An Improved Hidden Markov Model for Anomaly Detection Using Frequent Common Patterns

Afroza Sultana and Abdelwahab Hamou-Lhadj
Software Behavioural Analysis Research Lab
Department of Electrical and Computer Engineering
Concordia University
{af_sulta, abdelw}@ece.concordia.ca

Mario Couture
Software Analysis and Robustness Group
Defense Research and Development Canada
Valcartier, QC, Canada
mario.couture@drdc.gc.ca

***Abstract*-Host-based intrusion detection techniques are needed to ensure the safety and security of software systems, especially, if these systems handle sensitive data. Most host-based intrusion detection systems involve building some sort of reference models offline, usually from execution traces (in the absence of the source code), to characterize the system healthy behavior. The models can later be used as a baseline for online detection of abnormal behavior. Perhaps the most popular techniques are the ones based on the use of Hidden Markov Models (HMM). These techniques, however, require long training time of the models, which makes them computationally infeasible, the main reason being the large size of typical traces. In this paper, we propose an improved HMM using the concept of frequent common patterns. In other words, we build models based on extracting the largest n-grams (patterns) in the traces instead of taking each trace event on its own. We show through a case study that our approach can reduce the training time by 31.96%-48.44% compared to the original HMM algorithms while keeping almost the same accuracy rate.**

Keywords- Host-based Anomanly Detection Systems; HMM; N-gram extraction algorithm; Behavioral modeling.

I. INTRODUCTION

Intrusion detection refers to the ability to detect abnormal behavior in a system, often caused by security attacks, viruses, and the presence of design faults [1]. The consequences of not detecting these anomalies can be devastating in terms of system security and performance. A large body of research has been devoted to the analysis of network traffic, but these Network-based Intrusion Detection Systems (NIDS) are not always sufficient and can be easily evaded by "subtle" attacks that do not generate important network traffic, and hence go undetected by even the most advanced NIDS. To overcome this limitation, recently, there has been an important shift in this area to the techniques that permit the detection of intrusions at the host level, i.e., Host-based Intrusion Detection Systems (HIDS) [2, 3].

In HIDS, the intrusions are detected by monitoring and analyzing the system or application data that is collected from the host computer. Since an HIDS depends on the information about the host computer, it should, in principle, be able to detect an important range of anomalies that can cause the system to deviate from its normal behavior. Host-based intrusion detection techniques can be grouped into two categories: misuse or outlier detection [4] and anomaly detection [5]. The misuse detection techniques require prior knowledge of the potential intrusions (for example, virus, attacks, threats, etc.) of the system. They look for known intrusion patterns in the system from the prior knowledge of the intrusions (through signatures) and identify them as intrusions. One major drawback of misuse detection techniques is that the intrusion must be known beforehand to be identified. Therefore, any new intrusion, such as a new type of viruses will be unidentified by the misuse detection techniques.

The anomaly detection techniques, the second category and also the focus of this paper, operate by modeling the "normal" behavior of the host computer. The prior knowledge of normal or acceptable behavior of a system is modeled ensuring that the system is running in a safe environment without the presence of any intrusion. The system is then put in operation. The anomaly detection technique correlates the behavior of the system in operation with the one already built; it identifies any significant deviating behavior as an intrusion. The main advantage of the anomaly detection algorithms is that they do not require any prior knowledge of possible intrusions, hence, are able to identify any new virus attack, zero-day attacks, unknown system faults, and potential threats to the system.

There exist several techniques for building reference models such as machine learning [6-8], Hidden Markov Models [9-11], statistical profiling [5, 12- 14], and data mining [15-20]. Among these approaches, Hidden Markov Models (HMM) [21] have been shown to be very promising for anomaly detection over several other techniques because of their high accuracy in identifying intrusions [9]. However, the HMM-based algorithms suffer from long training time during the construction of the models, which hinders their efficiency [9].

In this paper, we present an Improved Hidden Markov Model (I-HMM) algorithm using the concept of frequent common patterns found in the trace sequences [8]. In other words, we use the frequent common patterns to build the HMM models instead of the trace events. By doing this, we reduce significantly the length of the training sequences, which in turn result in more compact HMM models. To extract these patterns, we use n-grams extraction algorithms, a concept used in text mining [22]. We show the effectiveness of our approach by applying it to building a behavioral model for a system called Gzip [23], which is a file compression and decompression software for Linux. Using the Linux Tracing Toolkit Next Generation (LTTng) [24,25] trace instrumentation tool, we collect traces of routine calls by exercising the system's features. We also use Weka 3.7.4 [26] for behavioral modeling and model verification for both HMM and I-HMM algorithms. Our study shows that our I-HMM algorithm reduces the model generation time by approximately 31.96% - 48.44% compared to the original HMM. Furthermore, the training time reduction gets even better in I-HMM when the trace coverage increases, hence, further improving the overall accuracy of the I-HMM.

The organization of this paper is as follows. Section II gives an overview of Hidden Markov Models (HMM). Our methodology for reducing the learning time of HMM is explained in Section III. Our case study is described in Section IV. The comparative results and analysis are presented in Section V. In Section VI, we discuss the conclusion and future work.

II. HIDDEN MARKOV MODELS

A Hidden Markov Model (HMM) is a double stochastic model [21]. The model is denoted by λ (A, B, π), where A is the set of observables, B is the set of hidden states, and π is the set of transition probabilities, i.e., the probabilities from going to one hidden state to another. This model is known as double stochastic since there is a hidden layer that contains some hidden states. This hidden layer follows the principles of Markov process. The other layer contains the states of the observables in a particular time t of the model construction. This is also a Markov process where the observable outputs can be seen, unlike the hidden layer.

The HMM algorithm works in two steps. The HMM is trained in the first step using the training sequences. At the initial state (at time t_0), the state transition probabilities and the observable output probabilities are randomly assigned. However, assigning these probabilities according to prior knowledge of the system, instead of the random assignment, can improve the performance of HMM. At this point, the model is denoted with λ_0. Then, applying the Baum-Welch algorithm, the HMM λ_0 is adjusted according to the input training sequences and construct the new model λ_1 [27]. After every adjustment of λ, the probability difference of the previous model and the adjusted model is calculated. If the difference is below the preset probability difference threshold, the model is known to be the final HMM. Otherwise, further adjustment is required. In the next step, the unknown sequences are applied to the model and the likelihood of the sequences (i. e., the probability of how much a sequence conforms the HMM) are determined. If the probability is above the predefined acceptable probability, the sequence is concluded as a non-anomalous sequence. Otherwise, it is concluded as an anomalous one. The HMM algorithm has very accurate prediction of anomaly and has been used for complex sequence analysis. However, the model training time is very high in HMM algorithm.

III. METHODOLOGY

As previously mentioned, in our research, we aim to minimize the training time while keeping the accuracy of the original HMM algorithm. Previous studies identified that the training time for the HMM algorithm depends on the number of the hidden states, the number of the observables and the length of the training sequences [11]. For these reasons, we intended to minimize these parameters in our I-HMM to make it relatively faster than the original HMM. We show our research methodology in Figure 1.

Our research methodology consists of three major steps: data collection, data processing and model construction. Our major contributions of this paper are in the data processing and behavioral model construction steps.

A. Data Collection

The data collection step consists of generating traces from a target system that will be used to build the model of the system. In this paper, we chose to focus on traces of routine calls, since the routine calls can reflect the presence of faults, unauthorized usage of resources or unusual function calls due to attacks. Same approach can be readily applicable to other types of traces.

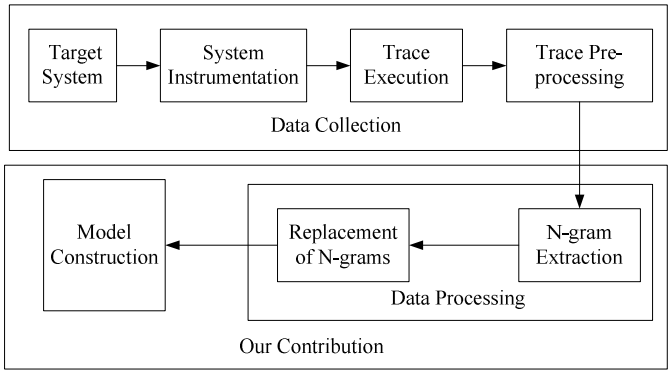

Figure 1. Methodology of our research.

There are different ways to generate traces including instrumenting the source code, using a debugger, or instrumenting the running environment (e.g OS). In this paper, we opted for program instrumentation in a running environment due to its simplicity and the availability of tools. Probes are inserted at the entry and exit of each routine.

For an anomaly detection algorithm to be effective, it is important to have a good coverage of the input data that is used to build the model. We achieve this by exercising the system by executing the test cases, which provide good coverage of the system.

Once the traces are generated, they are preprocessed to be used as input for an HMM system. For example, since HMM takes sequences of observables as input, we need to convert each raw trace into sequence of comma-separated routine calls. These sequences represent the exact sequence of routines that are called during the trace execution. Also some data cleansing is necessary such as the removal of contiguous repetitions to reduce the size of typical traces while keeping as much of the information they contain as possible.

B. Data Processing

As mentioned earlier, our strategy to reduce the size of traditional HMM (and therefore improve the learning time of the model) is based on n-gram extraction, which identifies the frequent common sub-sequences or patterns in a string; where, the length of the patterns can vary from one to n (the number of events in a trace).

There exist several n-gram extraction algorithms. In this paper, we adopt the one presented in [8]. This algorithm analyzes the training sequences, and extracts frequent patterns, i.e., n-grams, from them. Unlike the fixed length n-gram extraction algorithms for intrusion detection [29-30], it introduces a threshold α (varies from 0 to 1) to control the generalization ability of the model by allowing different lengths of the n-grams. At the beginning, the algorithm extracts all unique observables from the training sequences and labels them as 1-gram. For example, if ECDB, CDBA and EACDB are the input sequences, then A, B, C, D, E are the valid 1-grams.

In the consecutive steps, two n-grams of length k are combined to make an n-gram of length k+1. A sub-sequence or pattern p_{k+1} qualifies as an n-gram, if the frequency of p_{k+1} is greater than α multiplied by the minimum frequency of q_k and r_k. Here, p_{k+1} is constructed from q_k and r_k (two valid n-grams of length k).

Therefore, a model with a smaller α takes most of the n-grams, even with very low frequency, as valid n-grams and becomes very flexible. A very low value of α may lead to generate high false negative rate. Similarly, a model takes very few n-grams with high frequency, if the value α is significantly large. A very high value of α may lead to generate high false positive rate. If we take α = 0.6 in our previous example, and combine the two valid 1-grams E and C, we get EC that is present in the sequence. However, the frequency of EC is 1 in our input data which is less than α (= 0.6) * minimum frequency of E and C (= 2). Therefore, EC does not qualify as a valid 2-gram in the model. Whereas, CD is a composition of 2 valid 1-grams C and D, and the frequency of CD is 3 which is greater than α (= 0.6) * minimum frequency of C and D (= 3). Thus, CD is a valid 2-gram in the model. Similarly, DB is also another valid 2-gram in the model. Though, the 2-grams EC, BA, EA, AC are present in the input, they do not qualify as valid 2-grams because of there low frequency. In the next step, CD and DB are combined to make CDB. The 3-gram CDB is valid since the frequency 3 is higher than α (= 0.6) * minimum frequency of CD and DB (= 3). Since we do not have more than one 3-gram to compose a 4-gram, we stop at this point. That makes our highest n-grams to be 3-grams [8].

In our data processing step, we extracted all valid n-grams from our pre-processed trace sequences by setting α = 0.6. We marked each n-gram with a unique identification number for future use. Then, we replaced the n-grams in the trace sequences with their corresponding unique identification numbers (n-gram id). Before replacing the n-grams, as described in [8], we sorted the n-grams according to their lengths, where longer n-grams were replaced before the shorter ones. If there was a tie in their lengths, the one with higher frequency got the priority.

C. Model Construction

In this step, we construct the I-HMM. The process is similar to the construction of a traditional HMM. The set of observables in I-HMM are the n-grams instead of mere routine calls. Since common patterns (n-grams) are frequently found in the trace sequences, at most *n* number of routine calls can be replaced by one particular n-gram in I-HMM input sequences. Therefore, the longer is *n*, the shorter the training sequence for I-HMM is since this reduces the cardinality of the observable set. As we will show in the case study, these two factors result in minimizing the overall training time in I-HMM over HMM. We kept the number of hidden states in I-HMM the same as the number of hidden states in a traditional HMM. We randomly assigned the state transition probability and iteratively adjust the training model till it reaches the acceptable threshold [21, 27].

IV. CASE STUDY

The objective of the case study is to show whether I-HMM (i.e. an HMM based on n-grams – patterns of routine calls) improves over a traditional HMM (built based on mere sequences of routine calls) in terms of traning time and accuracy of the prediction. We achieve this by applying both approach to a system called Gzip [23]. All of our experiments are perfomr using an Intel Core 2 Duo Machine of 2.33 GHz with 4 GB of RAM.

A. Target System

As our target system to be modeled, we chose Gzip (GNU Zip) [23] for the case study. The Gzip software is a file compression and decompression tool for Linux that has similar functionalities as Winzip. We have chosen Gzip because it is written in C language, hence compatible with the LTTng (Linux Trace Toolkit Next Generation) instrumentation tool [24, 25].

B. Trace Generation and Pre-processing

We applied LTTng trace instrumentation for our trace generation as LTTng does not add significant overhead to the system [24]. In order to achieve a good coverage on Gzip data, we explored 200 individual test cases (e.g., open, decompress, uncompress, help, stdout, exit, etc.) from Gzip. All traces were collected in an intrusion-free environment (i.e. lab) to model the normal behavior of Gzip. Our LTTng trace instrumentation was able to record all entry and exit points of Gzip routines that were executed during trace collection. These records were saved as individual trace files for further study.

The generated raw traces needed pre-processing to act as the input data for both HMM and I-HMM. We wrote a parser in JAVA to extract all routine calls from each raw trace file and then to convert them into a sequence of comma-separated routine calls, maintaining the calling order. Furthermore, we wrote another JAVA program to remove the contiguous repeats of routine calls in each trace sequence.

C. HMM Construction

In our case study, we used the Weka 3.7.4 implementation of HMM (classifiers.bayes.HMM class) for model construction. This Weka implementation of HMM asks to specify the set of observables, the set of traces and the number of hidden states as inputs. We specified all routine calls as the set of observables and all pre-processed trace sequences as our input traces. We varied the number of hidden states from 5 to 20. However, our case study shows same accuracy for all varied number of hidden states. We kept the number of states as 5 to keep the training time the minimal. We constructed seven individual HMM models with 50, 75, 100, 125, 150, 175 and 200 healthy traces. During each model construction, we recorded the training time for each of the models.

D. I-HMM Construction

The I-HMM model construction required more data processing than the HMM model construction. We extracted all n-grams (see Section III for details) from the sequences of routine calls using a JAVA program implemented by us. We kept the value of α as 0.6 in our n-gram extractor, same as [8]. Then, we replaced the n-grams with their corresponding identification numbers in the trace sequences as described in Section III. Here, we also used the classifiers.bayes.HMM class of Weka 3.7.4 to implement the I-HMM model. We specified the n-gram ids as the set of observables and n-gram replaced traces as the input trace sequences. We also set the number of hidden states to be 5. We used 50, 75, 100, 125, 150, 175 and 200 healthy traces to constructed seven individual I-HMM models, like we did for HMM. We also documented the training time of each I-HMM model.

E. HMM and I-HMM Model Verification

After construction of each I-HMM and HMM models, we verified all of them by applying the cross validation technique of 5-folds [28]. We measured the accuracy of all 14 behavioral models (seven models of I-HMM and seven models of HMM) by taking the average accuracy calculated in all five folds. The result analysis of the experiments is described in the next section.

V. COMPARISON ANALYSIS

In this section, we present a comparative analysis of the performance of the HMM and I-HMM algorithms. We present the results of our experiments in terms of training time and accuracy of both algorithms.

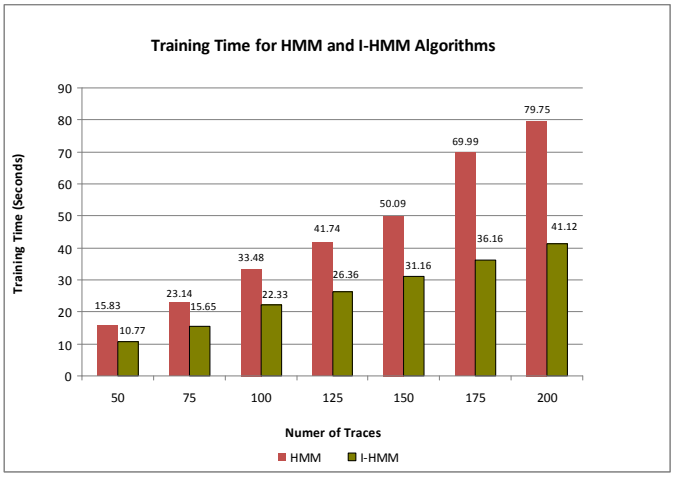

Figure 2. Comparative running time of original HMM and N-gram HMM.

In our experiments, we have seen that our I-HMM algorithm has always taken significant lower training time to build the software system behavioral models, compared to the original HMM algorithm (see Figure 2). More precisely, the I-HMM was able to reduce the training time from 31.96% to 48.44% of the original HMM algorithm. Another important observation from our experiments (shown in Figure 3) is that the training time differences between the HMM and I-HMM algorithms increased as we increased the number of traces for training the models. For example, our study shows that with 50 traces, the training time for HMM is 15.83 seconds and for I-HMM it is 10.77 seconds. Therefore, the training time reduces by 31.96%, if the model is built with 50 traces. After a gradual increase of training time reduction, there is a sharp rise (a rise from 37.79% to 44.38%), when the number of traces hits 175. Finally, the training time reduces by 48.44% (from 79.75 seconds to 41.12 seconds) when we construct a model from 200 traces.

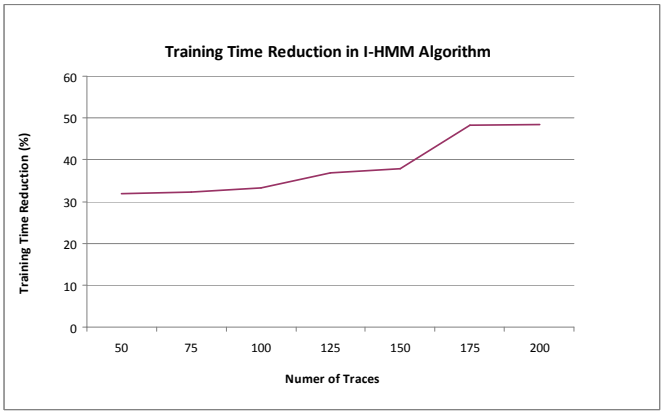

Figure 3. Comparative running time of original HMM and N-gram HMM.

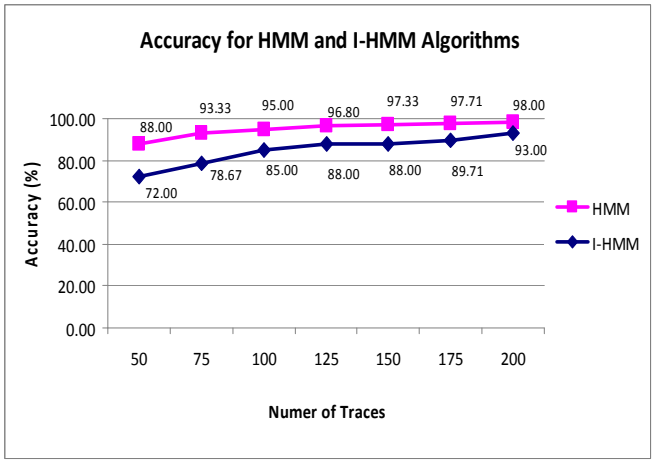

Figure 4. Comparative accuracy of original HMM and N-gram HMM.

As previously mentioned, the main advantage of the HMM algorithm is its accuracy, compared to other anomaly detection algorithms. In our experiment we also determined the accuracy of both HMM and I-HMM algorithms using the 5-fold cross validation feature available in Weka. Our experiments show that even though there are significant improvements in training time in the I-HMM algorithm, the original HMM algorithm achieves better accuracy. The compaction of HMM input sequences, caused by n-gram replacement, reduces the granularity of the input sequences in I-HMM. Therefore, I-HMM loses the ability to accurately identify anomalous behavior of the system as the original HMM does. In Figure 4, we can see that the accuracy of the original HMM is 88%, whereas it is noticeably low (72%) for the I-HMM algorithm with 50 traces. However, as we added more traces (i.e., more coverage) for behavioral model generation, the accuracy increases in both algorithms. Finally, with 200 traces, the accuracy achieved in the HMM is 98% and in the I-HMM it is 93%. The accuracy graph, shown in Figure 4, also reflects that using a good coverage of behavioral data for model generation ensures better accuracy in anomaly detection algorithms. Furthermore, the graph shows that as the coverage on training data increases, the I-HMM algorithm achieves comparable accuracy with the HMM algorithm.

VI. CONCLUSION AND FUTURE WORK

Intrusion detection is important in applying security measures on software systems and computer networks. Recent studies showed that the conventional Network-based Intrusion Detection Systems (NIDS) are not sufficient for identifying all types of intrusions, especially those that do not generate important network traffic. Therefore, along with the NIDS, the Host-based Intrusion Detection Systems (HIDS) has become an emerging area of research. Recent studies have also shown that the anomaly detection algorithms serve a key ingredient for intrusion detection systems. Hidden Markov Model (HMM) algorithm, for example, has shown to be very accurate in detecting attacks and faults.

However, the significantly large training time for behavioral model construction plays as a major obstacle for using HMM for anomaly detection. In order to ensure efficiency along with accuracy of HMM, we have introduced an improved HMM (I-HMM) where we replaced frequent common sequence of routine call observables with unique n-gram observables. These replacements considerably reduce the size of the observable sequences (i.e. trace) and the number of unique observables, hence

contribute to important reduction of training time. However, the use of n-grams in I-HMM results less accurate models than the models generated by the original HMM algorithm. Our preliminary studies show that the gaps between the accuracy of the HMM and I-HMM models can be reduced by improving the trace coverage during model construction. This ensures a fair tradeoff between the training time and accuracy in our I-HMM algorithm.

We are aware that this is a preliminary study and that more needs to be done. In the future, we will add more target systems (in addition to Gzip) with more trace coverage during model construction to test the accuracy, performance and the scalability of our model. Moreover, we will test our model with both anomalous and non-anomalous data and measure the accuracy. We will also conduct more experiments with changing the threshold α using during n-grams extraction and determine an optimum value for each of the different target systems.

Acknowledgement: This project is partly supported by the Canadian Natural Science and Engineering Research Council (NSERC) and Defence R&D Canada (DRDC). The authors would like to thank Dr. Shariyar Murtaza for providing the data used for the experiments.

REFERENCES

[1] V. V. Phohaha, "The Springer Internet Security Dictionary," Springer-Verlag, 2002.

[2] P. E. Proctor, "The Practical Intrusion Detection Handbook," Prentice Hall PTR, NJ, USA, 2001.

[3] V. Chandola, A. Banerjee, V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys, vol. 41(3), article: 15, July 2009.

[4] S. Kumar, and E. H. Spafford, "A pattern matching model for misuse intrusion detection," In Proceedings of the National Computer Security Conference, Baltimore, MD, 1994, pp. 11–21.

[5] D. E. Denning, "An Intrusion Detection Model," IEEE Transactions on Software Engineering, SE, vol. 13(2), 1987, pp. 222-232.

[6] S. Forrest, P. D'haeseleer, and P. Helam, "An immunological approach to change detection: Algorithms, analysis and implications". In Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, vol. 110, 1996.

[7] D. Endler, "Intrusion detection: applying machine learning to solaris audit data," In Proceedings of the IEEE Annual Computer Security Application Conference, Society Press, 1998, pp. 268 – 279.

[8] Guofei Jiang, Haifeng Chen, Cristian Ungureanu and Kenji Yoshihara, "Trace analysis for fault detection for application server", Handbook of Automatic Computing: Concepts, Infrastructures, and Applications, edited by S. Hariri, and P. Parashar, CRC Press, 2007.

[9] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternate data models," In Proceedings of the IEEE ISRSP. IEEE Computer Society, 1999, pp. 133 – 145.

[10] J. Hu, Q. Dong, X. Yu, and H. H. Chen, "A simple and efficient hidden markov model scheme for host-based anomaly intrusion detection," IEEE Netw. vol. 23(1), 2009, pp. 42 – 47.

[11] Jiankun Hu, "Host-Based Anomaly Intrusion Detection", Handbook of Information and Communication Security, Springer, 2010.

[12] S. Forrest, S. A. Hofmeyr, A. Somayaji. and T. A. Longstaff, "A sense of self for unix processes," In Proceedings of the IEEE ISRSP, 1996, pp. 120 – 128.

[13] E. Eskin, "Anomaly detection over noisy data using learned probability distributions," In Proceedings of the 17th International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., 2000, pp. 255–262.

[14] E. Eskin, W. Lee, and S. Stolfo, "Modeling system call for intrusion detection using dynamic window sizes," In Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX), 2001.

[15] A. K. Ghosh, and A. Schwartzbard, "A study in using neural networks for anomaly and misuse detection," In Proceedings of the 8th USENIX Security Symposium, 1999.

[16] N. Abouzakhar, A. Gani, G. Manson, M. Abutbel, and D. King, "Bayesian learning network approach to cybercrime detection," In Proceedings of the 2003 Post Graduate Networking Conference, Liverpool, United Kingdom, 2003.

[17] W. Hu, Y. Liao, and V. R. Vemuri, "Robust anomaly detection using support vector machines," In Proceedings of the International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., 2003, pp. 282–289.

[18] G. Stein, C. Bing, A. S. Wu, and K. A. Hua, "Decision tree classifies for network intrusion detection with GA-based feature selection," in Proceedings of the 43rd Annual Southeast Regional Conference, Georgia, 2005, pp. 136 – 141.

[19] Q. Xu, W. Pei, and Q. Zhao, "An intrusion detection approach based on understandable neural network trees," Journal of Electronics, 2007, pp. 574 – 579.

[20] R. C. Chen, K. F. Cheng, Y. H. Chen, C. F., Hsieh, "Using Rough Set and Support Vector Machine for Network Intrusion Detection System," In proceedings of the First Asian Conference on Intelligent Information and Database Systems, 2009, pp. 465 – 470.

[21] L. R. Rabiner and B. H. Juang, "An introduction to hidden markov models," IEEE ASSP Magazine, 1986.

[22] P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language", Computational Linguistics, vol. 18, pp. 467–479, 1992.

[23] Gzip Official Website http://www.gzip.org/

[24] M. Desnoyers, and M. R. Degenais, "The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux," In Proceedings of Ottawa Linux Symposium, Ottawa, Canada, July 19 – 22, 2006.

[25] LTTng Official Website. http://lttng.org

[26] Weka Official Website http://www.cs.waikato.ac.nz/ml/weka/

[27] Leonard E. Baum, Ted Petrie, George Soules and Norman Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains", The Annals of Mathematical Statistics, vol. 41(1), February, 1970, pp. 164 – 171.

[28] J. Han, and M. Kamber, "Data Mining: Concepts and Techniques," 2nd edition, San Francisco: Elsevier, 2006.

[29] S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, "A sense of self for Unix processes," In Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 6–8, 1996, pp. 120–128.

[30] C. C. Michael and A. Ghosh, "Simple, state-based approaches to program-based anomaly detection," ACM Transaction on Information and System Security (TIISSEC), vol. 5(3), pp. 203 – 237, August 2002.