S. Mukkamala and A. H. Sung. Feature selection for intrusion detection using neural networks and support vector machines. *Journal of the Transportation Research Board (of the National Academies)*. To appear.

[19] S. Mukkamala A. H. Sung. Identifying important features for intrusion detection using support vector machines and neural networks. In *Symposium on Applications and the Internet*, 2003.

[20] T. M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[21] X. Li and N. Ye. Decision tree classifiers for computer intrusion detection. *Journal of Parallel and Distributed Computing Practices*, 4(2), 2003.

[22] W. W. Cohen. Fast effective rule induction. In *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, 1995.
[23] K. Tan. The application of neural networks to unix computer security. *Proceeding of International Conference on Neural Networks*, 1995.
[24] J. Cannady. Artificial neural networks for misuse detection. In *Proceedings of the 1998 National Information Systems*, pages 443–456, 1998.
[25] J. Ryan, M. J. Lin, and R. Miikkulainen. Intrusion detection with neural networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.
[26] D. Siboni H. Debar, M. Becke. A neural network component for an intrusion detection system. In *In Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1992.
[27] J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
[28] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, Chichester, GB, 1998.
[29] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992.
[30] J. Platt. *Fast training of support vector machines using sequential minimal optimization*, pages 185–208. In B. Schlkopf, C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, MIT Press, Cambridge, MA, 1999.
[31] A H. Sung S. Mukkamala. Learning machines for intrusion detection: Support vector machines and neural networks. In *Proceedings of International Conference on Security and Management*, pages 525–531, 2002.
[32] A.H. Sung S. Mukkamala. Feature selection for intrusion detection using neural networks and support vector machines. *Journal of the Transportation Research Board (of the National Academies)*, 2003.
[33] J. Shawe-Taylor A. J. Smola B. Schölkopf, J. Platt and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.
[34] E. Eskin C. Leslie and W.S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *the Pacific Symposium on Biocomputing*, pages 564–575, 2002.
[35] S.V.N. Vishwanathan and A.J. Smola. Fast kernels for strings and trees. In *Neural Information Processing Systems*, 2002.
[36] P. Haffner C. Cortest and M. Mohri. Positive definite rational kernels. In *The Sixteenth Annual Conference on Learning Theory*, 2003.
[37] S. Forrest, S. A. Hofmeyer, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. *Proceedings 1996 IEEE symposium on Security and Privacy*, pages 120–128, May 1996. Oakland, CA.
[38] Y. Wang, A. S. Miner, and J. Wong. Comments on "Anomaly detection in embedded systems". *submitted to IEEE Transactions on Computer*, 2003. http://latte.cs.iastate.edu/Research/Intrusion/papers/Comm03.ps.
[39] C. C. Chang and C. J. Lin. Training nu-support vector regression: theory and algorithms. *Neural Computation*, 14(5):1959–1977, 2002.
[40] University of New Mexico. Compuer immune systems - data sets. online, 2003. http://www.cs.unm.edu/immsec/systemcalls.htm.