International Journal of Networks and Communications 2013, 3(3): 81-90
DOI: 10.5923/j.ijnc.20130303.02

A Discriminatory Model of Self and Nonself Network Traffic

Adetunmbi A. O, Olubadeji Bukky, Alese B. K, Adeola O. S*

Department of Computer Science, Federal University of Technology, Akure, Nigeria

Abstract The matrix of business and other transaction systems over the Internet makes computer security a critical issue in our day-to-day activities. In recent times, various approaches ranging from rule-based, expert system to data mining have been subjected to extensive research in handling security breaches on computer networks. Immune system (IS) presents a protection against the possibility of malfunctioning and failure of individual host cells. In mammals it keeps the organisms free of pathogens which are unfriendly foreign organisms, cells, or molecules. Two approaches to change detection which are based on the generation of T-cells were examined. One is an existing model while the other model is proposed by us, the one proposed by us is called immunological model, which is a protection model capable of autonomously detecting (Nonself) and opposing the attempts at intrusion and exploitation. The two models were implemented using C++ programming language and their feasibility determined on 1999 International Knowledge Discovery Intrusion Detection Datasets. The results reveal that our proposed model outperforms the existing model not only in terms of detection accuracy but also in terms of simplicity and generation of explainable rules inform of if ... then statements. The classification accuracy of our model christened IMSNT on training and test Datasets are 97.06% and 86.39% as against 89.65% and 85.70% on the Stephanie et al approach, which shows that it is a promising approach. The proposed system apart from its capability of detecting and monitoring the activities on the network can be used in extracting virus signature patterns.

Keywords Immune System, T-cells, Intrusion Detection, Self-Network and Non-self Network

1. Introduction

The Immune System (IS) is complex, and it has novel solutions for solving real-world problems. This can be applied as a solution to systems design in case there is an artificial system facing similar problems faced by the Immune System, which required reasonable understanding of immunology. The problem that the IS address is similar to the problem faced by computer security systems: the immune system protects the body from pathogens, and analogously, a computer security system should protect computers from intrusions. This analogy can be made more concrete by understanding the problems faced by computer security systems[1-2].

The word immunity (from Latin *immunitas*) means "freedom from". The main purpose of the immune system is to keep the organism free from unfriendly foreign organisms, cells, or molecules (collectively called pathogens). The innate immune system primarily is inborn which consists of the endocytic and phagocytic systems, which involve motile scavenger cells such as macrophages that ingest extra cellular molecules and materials, clearing the system of both debris and pathogens. Most of the inspiration for this research has been drawn from the adaptive immune system (IS), and as such we shall preview an adaptive immunity.

The adaptive immune system is so-called because it adapts or "learns" to recognize specific kinds of pathogens, and retains a "memory" of them for speeding up future responses. The learning occurs during a primary response to a kind of pathogen not encountered before by the Immune System. The primary response is slow, often first only becoming apparent ninety-six hours after the initial infection, and taking up to three weeks to clear the infection. After the primary response clears the infection, the IS retains a memory of the kind of pathogen that caused the infection. Should the body be infected again by the same kind of pathogen, the IS does not have to re-learn to recognize the pathogens, because it "remembers" their specific appearance, and so can mount a much more rapid and efficient secondary response.[3]. The secondary response is usually quick enough so that there are no clinical indications of a re-infection. Immune memory can confer protection up to the life-time of the organism (a canonical example is measles).

Also a model of intrusion detection is based on the principles of the immune system, that carry out both signature-based and anomaly detection which has

* Corresponding author:
deleadeola@yahoo.com (Adeola O. S)
Published online at http://journal.sapub.org/ijnc

mechanisms for detecting deviations from a set of normal patterns, and it has ways of storing and recalling specific patterns associated with previous pathogenic attacks. Though, the current Computer security system protects computers from intrusions but the growing scale of computer networks and sophisticated software codes make them more vulnerable to alien intrusions, such as computer viruses, intentional corruption, among others that could lead to serious failures of computer-based information and control systems. Majority of the computer security systems widely used are either rule-based or expert system based which are characterized by low accuracy in terms of detections of intrusions on computer system or network.

Various researchers have imbibed the concept of biological systems to resolve some facet of information security in a computer system and networking environment. Among these researchers include the works of[4] on Artificial Immune System for virus detection, and[5] on Artificial Immunity System for Network Security Situation Awareness Technology. Their findings show that it is a proving approach as it reduces false positive rate and cases of security incidence on computers and computer networks. The essence of the immune system is to keep the organisms free of pathogens which are unfriendly foreign organisms, cells, or molecules for survivability.

The adaptive immune system made up of lymphocytes, with the ability to learn, recognize specific kinds of pathogens, and retains memory of them for future responses. The immune system model used is based on T-cells approach. Here, intrusive traffics refer to as nonself stand for pathogens while the classification model developed with the ability to learn and generate patterns of intrusions represent the T-cells. In this paper, an attempt is made to develop an immunological model to differentiate benign and malicious network traffic, demonstrating the feasibilities of these approaches on the experiment performed on intrusion detection data available at Massachusetts Institute of Technology, University website, USA.

2. Biological versus Computer Immune Systems

There are different ways to interpret biological immune systems for security. The immune system is perhaps the most obvious system, which would have an analogy for security. Its role is to defend against attack, patch and clean up after an attack; thus appropriate for all of the threats.[6] reports antigens (foreign proteins) are recognized by antibodies (immune system detectors). The antibodies are highly specific, only binding to a small set of antigens if they do bind, then a complex set of events occur that result in the foreign protein being destroyed. Antibody cells are covered with antigen detectors and they are as theoretically likely to match and destroy healthy cells as foreign proteins. This would obviously be undesirable, and in most cases does not happen. The immune system thus appears to be able to discriminate between "self and "non-self".

[7] makes the analogy between self in the body and normal behaviour of a computer system (non-self is thus abnormal behaviour). Self is represented as a set of strings (with a variety of different representations) depending on the domain and antibodies or detectors are also represented as strings. The binding between the strings is modeled by a matching function, the most common one being contiguous bits, which returns true when two strings match in more than specified contiguous positions. This allows detectors to match a variety of strings.[7-8] use detectors for non-self (which is directly analogous to immune system), while in[9] uses detectors for self.

3. Review of Related Literatures

It is well known that there are vulnerabilities in computer and network systems due to design flaws that can lead to security hazards[10, 11, and 19]. These flaws are expensive to fix and it is difficult or nearly impossible to build a completely secure system void of design and programming errors[12];[11] and[13]. Even a truly secure system is vulnerable to abuse by insiders who abuse their privileges[14].

It is glaring that we are stuck with systems that have vulnerabilities for a while to come, the next direction is to employ intrusion detection as a last line of defense. The benefits of an intrusion detection system (IDS) include: Detecting attacks or break-ins on system as soon as possible preferably in real-time for appropriate actions: such as., shut down the connections, trace back to identify the intruders, or gather legal evidence to prosecute the intruder and prevention of similar attacks in the future.

Intrusion detection is a process of detecting security breaches by examining user and program activities in a computer system. The most popular way to detect intrusions has been by using the audit data generated by the operating system. An adult trail is a record of activities on a system that are logged to a file in chronologically suited order. Since almost all activities are logged on a system, it is possible that a manual inspection of these logs would allow intrusions to be detected. However, the incredibly large sizes of audit data generated make manual analysis impossible. IDS automate the drudgery of wading through the audit jungle. Audit trails are particularly useful because they can be used to establish guilt of attackers, and they are often the only way to detect unauthorized but subversive user activity. The main goal of effective IDS is to provide high rates of attack detection with very small rates of false alarms[15]. Here, IDS is simply categorized along two dimensions: Intrusion detection approach – Misuse or anomaly detection and protected system – host or network based.

4. Intrusion Detection Datasets

The development of intrusion detection system has been hampered due to lack of a common metric to gauge the performance of current systems. Evaluation has really helped to solve this problem in other developing technologies and have guided research by identifying the strengths and weaknesses of alternate approaches. Ideally IDS should be evaluated on a real network and tested with real attacks. However, it is difficult to repeat such test so that other researchers can replicate the evaluation. In doing this, network traffic would have to be captured and reused. This raises the issue of privacy because sensitive information such as email messages and passwords can be contained in real traffic[16].

The KDD Cup 1999[17] used for benchmarking intrusion detection problems is used in our experiment. The dataset was a collection of simulated raw TCP dump data over a period of nine weeks on a local area network. The training data was processed to about five million connection records from seven weeks of network traffic and two weeks of testing data yielded around two million connection records. The training data is made up of 22 different attacks out of the 39 present in the test data. The attacks types are grouped into four categories: DOS, Probe, R2L and U2R, since our focus is not to detect each attack type but the major category into which each falls. Table 1 gives the different attack types contained in the datasets.

Table 1. Different attack types for both training (known) and the additional attack types included for testing (novel) for the four categories Known and novel attack types

DOS	Probe	R2L	U2R
Known (attacks in training dataset)			
Back, land, Neptune, Pod, smurf, teardrop	ipsweepsata n, nmap, portsw-eep	ftp_write, guess_passwd, warezmaster warezclient, imap, phf, spy, multihop	rootkit, loadmodule, buffer_overflow, perl
Novel (additional attacks in test dataset)			
apache2, udpstorm, processtable, mailbomb	Saint, mscan	named, xlock, sendmail, xsnoop, worm, snmpgetattack, snmpguess	xterm,p.s., sqlattack, httptunnel

5. The Immunological Model

The model environment is defined over a universal set U, where U is a finite set of finite patterns and is partitioned into two sets, S and N, called self and nonself, respectively, such that $S \cup N = U$ and $S \cap N = \emptyset$. Self patterns represent acceptable or legitimate events, and nonself patterns represent unacceptable or illegitimate events.

A pattern $S \in$U is *normal* if it is in the memory, and is *anomalous* otherwise, that is,

$$\int(\mathrm{M},\mathrm{s}) = \begin{cases} normal & ifs \varepsilon m \\ anomalous & otherwise \end{cases} \quad (1)$$

where $\int$ is a binary classification function and M is a set of patterns drawn from U representing the memory of the detection system, $M \subset U$.

Basic Assumptions

In this work some of the assumptions proposed by[8] was adopted and used in building the system. All of the assumptions are justified below:

i. U is closed and finite. For any given problem domain, patterns must be represented in some fashion. A fixed size representation is used, and any fixed size representation implies a finite and closed universe.

ii. $S \cup N = U$ and $S \cap N = \phi$. If there are cases in which this assumption does not hold, which means that there will be patterns that are both self and nonself. It will be impossible for *any* detection system to correctly classify such *ambiguous* patterns, and so they will always cause errors.

iii. Every location has sufficient memory capacity to encode or represent any pattern drawn from U. Any location that has insufficient memory capacity to encode even a single pattern would be useless, and can be disregarded. If there is a subset of locations for which this assumption holds, then the analysis applies to those locations.

5.1. The Detection System

There are two separate, sequential phases of operation to the system: the first phase is called the training phase and the second is called the test phase. During the training phase, the detection system, D, has access to a training set, U_{trn}, which can be used to initialize or modify the memory of D. During the test phase, the detection system at each location l, attempts to classify the elements of an independent test set, $U_l \subseteq U$, with subsets $N_l \subset N$ and $S_l \subset S$, such that $N_l \cup S_l = U_l$. The performance of the detection system in terms of classification accuracy are measured during the test phase.

In real life situations data sets are made of discrete and continuous variables. In line with this Entropy, a supervised discretization technique is used in discretizing continuous attributes in data set. After, instances of redundant records were removed from the training data set; the classification model was obtained by matching the patterns of both self and nonself in other to obtain the signature patterns of nonself.

5.2. Entropy Based Discretization Technique

Entropy, a supervised splitting technique used to determine how informative a particular input attribute is about the output attribute for a subset, is calculated on the basis of the class label. It is characterized by finding the split with the maximal information gain[20]. It is simply computed thus:

Let D be a set of training data set defined by a set of attributes with their corresponding labels
The Entropy for D is defined as:

$$Entropy(D) = -\sum_{i=1}^{m} P_i \log_2(P_i) \quad (2)$$

where P_i is the probability of C_i in D, determined by dividing the number of tuples of C_i in D by |D|, the total number of tuples in D.

Given a set of samples D, if D is partitioned into two intervals D_1 and D_2 using boundary T, the entropy after partitioning is

$$E(D,T) = \frac{|D_1|}{D} Ent(D_1) + \frac{|D_2|}{D} Ent(D_2) \quad (3)$$

where | | denotes cardinality. The boundaries T are chosen from the midpoints of the attributes values Information gain of the split,

$$\text{Gain (D,T)} = \text{Entropy(D)} - \text{E(D,T)}. \quad (4)$$

In selecting a spilt-point for attribute A, pick an attribute value that gives the minimum information required which is obtained when E(D,T) is minimal. This process is performed recursively on an attribute the information requirement is less than a small threshold (0).

$$Ent(S) - E(T,S) > \delta \quad (5)$$

6. Generation of Nonself Patterns in Network Traffic

Adapting the concept proposed by[7]. The algorithm has two phases:

1. Training phase: the censoring stage is a stage to generate a set of detectors (D). Each detector is a string or pattern that distinctly recognizes nonself depicted in Fig. 1.

2. Testing Phase: The monitoring stage determines the performance of the proposed approach as depicted in Figure 2. If we view the set of data being protected (self) as a set of string over finite alphabet, we are proposing to generate detectors for all string not in the protected data set. Figures 1 and 2 depict the negative selection (Nonself) process.

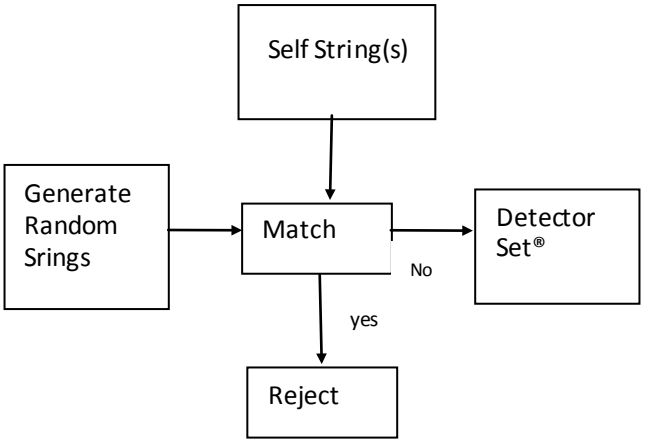

Figure 1. Generation of Valid Detector Set (Censoring) (culled from[7])

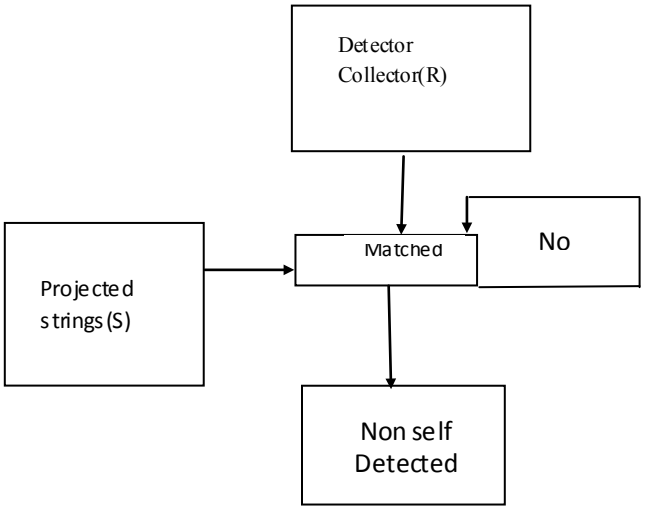

Figure 2. The monitoring stage

[7] developed a negative- selection algorithm for change detection based on the principles of self/non-self discrimination in a computer that is based on the ways that natural immune system distinguished self from non-self.

It is summarized as:

-Define self as a collection *S* of strings of length *l* over a finite alphabeth.

-Generate a set *R* of detectors, each of which fails to match any string in S.

-Monitor *S* for changes by continually matching the detectors in R against S.

To generate valid detectors, the self strings are splitted into equal-size segments.

For instance; breaking the following 32-bit string into eight substrings, each of length four:

(i) Discretization of the continuous variables in the training datasets made of self (normal) nonself (combination of various attacks in the dataset);

(ii) Seperate the discretized datasets into two distinct groups (self and nonself)

(iii) Divide each tuple in the discretized data of each group into eight (8) – the first 7 groups contains 5 strings each while the last contains 6 totalling forty-one which is the number of attributes in the dataset.

Redundant records of strings generated
were removed for each class

(vi) Determine the Detector R by comparing nonself strings generated (R) against the self strings (S) as depicted in Figure 2.

Figure 3.1a. The Training Phase for[7] approach

0010 1000 1001 0000 0100 0010 1001 0011 produces the collection *S* of self (sub) strings to be protected (S contains

all of the substrings). The second step is to generate random strings (call this collection R_0), and then match the strings of R_0 against the strings in *S*. Strings from R_0 that match self are eliminated. Strings that do generate random strings (call this collection R_0), and then match the strings of R_0 against the strings in *S*. Strings from R_0 that match self are eliminated. Strings that do not match any of the strings in S become members of the detector collection (R), also called the repertoire. Suppose R_0 contains the following four random strings: 0111, 1000, 0101, 1001. Then, R will consist of two strings, 0111 and 0101, the strings 1000 and 1001 being eliminated because they each match a string in *S*. Once a collection R of detector strings has been produced, the state of self can be monitored by continually matching strings in *S* against strings in R. Figures 3.1a and 3.1b depict the algorithm adopted in implementing[7] approach for network traffic analysis:

For the testing, the following approaches are adopted:

i. Each in coming traffic is subjected to cut Points btained from discretization technique described in section 3.2

ii. Divides each coming network traffic tuple into groups of eight as described in Fig. 3.1a

iii. Compare the strings obtained with the Detector R, if any matches then network traffic is regarded as nonself else regards traffic as self.

Figure 3.1b. The testing algorithm

7. The Proposed Model

Our proposed model was a modified version of[7], which exclude generation of strings into groups which are computationally intensive. Rather our proposed method depends majorly on frequency distribution of attribute values with regards to the class group to generate nonself signature as spelt out in[18]. Examples of intrusions in Table 2 are used in illustating the working principles of the method.

Table 2. Example of intrusion data

Object	Protocol	Service	Urgent	Category
0	Udp	Private	16	Nonself
1	Tcp	http	23	Nonself
2	Tcp	ftp	20	Self
3	Tcp	http	17	Self
4	Icmp	ecr_i	25	Nonself
5	Udp	domain_u	5	Self
6	Udp	Domain	5	Nonself
7	Udp	Private	6	Nonself
8	Icmp	ecr_i	5	Nonself
9	Tcp	Smtp	20	Self
10	Udp	Private	24	Self

Table.2. shows extraction of suitable features representing network connections based on the knowledge about the characteristics that distinguished self from nonself connections. It consists of three conditional features (protocol, services and urgent), one decision feature (class) and 10 objects.

The features are related to the network characteristics of the connection extracted from the TCP/IP headers of packets, which can be divided into two: intrinsic features, i.e. characteristics related to the current connection, and traffic features, related to a number of similar connections.

Table 3 which is an extract from table 1.0 shows the frequency distribution of attribute values of feature protocol. From this table, one could see that attribute value ICMP clearly identify nonself because self has 0 value and nonself has 2. Hence, ICMP becomes a signature pattern.

Table 3. Frequency distribution of the feature -protocol

Protocol	Self	Nonself
Udp	2	3
Tcp	3	1
Icmp	0	2

Also, Table 4 shows the frequency distribution of attribute values of feature service. From Table 4, one could see that attribute values ecr_i, domain, identify nonself, because self has 0s values, while nonself has 2 and 1 respectively. Hence, ecr_i, and domain have become a signature pattern for nonself.

Table 4. frequency distribution of the feature –service

Service	Self	Nonself
Private	1	2
http	1	1
ftp	1	0
ecr_i	0	2
Domain_u	1	0
Domain	0	1
Smtp	1	0

8. Experimental Setup and Results

The feasibility of this approach was demonstrated on the KDD '99 cup intrusion detection benchmark dataset earlier discussed. A total of 310,782 records were used for the experiment out of which 186,472 records randomly selected form the training dataset constituting 60.06% of the entire records used for experimental purpose; while the remaining 124,312 (39.94%) records carefully selected in the test dataset made up of all the attack types present.

All the attack types earlier mentioned are simply grouped as nonself for the purpose of this work while category normal is simply renamed self. Preprocessing is grouped into three steps. In the first step, categorical features like protocol_type (3 different symbols tcp, udp,icmp), Service (66 different symbols), and flag (11 different symbols) were mapped to integer values ranging from 1 to N where N is the total number of symbol variation in each feature. In the second

step, continuous-value attributes like duration, src_bytes, dst_bytes are standardized based on entropy earlier discussed. Appendix 1 shows the cutoff points of entropy on continuous attributes and the mapping obtained on the discretized dataset.

After preprocessing in our approach, instances of duplicated records were removed from the training dataset. A total of 4264 records set made up of 3188 self and 1076 nonself were actually used for training and in obtaining the signature patterns of nonself. While for the Stephanie approach, the entire dataset was used.

8.1. Result Discussions

Nonself signature patterns are obtained based on comparison of features in each network connection with the class label self with that of nonself. Results are presented in terms of variation(s) per attribute that achieved good levels of discrimination of self from nonself. This clearly distinguished a particular class label in the training data set. This can easily be achieved by generating the frequency of each variation per attribute against each class – self and nonself. Table 5 shows the signature pattern of non self obtained from the training dataset and a total of 12 attributes out of 41 presented for training are chosen.

Table 5. The signature pattern of nonself obtained from the training set

s/n	Column selected	Variations (x)	Variation(x) Translation
1	1: duration	4; and 6	$585 < x \leq 712$ $717.5 < x \leq 899.5$
2	4: flag	8 9	RSTO
3	5: src_bytes	16 23 25 6	$19.5 < x \leq 27.5$ $1031 < x \leq 1033.5$ $49080 < x \leq 132704$ $x > 882177$
4	6: dst_bytes	7	$142.5 < x \leq 144$
5	11: hot	1	$X > 2.5$
6	18: num_shells	1	$X > 0.5$
7	21: is_host_login		$X > 1$
8.	24: srv_count	25	$X > 419$
9	26: srv_serror_rate	1	$0.00499916 < x \leq 0.0149975$
10	30: diff_srv_rate	2	$0.0349961 < x \leq 0.104988$
11	37: dst_host_srv_diff_host_rate	2	$0.504944 < x \leq 0.634949$
12	‘	11	$0.824952 < x \leq 0.939942$

Figures 4, 5, and 6 show the dependency on the variations of attributes 4, 5 and 6 to buttress or ratified the content of table 5. The pattern in figure 4 reveals nonself at variation 8 and 9 of attribute 4. The frequencies of self at those variations are 0 while nonself is 276 and 92 respectively. These patterns can simply be presented using if .. then statements. Tables 6 and 7 show the confusion matrix obtained using the obtained signature of nonself in Table 5 on the training and test datasets. The computed accuracy obtained on the training and testing are satisfactory and thus shows that it is a promising approach. The training phase of the approach in[7] resulted in six hundred and one (601) Detectors R. Results obtained after matching the training and test datasets on Detector R are depicted in Table 8 and 9 respectively.

The performance measures or accuracy is computed thus:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

where

a. True Positives (TP), the number of self correctly classified as self

a. True Negatives (TN), the number of nonself correctly classified as nonself

b. False Positives (FP), the number of self falsely classified as nonself

c. False Negative (FN), the number of nonself falsely classified as self

Table 6. Classification obtained from Training dataset

Predicted as actual	Self	Nonself
Self (36354)	36354(100.00%)	0 (0.00%)
Nonself (150116)	5474 (3.65%)	144642(96.35%)

$$Accuracy = \frac{36354 + 144642}{36354 + 144642 + 0 + 5474}$$
$$= \frac{180996}{186470}$$
$$= 0.9706 = 97.06\%$$

Table 7. Classification obtained from Test dataset

Predicted as actual	Self	Nonself
Self (24235)	24198(99.85%)	37 (0.15%)
Nonself (100077)	16881(33.17%)	83196(66.93%)

$$Accuracy = \frac{24198 + 83196}{24198 + 16881 + 83196 + 37}$$
$$= \frac{107394}{124312} = 86.39\%$$

Table 8. Classification obtained from Training dataset

Predicted as actual	Self	Nonself
Self (36354)	31798(87.47%)	4556 (12.53%)
Nonself (150116)	14743 (9.82%)	135373(90.18%)

$$Accuracy = \frac{31798 + 135373}{31798 + 135373 + 4556 + 14743}$$
$$= \frac{167171}{186470} = 0.8965 = 89.65\%$$

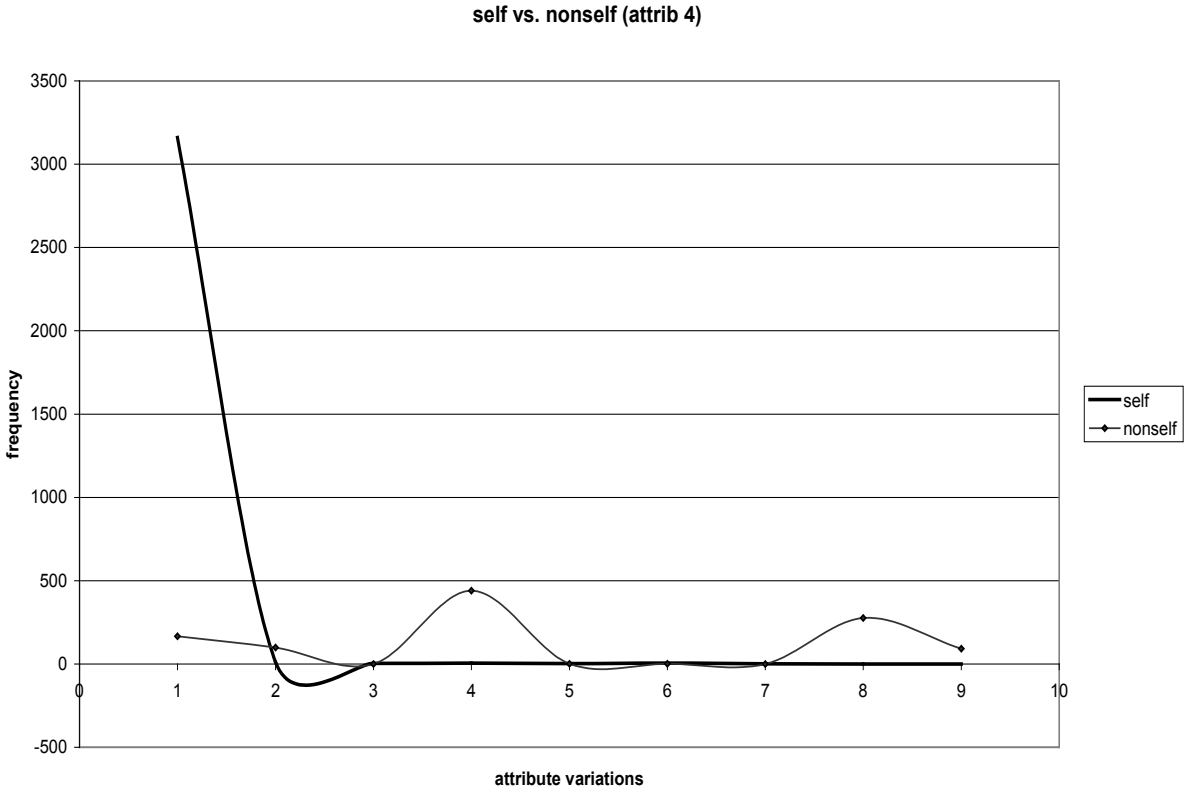

Figure 4. Attribute 4 variation dependency of self and nonself

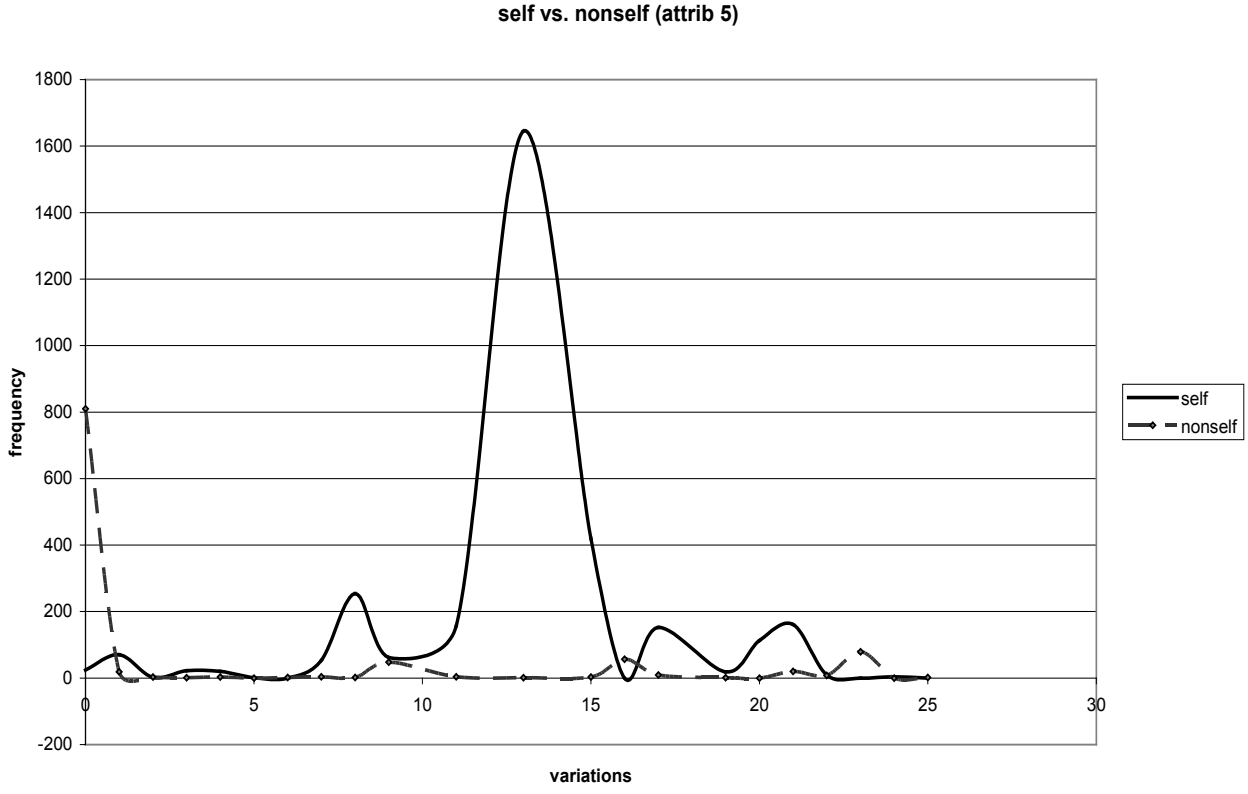

Figure 5. Attribute 5 variation dependency of self and nonself

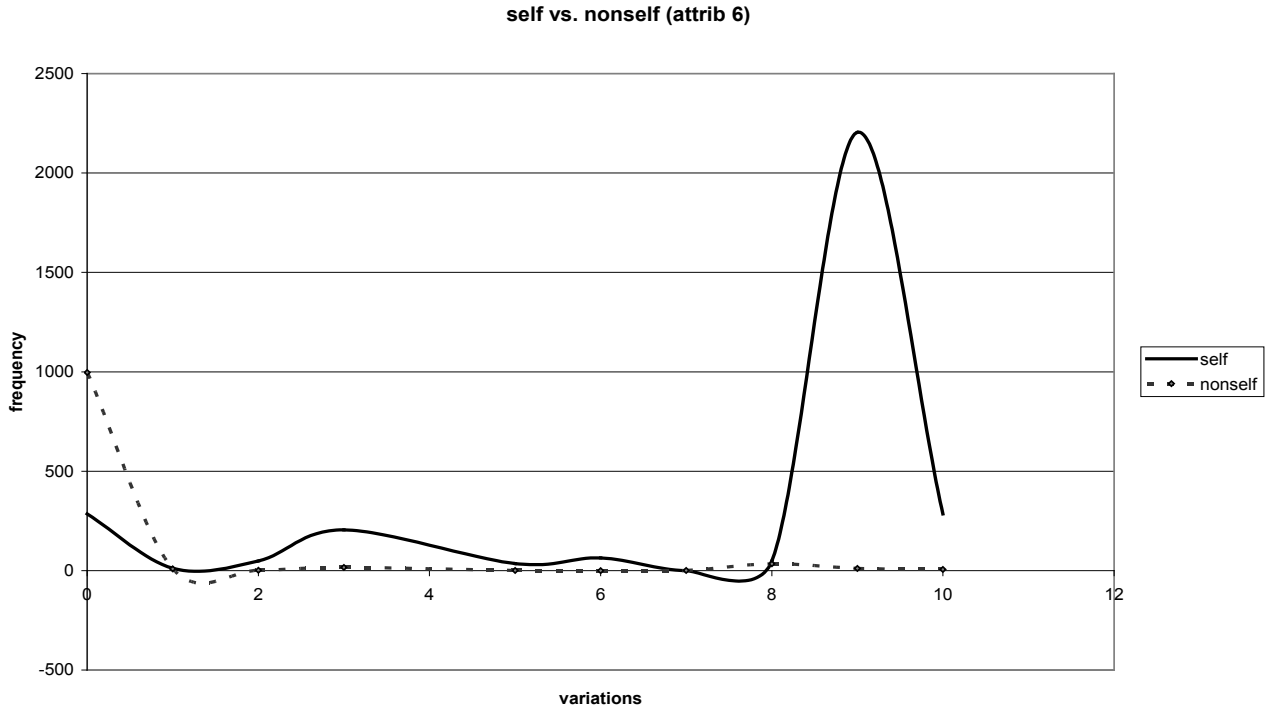

Figure 6. Attribute 6 variation dependency of self and nonself

Table 9. Classification obtained from Test dataset

Predicted as actual	Self	Nonself
Self (24235)	21074(86.95%)	3161 (13.05%)
Nonself (100077)	14614(%)	85463(85.40%)

$$Accuracy = \frac{21074 + 85463}{21074 + 85463 + 3161 + 14614}$$

$$= \frac{106537}{124312} = 86.39\%$$

[7] approach is computational intensive during testing which does not make it appropriate for practical use because it has to compare the newly generated eights strings with the six hundred and one earlier generated Detector-R in its repertoire. The best matching that could be obtained is 1 while in a worst case it has to carry out exhaustive comparison of 4808 matches. The probabilistic approach of this technique was not evaluated as mathematical analysis shows that it is more computationally expensive. The mathematical analysis is computed thus:

Assuming, there are 3 strings defined over the five alphabet (A,B,C,D,E) match at three contiguous locations. The number of three contiguous strings that could be obtained in a group of five alphabets = (number of strings in a group) – (number of contiguous strings) + 1 = 5-3+1 = 3

Hence number of exhaustive matching for a group
= DetectorR * number of contiguos * 3
= 601 * 3 * 3 = 5,409.

Hence, for the eights groups that make up a network traffic in this case = 601 * 5409 = 3 250, 809. Our proposed model is less computational intensive, simpler and more effective in terms of computational accuracy.

9. Conclusions

The need for effective and efficient security on our system cannot be over-emphasized. This position is strengthened by the degree of human dependency on computer systems and the electronic superhighway (Internet) which grows in size and complexity on daily basis for business transactions, source of information or research. This technique based on immune system for discriminating network traffic was implemented on Intel Pentium(R) 4, CPU 2.66GHz, 512 MB RAM using C++ programming language.

From the experiment, IMNST performances outweights that of Stephanie on both the training and testing sets as her accuracy stood at 97.06% and 88.06%, against 89.65% and 85.70% respectively. The immune algorithm proposed is easier in obtaining effective signature patterns for classifying network traffic. This method could as well be employed in obtaining virus signatures and in other classifying problems. The results of the developed tools are satisfactory though it can be improved upon. These tools will go a long way in alleviating the problems of security of data on computing systems.

Appendix 1: Cutoff Points Obtained on Continuous Features

cut_point1[9] =
{0.5,2,132.5,585,712,717.5,899.5,1100.5,4490.5,};

cut_point5[25] =
{0.5,3,5.5,9.00001,18,19.5,27.5,36.5,103,105,106,168,168.5,342,342.5,1031,1033.5,1480,1480.5,1563,2438,16050,49080,132704,882177,};
cut_point6[10] =
{0.5,2.5,35.5,105,106,114.5,142.5,144,148.5,15235,};
cut_point8[1] = {0.5,};
cut_point9[0] = {};
cut_point10[2] = {0.5,2,};
cut_point11[1] = {2.5,};
cut_point13[1] = {0.5,};
cut_point14[1] = {0.5,};
cut_point15[1] = {0.5,};
cut_point16[1] = {0.5,};
cut_point17[2] = {0.5,1.5,};
cut_point18[1] = {0.5,};
cut_point19[2] = {0.5,1.5,};
cut_point20[0] = {};
cut_point23[35] =
{0.5,2.5,3.5,8.50001,9.50001,21.5,26.5,34.5,37.5,38.5,39.5,41.5,42.5,43.5,44.5,45.5,46.5,47.5,48.5,53.5,54.5,66.5001,82.5001,85.5001,105.5,150.5,160.5,198.5,208.5,300.5,408.5,413.5,484.5,485.5,509.5,};
cut_point24[25] =
{0.5,1.5,3.5,5.5,8.50001,9.50001,13.5,16.5,18.5,19.5,20.5,25.5,28.5,40.5,54.5,61.5,65.5001,84.5001,102.5,112.5,153.5,155.5,162.5,206.5,419,};
cut_point25[18] =
{0.00499916,0.0149975,0.0249977,0.0349961,0.0449944,0.0549965,0.0649949,0.0749894,0.0849915,0.0949937,0.11499,0.144989,0.154984,0.164978,0.284973,0.954957,0.974976,0.994996,};
cut_point26[7] =
{0.00499916,0.0149975,0.0249977,0.0749894,0.149994,0.254975,0.974976,};
cut_point27[3] = {0.0499955,0.494995,0.994996,};
cut_point28[2] = {0.0249977,0.989991,};
cut_point29[6] =
{0.0349961,0.204987,0.219986,0.514954,0.909913,1,};
cut_point30[11] =
{0.00499916,0.0349961,0.104988,0.119995,0.144989,0.164978,0.204987,0.354981,0.464966,0.684937,1,};
cut_point31[6] =
{0.00499916,0.0149975,0.0249977,0.669922,0.709961,1,};
cut_point32[6] = {0.5,1.5,7.50001,23,218,255,};
cut_point33[13] =
{0.5,1.5,2.5,6.5,16.5,20.5,101,101.5,251,251.5,252.5,253.5,254.5,};
cut_point34[0] = {};
cut_point35[18] =
{0.00499916,0.0149975,0.0249977,0.0349961,0.0449944,0.0549965,0.0649949,0.0749894,0.0849915,0.0949937,0.11499,0.144989,0.154984,0.164978,0.284973,0.954957,0.974976,0.994996,};
cut_point36[9] =
{0.00499916,0.0149975,0.0549965,0.279968,0.294983,0.594971,0.964966,0.984986,0.994996,};
cut_point37[3] = {0.00499916,0.504944,0.634949,};
cut_point38[7] =
{0.00499916,0.0149975,0.0249977,0.0349961,0.0449944,0.0949937,0.994996,};
cut_point40[15] =
{0.00499916,0.0149975,0.0349961,0.0849915,0.124985,0.164978,0.414978,0.464966,0.704957,0.799927,0.814942,0.864991,0.884888,0.974976,0.984986,};
cut_point41[12] =
{0.00499916,0.0249977,0.0449944,0.0549965,0.0849915,0.11499,0.124985,0.454956,0.709961,0.749939,0.824952,0.939942,};

REFERENCES

[1] G. Meade, Department of Defense Trusted Computer System Evaluation Criteria, National Computer Security Service Centre, 1985. csfc.nist.gov/publications/history/dod85.pdf accessed February 2013

[2] S. Garfinkel, G. Spafford. Practical UNIX and Internet Security, 2nd Edition. O'Reilly and Associates, Inc. 1996

[3] C. Janeway, P. Traves, Immunobilogy, The Immune System in Health and Disease, 2nd Edition, Garland Science, New York, 1996

[4] C. Rui, T. Ying, A Virus Detection System Based on Artificial Immune System, International Conference on Computational Intelligence and Security, China. www.cii.pku.edu.cn/publication, 2009.

[5] N. Liu, D. Wang, X. Huang, S. Liu, K. Zhao, Network Security Situation Awareness\Technology based on Artificial Immunity System. International Forum on Information Technology and Applications, 2009.

[6] C.A. Janeway, P. Travers, M. Walport, M.J. Shilomchik, Immunology: The Immune System in Heath and desease, 5th Edition, New York: Garland, 2001.

[7] F. Stephanie, S.P. Alan, A. Lawrence, C. Rajesh, Self-Nonself Discrimination in a Computer. Proc of IEEE Symposium on Research in Security and Privacy. Oakland: IEEE Press, 1994. pp. 202 - 212.

[8] S.A. Hofmey, An Immunological Model of Distributed Detection and its Application to Computer Security, PhD Dissertation, University of New Mexico, 1999.

[9] S. Forrest, T.A. Longstaff, "A Sense of Self for Unix processes", Proceedings of IEEE Symposium on Computer Security and Privacy, Los Alamos, CA, 1996, pp.120-128.

[10] S. Bellovin, Defending against sequence number attacks internet engineering task force, May RFC 1948. number attacks internet engineering task force, May, RFC 1948, 1996

[11] L. Wenke, A data Mining Framework for Constructing Features and Models for Intrusion Detection Systems. PhD

dissertation, Columbia University, USA http://www.cc.gatech.edu/~wenke, 1999.

[12] S. Northcutt, J. Novak, Network Intrusion Detection: An Analyst's Handbook, Second Edition, New Riders Publishers, USA, 2001.

[13] S. Kumar, Classification and Detection of Computer Intrusions. PhD Dissertation, Department of Computer Science, Purdue University, 1995.

[14] H. Debar, What is behavior based intrusion detection? IBM Zurich Research Laboratory, www.sans.org/privacy.php, 2003

[15] S. Axelsson, Intrusion Detection Systems: A survey and Taxonomy, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden. Technical Report TR-99-15,2000.

[16] M. V. Mahoney, A machine Learning Approach to Detecting Attacks by Identifying Anomalies in Network Traffic, College of Engineering at Florida Institute of Technology, USA, PhD Dissertation, 2003.

[17] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup

[18] A.O. Adetunmbi, S.O. Adeola, O.A. Daramola Relevance Features Selection for Intrusion Detection, Intelligent, Automation and System Engineering, Lecture Notes in Electrical Engineering, *(Boston Springer*), vol. 103, 2011, pp. 407 – 418.

[19] A.O. Adetunmbi, S.O. Falaki, O.S. Adewale, B.K. Alese, Intrusion Detection based on rough Set and k-Nearest Neighbour, *International Journal of Computing and ICT Research*, vol. 2 No. 1, 2008, pp. 60-66.

[20] H. Jiawei, K. Micheline, Data Mining: Concepts and Techniques, Second Edition, Elsevier Inc., 2006.