An intelligent approach for intrusion detection system using KNN classifier

Ramasamy Mariappan [author for correspondence]
Assistant Professor, Department of Computer Science & Engineering,
Sri Venkateswara College of Engineering,
Sriperumbudur, Pennalur Post – 602 105 PIN
Chennai, Tamilnadu, INDIA
Email: rmn@svce.ac.in

Abstract – **A new similarity measure for the Anomaly- Based Intrusion Detection Scheme using sequences of system calls is used. With the increasing frequency of new attacks, it is getting difficult to update the signatures database for Misuse-Based Intrusion Detection System (IDS). Though, Anomaly-based IDS has a very important role to play, the high rate of false positives remains a cause of concern. Our work defines a similarity measure that considers the number of similar system calls, frequencies of system calls and ordering of system calls made by the processes to calculate the similarity between the processes. We propose the use of Kendall Tau distance to calculate the similarity in terms of ordering of system calls in the process. The k nearest neighbour (kNN) classifier is used to categorize a process as either normal or abnormal.**

Keywords: **anomaly, intrusion, KNN classifier, frequency, ordering**

I. INTRODUCTION

Intrusion detection has played an important role in computer security research. Intrusions are attempts at compromising the confidentiality and integrity or bypassing the security mechanisms of a computer or network. Intrusion Detection is the process of monitoring the events in a computer or network and analyzing them for signs of Intrusions. With the rapid growth of attacks on the computers, Intrusion detection Systems (IDS), which are software or hardware products that automate this monitoring and analysis process, have become a critical component of Security Architecture. In the present work, we propose a Novel Intrusion Detection Process for a Host-Based system. This work draws inspiration from one of the recent proposals by Liao et al on an Anomaly Based Intrusion Detection System. Anomaly detection, on the other hand, looks for patterns that deviate from the normal. In spite of their capability of detecting unknown attacks, anomaly detection systems suffer from a basic difficulty in defining what is ``normal". Methods based on anomaly detection tend to produce many false alarms because they are not capable of discriminating between abnormal patterns triggered by an otherwise authorized user and those triggered by an intruder.

II. APPROACH OF THE METHOD

Learning program behaviour and building program profiles is one important possibility to detect intruder. Indeed building program profiles, especially those of privileged programs, has become a popular alternative to building user profiles in intrusion detection. Capturing the system call history associated with the execution of a program is one way of creating the execution profile of a program. Program profiles appear to have the potential to provide concise and stable descriptions of intrusion activity [4]. To date, almost all the research in this area has been focused on using short sequences of system calls generated by individual programs. The local ordering of these system call sequences is examined and classified as normal or intrusive. There is one theoretical and one practical problem with this approach. Theoretically, no justification has been provided for this definition of 'normal' behaviour. Notwithstanding this theoretical gap, this procedure is tedious and costly. Although some automated tools may help to capture system call sequences, it is difficult and time consuming to learn individually the behaviour profiles of all the programs (i.e., system programs and application programs). While the system programs are not generally updated as often as the application programs, the execution traces of system programs are likely to be dynamic also, thus making it difficult to characterize 'normality'. This method treats the system calls in different manner. Instead of looking at the local ordering of the system calls, our method uses the frequencies of system calls to characterize program behaviour for intrusion detection. This stratagem allows the treatment of long stretches of system calls as one unit, thus allowing one to bypass the need to build separate databases and learn individual program profiles. Using the text processing metaphor, each system call is then treated as a 'word' in a long document and the set of system calls generated by a process is treated as the 'document'. This analogy makes it possible to bring the full spectrum of well-developed text processing methods to bear on the intrusion detection problem. One such method is the *k*-Nearest Neighbour classification method. Adopting Liao's method, we make use of k-Nearest Neighbour scheme with our new similarity measure, thus gracefully extending the result of Liao et al, which doesn't consider the ordering of system calls. We prove that the similarity matrix proposed by Liao may result in inaccurate conclusion for intrusion detection and our proposed similarity matrix overcomes this flaw. The major contributions of our work are:

i. A combined similarity measure for frequency and occurrence.

ii. A similarity measure for sequence.

III.RELATED WORK

Anomaly-Based IDS has the capability to identify new attacks, as any attack will differ from the normal activity. However, such systems have a very high rate of false positive. Hence, a lot of research is being done in the area of Anomaly-based Intrusion Detection. The pioneering work in the field of Anomaly Detection by Denning describes a model for detecting computer abuse by

monitoring the system's audit records. In this approach, profiles of subjects (users) are learnt and statistical methods used to calculate deviations from the normal behavior. Lane et al proposes another approach that captures user's behavior. A database of sequence of UNIX commands that a user issues normally, is maintained for each user. Any new command sequence is compared with this database using a similarity measurement. Though the scheme gives good results, it is rather difficult to profile all the users, especially of big organizations. Moreover, since the behavioral pattern of new users is not very stable, such models may give a high rate of false positives.

Ko et al. [1],[2] at UC Davis first proposed to specify the intended behavior of some privileged programs (setuid root programs and daemons in UNIX) using a program policy specification language. During the program execution, any violation of the specified behavior was considered ``misuse". The major limitation of this method is the difficulty of determining the intended behavior and writing security specifications for all monitored programs. Nevertheless, this research opened the door of modeling program behavior for intrusion detection. Uppuluri et al. applied the specification-based techniques to the 1999 DARPA BSM data using a behavioral monitoring specification language. Without including the misuse specifications, they were able to detect 82% of the attacks with 0% false positives. The attack detection rate reached 100% after including the misuse specifications. Forrest's group at the University of New Mexico introduced the idea of using short sequences of system calls issued by running programs as the discriminator for intrusion detection. The Linux program *strace* was used to capture system calls. Normal behavior was defined in terms of short sequences of system calls of a certain length in a running UNIX process, and a separate database of normal behavior was built for each process of interest. A simple table look-up approach was taken, which scans a new audit trace, tests for the presence or absence of new sequences of system calls in the recorded normal database for a handful of programs, and thus determines if an attack has occurred. Lee et al [9],[10] extended the work of Forrest's group [3] and applied RIPPER, a rule learning program, to the audit data of the UNIX *send mail* program. Both normal and abnormal traces were used. Warrender et al introduced a new data modeling method, based on Hidden Markov Model (HMM), and compared it with RIPPER and simple enumeration method. For HMM, the number of states is roughly the number of unique system calls used by the program. Although HMM gave comparable results, the training of HMM was computationally expensive, especially for long audit traces. Ghosh and others employed artificial neural network techniques to learn normal sequences of system calls for specific UNIX system programs using the 1998 DARPA BSM data. More than 150 program profiles were established. For each program, a neural network was trained and used to identify anomalous behavior.

Wagner et al [8] proposed to implement intrusion detection via static analysis. The model of expected application behavior was built statically from program source code. During a program's execution, the ordering of system calls was checked for compliance to the pre-computed model. Dynamic linking, thread usage and modeling library functions pose difficult challenges to static analysis. Another limitation of this approach is the running time involved in building models for individual programs from lengthy source code. Unlike most researchers who concentrated on building individual program profiles, Asaka et al introduced a method based on discriminated analysis. Without examining all system calls, an intrusion detection decision was made by analyzing only 11 system calls in a running program and calculating the program's Mahalanobis distances to normal and intrusion groups of the training data. There were four instances that were misclassified out of 42 samples. Due to its small size of sample data, however, the feasibility of this approach still needs to be established.

Ye et al [11] attempted to compare the intrusion detection performance of methods that used system call frequencies and those that used the ordering of system calls. The names of system calls were extracted from the audit data of both normal and intrusive runs, and labeled as normal and intrusive respectively. It is our impression that they did not separate the system calls based on the programs executing. Since both the frequencies and the ordering of system calls are program dependent, this oversimplification limits the impact of their work. Our approach employs a new technique based on the k-Nearest Neighbor classifier proposed by Liao and Vemuri for learning program behavior for intrusion detection. The frequencies of system calls executed by a program are used to characterize the program's behavior. Text categorization techniques are adopted to convert each process to a vector and cosine similarity measure is used to calculate the similarity among processes. Then the k-Nearest Neighbor classifier, which has been successful in text categorization applications, is used to categorize each new program behavior into either normal or intrusive class.

IV.ALGORITHM

Implementation is based on the k-Nearest Neighbor (kNN) classifier, which is used to classify program behavior as normal or intrusive. Others to characterize a program's normal behavior before have used short sequences of system calls. However, separate databases of short system call sequences have to be built for different programs, and learning program profiles involves time-consuming training and testing processes. With the kNN classifier, the frequencies of system calls are used to describe the program behavior. Text categorization techniques [5] are adopted to convert each process to a vector and calculate the similarity between two program activities. Since there is no need to learn individual program profiles separately, the calculation involved is largely reduced. Preliminary experiments with 1998 DARPA BSM audit data show that the kNN classifier can effectively detect intrusive attacks and achieve a low false positive rate

A. KNN Classifier

Text categorization is the process of grouping text documents into one or more predefined categories based

on their content. A number of statistical classification and machine learning techniques [6],[7] have been applied to text categorization, including regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines. The first step in text categorization is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. The most commonly used document representation is the so-called vector space model. In this model, a vector of words represents each document. A word-by-document matrix A is used for a collection of documents, where each entry represents the occurrence of a word in a document, i.e., A = (aij), where aij is the weight of word i in document j. There are several ways of determining the weight aij. Let fi be the frequency of word i in document j, N the number of documents in the collection, M the number of distinct words in the collection, and ni the total number of times word i occurs in the whole collection. The simplest approach is Boolean weighting, which sets the weight nij to 1 if the word occurs in the document and 0 otherwise. Another simple approach uses the frequency of the word in the document, i.e., aij = fij. A more common weighting approach is the so-called tf idf (term frequency - inverse document frequency) weighting:

$$\mathbf{aij = fij * log (N/ ni)} \qquad \text{---------- (1)}$$

A slight variation of the tf idf weighting, which takes into account that documents may be of different lengths, is the following:

$$a_{ij} = \frac{\mathbf{fij * log (N/ nij)}}{\sqrt{\sum_{i=1}^{M} \mathbf{fij * fij}}} \qquad \text{---------- (2)}$$

For matrix A, the number of rows corresponds to the number of words *M* in the document collection. There could be hundreds of thousands of different words. In order to reduce the high dimensionality, stop-word (frequent word that carries no information) removal, word stemming (suffix removal) and additional dimensionality reduction techniques, feature selection or re-parameterization, are usually employed. To classify a class-unknown document X, the k-Nearest Neighbor classifier algorithm ranks the document's neighbors among the training document vectors, and uses the class labels of the *k* most similar neighbors to predict the class of the new document. The classes of these neighbors are weighted using the similarity of each neighbor to X, where similarity is measured by Euclidean distance or the cosine value between two document vectors. The cosine similarity is defined as follows:

$$\mathbf{Sim\ (X, Dj)} = \frac{\Sigma_{ti}\, \varepsilon\, (\mathbf{X} \cap \mathbf{Dj})\ \mathbf{xi * dij}}{\| \mathbf{X} \|_2 * \| \mathbf{Dj} \|_2} \qquad \text{---------- (3)}$$

where *X* is the test document, represented as a vector; Dj is the jth training document; ti is a word shared by *X* and Dj; xi is the weight of word ti in X; dij is the weight of word ti in document Dj; || X ||2 = Ö (x1) 2+(x2) 2+ (x2) 2 + ... is the norm of X, and || Dj ||2 is the norm of Dj. A cutoff threshold is needed to assign the new document to a known class. The kNN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. Compared to other text categorization methods such as Bayesian classifier, [5] kNN does not rely on prior probabilities, and it is computationally efficient. The main computation is the sorting of training documents in order to find the *k* nearest neighbors for the test document. We seek to draw an analogy between a text document and the sequence of all system calls issued by a process, i.e., program execution. The occurrences of system calls can be used to characterize program behavior and transform each process into a vector. Furthermore, it is assumed that processes belonging to the same class will cluster together in the vector space. Then it is straightforward to adapt text categorization techniques to modeling program behavior. Table 1 illustrates the similarity in some respects between text categorization and intrusion detection when applying the kNN classifier. There are some advantages to applying text categorization methods to intrusion detection. First and foremost, the size of the system-call vocabulary is very limited. There are less than 100 distinct system calls in the DARPA BSM data, while a typical text categorization problem could have over 15000 unique words. Thus the dimension of the word-by-document matrix **A** is significantly reduced, and it is not necessary to apply any dimensionality reduction techniques. Second, we can consider intrusion detection as a binary categorization problem, which makes adapting text categorization methods very straightforward.

B. Binary Similarity measure

A similarity score is defined μ (Pbi, Pbj) between any two processes Pbi & Pbj as follows:

$$\mu\ \mathbf{(Pbi, Pbj)} = \frac{\Sigma\ \mathbf{Pbi} \wedge \mathbf{Pbj}}{\Sigma\ \mathbf{Pbi} \vee \mathbf{Pbj}} \quad \mathbf{for\ 0\ \ \mu\ \ 1} \qquad \text{---------- (4)}$$

It may be noticed that the value of μ increases when there are more similar calls than the dissimilar ones (due to the numerator) and value of μ decreases when there are more dissimilar system calls that the similar ones (due to denominator) in Pbi and Pbj.

B. Frequency similarity measure

Another similarity score, known as cosine similarity measure λ (Pi, Pj) between the processes Pi and Pj,where Pi and Pj are obtained from A, is defined as follows:

$$\lambda\ \mathbf{(Pi, Pj)} = \frac{\mathbf{Pi * Pj}}{\| \mathbf{Pi} \|\ \| \mathbf{Pj} \|} \qquad \text{---------- (5)}$$

It may be noted that this equation represents the same similarity measure as used in the kNN classifier. So the new similarity measurement is as follows:

$$\mathbf{Sim\ (Pi, Pj)} = \mu\ \mathbf{(Pbi, Pbj)} * \lambda\ \mathbf{(Pi, Pj)} \qquad \text{---------- (6)}$$

The motive behind multiplying μ & λ is that λ (Pi, Pj) measures the similarity based on the frequency and

μ (Pbi, Pbj) is the weight associated with Pi & Pj. In other words, μ (Pbi, Pbj) tunes the similarity score, according to the number of similar and dissimilar system calls. Therefore, similarity measure Sim (Pi, Pj) takes frequency and number of similar system calls into consideration while calculating similarity between two processes. The flow of algorithm is shown in fig.1

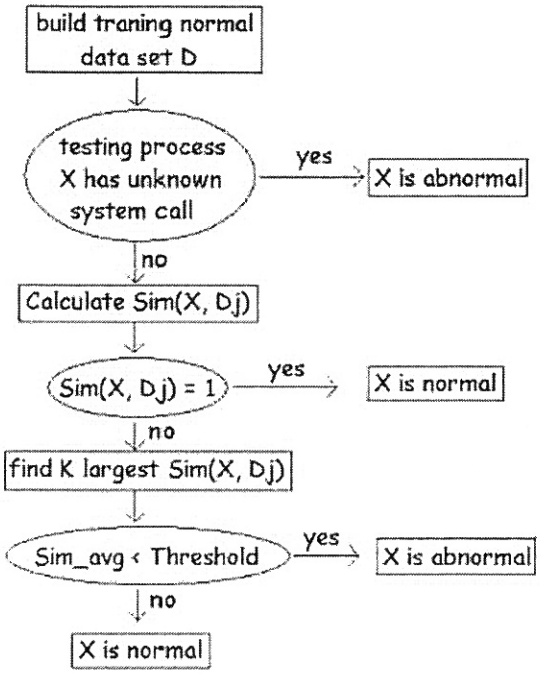

Fig.1 Flow of algorithm

V.RESULTS AND DISCUSSION

A. Data set

The TCPDUMP and BSM audit data were collected on a network that simulated the network traffic of an Air Force Local Area Network. The audit logs contain seven weeks of training data and two weeks of testing data. There were 38 types of network-based attacks and several realistic intrusion scenarios conducted in the midst of normal background data. The Basic Security Module (BSM) was used to collect audit data from a victim Solaris machine inside the simulation network. The BSM audit logs contain information on system calls produced by programs running on the Solaris machine. The names of system calls were recorded. Other attributes of BSM events, such as arguments to the system call, object path and attribute return value, etc., were not used here, although they could be valuable for other methods. The DARPA data was labeled with session numbers. Each session corresponds to a TCP/IP connection between two computers. Individual sessions can be programmatically extracted from the BSM audit data. Each session consists of one or more processes. A complete ordered list of system calls is generated for every process. A sample system call list for the process id: 994 are shown in the table 1 below. The first system call issued by Process 994 was *close*; *execve* was the next, then *open*, *mmap*, *open* and so on. The process ended with the system call *exit*.

Table 1. Process ID: 994

close	munmap	open	munmap	chmod
execve	mmap	mmap	open	close
open	mmap	mmap access	ioctl	close
mmap	close	munmap	access	close
open	open	mmap	chown	close
mmap	mmap	close	ioctl	close
mmap	close	close	access	exit

The numbers of occurrences of individual system calls during the execution of a process were counted. Then text-weighting techniques were ready to transform the process into a vector. During our off-line data analysis, our data set included system calls executed by all processes except the processes of the Solaris operating system such as shells which usually spanned several audit log files.

B. Anomaly detection

First we implemented intrusion detection solely based on normal program behavior. In order to ensure that all possible normal program behaviors are included, a large training data set is preferred for anomaly detection. On the other hand, a large training data set means large overhead in using a learning algorithm to model program behavior. There are 5 simulation days that are free of attacks during the seven-week training period. We arbitrarily picked 4 of them for training, and used the fifth one for testing. Our training normal data set consists of 606 distinct processes running on the victim Solaris machine during these 4 simulation days. There are 50 distinct system calls observed from the training data set, which means each process is transformed into a vector of size 50. The table lists the 50 distinct system calls that appear in the training data set is unable to produce here because of page constraint. Once we have the training data set for normal behavior, the kNN text categorization method can be easily adapted for anomaly detection. We scan the test audit data and extract the system call sequence for each new process. The new process is also transformed to a vector with the same weighting method. Then the similarity between the new process and each process in the training normal process data set is calculated using Equation (5). If the similarity score of one training normal process is equal to 1, which means the system call frequencies of the new process and the training process match perfectly, then the new process would be classified as a normal process immediately. Otherwise, the similarity scores are sorted and the k nearest neighbors are chosen to determine whether the new program execution is normal or not. We calculate the average similarity value of the k nearest neighbors (with highest similarity scores) and set a threshold. Only when the average similarity value is above the threshold, is the new process considered normal.

C.Performance measure

In intrusion detection, the Receiver Operating Characteristic (ROC) curve is usually used to measure the performance of the method. The ROC curve is a plot of intrusion detection accuracy against the false positive probability. It can be obtained by varying the detection threshold. We formed a test data set to evaluate the performance of the kNN classifier algorithm. The BSM data of the third day of the seventh training week was chosen as part of the test data set (none of the training processes was from this day). There was no attack launched on this day. It contains 412 sessions and 5285 normal processes. The rest of the test data set consists of 55 intrusive sessions chosen from the seven-week DARPA training data. There are 35 clear or stealthy attack instances included in these intrusive sessions (some attacks involve multiple sessions), representing all types of attacks and intrusion scenarios in the seven-week training data. Stealthy attacks attempt to hide perpetrator's actions from someone who is monitoring the system, or the intrusion detection system. Some duplicate attack sessions of the types *eject* and *warezclient* were skipped and not included in the test data set. When a process is categorized as abnormal, the session that the process is associated with is classified as an attack session. The intrusion detection accuracy is calculated as the rate of detected attacks. Each attack counts as one detection, even with multiple sessions. Unlike the groups who participated in the 1998 DARPA Intrusion Detection Evaluation program, we define our false positive probability as the rate of mis-classified processes, instead of mis-classified sessions. The performance of the kNN classifier algorithm also depends on the value of k, the number of nearest neighbors of the test process. Usually the optimal value of k is empirically determined. We varied k's value from 5 to 25. Figure 2 shows the ROC curves for different k values. For this particular data set, k=10 is a better choice than other values in that the attack detection rate reaches 100% faster. For k=10, the kNN classifier algorithm can detect 10 of the 35 attacks with zero false positive rate. The detection rate reaches 100% rapidly when the threshold is raised to 0.72 and the false positive rate remains as low as 0.44% (23 false alarms out of 5285 normal processes) for the whole simulation day. The RSTCORP group gave good performance during the evaluation of the 1998 DARPA BSM data.

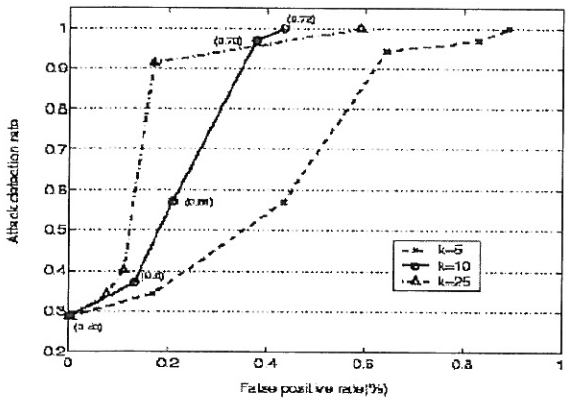

Fig.2 Performance of KNN classifier method expressed in ROC curves.

By learning normal sequences of system calls for more than 150 programs, their Elman neural networks were able to detect 77.3% of all intrusions with no false positives, and 100% of all attacks with about 10% miss-classified normal sessions, which means 40 to 50 false positive alarms for a typical simulation day with 500 sessions. Their test data consisted of 139 normal sessions and 22 intrusive sessions. Since different test data sets were used, it is difficult to compare the performance of our kNN classifier with that of the Elman networks. Although the kNN classifier has lower attack detection rate at zero false positive rate, the attack detection rate reaches 100% quickly, and hence a low false alarm frequency can be achieved.

D. Anomaly detection combined with signature verification

It is shown that the *k*NN classifier algorithm can be implemented for effective abnormality detection. The overall running time of the *k*NN method is *O(N)*, where N is the number of processes in the training data set (usually k is a small constant). When N is large, this method could still be computationally expensive for some real-time intrusion detection systems. In order to detect attacks more effectively, the *k*NN anomaly detection can be easily integrated with signature verification. The malicious program behavior can be encoded into the training set of the classifier. After carefully studying the 35 attack instances within the seven-week DARPA training data, we generated a data set of 19 intrusive processes. This intrusion data set covers most attack types of the DARPA training data. It includes the most clearly malicious processes, including *ejectexploit*, *formatexploit*, *ffbexploit* and so on. For the improved *k*NN algorithm, the training data set includes 606 normal processes as well as the 19 aforementioned intrusive processes. The 606 normal processes are the same as the ones in described case. Each new test process is compared to intrusive processes first. Whenever there is a perfect match, i.e., the cosine similarity is equal to 1.0; the new process is labeled as intrusive behavior (one could also check for near matches). Otherwise, the abnormal detection procedure described is performed. Due to the small amount of the intrusive processes in the training data set, this modification of the algorithm only causes minor additional calculation for normal testing processes. The performance of the modified *k*NN classifier algorithm was evaluated with 24 attacks within the two-week DARPA testing audit data. The DARPA testing data contains some known attacks as well as novel ones. Some duplicate instances of the *eject* attack were not included in the test data set. The false positive rate was evaluated with the same 5285 testing normal processes as described. Table 2 presents the attack detection accuracy for k=10 and the threshold of 0.8. The false positive rate is 0.59% (31 false alarms) when the threshold is adjusted to 0.8. The two missed attack instances were a new denial of service attack, called *process table*. They matched with one of training normal processes exactly, which made it impossible for the *k*NN algorithm to detect. The *process table* attack was implemented by establishing connections to the telnet port of the victim machine every 4 seconds and exhausting its

process table so that no new process could be launched. Since this attack consists of abuse of a perfectly legal action, it did not show any abnormality when we analyzed individual processes. Among the other 22 detected attacks, eight were captured with signature verification. These eight attacks could be identified with each of the normal processes in the training data set

Table 2: Attack detection rate for DARPA testing data (k=10 and *threshold*=0.8) when anomaly detection is combined with signature verification.

Attack	Instances	Detected	Detection rate
Known attacks	16	16	100%
Novel attacks	8	6	75%
Total	24	22	91.7%

VI. CONCLUSION

This approach is predicated on the following properties. The frequencies of system calls issued by a program appear consistently across its normal executions and unseen system calls will be executed or unusual frequencies of the invoked system calls will appear when the program is exploited. With the *k*NN classifier method, each process is classified when it terminates. We argue that it could still be suitable for real-time intrusion detection. Each intrusive attack is usually conducted within one or more sessions, and every session contains several processes. Since the *k*NN classifier method monitors the execution of each process, it is highly likely that an attack can be detected while it is in operation. At any instant of time, the threshold value is updated by observing the similarity value between the training processes and process at that time. With this method, the threshold is chosen in dynamic way by learning approach using artificial neural networks and hence this approach is specifically called as an intelligent approach. The accuracy and effectiveness of this method is proved through ROC curves drawn between false positive rates vs. attack detection rate for different number of neighbors, which is shown in figure.2

VII.REFERENCES

[1] C. Ko, P. Brutch, J. Rowe, G. Tsafnat, and K. N. Levitt. "System Health and Intrusion Monitoring Using a Hierarchy of Constraints," Recent Advances in Intrusion Detection (RAID) 2001, Lecture Notes in Computer Science, W. Lee, L. Me, and A. Wespi, eds., Vol. 2212, pp. 190-203.

[2] C. Ko, G. Fink and K. Levitt, ``Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring", Proceedings of 10th Annual Computer Security Applications Conference, Orlando, FL, Dec 134-144, 1994.

[3] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Logstaff, "A Sense of Self for Unix process", Proceedings of 1996 IEEE Symposium on Computer Security and Privacy, 120-128, 1996.

[4] J. E. Just, J. C. Reynolds, L. A. Clough, M. Danforth, K. N. Levitt, R. Maglich, and J. Rowe. "Learning Unknown Attacks – A Start." Proc. of the 5th International Symposium, RAID 2002, Recent Advances in Intrusion Detection, A. Wespi, G. Vigna, and L. Deri, eds., Zurich, Switzerland, October 16-18, 2002, pp. 158-176.

[5] Giacinto, G.; Roli, F, " Intrusion detection in computer networks by multiple classifier systems", Proc. of 16th International Conf. on Pattern Recognition, 002., Volume: 2 , 11-15 Aug. 2002 ,volume-2,Pages:390 - 393

[6] Mukkamala, S.; Sung, A.H, "Artificial intelligent techniques for intrusion detection Systems", IEEE International Conference on Man and Cybernetics, Volume: 2, 5-8 Oct. 2003,Pages: 1266 - 1271

[7] Yan-Heng Liu; Da-Xin Tian; Al-Nun Wang, ANNIDS: "intrusion detection system based on artificial neural network", International Conference on Machine Learning and Cybernetics, 2003, Volume: 3, Nov. 2-5, 2003, Pages: 1337 – 1342

[8] D. Wagner and D. Dean, "Intrusion Detection via Static Analysis", Proc. of IEEE Symposium on Research in security and Privacy, Oakland, CA, May 2001, pp. 156-168

[9] Eleazar Eskin, Wenke Lee, Salvatore J. Stolfo, "Modelling System Calls for Intrusion Detection with Dynamic Window Sizes", Proceedings of the DARPA Conference and Exposition on Information Survivability. DISCEX '01, 2001

[10] W. Lee, S. J. Stolfo and P. K. Chan, "Learning Patterns from Unix Process Execution Traces for Intrusion Detection", Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management, 50-56, 1997.

[11] N. Ye, X. Li, Q. Chen S. M. Emran and M. Xu, "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data", IEEE Trans. SMC-A, Vol. 31, No. 4, 266-274, 2001.