Improving Worm Detection with Artificial Neural Networks through Feature Selection and Temporal Analysis Techniques

Dima Stopel, Zvi Boger, Robert Moskovitch, Yuval Shahar, and Yuval Elovici

Abstract—**Computer worm detection is commonly performed by antivirus software tools that rely on prior explicit knowledge of the worm's code (detection based on code signatures). We present an approach for detection of the presence of computer worms based on Artificial Neural Networks (ANN) using the computer's behavioral measures. Identification of significant features, which describe the activity of a worm within a host, is commonly acquired from security experts. We suggest acquiring these features by applying feature selection methods. We compare three different feature selection techniques for the dimensionality reduction and identification of the most prominent features to capture efficiently the computer behavior in the context of worm activity. Additionally, we explore three different temporal representation techniques for the most prominent features. In order to evaluate the different techniques, several computers were infected with five different worms and 323 different features of the infected computers were measured. We evaluated each technique by preprocessing the dataset according to each one and training the ANN model with the preprocessed data. We then evaluated the ability of the model to detect the presence of a new computer worm, in particular, during heavy user activity on the infected computers.**

***Keywords*—**Artificial Neural Networks, Feature Selection, Temporal Analysis, Worm Detection.

I. Introduction

THE detection of malicious code transmitted over computer networks has been substantially researched during the past years. Commonly, the term "malicious code" (malcode) refers to different types of codes, such as executables or scripts, which contain some code having a malicious purpose. One type of malcode is *worms*, which actively propagate, exploiting vulnerability in the operating system, through communication protocols. Other types of malcode are *viruses*, which inject their code into an innocent file (a host) and are activated whenever the file is executed. Unlike *worms*, *viruses* require user intervention to propagate. Other recently disseminated malicious codes include *Trojans*, which are computer programs that have a useful functionality but also have some hidden malicious goal, and *backdoors*, which enable remote access and control with the aim of gaining full or partial access to the infected system.

Nowadays, *known* malcodes are mainly detected and removed by *antiviruses*. Antiviruses search the executables for known patterns, also called *signatures*. Antiviruses are helpless when facing an *unknown* malicious executable. After the appearance of the *new worm*, a new signature is created by the antivirus system company. The antivirus signature base is periodically updated. However, since worms spread rapidly, the signature update action is often taken too late, and expensive damage may already have been done by the *worm*. *Trojans* and *backdoors* have other malicious motivations, such as *economic*, *terrorist*, or *criminal*. They are commonly installed on a relatively small number of hosts, and thus do not attract the attention of *antivirus* systems and remain undetected. For example, a student can demonstrate a problem he encountered in his homework using a *portable memory device* on his/her professor's computer. During this demonstration the student installs a *backdoor* that allows him to access the professor's computer and search for a copy of an upcoming examination.

A recent survey of intrusion detection [1] suggests using artificial intelligence (AI) techniques to recognize *malicious software* (malware) in single computers and in computer networks. It describes the research done in developing these AI techniques, and discusses their advantages and limitations. One of the critical requirements of such an AI technique is that it operates efficiently in real-time.

One of the AI techniques mentioned in this survey is Artificial Neural Networks (ANN). Other research studies referenced in the survey used the Self Organizing Map (SOM) method [2]-[4], the main challenge of which as reported by the authors was overcoming the high dimensional inputs. Linear techniques, such as Principle Component Analysis (PCA), Singular Value Decomposition or Support Vector

Manuscript received August 30, 2006.

This work was supported by Deutsche Telekom Co.

Dima Stopel (corresponding author) is an M.Sc. student, Deutsche Telekom Laboratories at Ben-Gurion University, Be'er Sheva, 84105 Israel. Phone: +972-54-5959320; email: stopel@cs.bgu.ac.il.

Zvi Boger, president of OPTIMAL – Industrial Neural Systems Ltd., Be'er Sheva, 84243 Israel; email: zboger@bgu.ac.il

Robert Moskovitch, Ph.D. student, Deutsche Telekom Laboratories at Ben-Gurion University, Be'er Sheva, 84105 Israel; email: robertmo@bgu.ac.il

Yuval Shahar, Deutsche Telekom Laboratories at Ben-Gurion University, Head of the Department of Information Systems Engineering, Ben-Gurion University of the Negev, Be'er Sheva, 84105 Israel; email: yshahar@bgu.ac.il

Yuval Elovici, Deutsche Telekom Laboratories at Ben-Gurion University, Head of the Software Engineering program, Ben-Gurion University of the Negev, Be'er Sheva, 84105 Israel; email: elovici@inter.net.il.

Machine [5], [6] are used to reduce the input dimensionality and may result in a less accurate modeling.

Other aspects discussed in the survey [1] were the detection of an intrusion or a malware presence by analyzing executable files on local storage devices [7], analyzing the content of the packets sent or received by the computer [8,2], or looking at the system calls that were invoked by processes running on the system [9].

In this study a different approach is suggested. The detection of the presence of a malware in a computer is performed by analyzing the overall computer *behavior*. We define the computer *behavior* by a variety of different values which can be measured in the computer while it is operating.

The main ANN advantages are a high level of accuracy in real-time operation, low CPU resources utilization during the classification phase, and the ability to generalize, in order to detect and identify, any previously unseen classes. The fact that worms propagate very fast on networks makes them one of the most challenging malwares. Fast detection of computers infected with worms is critically important on local networks. For these reasons we propose employing ANN for the detection of worm activity in real time.

It was already shown in our previous work that the detection of new worms using ANN techniques is possible and effective [10]. However, continuous monitoring of large number of measured features may demand a significant part of the computational resources. In this work we reduce the number of features significantly by using various *feature selection* techniques, while increasing the detection accuracy. We use the Causal Indices (CI) technique to estimate the contribution of each input feature to each classification output. Additionally, we evaluate the effect of several temporal analysis preprocessing techniques on the detection problem.

The structure of this paper is as follows: In section II the classification, feature selection and temporal analysis methods are described. In section III we describe the creation of the datasets we used in the study. In section IV we present the techniques we used to evaluate the different methods, and we describe the CI technique in this section. Section V describes the experimental sequence and the purpose of each particular experiment. In section VI we present the results of the experiments. In section VII we discuss the experiment results and conclude with recommendations for further research.

II. Classification, Feature Selection and Temporal Analysis methods

A. Classification method

We used a typical feed forward neural network, together with the Levenberg-Marquardt training method [11]. The number of hidden neurons was set empirically to *six*.

B. Feature selection techniques

Generally, there are two different approaches to feature selection: the *wrapper* and the *filter approach*. The *wrapper* approach searches for the optimal *subset* of features of a given dataset, for a specific classification algorithm. The main drawback of this approach is the relatively long computation time. The *filter* approach ranks the features according to a certain measure independent of any classification algorithm. Thus, after the calculation of ranks, one can use any subset of features based on their ranks.

For features selection we used three different *filter* techniques. These techniques are described below.

1) The relation between the inputs and the hidden neuron's relative variance

This knowledge extraction and dimensionality reduction technique involves ranking the inputs according to their relevance to the ANN prediction accuracy. It is based on the observation that in a trained ANN model a less relevant input contributes a small proportion of the variance in the hidden layer neuron's activities. This may be the result of either the small relative variance of the input value or the small final connection weights to *all* hidden neurons assigned to this input by the trained ANN.

The contribution of an input *i* to the total variance of the hidden layer inputs is presented in (1). The $(W_H)_i^T$ is the *i*'th row of the transpose of W_H, i.e., the *i*'th column of the *input-to-hidden* connection weights expressed as a row vector; *R* is the covariance matrix for the network inputs *x*, estimated from the training set.

$$(V_I)_i = (W_H)_i^T W_H R_i^T \tag{1}$$

The relative contribution of an input *i* to the variance of the hidden layer inputs is calculated using (2). In (2) *j* is the index of each one of the *n* hidden neurons.

$$(V_I)_i^{rel} = \frac{(V_I)_i}{\sum_{j=1}^{n}(V_I)_j} \tag{2}$$

The less contributing inputs can be discarded and the ANN re-trained with the reduced input set, often yielding better prediction accuracy. This technique is described in detail in [12].

2) The Fisher score ranking.

The Fisher score ranking technique calculates the difference, described in terms of mean and standard deviation, between the positive and negative examples relative to a certain feature. Equation (3) defines the Fisher score, in which R_i is the rank of feature *i,* describing the proportion of the substitution of the mean of the feature *i* values in the positive examples (*p*) and the negative examples (*n*), and the sum of the standard deviation. The bigger the R_i, the bigger the difference between the values of positive and negative examples relative to feature *i;* thus, this feature is more *important* for separating the positive and negative examples. This technique is described in detail in [13].

$$R_i = \frac{\left|\mu_{i,p} - \mu_{i,n}\right|}{\sigma_{i,p} + \sigma_{i,n}} \quad (3)$$

3) *Gain Ratio filter*

The Gain Ratio measure is based on the Information Gain (IG) measure, which is based on measuring the relative entropy reduction. This method requires discretization to be applied to the continuous data in advance. Equation (4) defines the classic *entropy* measure, in which S is the entire dataset, C is the class attribute, and S_c is a subset of S in which the value of C is c.

$$E(S) = \sum_{c \in C} -\frac{|S_c|}{|S|} \log_2 \frac{|S_c|}{|S|} \quad (4)$$

Equation (5) defines the IG rank. IG shows how much information we gain by splitting the dataset relative to attribute A. In this equation, $V(A)$ is the set of unique values of attribute A, and S_v is the subset of S in which the value of attribute A is v.

$$IG(S,A) = E(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} E(S_v) \quad (5)$$

The problem of the IG method is that it gives higher ranks to attributes with a large number of unique values, i.e., $V(A)$ is high. Gain Ratio overcomes this bias by using an extra term which represents the way an attribute splits the data. Equation (6) defines this special term and (7) defines the ranking of Gain Ratio.

$$SI(S,A) = -\sum_{v \in V(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|} \quad (6)$$

$$GR(S,A) = \frac{IG(S,A)}{SI(S,A)} \quad (7)$$

When $SI(S,A)$ is zero, it is defined as equal to $IG(S,A)$. Additional information about this technique can be found in [14].

C. *Temporal analysis techniques*

Since worm propagation involves a temporal characteristic, we thought about improving the detection accuracy by considering the temporal dimension. We present three temporal analysis techniques which use different types of *sliding windows*. We evaluated these temporal techniques using the reduced datasets that are based on the most-contributive features selected at the feature selection stage. The challenge here is to represent the temporal dimension authentically while using a static classification algorithm. All three techniques are described below.

1) *Simple sliding window*

This technique proposes constructing a single window from the given dataset. Given a window size k, the new instance will include k original sequential instances. For example, with a dataset having instances [1,2,3,4,5] and window size k=3 we create *three* instances: [1,2,3], [2,3,4], and [3,4,5], respectively. However, note that the current instance will include k multiplied by n attributes, where n is the number of attributes in an original instance.

2) *Simple exponential compression.*

While being simple, the drawback of the simple sliding window representation is that the number of instances in each window grows linearly with the size of the window (k). Working with many instances could be problematic due to the long computation times. The simple exponential compression (*SEC*) technique overcomes this problem by averaging the "*older*" instances. The number of averaged instances will be 2^n where n is the place in the window starting with zero. For example, with a dataset [1,2,3,4,5,6,7] and a window size of 3 the resulting *one* instance will look as follows: [7, avg(5,6), avg(1,2,3,4)] where 7 is the "youngest" original instance. Generally speaking, each instance represents precisely the current features values, and in a more abstractive and summarized representation, the earlier values. Thus, the number of original instances the window covers grows *exponentially* with the size of the window (k).

3) *Poisson exponential compression.*

We propose a technique that we have named *Poisson exponential compression* (PEC). The PEC technique is similar to SEC in the way that the instances are compressed in an exponential rate. However, this technique does not show precisely the "youngest" instances but the instances with higher Poisson probability density function (PDF). Poisson PDF is presented in (8). k is the index of the instance starting from the "youngest," λ is a constant and was 2 in the experiment, and e is the base of the natural logarithm.

$$R(k) = \frac{\lambda^k}{k!} e^{-\lambda} \quad (8)$$

The motivation for using such a technique was the hypothesis that the most important instance in the window is not necessarily the "youngest." The amount of averaged instances in place n is still 2^n. For example, having a dataset [1,2,3,4,5,6,7] where 7 is the "youngest" instance, we can calculate the array of *ranks*, using the Poisson PDF, which will be [0.01,0.04,0.09,0.18,0.27,0.27,0.14]. Now, given a window size of 3, the resulting *one* instance will look as follows: [6, avg(5,4), avg(1,2,3,7)].

III. Dataset description

Since no public standard dataset was available for this study, we had to create our own dataset. We created a computer network environment consisting of a variety of computers (configurations). We injected worms into the network environment, and monitored various computer features in each of the infected and non-infected computers.

The computer network environment consisted of seven computers, which contained heterogenic hardware, and a server simulating the Internet.

We used the *MS Windows performance tool*, which enables monitoring of system features that appear in these main categories: *Internet Control Message Protocol* (ICMP), *Internet Protocol* (IP), *Memory*, *Network Interface*, *Physical Disk*, *Processes*, *Processor*, *System*, *Transport Control Protocol* (TCP), *Threads*, *User Datagram Protocol* (UDP). We also used *VTrace* [15], a software tool which can be installed on a PC running Windows. VTrace collects traces of the file system, the network, the disk drive, processes, threads, inter-process communication, writable objects, cursor changes, windows, and the keyboard. The Windows performance tool was configured to measure the features every second and store them in a log file as vectors. VTrace stored time-stamped events, which were collected into a second file. Both files were merged and contained a vector of *323* features for every second.

In order to perform the evaluation in a realistic environment, we considered three major aspects: *computer hardware configuration*, *constant background application* requiring high computational resources, and *user activity*. For each dataset related to an aspect the data were collected when each one of the five worms was injected separately, for a constant length of time. In addition there was a time period in which no worm was activated. In the rest of the paper we will refer to this clean period as a state named "Clean." The resulting datasets were combined into a single dataset.

Computer hardware configuration: Both computers ran MS Windows XP, since we considered it to be the most commonly used operating system. There were two configurations: *old*, using a PC based on Pentium III 800 MHz CPU, bus speed 133 MHz and memory 512 Mb, and *new*, using a PC based on Pentium IV 3 GHz CPU, bus speed 800 MHz and memory 1 GB.

Background application activity: There were two configurations: with *background application* activity and without. We ran the *WEKA* mathematical processing application software [16] which mainly effected the following features: *Processor Time* (usage of 100%); *Page Faults/sec*; *Avg Disk Bytes/Transfer*; *Avg Disk Bytes/Write*; *Disk Writes/sec*.

User activity: A user opened several applications, including Internet Explorer, Word, Excel MSN messenger, and Windows Media Player in a scheduled order. There were two configurations: with *user activity* and without. The exact schedule of the user activity is described in Table I.

TABLE I
User activity schedule

Time period	User operations
0-5 minutes	- Opening 10 MS Word instances - Downloading two files simultaneously
5-10 minutes	- Opening 5 instances of MS Excel - Generating random number in MS Excel - One file downloading - Listening to internet radio
10-15 minutes	- Opening 12 instances of MS Word - Downloading one file
15-20 minutes	- Opening 9 instances of MS Excel - Generating random numbers in MS Excel - Browsing the internet (using MS IE)

In each time period all the user operations were performed simultaneously.

During the evaluation we used five different *real* worms. The description of each one is presented below.

1. *Dabber.A* (Daber.A)

This worm scans networks for random IP addresses, searching for victim machines that have the ftp component of the Sasser worm installed on port 5554.

When the worm finds a suitable victim machine, it sends a vulnerability exploit to it to infect the system. It then launches the command shell on port 8967. It also installs a backdoor on port 9898 to receive external commands.

2. *W32.Sasser.D* (Sasser.C)

This worm spreads by generating random IP addresses using 128 threads. The IP addresses are generated so that 48% of them should be close to the current computer by using the current computer's IP and 52% of them completely random. It connects to the remote computer using TCP port 445 and if the connection is established, a remote shell is opened. The remote shell is used to connect to the infected computer's FTP server and transfer the worm.

3. *W32.Deborm.Y* (DebormY)

This worm scans the local network and tries to propagate to other computers on the local network. It attempts to share C$ (C drive) using the accounts of the administrator, owner or guest (it succeeds if a certain account does not have a password).

4. *W32.Korgo.X* (PadobotKorgoX)

This worm generates random IP addresses and exploits the LSASS Buffer overrun vulnerability using TCP port 445. If it succeeds in taking over a computer, the newly infected computer will send a request to download the worm from the infecting computer by using a random TCP port.

5. *Slackor.A* (Slackor.A)

When the Slackor worm is run, it sends a SYN TCP packet to randomly generated IP addresses through port 445 to search for the systems using Server Message Block (SMB). It then attempts to connect to the Windows default shares on these systems by using the username and password pair that it carries. If successful, it tries to copy the worm to the system.

IV. Evaluation techniques

A. Evaluation Measures

In order to perform the comparisons of the methods described, we employed the commonly used evaluation measures: *True Positive Rate* (*TPR*), shown in equation (9), *False Positive Rate* (*FPR*), shown in equation (10), and *Accuracy*, shown in equation (11). TP is the number of true positive examples. FP is the number of false positive examples. TN is the number of true negative examples. FN is the number of false negative examples.

$$TPR = TP/(TP + FN) \tag{9}$$

$$FPR = FP/(FP + TN) \tag{10}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

Additionally we used *Receiver Operating Characteristic* (ROC) curves in order to evaluate different methods. An ROC curve is a graphical representation of the trade off between the false negative and false positive rates for every possible cut off. Equivalently, the ROC curve is the representation of the tradeoffs between Sensitivity and Specificity.

B. Causal Indexes analysis

The Causal Indexes (CI) method estimates the influence of each input feature on the classification output. The quasi-quantitative influence of each input on each output is calculated based on the connection weights of a trained ANN [17]. The CI is calculated as the sum of the product of all "pathways" between each input to each output:

$$CI_{ik} = \sum_{j=1}^{n} w_{ij} \cdot w_{jk} \tag{12}$$

Equation (12) presents the basic formula of the CI, from input *i* to output *k*. w_{ij} is the connection weight from input *i* to the hidden neuron *j*. w_{jk} is the connection weight from hidden neuron *j* to output *k*. The product of these is computed for all the *n* hidden neurons. The CI reveals the influence direction (positive or negative) and the relative magnitude of the relationship of any input and any output.

V. Experimental sequence

We performed two types of experiments. First, we evaluated several feature selection methods, as a preprocessing stage. Second, we evaluated several temporal representations.

A. Feature selection experiments

The aim of these experiments was to find the best of the three different feature selection methods: Hidden neurons relative variance based method, Fisher's score, and Gain Ratio. Gain Ratio improved the performance of ANN as a classification method in [18] more than any other feature selection method. In addition to these three feature selection methods, we used a common feature extraction method known as Principal Component Analysis (PCA) for comparison. For each feature selection method we chose *six* different subsets of features ranked by this method: *top5*, *top10*, *top20*, *top30*, *top50*, and *full*. We did the same for PCA by choosing the most significant *principal components*. Each of the sets contained respectively top 5, 10, 20, 30, 50, and 323 features. Thus, the total amount of subsets obtained was *4x6=24*. For each subset we performed *five* different experiments. In each experiment one of the five different worms was removed from the training set. During each experiment we trained the ANN using a dataset constructed from the instances of *four* worms and 80% of the *clean* instances. The test set contained *only* instances of the *fifth* worm and an additional 20% of *clean* instances. Thus, the total number of experiments in this section was *4x6x5=120*. Note that the test sets on which the ANN was tested contained only a *new* worm, which did not appear in the training set.

B. Temporal analysis experiments

After the best feature selection algorithm was identified, we examined the potential of improving the detection performance by using different temporal representation techniques. We used three techniques: Simple sliding window, Simple exponential compression, and Poisson exponential compression, which were explained earlier in section II.C. Using the best feature selection algorithm, we chose the *20* best attributes. We used the reduced dataset to create three different datasets by applying three different temporal representation techniques. With each one of these datasets we performed five experiments, by moving one worm, out of five, from the training set to the test set. As in the feature selection case, we included 20% of *clean* examples in the test set. Thus, the total number of experiments was *3x5=15*. The sliding window size in all experiments was *5*; thus the input vector size was constant and equal to *20x5=100*.

VI. Results

A. Feature selection results

The worm classification results using the feature selection techniques are summarized in Table II. Each cell represents *five* different experiments, one for each missing worm. The values in the cells are the detection accuracy averages of these five experiments. It can be seen that Fisher's score method outperformed the other techniques.

TABLE II

WORM CLASSIFICATION WITH FEATURE SELECTION RESULTS

	Top5	Top10	Top20	Top30	Top50	Full	Avg
R.V.	0.67± 0.35	0.71± 0.35	0.85± 0.14	0.56± 0.31	0.77± 0.23	**0.64± 0.23**	0.70± 0.28
Fisher	**0.90± 0.05**	**0.87± 0.14**	0.84± 0.21	0.80± 0.23	**0.81± 0.19**	0.61± 0.35	**0.81± 0.21**
GR	0.81± 0.24	0.86± 0.23	**0.87± 0.20**	**0.86± 0.20**	0.60± 0.34	0.43± 0.22	0.74± 0.24
PCA	0.57± 0.34	0.71± 0.37	0.56± 0.37	0.72± 0.28	0.74± 0.32	0.63± 0.34	0.66± 0.34

R.V. stands for hidden neurons relative variance method.

Surprisingly, the best accuracy was achieved by using only *five* attributes selected by the Fisher's score method. The average accuracy of *new* worm detection using these attributes is *0.90* as shown in Table II. These five attributes are presented in Table III. They are related to memory management, and number of system context switches.

TABLE III

BEST FIVE FEATURES SELECTED BY FISHER'S SCORE

Attribute №	Attribute name	Fisher's score
1	Perf_MemoryPool_Paged_Allocs	23.80
2	Perf_MemoryCache_Bytes	17.09
3	Thread_Total_Context_Switches_per_sec	15.24
4	SystemContext_Switches_per_sec	14.54
5	Perf_MemorySystem_Driver_Total_Bytes	13.66

In Fig. 1 the averaged ROC curves of each one of four feature selection methods are presented. It is obvious that the separation level of Gain Ratio is significantly lower than that of the other three techniques. However, the separation levels of the hidden neurons Relative Variance technique and Fisher's score technique are very close to the separation level of PCA.

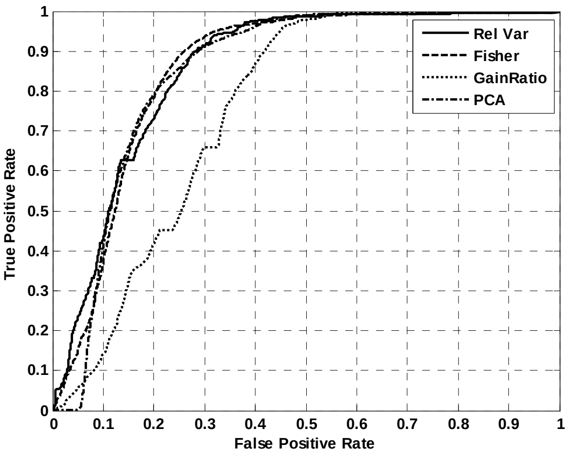

Fig. 1. Averaged ROC curves for four different feature selection techniques.

The areas under the ROC curves are presented in Table IV. Despite the fact that Fisher's score attained the best accuracy value, the Relative Variance method of selecting the reduced feature set has the best separation level.

TABLE IV

THE AREAS UNDER THE ROC CURVES

Relative Var.	Fisher's score	Gain Ratio	PCA
0.90	0.86	0.77	0.86

B. Causal index analysis results

The CIs were calculated from the ANN model that used the five best features selected by the Fisher's score method. The results are presented in Table V where CIs higher than 5 are marked bold and defined *high*, and those lower than -5 are underlined and defined *low*. The attribute numbers are related to these in Table III and the worm numbering is presented in Section III.

TABLE V

CI VALUES OF TOP 5 ATTRIBUTES

Att. №	Clean	Worm № 1	Worm № 2	Worm № 3	Worm № 4	Worm № 5
1	2.82	<u>-7.23</u>	<u>-12.82</u>	<u>-7.41</u>	**9.67**	-4.17
2	**6.44**	**6.86**	**11.51**	<u>-8.99</u>	<u>-6.56</u>	<u>-6.21</u>
3	<u>-17.20</u>	**18.76**	<u>-7.65</u>	<u>-7.70</u>	-3.13	-1.91
4	<u>-15.16</u>	**15.57**	<u>-16.40</u>	<u>-8.30</u>	1.13	-1.39
5	<u>-10.85</u>	-0.38	<u>-14.50</u>	2.62	**6.68**	-3.84

If a certain CI is positive with a relatively high magnitude, then the related input influences the related output in the same direction, i.e., if the input is high the output will rise also. Alternatively, if the CI is negative with a relatively high magnitude, then the related input influences the related output in the opposite direction, and thus if it is high the output will be relatively low.

From the CI's presented in Table V the following "rules" can be formulated:

- If 2 is *high* and ,3,4,5 are *low*, then Clean
- If 2, 3 and 4 are *high* and 1 is *low*, then DabberA
- If 2 is *high* and 1,3,4,5 are *low*, then SasserC
- If 1, 2, 3 and 4 are *low*, then DabormY
- If 1 and 5 are *high* and 2 is *low*, then Padobot
- If 2 is *low* and 1,3,4,5 are *average* then SlakorA

C. Temporal analysis results

The evaluation results of different temporal preprocessing techniques are presented in Table VI. Each cell in this table represents the averaged accuracy of *five* experiments, one for each missing worm. Simple Window showed the best results in this category, achieving an accuracy of *0.85*. As already mentioned, we took 20 best attributes selected by Fisher's score as raw data and preprocessed them with different preprocessing techniques. Thus, it is interesting to compare the accuracy of Simple Window technique, *0.85,* to the accuracy of regular Fisher's score with 20 attributes, *0.84*. Although there is an increase in the accuracy, it is very minor. Such a phenomenon can be explained by the fact that it takes a very *short* time for a worm to initialize and after the initialization phase it starts to operate in a constant manner. Thus, in this case, the temporal preprocessing does not help much, but only adds attributes, thus making the ANN more complicated for training.

TABLE VI
TEMPORAL ANALYSIS RESULTS

	Averaged accuracy
Without temp. preprocessing	0.84±0.21
Simple Window	**0.85±0.17**
Simple Exponential	0.83±0.24
Poisson Exponential	0.81±0.23

VII. DISCUSSION AND CONCLUSION

In this paper we presented three different feature selection techniques for selecting computer behavior features that can be used for detecting the presence of worms. We showed that the accuracy of worm detection may increase when the detection process is using only the most *important* features. We presented the five most important attributes and derived different rules, related to these attributes, from the trained ANN by using the Causal Indices method. Additionally, we used three temporal preprocessing techniques to see whether they are able increase the accuracy of the detection.

We showed that Fisher's score, despite its simplicity, appears to be a good feature selection method for computer behavior data. It demonstrated the best accuracy and minimal standard deviation using only the top five features. Additionally, we showed that preliminary temporal processing does not increase the detection accuracy significantly and may even decrease it. This may happen due to the fact that the initialization period of the worm is very short and after this period of time worm operation is stable and temporal preprocessing becomes irrelevant.

For future work, we propose the evaluation of the system using additional types of malwares such as Viruses, Trojans and so on.

ACKNOWLEDGMENT

This work was done as a part of a Deutsche-Telekom Co. /Ben-Gurion University joint research project.

We would like to thank Clint Feher, for providing the worm software and for creating the large number of security datasets we used in this study.

REFERENCES

[1] P. Kabiri and A.A. Ghorbani, "Research on intrusion detection and response: A survey," *International Journal of Network Security*, vol. 1(2) Sept. 2005, pp. 84-102.

[2] S. Zanero and S.M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," *Proc. 2004 ACM symposium on Applied Computing,* 2004, pp. 412–419.

[3] H.G. Kayacik, A.N. Zincir-Heywood and M.I. Heywood "On the capability of an SOM based intrusion detection system," *Proc. Int. Joint Conf. Neural Networks* Vol. 3, 2003, pp. 1808–1813.

[4] J. Z. Lei and A. Ghorbani, "Network intrusion detection using an improved competitive learning neural network," *Proc. Second Annual Conf. Communication Networks and Services Research (CNSR04),* 2004, pp. 190–197.

[5] P. Z. Hu and Malcolm I. Heywood, "Predicting intrusions with local linear model," *Proc. Int. Joint Conf. Neural Networks*, Vol. 3, 2003, pp. 1780–1785.

[6] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," *Proc. High Performance Computing Symposium - HPC 2002*, pp 178-183.

[7] I. Yoo. "Visualizing windows executable viruses using self-organizing maps," *Proc. 2004 ACM Workshop on Visualization and Data Mining for Computer Security*. 2004.

[8] U. Ultes-Nitsche and I. Yoo. "An Integrated Network Security Approach: Pairing Detecting Malicious Patterns with Anomaly Detection," *Proc. Conference on Korean Science and Engineering Association in UK*.

[9] Z. Liu, S.M. Bridges and R.B. Vaughn "Classification of anomalous traces of privileged and parallel programs by neural networks," *Proc. FuzzIEEE* 2003, pp. 1225-1230.

[10] D. Stopel, Z. Boger, R. Moskovitch, Y. Shahar and Y. Elovici. "Application of Artificial Neural Networks Techniques to Computer Worm Detection," *Proc. International Joint Conference on Neural Networks*, Vancouver, 2006.

[11] M.B. Hagan, M.T. Menhaj. "Training feed forward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks,* Vol. 5(6), 1994, pp. 989-993.

[12] Z. Boger. "Selection of the quasi-optimal inputs in chemometric modeling by artificial neural network analysis," *Analytica Chimica Acta* 490(1-2) (2003) 31-40

[13] T. Golub, D. Slonim, P. Tamaya, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. Lander. "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring," *Science*, 286:531-537, 1999.

[14] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[15] J. Lorch, A. J. Smith. "The VTrace tool: building a system tracer for Windows NT and Windows 2000," *MSDN Magazine*, 15(10):86–102, October 2000.

[16] I.H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[17] K. Baba, I. Enbutu, M. Yoda. "Explicit representation of knowledge acquired from plant historical data using neural network," *Proc. International Joint Conference on Neural Networks,* Vol. 3 (1990) 155-160

[18] (342/2006) R. Moskovitch, I. Gus, S. Pluderman, D. Stopel, C. Glezer, Y. Shahar, Y. Elovici. "Detection of Unknown Computer Worms Activity Based on Computer Behavior using Machine Learning Techniques," Department of Information System Engineering, Ben-Gurion University of the Negev, Israel (2006)