


Using K-Nearest Neighbor Classifier for Intrusion Detection

Yihua Liao, V. Rao Vemuri
Department of Computer Science
University of California, Davis
One Shields Avenue, Davis, CA 95616
{yhliao, rvemuri}@ucdavis.edu

Abstract

A new approach, based on the k -Nearest Neighbor (k NN) classifier, is used to classify program behavior as normal and intrusive. Short sequences of system calls have been used to characterize a program's normal behavior. 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 k NN classifier, the frequencies of system calls are used to describe the program behavior.

 Text categorization techniques are adopted to convert each program execution to a vector and calculate the similarity between two program activities. Since there is no need to generate individual program profiles, the calculation involved is largely reduced. Preliminary experiments with 1998 DARPA BSM audit data show that the k NN classifier can effectively detect intrusive attacks and achieve a very low false positive rate. Timing and scaling properties of this method are currently under investigation.

1 Introduction

Intrusion detection has always played an important role in computer security research [1]. Two general approaches to intrusion detection are currently popular: misuse detection and anomaly detection. In misuse detection, ba-

sically a pattern matching method, a user's activities are compared with the known signature patterns of intrusive attacks. Those matched are then labeled as intrusive activities. That is, misuse detection is essentially a model-reference procedure. While misuse detection can be effective in recognizing known intrusion types, it tends to give less than satisfactory results in detecting novel attacks.

Anomaly detection, on the other hand, looks for patterns that deviate from the normal (for example, [2, 3]). In spite of their capability of detecting unknown attacks, anomaly detection systems suffer from the 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 [4].

Regardless of the approach used, almost all intrusion detection methods rely on some sort of signature tracks of activity left behind by users. People trying to outsmart an intrusion detection system can deliberately cover their tracks by consciously changing their behavior patterns. Some examples of obvious features that a user can manipulate are the identity of the host machine from where an attack originated, time of log-in and the command set used [5]. This, coupled with factors emanating from privacy issues, makes the modeling of user activities a less attractive option.

Learning program behavior and building program profiles is another possibility. Indeed building program profiles, especially those of privileged programs, has become a popular alternative to building user profiles in intrusion detection [6, 7, 8, 9]. 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. Furthermore, they are less prone to subjectivity than user behavior profiles because it would be difficult for attackers to cover their signature tracks left in system call history. To date, almost all the research in this area was focused on using short sequences of system calls generated by individual programs. The local ordering of these system call sequences was then examined and classified as normal and intrusive. There is one theoretical and one practical problem with this approach. Theoretically, no justification was provided for this definition of “normal” behavior. Notwithstanding this theoretical gap, this procedure is tedious and costly because it is difficult and time consuming to build and maintain profiles to all the programs (i.e., system programs and application programs). Although 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 paper treats the system calls differently. Instead of looking at the local ordering of the system calls, this method uses the frequencies of system calls to characterize program behavior. This stratagem allows the treatment of long stretches of system calls as one unit thus allowing one to bypass the need to build and maintain separate databases for each program. 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

[10, 11] to bear on the intrusion detection problem. One such method is the k -Nearest neighbor classification method.

The rest of this paper is organized as follows. In Section 2 we review some related work. Section 3 is a brief introduction to the k NN text categorization method. Section 4 describes details of our experiments with the 1998 DARPA data, and Section 5 contains our conclusions and some thoughts on future work.

2 Related Work

Ko et al. at UC Davis first proposed to monitor execution of privileged programs using audit trails and detect exploitations of vulnerabilities in those security-critical programs [12]. A program policy specification language is used to specify the intended behavior of some privileged programs (setuid root programs and daemons) in Unix. Any violation of the specified behavior is considered “misuse” of privileged programs. It’s claimed that this specification-based intrusion detection approach detects known attacks as well as unknown vulnerabilities. The main drawback of this method is the difficulty of determining the intended behavior and writing security specifications for all monitored programs. Despite its disadvantage, this research opened the door of modeling program behavior for intrusion detection. Uppuluri et al. applied the specification-based techniques to the 1999 DARPA data using a behavioral monitoring specification language [13].

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 [6]. 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. [8] extended the work of Forrest's group and applied RIPPER, a rule learning program, to the audit data of the Unix *sendmail* program. Both normal and abnormal traces were used. Warrender et al. [7] 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 gives comparable results, the training of HMM is computationally expensive, especially for long audit traces. Ghosh and others [9] employed artificial neural network techniques to learn program behavior profiles for the 1998 DARPA BSM data. More than 150 program profiles were established. For each program, a neural network was trained and used for anomaly detection.

Wagner et al. proposed to implement intrusion detection via static analysis[14]. 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. The main limitation of this approach is the run-time overhead involved in building models for individual programs from lengthy source code.

Unlike most researchers who concentrated on building individual program profiles, Asaka et al. [15] introduced a simple method based on discriminant 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. Due to its small size of sample data, however, the feasibility of this approach still needs to be established.

Ye et al. intended to compare the intrusion detection performance of using frequency property and ordering property of system calls[16]. The names of system calls were extracted from the audit data of both normal and intrusive runs, and labeled as normal and intrusive respectively. This work neglected the fact that both frequency and ordering properties of system calls are program dependent, and a single system call within an intrusive run might be perfectly normal. This oversimplification makes their methodology and results questionable.

Lastly, it is worth pointing out that our work differs from [17] in that the k -Nearest neighbor classification is implemented in our work, whereas in [17] the false alarm rate was used as a heuristic to classify test data when applying instance based learning techniques to learn Unix Shell command sequences.

3 Review of K-Nearest Neighbor Text Categorization Method

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 have been applied to text categorization, including regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines[18].

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, documents are represented by vectors of words. A word-by-document matrix \mathbf{A} is used for a collection of documents, where each entry represents the occurrence of a word in

a document, i.e., $\mathbf{A} = (a_{ik})$, where a_{ik} is the weight of word i in document k . There are several ways of determining the weight a_{ik} . Let f_{ik} be the frequency of word i in document k , N the number of documents in the collection, M the number of distinct words in the collection, and n_i the total number of times word i occurs in the whole collection. The simplest approach is Boolean weighting, which sets the weight a_{ik} 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., $a_{ik} = f_{ik}$. A more common weighting approach is the so-called *tf · idf* (term frequency - inverse document frequency) weighting:

$$a_{ik} = f_{ik} \times \log \left(\frac{N}{n_i} \right) \quad (1)$$

A slight variation [19] of the *tf · idf* weighting, which takes into account that documents may be of different lengths, is the following:

$$a_{ik} = \frac{f_{ik}}{\sqrt{\sum_{j=1}^M f_{jk}^2}} \times \log \left(\frac{N}{n_i} \right) \quad (2)$$

For matrix \mathbf{A} , the number of rows corresponds to the number of words M , i.e., the size of the vocabulary, 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 reparameterization, are usually employed.

To classify a class-unknown document vector 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:

$$\text{sim}(X, D_j) = \frac{\sum_{t_i \in (X \cap D_j)} x_i \times d_{ij}}{\|X\|_2 \times \|D_j\|_2} \quad (3)$$

where X is the test document; D_j is the j th training document; t_i is a word shared by X and D_j ; x_i is the weight of word t_i in X ; d_{ij} is the weight of word t_i in document D_j ; $\|X\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots}$ is the norm of X , and $\|D_j\|_2$ is the norm of D_j . A cutoff threshold is needed to assign the new document to a known class.

Compared to other text categorization methods, k NN doesn't rely on prior knowledge, 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. More importantly, in a dynamic environment that requires frequent additions to the training document collection, incorporating new training documents is easy for the k NN classifier.

We seek to draw an analogy between a text document and the sequence of all system calls issued by a program. 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 k NN classifier. There are some advantages to apply text categorization methods to intrusion detection. First and foremost, the size of the system-call vocabulary is very limited. There are only less than 100 distinct system calls in the DARPA BSM data, while a typical text categorization problem could have over 15000 unique words[18]. Thus the dimension of the word-by-document matrix \mathbf{A} is significantly reduced. It is not necessary to apply any dimensionality reduction techniques. Secondly, intrusion detection is a simple binary categorization problem, which makes adapting text categorization methods very straightforward.

Terms	Text categorization	Intrusion Detection
N	total number of documents	total number of processes (programs)
M	total number of distinct words	total number of distinct system calls
n_i	number of times i th word occurs	number of times i th system call was issued
f_{ik}	frequency of i th word in document k	frequency of i th system call in process k
D_j	j th training document	j th training process
X	test document	test process

Table 1: Analogy between text categorization and intrusion detection when applying the k NN classifier.

4 Experiments

4.1 Data Set

We applied the k -Nearest Neighbor classifier to the 1998 DARPA data. The 1998 DARPA Intrusion Detection System Evaluation program provides a large sample of computer attacks embedded in normal background traffic[20]. The TCPDUMP and BSM audit data were collected on a simulation network that simulates the network traffic of an Air Force Local Area Network. It consists of 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.

We used the Basic Security Module (BSM) audit data collected 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. See [21] for a detailed description of BSM events. We only recorded the names of system calls. Other information of BSM events, such as arguments to the system call, object path and attribute, return value, etc., was not used.

The DARPA data was labeled with session numbers. Each session corresponds to a TCP/IP connection between two computers. There were about 500 sessions recorded by the BSM tool of the Solaris machine each simula-

tion day. 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 is shown 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 a system call *exit*.

```
Process ID: 994
close  execve  open    mmap   open
mmap   mmap    munmap  mmap   mmap
close  open    mmap    close  open
mmap   mmap    munmap  mmap   close
close  munmap  open    ioctl  access
chown  ioctl    access  chmod  close
close  close   close   close  exit
```

The numbers of occurrences of individual system calls during the execution of a process were counted. Then text weighting techniques are ready to transform the process into a vector. We used formula (2) to encode the processes.

Preselection of processes was conducted during our off-line data analysis. We only chose the ones with the *execve* system call. We found it was an effective way to separate the user-level processes from the kernel-level processes in the voluminous BSM logs. In addition, Some trivial processes that occurred frequently and yet are not relevant to attacks, such as *date*, *sleep* and *ps*, were skipped. After preselection, the number of processes of one simulation day

is somewhere between 2400 and 4000.

4.2 Anomaly Detection

First we implemented intrusion detection 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, large training data set means large overhead in using a learning algorithm to model program behavior. We arbitrarily chose 3556 processes from the normal background data of the training period. Our normal data set is about one simulation day's load on the victim Solaris machine. We believe it is a typical data set. There are 49 distinct system calls observed from the training data set.

Once we have the training data set for normal behavior, the k NN 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 formula (3). If the similarity score of one training normal process is equal to 1, which means the system call lists of the new process and the training process match perfectly, then the new process would be classified as 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, the new process is considered normal. The pseudo code for the adapted k NN algorithm is presented in Figure 1.

In intrusion detection, the Receiver Operating

```
build the training normal data set  $D$ ;  
for each process  $x$  in the test data do  
  if  $x$  has an unknown system call then  
     $x$  is abnormal;  
  else then  
    for each process  $D_j$  in training data do  
      calculate  $sim(x, D_j)$ ;  
      if  $sim(x, D_j)$  equals 1.0 then  
         $x$  is normal and exit;  
    find  $k$  biggest scores of  $sim(x, D)$ ;  
    calculate  $sim\_avg$  for  $k$ -nearest neighbors;  
    if  $sim\_avg$  is greater than  $threshold$  then  
       $x$  is normal;  
    else then  
       $x$  is abnormal;
```

Figure 1: Pseudo code for the k NN classifier algorithm for anomaly detection.

Characteristic (ROC) curve is usually used to measure the 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 k NN classifier algorithm. The BSM data of the first 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 456 sessions and 2436 normal processes. The rest of the test data set consists of 55 intrusive sessions chosen from the seven-week DARPA training data. There are 42 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. Some attack sessions of the same types are skipped for simplicity. When a process is categorized into abnormal class, 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 attacks counts as one detection, even with multiple sessions. Unlike the groups who participated in the 1998 DARPA

Intrusion Detection Evaluation program[22], we define our false positive probability as the rate of mis-classified processes, instead of of mis-classified sessions.

The performance of the k NN 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 3 to 20. Figure 2 shows the ROC curves for different k values. Obviously, $k = 5$ is a better choice than other values. For $k = 5$, the k NN classifier algorithm can detect 64.3% of the attacks with zero false positive rate and threshold of 0.74. And the detection rate reaches 100% rapidly when the threshold is raised to 0.87 and the false positive rate remains as low as 0.082% . In other words, all the attacks can be detected at the cost of only 2 to 3 false alarms per day, considering the total number of processes is around 3000 per simulation day.

The RSTCORP group [9] gave the best performance during the evaluation of the 1998 DARPA BSM data [22]. 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 per day. Compared to the Elman networks, our k NN classifier has slightly lower attack detection rate at zero false positive rate, but the attack detection rate reaches 100% much faster than the Elman nets, and the number of false alarms can be reduced by one order of magnitude. This is a significant improvement.

4.3 Anomaly Detection Combined with Signature Verification

We have just 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 num-

ber of processes in the training data set. The algorithm of finding k biggest numbers out of N numbers is, in the main, responsible for this. When N is large, this method could still be computationally expensive for some real-time intrusion detection systems. In order to detect attacks more efficiently, 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 42 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*, *ffbeexploit* and so on.

For the improved k NN algorithm, the training data set includes 3556 normal processes as well as the 19 aforementioned intrusive processes. The 3556 normal processes are the same as the ones in subsection 4.2. 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. Otherwise, the abnormal detection procedure in Figure 1 is performed. Clearly, this method reduces the calculation further and speeds up the intrusion detection.

The performance of the improved 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 intrusive sessions of the same types were not included in the test data set. Table 2 presents the the attack detection accuracy for $k = 5$ and the threshold of 0.87. The false positive rate remains the same.

The two missed attack instances were a new denial of service attack, called *processtable*. They matched with one of training normal processes exactly, which made it impossible for the k NN

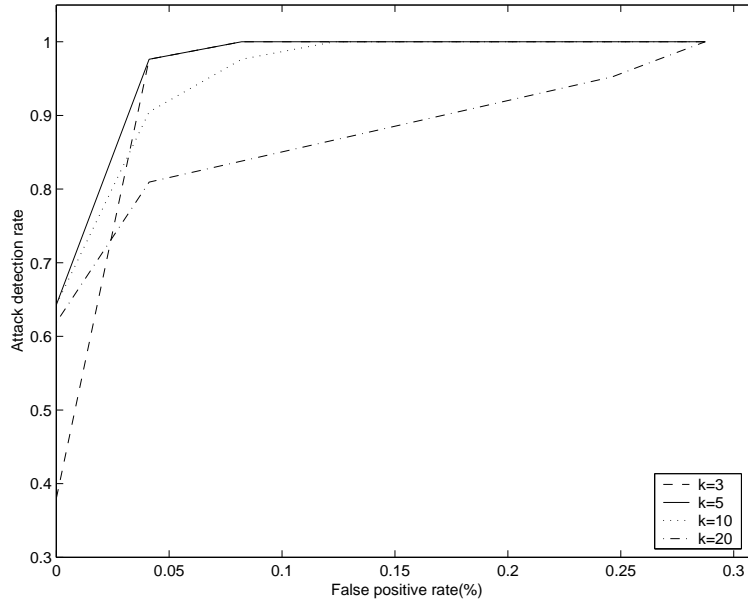


Figure 2: Performance of the k NN classifier method expressed in ROC curves. False positive rate vs attack detection rate for $k = 3, 5, 10$ and 20 .

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

Table 2: Attack detection rate for DARPA testing data ($k = 5$ and $threshold = 0.87$) when anomaly detection is combined with signature verification.

algorithm to detect. We suspect that these two instances were at very early stage. Therefore they didn't show any abnormality. Among the other 22 detected attacks, one was captured with signature verification.

5 Discussion



In this paper we have proposed a new algorithm for modeling program behavior in intrusion detection, which is based on the k -Nearest Neighbor classifier method, a method that was found to be very effective in text categorization. Our preliminary experiments with the 1998 DARPA BSM audit data have shown that this approach is able to effectively detect intrusive program behavior. Compared to other methods using short system call sequences, the k NN classifier doesn't have to build separate profiles of short system call sequences for different programs, thus the calculation involved with classifying new program behavior is largely reduced. Our results also show that the false positive rate can be decreased by an order of magnitude, which is a significant improvement. In addition, the k NN classifier method works well with dynamic environments that requires frequent updates of the training data, which makes it attractive for intrusion detection.

In our current implementation, the audit data analysis is performed offline. However, the k NN classifier method is well 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 very likely that an attack can be detected while it is in operation.

The results reported here are preliminary. Further research is in progress to address issues such as reliability, timing and scaling properties of the k NN classifier method, and quanti-

tative comparison with other machine learning and statistical techniques.

6 Acknowledgment

We thank Dr. Marc Zissman of Lincoln Laboratory at MIT for providing us the DARPA training and testing data. This work is supported in part by the AFOSR grant F49620-01-1-0327 to the Center for Digital Security of the University of California, Davis.

References

- [1] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy", <http://citeseer.nj.nec.com/axelsson00intrusion.html>, 2000.
- [2] H.S. Javitz and A. Valdes, *The NIDES Statistical Component: Description and Justification*, Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1994.
- [3] H.S. Vaccaro and G.E. Liepins, "Detection of Anomalous Computer Session Activity", *Proceedings of 1989 IEEE Symposium on Security and Privacy*, 280-289, 1989.
- [4] E. Lundin and E. Johnsson, "Anomaly-based intrusion detection: privacy concern and other problems", *Computer Networks*, vol. 34, 623-640, 2000.
- [5] Dao Vu and V Rao Vemuri, "A Performance Comparison of Different Back Propagation Neural Networks Methods in Computer Network Intrusion Detection", *Differential Equations and Dynamical Systems*, December 2001 (to appear)
- [6] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Logstaff, "A Sense of Self for Unix pro-

- cess”, *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, 120-128, 1996.
- [7] C. Warrender, S. Forrest and B. Pearlmutter, “Detecting Intrusions Using System Calls: Alternative Data Models”, *Proceedings of 1999 IEEE Symposium on Security and Privacy*, 133-145, 1999.
- [8] 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*, 1997.
- [9] A. K. Ghosh, A. Schwartzbard and A. M. Shatz, “Learning Program Behavior Profiles for Intrusion Detection”, *Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA, April 1999, <http://www.rstcorp.com/anup/>.
- [10] Y. Yang, *An Evaluation of Statistical Approaches to Text Categorization*, Technical Report CMU-CS-97-127, Computer Science Department, Carnegie Mellon University, 1997.
- [11] Y. Yang, “Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval”, *Proceedings of 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’94)*, 13-22, 1994.
- [12] 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.
- [13] P. Uppuluri and R. Sekar, “Experiences with Specification-Based Intrusion Detection”, *Recent Advances in Intrusion Detection (RAID 2001)*, LNCS 2212, Springer, 172-189, 2001.
- [14] D. Wagner and D. Dean, “Intrusion Detection via Static Analysis”, *Proceedings of IEEE Symposium on Research in Security and Privacy*, Oakland, CA, 2001.
- [15] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa and S. Goto, “A New Intrusion Detection Method Based on Discriminant Analysis”, *IEEE TRANS. INF. & SYST.*, Vol. E84-D, No. 5, 570-577, 2001.
- [16] 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.
- [17] T. Lane and C.E. Brodley, “Temporal sequence learning and data reduction for anomaly detection”, *ACM Transactions on Information and System Security*, vol. 2, 295-331, 1999.
- [18] K. Aas and L. Eikvil, *Text Categorisation: A Survey*, <http://citeseer.nj.nec.com/aas99text.html>, 1999.
- [19] J. T.-Y. Kwok, “Automatic Text Categorization Using Support Vector Machine”, *Proceedings of International Conference on Neural Information Processing*, 347-351, 1998.
- [20] MIT Lincoln Laboratory, <http://www.ll.mit.edu/IST/ideval/>.
- [21] Sun Microsystems, *SunShield Basic Security Module Guide*, 1995.
- [22] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Webber, S. Webster, D. Wyschograd, R. Cunningham and M. Zissan, “Evaluating Intrusion Detection Systems: the 1998 DARPA off-line Intrusion Detection Evaluation”, *Proceedings of the DARPA Information Survivability Conference and Exposition, IEEE Computer Society Press*, Los Alamitos, CA, 12-26, 2000.