Anomaly Detection Using System Call Sequence Sets

Surekha Mariam Varghese,
Dept of Computer Sc. and Engg, M.A. College of Engineering, .Kothamangalam, India.
email: surekha.laju@gmail.com

K.Poulose Jacob,
Dept of Computer Science, Cochin University of Science and Technology, Kochi, India
email: kpj@cusat.ac.in

Abstract-- **This paper discusses our research in developing a generalized and systematic method for anomaly detection. The key ideas are to represent normal program behaviour using system call frequencies and to incorporate probabilistic techniques for classification to detect anomalies and intrusions. Using experiments on the *sendmail* system call data, we demonstrate that concise and accurate classifiers can be constructed to detect anomalies. An overview of the approach that we have implemented is provided.**

Index Terms **-Intrusion, Security, Anomaly**

I. INTRODUCTION

Over the past few years, the scope and importance of security technologies has increased. But many of the modern computer systems are crammed with high security vulnerabilities. Most of the common applications and operating systems are full of security flaws at many levels. These vulnerabilities allow an attacker to gain unauthorized privileges, gain unauthorized access to protected data or interfere with the work of others. Many attacks make use of techniques based on buffer overflows and race conditions.

Detection attempts to compromise the integrity, confidentiality, or availability of computing and communication networks are an extremely challenging problem [1]. Intrusion detection and prevention generally refers to a broad range of strategies for defending against malicious attacks [2]. Intrusion detection can be categorized into misuse detection and anomaly detection.

Misuse detection techniques build signatures of all known intrusions and use these signatures for detecting attempts of intrusions. The main drawback of such systems is that they cannot detect new intrusions whose intrusion patterns are unknown. The need for storing the intrusion signatures for each type of intrusion and the requirement of instant updating of the intrusion signatures impose severe performance bottleneck on misuse detection techniques.

Anomaly based techniques have been useful for Intrusion Detection to detect intrusions without known signatures. However, Anomaly detection techniques suffer from higher false alarm rate compared to misuse intrusion detection techniques. Although many Anomaly Detection techniques have been proposed to date, no single no single technique can effectively detect all types all types of intrusions under various scenarios.

In this paper, the concept of sequence sets is introduced to address the problem. A process can be profiled with frequencies of different system calls in different sequence sets. The representation of a process into different sequence sets and the corresponding frequency distribution has significantly enhanced the detection rate, and lowered the false alarms. To evaluate the effectiveness of the concept, a simple probabilistic model for anomaly detection is proposed.

II. BACKGROUND

The most commonly exploited vulnerability in general-purpose operating systems is the buffer overflow. It is encountered due to insufficient bounds checking on arguments that are supplied by users and it occurs whenever a request for a buffer access crosses the array / buffer boundary that was allocated for it. For example, after it was first reported many years ago, exploitable “buffer overflow” still exists in some recent system software due to programming errors.

In an overflow attack, the objective of the attacker is to corrupt the information in a carefully designed manner. Commonly they make use of functions that do not check the size of the arguments and pass very large strings as arguments to these functions, which either overwrites the function return address or places an executable code in the stack.

The correct method to prevent such attacks is to provide range checking for arrays or buffers used. Owing to the heavy overhead involved, this is usually not preferred. For languages like C, where the size of arguments are unknown in most cases, range checking is not a good option. The usual method is to go for static

© 2007 ACADEMY PUBLISHER

code checking to determine risky functions and to substitute them with safer functions.

Our experiments verify whether buffer overflows alter the execution sequence of a process, and attempt to detect anomalies caused by buffer overflows by analyzing system call sequences. A Bayesian network is used for the determination of anomalous sequences.

Bayesian Belief Networks have attracted much recent attention as a possible solution for the problems of decision support under uncertainty. They are called Bayesian networks because they make use of Bayes rule for probabilistic inference [3].

The Bayesian network model can represent dependencies among the different objects into its structure. It is made up of a set of variables (nodes) and a set of directed edges between variables. Each node has a number of states and a conditional probabilistic table that describes the probabilistic distribution of the states for the corresponding variable given the states of its parent nodes. Graphically a Bayesian network can be described by a directed acyclic graph [3, 4, 5]. A Bayesian network can effectively represent the dependence between variables and can give a concise specification of the joint probability distribution.

III. Anomaly Detection

Anomaly Detection problem can be defined as classifying each process Pi into normal and abnormal and associating a label L depending on the class to which Pi belongs. Anomaly detection basically involves collecting, organizing and profiling the behavior of processes, and classifying them according to the profiles.

Recently, there has been much research on monitoring program behavior to detect intrusions. Program-based intrusion detection uses the philosophy that normal program behavior can be characterized in an unambiguous way.

Unlike the behavior of a human user or the behavior of network traffic, the behavior of a program ultimately stems from a series of machine instructions whose meanings we know. The observed programs are usually system programs, and their behavior should not change without our knowledge. Thus, if intrusions can be detected as deviations from normal program behavior, such an intrusion detection technique would be free from false alarms caused by changes in user behavior patterns, and free as well from missed intrusions caused by attackers that mimic benign users [6].

Intrusion detection in such systems is done by comparing the profile of the input process behavior against the normal profile and taking actions according to some predetermined security policies. To profile normal usage patterns, Anomaly detection systems such as IDES[7] makes use of statistical measures on system features like the CPU and I/O activities with respect to a particular user or a program. Decision making System features as well as inter-relationships among different events and features, vary highly in different computing environments [8].

System call traces are a common type of audit data collected for performing intrusion detection. A system call trace is the ordered sequence of system calls that a process performs during its execution. The trace for a given process can be collected using system utilities such as strace. System call traces are useful for detecting a user trying to root exploit or attack. In this type of exploit, a user exploits a bug in a privileged process using a buffer overflow to create a root shell. Typically, the system call trace for a process being exploited is drastically different from the program process under normal conditions. This is because the buffer overflow and the execution of a root shell typically call a very different set of system calls than the normal execution of the program.

Because of these differences, we can detect when a process is being exploited by examining the system calls. Traditionally, these methods typically build models over short contiguous subsequences of the system call trace. There have been many different methods proposed for building models over these short contiguous subsequences.

Reference [9] describes a simple method to determine the normal behavior for privileged processes using local ordering of system calls. Normal sequences are represented with the help of look ahead pairs in [9] and contiguous sequences in [10]. [11] Presents a statistical method for misuse detection by locating sequences which occur more frequently in intrusion data as opposed to normal data. All these methods predict the probability for a subsequence to belong to a normal process or an exploit. In [2] an alternative representation for system call traces using a bag of system calls is introduced. In [6] and [12] a state based approach is used.

One potential drawback of using fixed length subsequences for detecting intrusions is that the size of the database grows exponentially. To overcome the disadvantages of representing system call sequences using subsequences, a method to profile process behavior similar to the bag of system calls presented in [2] was developed and implemented [13]. This representation is called sequence sets.

A.. Sequence Sets

System call trace for a particular process is termed as a sequence. A collection of similar sequences is called a sequence set. Sequences that start with the same sequence of system calls will be in the same sequence set if they continue to follow the same sequence of system calls. Certain sequences differ only in the number of times of execution of certain subsequences. These sequences, which differ only in the number of execution of subsequence of system calls, will be in the same sequence set. Consider the sequences described in Fig 1. The sequences, sequence 1, sequence 2 and sequence 3, differ only in the number of times the subsequence S4, S5, S6 is executed. The system calls and the order in which these system calls are executed are same in all the three sequences. Hence they belong to the same sequence set.

Determining which all sequences belong to a particular sequence set is a very complicated process. Determination of sequence sets can be considered as a classification problem. The given input sequences have to be divided into disjoint classes known as sequence sets. All known algorithms for classification can be applied to the classification of sequences.

© 2007 ACADEMY PUBLISHER

In our approach, the input sequence is converted into frequency components of the system calls Xi = {f_1,f_2,- - -, f_n} where n is the total number of possible system calls. The ordering information of adjacent system calls in the input sequence is lost and only the frequency of each system call in the sequence is preserved. Intrusion in this representation is defined according to frequency count of system calls.

Sequence 1	Sequence 2	Sequence 3
S1	S1	S1
S2	S2	S2
S3	S3	S3
S4	S4	S4
S5	S5	S5
S6	S6	S6
S7	S4	S4
S8	S5	S5
S9	S6	S6
S10	S7	S4
S11	S8	S5
S12	S9	S6
S13	S10	S7
	S11	S8
	S12	S9
	S13	S10
		S11
		S12
		S13

Figure 1. A set of system call sequence

More formally, let s_1,s_2,s_3,- ---,s_k be the system call trace for a particular process. if i_1,i_2, i_3, - -- ,i_n be the possible system calls in the sequence. The profile P_{seq} is defined as: P_{seq} = f_{i1},f_{i2},f_{i3}, - - f_{in} ,where f_{ir} represents the frequency of the system call i_r in the sequence.

IV. Initial Experiments

Initial experiments were conducted in Redhat Linux on simple processes to verify whether buffer overflows can be detected from system call traces. Buffer overflows were created by passing very large strings with intrusion code. System call traces were collected using the command "strace". A simple C program which is vulnerable to buffer overflow, the buffer overflow code passed to the buffer and the corresponding shell code are given in Fig 2.

System call traces obtained during normal execution and abnormal execution of the vulnerable program are shown in Fig 3. The portion of the system call trace which is altered in the intrusion trace is shown in bold face.

Vulnerable C Program

```
int overflow( char *data)
{
char final[100];
strcpy(final, data);
return;
}
main ()
{
char initial[160];
gets(initial);
overflow(initial);
return;
}
```

Overflow Code

Jump shellcode
Setreuid(0,0)
Exit(0)

Shellcode Used

"\x31\xdb\x31\xc9\x31\xc0\xb0\x46\xcd\x80\x31\xdb\x31\xc0\xb0\x01\xcd\x80"

Figure 2. Buffer Overflow Demonstration

It is discovered that the sequences of system calls made by a program during its normal executions are very consistent, and different from the sequences of its abnormal executions as well as the executions of other programs. It is also observed that buffer overflows cause deviation in normal program flow and system call sequence. Therefore a database containing the possible normal sequences with permissible deviations can be used as the definition of the normal behavior of a program and a profile of this database can be used as the basis to detect anomalies. These findings motivated us to search for simple and accurate intrusion detection models based on system call frequencies.

Normal ExecutionTrace

Execve, Uname, Brk, Old_mmap, Open,Open, Fstat64,Old_mmap, Close, Open,Read, Fstat64 Old_mmap,Old_mmap, Old_mmap, Close, Set_thread_area, Munmap,Mmap2, Read, **Fstat64, Mmap2,Write, Munmap, Exit_group**

Intrusion Trace

Execve, Uname,Brk, Old_mmap, Open,Open, Fstat64,Old_mmap, Close, Open,Read, Fstat64 Old_mmap,Old_mmap, Old_mmap, Close, Set_thread_area, Munmap,Mmap2,Read, **Setreuid, Exit**

Figure 3. System Call Trace Analysis

© 2007 ACADEMY PUBLISHER

V. EXPERIMENTS USING SENDMAIL

To conduct further experiments, as a commonly used program that is vulnerable to common exploits and buffer overflows, Sendmail daemon was used for studying the normal behaviour and to detect anomalous behaviour. The syslog intrusions [14] are simple examples of buffer overflows in sendmail. Though patches are currently available for the most of the vulnerabilities, sendmail and the buffer overflow attacks on Sendmail are good cases for experimental study. Sendmail daemon was examined for detection of buffer overflow attacks.

In order to construct a good classifier, we need to gather sufficient amount of training data and identify the set of meaningful features. Due to the unavailability of enough varieties of intrusion trace data, further experiments were conducted using data sets available at University of New Mexico [15].

UNM data sets consist of system call traces for many processes. Synthetic data for sendmail, used in the experiments, were collected at UNM on SUN SPARC stations running unpatched SUNOS 4.1.1 and 4.1.4. System calls generated by a process and its children are stored in the same trace. Each trace is a sequence of (process id, system call number). System call numbers are stored in the order in which it is executed. There is a mapping file that associates the system call numbers to the corresponding system call names. The set include normal traces and abnormal traces. A normal trace consists of several invocations of the sendmail program. The abnormal traces used are from syslog-remote intrusion and syslog-local intrusion.

TABLE I.
A SHORT SEQUENCE FROM SENDMAIL NORMAL DATASET

3750 5	3752 105	3752 104	3752 104	3752 106

The abnormal traces include local and remote intrusions, each with variety of commands executed during the attack.

A. Data Preparation

System call trace for a particular process is represented as a sequence. Each trace in the data set is a collection of several sequences. Sequences are separated and a frequency chart of system calls for each sequence is prepared. Each sequence is characterized by the start sequence. All sequences with similar starting sequences are grouped into a sequence set. It is assumed that the number of possible normal sequence sets for a particular process is limited. As per the UNM data sets, the number of possible sequence sets for Sendmail is nine and the first seven system calls in the sequence and the frequency of the first system call is used to identify the sequence set to which the particular sequence belongs. Table II shows a fragment from frequency chart of sequence set1.

B. Anomaly Status Determination

Frequency of individual system calls in the execution trace of a process is used for determining the anomaly status of the particular process. Frequency of each system call in the input execution trace is determined and matched with a normal profile. Details of deviations in frequencies of the input execution trace are fed to the Bayesian network. The Bayesian model computes the anomaly score using system call frequencies and prior probability distributions and if the anomaly score is above threshold value, marks it as an anomalous situation.

TABLE II.
A FRAGMENT FROM THE FREQUENCY CHART OF SEQUENCE SET1

	Process id					
	System Call Number and Name	**3772**	**3805**	**3827**	**3783**	**3794**
Start-sequence		4	4	4	4	4
		2	2	2	2	2
		66	66	66	66	66
		66	66	66	66	66
		4	4	4	4	4
		138	138	138	138	138
		66	66	66	66	66
Frequency Details	**1-fork**	1	1	1	1	1
	2-read	26	26	26	33	59
	3-write	8	8	8	15	41
	4-open	29	29	29	29	29
	5-close	98	98	98	98	98

A Bayesian Network defines the probability of anomaly for different combinations of system call frequencies. The Bayesian network makes use of a number of model parameters to detect anomalous sequences. The model parameter values are different for different sequence sets.

System call frequencies vary highly in different sequences. Even with in the same sequence set, for certain system calls this variation can be unlimited. But for certain system calls, with in the same sequence set, the variation in the frequency is relatively less or limited during normal executions. During a buffer overflow, it is often necessary to insert new code resulting in insertion, deletion or modification of the normal system call sequence. As a consequence, frequency of certain system calls in the sequence deviate from the normal. In most cases, frequencies of system calls can be used to detect anomalous sequence.

System calls are categorized into three groups depending on their frequency variation in anomalous situations. Each model is concerned about a particular category of system calls.

C. Model Parameters

Underlying model parameters, their detection mechanisms and significance are described in the following section.

1) Matching Profile :

System calls that has limited or no variation with in the same sequence set are considered in the matching profile

© 2007 ACADEMY PUBLISHER

model. This model approximates and profiles the distribution of frequencies of system calls during normal executions. The goal of this model is to approximate the distribution of the frequencies of system calls of each sequence set and detect instances that significantly deviate from the observed normal behaviour.

For each sequence set there is a normal profile. The profile stores the minimum value for normal and maximum possible deviation for each system call frequency component, of the particular sequence set. Each input sequence is matched with corresponding normal profile. If the frequency components of the input sequence match with the normal profile, with permissible variations, it is treated as a normal sequence by the matching profile model.

2) Frequency Pattern

Frequencies of certain system calls and subsequences will vary highly even with in the same sequence set. This variation can be considered as normal if this variation is relative to the frequencies of similar system calls. Variation in the frequency of system call Read with system call number 2 in the sequence set1 as shown in Table II can be considered as example for this case. A Fragment from the frequency chart of sequence set4, with identifying sequence "105, 104, 104, 106, 105, 108, 112, 1" is listed in Table III to demonstrate the variation in frequencies.

3) Irregularity count /Presence of system calls

Many of the system calls will not appear in the execution sequence of a particular process and will have zero frequency value. This model takes care of system calls absent in all the normal sequences encountered during training phase. The model examines the input sequence for presence of anomalous system calls and outputs an abnormal value if found.

Once the parameters are correctly identified probability tables can be constructed for predicting the anomaly score. The anomaly score is a value that specifies the extent of the deviation of the received request from the expected profile. It is a compound value that is obtained from the joint probability table. The anomaly score for each request can be in a range from 0.00 to 1.00, where 0.00 represents a completely secure state and 1.00 a sure anomalous state.

The frequency distribution model captures the concept of a 'normal' system call frequency for such system calls by looking at the relative ranking of the frequency component. It is based on the observation that repeating subsequences will increase the frequency of every system call in the subsequence. The analysis is based only on the relative order of the frequency values and does not rely on the value of the individual system calls. Table IV lists the ranking of the system call frequencies, as used by the frequency pattern model, corresponding to the processes listed in Table III.

D. Training

Training involves determination of the sequence sets, system calls used by each of the models and , the structure as well as probabilities associated with each of the nodes in the Bayesian Model. The success of anomaly detection depends on the determination of the correct sequence sets and actual probabilities associated with each of the nodes in the Bayesian Network.

System calls used by each of the models and the sequence sets involved are determined by analyzing the variations in system call frequencies and by matching against the identifying sequence.

The Bayesian Network uses a separate node for each model parameter in each sequence set. The joint probability table associated with a node involving variables X_1 to X_k is estimated from the training data as follows

$$P(X_1 = I_1, \ldots X_k = I_k) = \frac{N_{X_1 = I_1, \ldots X_k = I_k}}{N}$$

where $N_{X_1 = I_1, \ldots X_k = I_k}$ is the no of observations in which $X_1 \ldots X_k$ are in states $I_1 \ldots I_k$.

TABLE III.
A FRAGMENT FROM THE FREQUENCY CHART OF SEQUENCE SET4

System Call Number	1492	1575	1408	1423
2	32	32	12	14
3	15	15	10	11
19	4835	4835	190	670
50	16	16	17	17
78	4814	4814	168	648
104	16052	16052	564	2164
105	9637	9637	344	1304
106	8027	8027	283	1083
108	1610	1610	61	221
112	4830	4830	187	667
128	8	8	10	10

TABLE IV.
A FRAGMENT FROM THE FREQUENCY PATTERN CHART OF SEQUENCE SET4

System Call Number	1492	1575	1408	1423
2	8	8	9	9
3	10	10	10	10
19	4	4	4	4
50	9	9	8	8
78	6	6	6	6
104	1	1	1	1
105	2	2	2	2
106	3	3	3	3
108	7	7	7	7
112	5	5	5	5
128	11	11	10	10

VI. PERFORMANCE EVALUATION

A concept prototype was developed and implemented to detect buffer overflows. Sequences were identified with their process-ids. Only normal sequences were used for training. Samples were selected using random() function from the list of normal sequences.

© 2007 ACADEMY PUBLISHER

On analyzing the data set, it was observed that the number of normal profiles is limited. For evaluating the performance, a number of parameters were measured under different conditions. Sequences were identified with their process-ids. Only normal sequences were used for training. Samples were selected using random() function from the list of normal sequences. The prototype was tested using random samples from the list of normal and abnormal sequences.

Performance was measured using cross validation method by varying training and testing data. Only normal datasets were used for training. Cross validation is a good method of evaluation when the predictions are based on new data [16].

The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The normal profile is trained using the training set only. Then the data in the testing set is matched against the profile. The testing data is new to detection mechanism. The errors made by the model are accumulated to give the mean test set error, which is used to evaluate the model. However, the evaluation may depend heavily on the data points which are used as training data and which are used as test data.

To overcome the disadvatages of holdout method K-fold cross validation with random division was used. In K-fold validation the data set is divided into *k* subsets, and the holdout method is repeated *k* times. Each time, one of the *k* subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all *k* trials is computed. Instead of running the algorithm k times, in random division method the data is divided randomly into a test and training set *k* different times. The advantage of this metod is that the size of the test set and the number of trials can be independenly selected.

For evaluation 100,000 runs were used. This is because of the high variation observed in the performance of the different runs. Data sequences in each run are divided into training and testing sets. For experiments concerned with training percentage, size of the training data set was selected and all remaining data was used for testing. For all other experiments a total of 10,00,000 random data sequences were selected randomly from all the 10,000 runs. The parameters are totaled and average of all the runs was computed.

The parameters considered for performance measurement were accuracy, detection rate and false positive rate. [17] has defined accuracy, detection rate and false positive rate as follows.

Accuracy is a fraction of accurate identifications.

$$\text{Accuracy} = \frac{\text{Number of true positives} + \text{Number of true negatives}}{\text{Number of input sequences}}$$

Detection rate is a fraction of the intrusions identified.

$$\text{Detection Rate} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false negatives}}$$

False positive rate is a fraction of the normal data misidentified as intrusions.

$$\text{False Positive Rate} = \frac{\text{Number of false positives}}{\text{Number of true positives} + \text{Number of false positives}}$$

where true positives are the number of abnormal sequences detected as abnormal, true negatives are the abnormal sequences detected as abnormal, false positives are the normal sequences detected as abnormal and false negatives are the as abnormal sequences detected as normal.

A. Detection of abnormal sequences

The system assumes a threshold value of zero. The system was able to detect all abnormal sequences, keeping the number of false positives small. The false positives are caused by system call sequences, which significantly deviate from all examples encountered during the training phase. This is a common problem in intrusion detection practice as pointed out in [17] that the available intrusion data is not quite balanced. In such cases the detection rate and false positive rate will not be optimal. This is due to the huge disparity between the numbers of normal sequences belonging to different sequence sets of the dataset used for evaluation. If this disparity can be removed by selecting all the different varieties of data sequences for training, the number of false positives will be zero.

Most of the false positives were caused by the absence of enough varieties of samples. A typical situation from model2, while considering the frequency ranks of sequence set5 is shown in Table V. Table VI lists the corresponding frequency ranking.

TABLE V.
AN EXTRACT FROM THE FREQUENCY CHART OF SET 5 SHOWING FREQUENCY

System Call Number	Frequency Values			
	PID 1408	PID 1393	PID 1423	PID 2905
2	12	12	14	25
3	10	10	11	16
19	190	73	670	3070
50	17	17	17	17
78	168	51	648	3048
104	564	174	216	1016
105	344	110	130	6104
106	283	88	108	5083
108	61	22	221	1021
112	187	70	667	3067
128	10	10	10	10

Here the false positives were caused due to the specialty of the training data set pertaining to sequence set 5. There are only 2 processes, 1423 and 2905 with a

© 2007 ACADEMY PUBLISHER

different frequency sequence for the system calls .If both 1423 and 2905 do not appear in the randomly selected training data, the frequency rank profile for sequence set5 will be unique . In this situation frequency sequences corresponding to both 1423 and 2905 will be tested as false positives, since they differ from the unique profile of sequence set5. A similar situation occurs in sequence set1 also. The increase in the number of false positives was caused by the lack of sufficient number of variety data in the training data set.

TABLE VI
AN EXTRACT FROM THE FREQUENCY CHART OF SET 5 SHOWING RANKS

System Call Number	Frequency Ranking			
	PID 1408	PID 1393	PID 1423	PID 2905
2	9	9	9	8
3	10	10	10	10
19	4	4	4	4
50	8	8	8	8
78	6	6	6	6
104	1	1	1	1
105	2	2	2	2
106	3	3	3	3
108	7	7	7	7
112	5	5	5	5
128	10	10	10	11

A similar situation occurs in sequence set1 in model1. Table VII and Table VIII show the corresponding situation.

TABLE VII.
AN EXTRACT FROM THE FREQUENCY CHART OF SEQUENCE SET1 SHOWING FREQUENCY

System Call Name	System Call Number	Frequency Values			
		PID 4176	PID 4187	PID 3783	PID 3794
Read	2	26	26	33	59
Write	3	8	8	15	41

TABLE VIII
AN EXTRACT FROM THE FREQUENCY CHART OF SEQUENCE SET1 SHOWING RANKS

System Call Name	System Call Number	Frequency Ranking			
		PID 4176	PID 4187	PID 3783	PID 3794
Read	2	1	1	1	1
Write	3	2	2	2	2

The situation is caused due to the similarity of the processes appearing in sequence set1. There are only two processes 3783 and 3794 in sequence set1 with a different frequency profile. All other processes belonging to sequence set1 has a fixed pattern of frequency values for all system calls. In sequence set1 there are only two system calls, which are handled by the frequency rank model or model2. So if both 3783 and 3794 did not appear in the training data, profile for sequence set1 will contain fixed frequency values for all system calls with non zero frequency values and are handled by model1 itself. In this special situation, even system call 2 and 3 which were supposed to be handled by model2 will be handled by model1. Model1 signals both 3783 and 3794 as false positives, due to the difference in the frequency values of system call 2 and system call 3.

B. Effect of Training Ratio on performance

In the first set of experiments, performance was measured by varying training percentage. The prototype was tested using random samples from the list of normal and abnormal sequences. Fig 4 shows the corresponding chart.

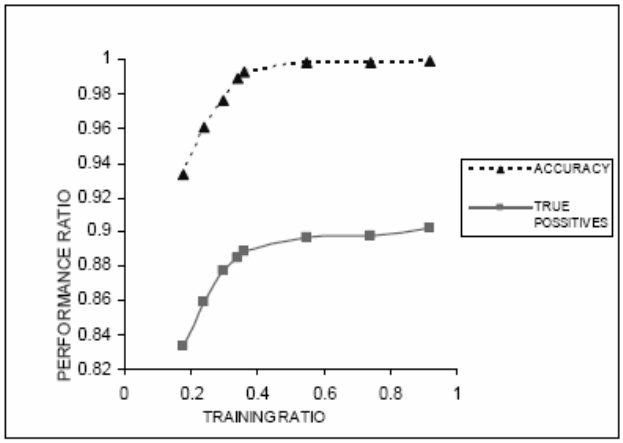

Figure 4. Performance Evaluation1

C. Comparison of False Positives

The number of false positives is a very important criterion to determine the success of the method.

1) Effect of Training Ratio

The effect of training ratio on false positives was studied. Fig 5 shows the corresponding chart.

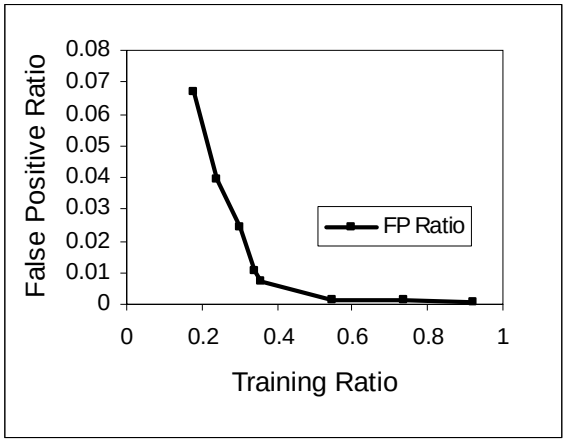

Figure 5 Performance Evaluation2

The emphasis has been on determining the success of the approach; efficiency issues were not much considered. It was clear from the experiments that frequencies of system calls are good discriminators to

© 2007 ACADEMY PUBLISHER

detect abnormal behaviour due to buffer overflows. The approach can be extended to other processes and for different types of intrusions.

VII. CONTRIBUTIONS

Based on the observation that Anomaly detection approach is more suitable to the detection of buffer overflow attacks, we identified a method to detect anomalies caused by buffer overflows. The idea is to analyze system call traces generated by the process. A representation for normal behaviour of processes using frequencies of system calls was developed for this purpose. A new idea to profile processes using sequence sets was introduced. We developed a Bayesian network on frequency variations to detect induced buffer overflows. We evaluate the proposed method on UNM data sets to confirm the performance.

VIII. CONCLUSION

What generally occurs is a race between intrusion techniques and detection techniques. As more efficient detection techniques are discovered, more complicated intrusion techniques will evolve. The approach aims at building process profiles with system call frequencies and to detect anomalies by measuring deviations from the process profile. The use of Bayesian network, incorporating different complex possibilities, improves detection and reduces false alarms. The accuracy of the detection models depends on sufficient training data and the right feature set. Preliminary experiments of using the approach on Sendmail data provided at the UNM site showed promising results.

ACKNOWLEDGMENT

We are very grateful to Anil Somayaji and Stephanie Forrest, for helping us with the necessary intrusion data and providing the details of their experiments at University of New Mexico.

REFERENCES

[1] D.E. Denning, An intrusion-detection model, IEEE Transactions on software Engineering, Vol:13, No:2,pp. 222-232,1987

[2] H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991.

[3] Finn V.Jensen, An Introduction to Bayesian Networks, Springer, 1996.

[4] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1997.

[5] Nong Ye , Mingming Xu, Probabilistic Networks with Undirected Links for Anomaly Detection, www.itoc.usma.edu/workshop/2000/Abstracts/WA1_2.pdf Probabilistic networks-with-undirected.pdf

[6] C. C. Michael And Anup Ghosh, Simple, State-Based Approaches to Program-Based Anomaly Detection, ACM Transactions on Information and System Security, Vol. 5, No. 3, August 2002.

[7] Dae-Ki Kang, Doug Fuller, Vasant Honavar, Learning Classifiers for Misuse and Anomaly Detection Using a Bag of System Calls Representation, In the Proceedings of the 2005 IEEE Workshop on Information Assurance and Security,United States Military Academy, West Point, *NY,,pp118-125*.

[8] W. Lee, S. Stolfo, and K. Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. In *Proceedings of the* 5*th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '99)*, San Diego, CA, August 1999.

[9] S. Forrest. A Sense of Self for UNIX Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[10] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. Journal of Computer Security, 6:151–180, 1998.

[11] P. Helman and J. Bhangoo. A statistically based system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(4):449–466, July 1997.

[12] Sekar, R., Bendre, M., Dhurjati, D., And Bollineni, P, A Fast Automaton-Based Method For Detecting Anomalous Program Behaviors. In *Proceedings Of The 2000 IEEE Symposium On SecurityAnd Privacy*. IEEE Computer Society, Los Alamitos, Calif., 144–155.

[13] Surekha Mariam Varghese, Poulose Jacob,' Process Profiling Using Frequencies of System calls'

[14] CERT Syslog vulnerability-a workaround for sendmail,http://www.cert.org/advisories/CA-95.13.syslog.vul.html,October 19,1995.

[15] Computer Immune Systems - Data Sets and Software http://www.cs.unm.edu/~immsec/systemcalls.htm

[16] Jeff Schneider, Cross Validation, 7 February 1997, http://www.cs.cmu.edu/~schneide/tut5/node42.html.

[17] Learning Classifiers for Misuse and Anomaly Detection Using a Bag of System Calls Representation, Dae-Ki Kang, Doug Fuller, and Vasant Honavar

Ms Surekha Mariam Varghese received B-Tech in Computer Science and Engineering from College of Engineering, Trivandrum, India in 1990 and M-Tech in Computer and Information Sciences from Cochin University of Science and Technology, Kochi, India in 1996. She is currently a PhD Scholar in the Department of Computer Science, Cochin University of Science and Technology.

She has worked as Lecturer at Manipal Institute of Technology, Manipal, India during the period 1990-91. She joined M.A. College of Engineering, Kothamangalam, Kerala, India in the year 1991 in Computer Science and Engineering department and is continuing there. Her research interests include Computer Security, Operating Systems, Computer Algorithms and Database Systems.

Dr. K. Poulose Jacob, Professor of Computer Science at Cochin University of Science and Technology has more than 27 years of teaching experience and is currently the Director of School of Computer Science Studies. His studies and research in Multimicroprocessor Applications earned him his PhD from Cochin University of Science and Technology. He is presently Chairman, Board of Studies in Computer Science, and a member of the Academic Council.

Dr. K.Poulose Jacob is a permanent professional member of the ACM and a Life Member of the Computer Society of India. His research interests are in Information Systems Engineering, Intelligent Architectures and Networks.

© 2007 ACADEMY PUBLISHER