Article can be accessed online at http://www.publishingindia.com

An Architecture for Host-based Intrusion Detection Systems using Fuzzy Logic

Maryam Rostamipour*, Babak Sadeghiyan**

Abstract

Intrusion Detection Systems (IDSs) are key parts of computer system defences used to detect malicious activities or policy violations and produce reports to a management station. In this paper, we propose a host-based IDS to detect with a fuzzy logic approach. The novelty of our proposed system is that multiple features are extracted for each session in order to identify attacks, and then fuzzy inference expert systems are used to detect intrusion. Selected features are extracted based on system call arguments and used to detect the buffer overflow attack in UNIX system. Because of the difficulty of specifying the exact amounts of them, fuzzy inference expert system is used to detect intrusion. The extracted features from audit trail are related to the different stages of attack scenario, so the output of proposed system is suitable for forensic investigation. Our Host-based Intrusion Detection System (HIDS) is tested experimentally against DARPA 98 and 99 intrusion detection datasets. A comparison with other learning-based approaches is performed. The final results show that our system is efficient.

Keywords: Network Security, Buffer Overflow Attack, Host-based IDS, Fuzzy Intrusion Detection, Fuzzy Logic

1. Introduction

Intrusion detection has emerged as a significant field of research, because it is not theoretically possible to set up a system with no vulnerabilities (Bridges, & Vaughn, 2000). There are two fundamental approaches employed in the detection analysis engine of IDSes, i.e., misuse detection and anomaly detection. A system which employs the first approach detects intrusion events which try to model abnormal behaviour. However, an anomaly detection based system recognises patterns of activities that appear to be normal, and detects intrusion events which do not follow normal patterns (Yao et al., 2005). In misuse detection systems, the knowledge of past intrusions can be used by human experts into expert system rules.

IDS can also be categorised as network-based and host-based intrusion detection system. In network-based IDS (NIDS) such as (Denning, 1987), header fields of the various network protocols and payloads are used to detect intrusions. In the latter approach (Paxson, 1998; Lindqvist & Porras, 1999), operate on information collected from within an individual computer system. In HIDS most of the present techniques revolve around sequences of system calls. These techniques often are based upon the observation that an abnormal activity results in sequence of system calls, for example (Forrest *et al*., 1966; Wanger & Dean, 2001). Recently, it has been shown how these systems can be bypassed by triggering attacks that execute legitimate system call traces (Mutz *et al*., 2006). Mimicry attacks operate by crafting the injected malicious code in a way that imitates can be easily bypassed (Kruegel *et al*., 2003). Other possible approaches to create models of program behaviour are extracted system calls and their arguments, path, return value and error status, etc. (Mutz *et al*., 2006; Kruegel *et al*., 2003; Varghese *et al*., 2007).

* M.Sc. Student of Information Security, Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran. E-mail: rostamipoor@aut.ac.ir

** Associate Professor in Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran. E-mail: basadegh@aut.ac.ir

Many researches have been done on anomaly-based HIDS. Examples include Bayesian networks to perform system call classification (Varghese *et al*., 2007), mining on system call data (Akbarpour & Maarof, 2009; Portnoy *et al.*, 2001), Hidden Markov Model (HMM) to differentiate between the normal and abnormal behaviours (Maggi *et al*., 2010) and k-Nearest Neighbour method to model the normal user programs (Sharma *et al*., 2007).

In this paper, the features are extracted to detect the buffer overflow attack in UNIX system. Because of the difficulty of extracting all attributes in the attack scenario and specifying the exact amounts of them, fuzzy inference expert system is used to detect intrusion. We extract these features for each phase of buffer overflow attacks and use them as inputs of fuzzy system to detect intrusions. Fuzzy systems have a number of abilities which make them suitable as intrusion detection systems. However, there are very few HIDSs that use fuzzy logic. These systems often detect intrusion based on abnormal sequence of system calls. For example, Akbarpour & Maaruf (2009) have proposed an algorithm to improve complexity, flexibility and accuracy of the system utilizing data mining and Fuzzy Inference Engine. Instead of analyzing the system call sequences to model the normal profiles, this paper presents a novel misuse HIDS that uses the information contained in system call arguments. Five features are extracted for each session in order to identify the buffer overflow attacks. Each of the features corresponds to one phase of scenario attacks. Three phases are considered for buffer overflow attack in UNIX system. Fuzzy rules are generated to detect intrusion based on the feature values. We test our system against DARPA 98 and 99 datasets (2003). Our results show a considerable success in detection of buffer overflow attacks.

The rest of the paper is organised as follows: The second section defines the buffer overflow attacks. The third section introduces our proposed architecture for attacks detecting. The implementation of proposed intrusion detection system using fuzzy logic is given in the fourth section. Experimentation and performance analysis of the proposed system is discussed in fifth section. Finally, the conclusion is given in the sixth section.

2. Buffer Overflow Attacks

Buffer overflows occur when a sequence of bytes of length n is placed into an array, or buffer, of length less than n (Bishop *et al*., 2012). The goals of buffer overflows are to change variables, function pointers in the area or return addresses on the stack. In the paper by Bishop *et al*. (2012), two classes of buffer overflow attacks have been suggested. In first class, a data buffer overflow occurs when input overwrites the existent data to violate the security policy. The executable buffer overflow is another class that occurs when executable code is loaded into a buffer, and a return address or function pointer is altered to cause that code to be executed.

In DARPA 98 dataset, second class of buffer overflow attacks has been applied. The total of four U2R attacks against the Solaris includes: Eject fdformat, ffbconfig, and ps attacks. The 'ps' attack was present only in testing dataset. All these attacks exploit root-owned programs accessible to normal users on the system [18]. Every program has a 'set user-ID' (suid) bit associated with it. If suid bit is set, a user will be able to access the program with privileges of root. Setting this bit can cause problems.

Assume that a user executes an attack through a process owned by a root user having suid bit set. Now if the user is able to stop the process before it finishes execution, shall have privileges of root since suid bit was set. All buffer overflow attacks use this mechanism (Sabhnani & Serpen, 2003). Usually the attacker utilises an exploit code to invoke the vulnerable program using a shellcode as input. These exploit code is compiled and then executed. Executing this exploit code causes the address points return to the beginning of the shellcode. This mechanism alters the arguments of system call or number of system call. We use these changes to extract the features for detection of attacks.

3. Proposed Architecture

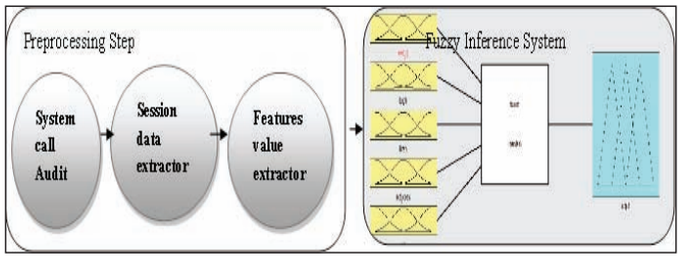
In this paper, we propose architecture shown in Figure 1 for HIDS that uses fuzzy inference engine to detect buffer overflow attacks in UNIX system. The architecture has two steps. The first step includes system call audit, session data extractor and value extractor modules.

The system call audit module keeps track of the system calls in audit log file. These audit log file are sent to the next module. The second module drives sessions from audit log file. Then, the features value extractor module initializes the value of selected features. Five features are selected in our system for detection of buffer overflow

attacks. Selected features are extracted based on the system call arguments.

In second step, fuzzy inference expert system is used to detect intrusion based on the selected features. In the following subsections, we will describe each step.

Figure 1: Proposed Architecture



3.1. Preprocessing Step

In this step, preprocessing is performed on raw data gathered from user activity. Preprocessing consists of three modules which will be explained in the following.

3.1.1. System Call Audit

For different operating systems, there are a variety of methods and tools to audit user activity. Some of them come preinstalled within common distributions, some can be downloaded as freeware, and some are commercially available products. It is assumed that call arguments are stored in audit log files.

3.1.2. Session Extractor

This module extracts the sessions from audit log file. Each session corresponds to a connection between host and user is connecting remotely or locally. Individual sessions can be extracted from the audit data. Extracted sessions are given as input to the next module.

3.1.3. Features Value Extractor

Some attacks can be detected through their impacts on session, e.g., some user-to-root attacks. Changing some features of a session could announce an unusual event. In this step, it is assumed that these attributes are defined already. The feature values are initialized for each session extracted from the previous step.

3.2. Fuzzy Inference System

Fuzzy logic is a powerful technique for dealing with human reasoning and decision-making (Liao *et al*., 2009). A fuzzy expert system is a collection of membership functions and rules that are used for reasoning expert knowledge. In a fuzzy expert system, the inference process is a combination of four sub-processes: fuzzification, inference, composition, and defuzzification (Liao *et al*., 2009). There are several methods to perform each of the sub-processes. Set of "if-then" rules are employed to reason about data. Rules are created using expert knowledge or automated methods such as Genetic algorithm (GA) or clustering.

In proposed architecture, fuzzy inference expert system is applied to detect intrusion. The fuzzy system uses feature values to make decisions about the intrusion. Output of fuzzy system is a value between [0, 1] which indicates whether an attack has occurred.

4. The Implementation of Model

In this paper, we apply the proposed model for detecting the buffer overflow attacks in UNIX. The model is fully implemented. Java language is used to implement preprocessing step. We apply Matlab fuzzy toolbox to create a fuzzy inference system and link it to preprocessing modules by java programming. The DARPA 98 and 99 datasets are used for purpose of training and testing the proposed model. The MIT Lincoln Laboratory collected network traffics and system logs. The dataset is distributed for evaluation of computer network Intrusion Detection Systems (IDS) (Thomas *et al*., 2008).

The implementation details of the model are described in the following.

4.1. Preprocessing Step

In UNIX operating system, there are many tools and methods to audit events. In this paper, we used BSM audit data collected from a victim Solaris machine in DARPA dataset 98 instead of system call audit module. The BSM audit logs contain information on system call produced by process running. There are many kernel and user events that the system executes, including: execve, mmap, open,

close, munmap, stat, etc. (Sabhnani & Serpen, 2003). Each of events includes the name of system calls, owner of process, effective user id, real user id, path of process, etc.

In audit record of UNIX system, each system call begins with a `header` token and ends with a trailer token. The header token includes the total length of event and hence suggests whether a large argument was passed to the program. It also contains other arguments such as time, name of system call, etc. The "*path*" token specifies which program was executed and contains access path information for an object. The "*attribute*" token points to whether suid bit is set and also indicates the owner of the process. The '*exec_args'* token contains system call argument information. It will indicate how many and which arguments are passed to the program. Subject token will indicate the effective user id (euid), the real user id (ruid), and the session number (session). The ruid is the UID of the user or process that created this process but euid is the UID of the user or process that executed this process. The euid is used to evaluate privileges of the process to perform a particular action.

In second module, session data extractor module extracts sessions from BSM audit log file. Each session is isolated based on sid placed in the header token of system call. This module is implemented by a java class in our system. The specified features are initialized by next module. These attributes must be defined already. We extract five features for each telnet and local sessions, and then, use them to detect the buffer overflow attacks. Each of the features corresponds to the one phase of scenario attacks.

Three phases are considered for buffer overflow attack in UNIX system. In the first phase, the exploited code is compiling in system with gcc library. In the second step, attacker uses the compiled code to exploit the vulnerable program. In this step attacker gains root access. Finally, the attacker can abuse the root access obtained.

In our model, the feature extraction is performed based on the review of attack and normal records and knowledge of how the buffer overflow attack occurs.

For the first phase of attack, we select two features to represent this phase. These features were labeled with "*library*" and "*usr*". The *library* feature shows number of system calls on "/usr/lib/gcc" or "/opt/local/lib/gcc-lib/" paths that correlated to gcc library in UNIX system. These features indicate that a file is compiled with gcc library.

Another feature is related to the number of system calls on "/usr/ccs/" path. While building the executable file, the static linker searches for any libraries in the "/usr/ccs/lib" path. In UNIX, all binary files are stored in appropriate `bin` directories such as "/usr/ccs/bin". For executing of a binary file, these directories are searched. So, there are a lot of system calls on "usr/css" path in audit file. The "*usr*" feature shows a binary file is created and executed.

In the second phase, the attacker uses the buffer overflow vulnerability of program in the victim system. In the papers by Sabhnani & Serpen (2003) and Lindqvist & Porras, (1999), heuristic rules have been presented to detect buffer overflow attack in DARPA 98 dataset. The proposed heuristic rule by Sabhnani & Serpen (2003) is shown in Figure 2.

(event = EXECVE) ∧
(euid != ruid) ∧
(owner = "root") ∧
(suid = 1) ∧
(length > 400) ∧
(arg_num < 6) ⇒ Buffer overflow attack

The threshold for length was set to 400. Only few non-attack records had length greater than 400 but those records required much larger value for parameter "*args_num*". With our assessment, this heuristic rule would generate the false alarms. However, the authors of this paper had claimed the false alarm is zero. If the threshold for length is set to 500, then false alarm is zero in DARPA 98 dataset. The problem of these heuristic rules is that intervals must be specified explicitly.

Figure 2: The Proposed Heuristic Rules for Buffer Overflow Attack Detection (Sabhnani & Serpen, 2003)

Missinng

We use this heuristic rule to extract two features for detecting the second phase of attack. In this model we extract the features for one session so only one "execve system call" must be selected. This system call shows whether the attacks accrued or not. In the session, a lot of execve system call exists. We select one execve system

Table 1: Type and Values of the Parameters for Membership Functions

Channels	*Interval*	*Low (gaussmf)*	*Med(gaussmf)*	*High (Smf)*
Exec_arg	[0 40]	[1.84 4.24]	[3.52 13.3]	[12.16 24.96]
Length	[0 900]	[54.88 108]	[85.3 341.2]	[363.1 677.9]
Library	[0 200]	[7.26 21.3]	[9.94 52.05]	[101 152]
Usr	[0 200]	[3.6 8.727]	[7.74 23.8]	[45.25 87.21]
Root_access	[0 50]	[2.656 6.33]	[4.983 18.36]	[25.2 38]
Output	[0 1]	[0.183 2.5 6.94e-018]	[0.116 0.4074]	[0.504 0.753]

call with minimum "*args_num*" and maximum "*length*". This system call must be chosen between the execve system calls with the root owner and the euid that is not equal with the ruid.

For detection of third phase of attack, the number of system calls in the session with root euid is measured. The "*root_access*" feature is useful for misuse detection of root access obtained. Five extracted features send to fuzzy inference engine as inputs to detect attacks.

4.2. Fuzzy Inference Engine

Using fuzzy systems enables the ability to formulate expert knowledge to the system with simple rules. A fuzzy expert system is applied to detect the intrusions using the five extracted features, reached from the preprocessing step, as input. A fuzzy set is considered for each of the extracted features. So, our fuzzy system has five input fuzzy sets and one output fuzzy set that determines whether the attack accrued or not. We consider three membership functions of Low, Medium and High for each of the fuzzy sets. The type and values of the parameters for membership functions are shown in Table 1.

We make our fuzzy inference engine employing fuzzy Singleton, Mamdanie's inference, Mamdanie's product implication, and the center of average de-fuzzifier. The non-linear mapping of the system has the following form:

$$f(x) = \frac{\sum_{l=1}^{M} \bar{y}^{l} \left(\subseteq_{i=1}^{n} \mu_{A_i^l}(x_i) \right)}{\sum_{l=1}^{M} \left(\subseteq_{i=1}^{n} \mu_{A_i^l}(x_i) \right)} \quad (1)$$

In Figure 3 and Figure 4, membership functions for two features are shown.

Figure 3: Membership Functions for "Length"

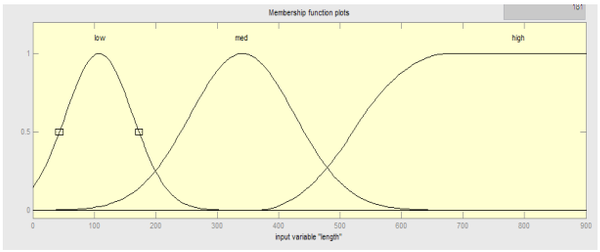
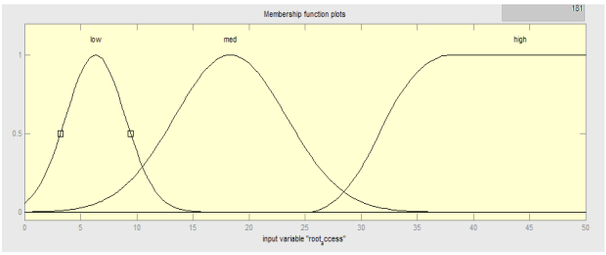
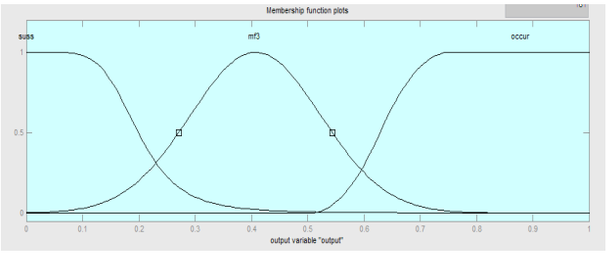


Figure 4: Membership Functions for "Root_access"



The range of output membership functions are distributed uniformly in the interval [0, 1] and are shown in Figure 5.

Figure 5: Membership Functions for "Output"



Setting the parameters of membership functions are carried out with the aim of reducing the number of rules. The six rules are extracted based on normal and attack telnet sessions in training dataset. Best results are obtained using these rules. The rules are listed in Table 2.

Table 2: Extracted Rules

Exec_arg	*Length*	*Library*	*Root_access*	*Usr*	*Output*
low	high	high	med	High	occur
low	high	med	high	Med	occur
low	high	med	med	Med	occur
low	high	high	high	High	occur
low	low	high	low	Med	not
med	med	low	low	Low	not

5. Results

This section describes the experimental results and performance evaluation of the proposed system. The goal of our tools is to detect the buffer overflow attacks in the UNIX environment. Two tests have been designed to evaluate the system. The aim of the first experiment is to use the data set to allow the comparison of the performance of the system with that of other systems. For experimental evaluation, we have taken DARPA 98 and 99 datasets, which are mostly used for evaluating the performance of the HIDS.

For first experiment we use DARPA 98 dataset early. Our system is evaluated with three criteria. These criteria include false positive rate, detection rate and accuracy. The overall accuracy of the proposed system is computed based on the definitions of detection rate, number of false alarm and false alarm rate. The criteria are defined as follows:

$$\text{detectionrate} = \frac{\#\text{ of true positive}}{\#\text{ of true positives} + \#\text{ of false negatives}} \quad (2)$$

$$\text{false positive rate} = \frac{\#\text{ of false positives}}{\#\text{ of true positives} + \#\text{ of false positives}} \quad (3)$$

The training DARPA 98 dataset was applied to extract the rules for the proposed system. The system with the threshold value of 0.65 has the detection rate and accuracy as 100%. With applying these rules, the false alarm is 0%. In comparison with heuristic rules proposed in (Forrest *et al*., 1966) and (DARPA 99, 2003), our system has better results. The detection rate and false alarm of system are 80% and 0% respectively in the paper by Forrest *et al*. (1966). In DARPA 99 (2003), they have claimed the detection rate and false alarm have been obtained 100% and 0% respectively. However we observed several sessions that their system produce a false alarm.

Then, we tested our system with DARPA 99 dataset to compare it with other systems. The attacks are similar to the first experiment. We compare our system with the system used by Mutz *et al.* (2006) that has applied the same dataset to it. To compare our system with some other well-known intrusion detection systems, we take advantage of the report presented in the paper by Mutz *et al*. (2006). In this paper, unsupervised intrusion detection systems have been selected. These systems operate on unlabeled training data to allow a fair comparison to proposed technique in the paper by Mutz *et al*. (2006). The comparison of results is shown in Table 3.

The first system (Kang *et al*., 2005) extends the system call sequences to bags of system calls. The technique produces less false positives and paid for by a higher number of missed attacks. The second system (Portnoy *et al.*, 2001) uses machine learning techniques to identify outliers in a high-dimensional vector space. The last system combines multiple anomaly scores using a Bayesian modeling approach. Each Bayesian network requires a probability threshold that allows it to distinguish between attacks and normal system calls.

As can observe in Table 3, our system shows excellent results against the buffer overflow attacks. In the proposed system, there are no false alarms. The system with the threshold value have been set to 0.65 has the detection rate and accuracy as 100%. With applying the six rules, the false alarm is 0%.

The overhead of proposed fuzzy system is low due to the low number of rules and inputs. Thus efficiency and overhead of our system depends on the sessions and feature extraction algorithms. An efficient algorithm can increase system efficiency.

Table 3: Comparison of Results

Application	*Syscall Bags[21]*		*Clusters[12]*		*Bayesian classification[8]*		*Output*	
	FN	FP	FN	FP	FN	FP	FN	FP
eject	1	1	0	1	0	0	0	0
fdformat	2	0	0	0	0	0	0	0
ffbconfig	0	0	0	0	0	0	0	0
ps	0	0	12	25	0	0	0	0

As observed in Table 3, our system shows excellent results against the buffer overflow attacks. In the proposed system, there are no false alarms. The system with the threshold value have been set to 0.65 has the detection

rate and accuracy as 100%. With applying the six rules, the false alarm is 0%.

The overhead of proposed fuzzy system is low due to the low number of rules and inputs. Thus efficiency and overhead of our system depends on the sessions and feature extraction algorithms. An efficient algorithm can increase system efficiency.

In the second experiment, the goal is to investigate the system performance in the real world. Simulated environment consisted of a target system with Sun Solaris 10 operating system with snort installed, the attacker system with Windows 7 operating system and the host system with Ubuntu operating system. In order to implement the scenario, a vulnerable program with length of 300 was created. Using this program, the attacker obtains the high level access to the system. The proposed system sets the condition of this session to occur, and will detect the attacker and attack occurrence.

6. Conclusion

The system calls and their arguments have been used to model the normal behaviour of process for anomaly-based HIDS. Recently, it has been shown how these systems can be bypassed by triggering attacks that execute legitimate system call traces. In this paper, we proposed a host-based IDS system to detect the Buffer Overflow attacks. The novelty of our proposed system is that multiple features are extracted based on system call arguments for each session in order to identify attacks. Then fuzzy inference expert system has been used to detect intrusion. The extracted features from audit trail are related to different stages of attack scenario, so the output of proposed system is suitable for forensic investigation. Furthermore, we used the training DARPA 98 dataset to extract the six rules for the proposed system. With applying test DARPA 98 dataset, the proposed system has the detection rate and accuracy are set to 100% with false alarm as 0%. Additionally, we performed a comparison of our system to five other learning-based approaches on DARPA 99 data set. This comparison showed that our system has the higher detection capability. Because of the low number of rules and inputs, the overhead of proposed fuzzy system is low. Also, the second experiment shows the system is efficient in the real world.

References

Akbarpour, S. M., & Maarof, M. A. (2009). *Fuzzy Intrusion Detection System via Data Mining Technique with Sequences of System Calls.* 5th International Conference on Information Assurance and Security, (pp. 154-157).

Bishop, M., Engle, S., Howard, D., & Whalen, S. A. (2012). *Taxonomy of Buffer Over-flow Characteristics*. IEEE Transactions on Dependable and Secure Computing, May-June, 9(3), 305-317.

Bridges, S. M., & Vaughn, R. B. (2000). *Fuzzy Data Mining and Genetic Algorithms Applied to Intrusion Detection.* In Proceedings of National Information Systems Security (NISSC), (pp.16-19).

DARPA Data Set, 1998, (2003). Retrieved from http://www.ll.mit.edu/IST/-ideval/data/1998/1998_data_index.html.

DARPA Data Set, 1999, (2003). Retrieved from http://www.ll.mit.edu/IST/-ideval/data/1999/1999_data_index.html.

Denning, D. E. (1987). *An Intrusion Detection Model.* IEEE Transactions on Software Engineering, 13(2), 222-232.

Forrest, S., Hofmeyr, S. A., Somayaji, A. & Longsatff, A. (1966). *A Sense of Self for UNIX Process*. IEEE Symposium on Security and Privacy, (pp. 120-128).

Kang, D. K., Fuller, D., & Honavar, V. (2005). *Learning Classifiers for Misuse and Anomaly Detection using a Bag of System Calls Representation*. In Proceedings of 6th Annual IEEE SMC, (pp. 118-125).

Kruegel, K. C., Mutz, D., Valeur, F., & Vigna, G. (2003). *On the Detection of Anomalous System Call Argument.* In Proceedings of 8th European Symposium on Research in Computer Security, (pp. 326-343).

Liao, N., Tian, S., & Wang, T. (2009). Network forensics based on fuzzy logic and expert system. *Computer Communications*, November, 32(17), 1881-1892.

Lindqvist, U., & Porras, P. A. (1999). *Detecting Computer and Network Misuse with the Production-Based Expert System TTOLSET (P-BEST).* IEEE Symposium on Security and Privacy, Oakland, California, (pp 146-161).

Maggi, F., Matteucci, M., & Zanero, S. (2010). *Detecting Intrusions through System Call Sequence and Argument Analysis*. IEEE Transactions on Dependable and Secure Computing, 7(4), 381-395.

Mutz, D., Valeur, F., Kruegel, C., & Vigna, G. (2006). *Anomalous System Call Detection*. ACM Transactions on Information and System Security (TISSEC), 9, (pp. 61-93).

Portnoy, L., Eskin, E., & Stolfo, S. (2001). *Intrusion Detection with Unlabeled Data Using Clustering*. In ACM CSS Workshop on Data Mining Applied to Security (DMSA).

Sabhnani, M., & Serpen, G. (2003). *Formulation of a Heuristic Rule for Misuse and Anomaly Detection for U2R Attacks in Solaris Operating System Environment*. In Proceedings of Security and Management, (pp. 390-396).

Sharma, A., Pujari, K., & Paliwal, K. K. (2007). Intrusion detection using text processing techniques with a kernel based similarity measure. *Computers & Security*, 26, 488-495.

Thomas, C., Sharma, V., & Balakrishnan, N. (2008). *Usefulness of DARPA Dataset for Intrusion Detection System Evaluation*. In Proceedings of SPIE Defense and Security Symposium, 6973, pp. 69730G-8.

V. Paxson, Bro: (1998), *A System for Detecting Network Intruders in Real-Time*. In Proceedings of 7th USENIX Security Symposium, San Antonio, TX.

Varghese, S. M., & Jacob, K. P. (2007). Anomaly detection using system call sequence sets. *Journal of Software*, December, 2(6), 14-21.

Wanger, D., & Dean, D. (2001). Intrusion Detection via Static Analysis. IEEE Symposium on Security and Privacy on 2001. (pp. 156-168).

Yao, J. T., Zhao, S. L., & Saxton, L. V. (2005). *A Study on Fuzzy Intrusion Detection. In Data Mining, Intrusion Detection, Information Assurance, and Data Network Security 2005*. Proceedings of the International Society for Optical Engineering, (pp. 23-30).