Application of Artificial Neural Networks Techniques to Computer Worm Detection

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

Abstract—**Detecting computer worms is a highly challenging task. Commonly this task is performed by antivirus software tools that rely on prior explicit knowledge of the worm's code, which is represented by signatures. We present a new approach based on Artificial Neural Networks (ANN) for detecting the presence of computer worms based on the computer's behavioral measures. In order to evaluate the new approach, several computers were infected with seven different worms and more than sixty different parameters of the infected computers were measured. The ANN and two other known classifications techniques, Decision Tree and *k*-Nearest Neighbors, were used to test their ability to classify correctly the presence, and the type, of the computer worms even during heavy user activity on the infected computers. The comparisons between the three approaches suggest that the ANN approach have computational advantages when real-time computation is needed, and has the potential to detect previously unknown worms. In addition, ANN may be used to identify the most relevant, measurable, features and thus reduce the feature dimensionality.**

I. Introduction

The increasing dependence of the modern society on Information and Communication Technology (ICT) highlights the importance of protection against malware (*mal*icious soft*ware*) attacks. A single malware in a computer, which is a part of a computer network, can result in the loss, or unauthorized utilization or modification, of large amounts of data and cause users to question the reliability of all of the information on the network.

A recent survey on intrusion detection [1] suggests using artificial intelligence (AI) techniques to recognize 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 from such AI technique is operating efficiently in real-time.

One of the AI technique mentioned in this survey is the Artificial Neural Networks (ANN). Several research papers referenced in the survey are using the Self Organizing Map (SOM) ANN method [2]-[4]. The main challenge mentioned by the authors in using the ANN was to overcome the difficulties in clustering data having high input dimensionalities. Linear techniques such as Principle Component Analysis, Singular Value Decomposition or Support Vector Machine [5], [6] are used to reduce the input dimensionality, which may result a less accurate modeling.

The survey refers to several studies, in which the researches tried to detect an intrusion or a malware presence by analyzing executable files on local storage devices [7], by analyzing the content of the packets sent or received by the computer [8], [2], or by looking at the system calls that were accessed by 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 computer *behavior*. We have not limited the research to a sub-group of features we measure. Instead we measured as many features as possible in order to determine computer's overall *behavior* and analyze it in order to detect the presence of a malware.

ANN modeling is used for analyzing data when no mathematical relationships between the inputs and the outputs of a system are known. They have been used for modeling and clustering of real-life industrial, commercial and textual systems [10], [11]. It was already demonstrated that high dimension datasets, with thousands input attributes can be reliably modeled by using non-random initial connection weights ANN and proprietary algorithms that avoid and escape local minima during the training [12]-[14]. This capability can be used for data-mining applications, when the relationships in large databases are sought, or for clustering, when the number of clusters in the data is unknown [15], [16]. New knowledge can be acquired by analyzing the trained ANN, either by Causal Indices that relate the changes in each input to the change of each output [17], or by identifying the input attributes that can be discarded without reducing the accuracy of the re-trained ANN model [18], [19].

The main ANN advantages are good accuracy in real-time operation, low CPU resources utilization during the classification and the ability to detect and identify previously unseen classes. For this reason we propose employing ANN

Manuscript received January 31, 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, PhD 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.

for the detection of malware activity in real time. We use clustering classification, which is described later in II-B, among with supervised learning in order to utilize the ability of the ANN to report new threats and its relatively fast training. In this study we compare the ANN approach to two well-known classification methods – Decision Trees (DT) and *k*-Nearest Neighbors (KNN) [20]. We describe these methods in section II.

Among other malwares, *computer worm* is a self-replicating computer program. Computer worm (or simply a worm) is a fully stand-alone program that does not need to be a part of other program in order to propagate. The fact that worms propagate very fast on networks makes them one of the most challenging malwares to intercept. Fast detection of computers infected with worms is critically important on local networks.

We focus on the application of ANN in the task of worm detection within the environment of Microsoft Windows® operating system, which is currently mostly used. Thus, after a detection of *suspicious* behavior the suspected computer can be disconnected from the network in order to avoid the worm from spreading.

The structure of this paper is as follows: In section II the classification methods used in the study are described. In section III we illustrate how we obtained the data sets that were used in the study. In section IV we present the evaluation techniques we used in order evaluate the classification methods. In addition, in section IV we describe the purpose of each particular experiment we made. In section V we present the results of the experiments. In section VI we discuss the results and propose future work.

II. CLASSIFICATION METHODS USED IN THE STUDY

In this section we review three methods that we used in this study for detecting the presence of a worm, based on various parameters that were collected from the infected computer.

A. Artificial Neural Networks

An Artificial Neural Network is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. General description can be found in [10].

The main advantages of the ANN are the ability to find patterns in highly non-linear problems and the very fast classification time. Supervised learning, in which each output unit is told what its desired response to input signals ought to be, may not detect a new threat, or worm. As it was not included in the training set, no output unit has been assigned to such a worm. One way to overcome this limitation, which is also common to the DT and KNN methods, is to utilize the hidden neurons outputs as pattern detectors

The hidden neurons outputs have to condense the information in the feature space into smaller spaces, orthogonal to each other, in order to generate the correct outputs of the ANN model, thus they may learn concepts [21]. The maximum information content of the hidden neuron outputs is reached when their outputs are close to zero or one (for sigmoid as a processing function), minimizing the entropy calculated as the sum over all hidden neurons of $p * ln(p)$, when p are the hidden neurons outputs [22]. It was found in many cases that in a well-trained ANN, the hidden neuron's output values tend to be close to either one or zero [14]. Thus the hidden neurons outputs may be rounded into binary patterns, giving a maximum of 2^h possible classes, when h is the number of hidden neurons.

These "binary" patterns have been used successfully to form clusters in many ANN applications. Typically these patterns are formed by unsupervised ANN. These ANN learn with no external teacher, where the feature vector is presented both as the input and the output. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties, such as in Auto Associative ANN (AA-ANN). Examples giving identical hidden neuron's output pattern are assumed to belong to the same cluster [16].

High-dimensional AA-ANN may be difficult to train to a low modeling error, thus we propose, for the case of detecting new worm, to train supervised ANN using known threats, and then learn the "binary" patterns of the hidden neurons. When an unknown new threat is presented to the trained ANN, it may generate a new "binary" pattern, or its behavior would be similar to known threats. As long as this pattern is not similar to the binary patterns resulting from normal computer operational behaviors, no False Negative (FN) error will result. This method described in a more detailed way in subsection II-B.

In this preliminary study we have used the Levenberg-Marquardt method [23]. This method considered to be one of the best free algorithms for training ANN, and it is available as part of the MATLAB® Neural Network Toolbox [24]. As it uses second-order derivatives, it may require high computation resources, but as the training is done off-line, not in real time, the modern high-speed PC computing power is sufficient for this purpose. Once trained, the ANN processing capabilities are very fast and are suitable for real-time worm detection.

B. Classification by clustering

After training a supervised ANN it is possible to measure the accuracy of the network detection by analyzing the outputs of the hidden neurons instead of analyzing the outputs of the output neurons. Such an approach is useful because it enables to determine the ability of ANN to detect an unknown worm.

For each sample propagated through the *trained* network the output of hidden neurons with the sigmoid processing function are measured and rounded in order to obtain a binary pattern.

After the propagation of all of the samples, each sample has its associated binary pattern that represents the cluster

this sample belongs to. Now it is possible to build a table (clusters matrix) that represents the obtained clusters. Each row in this table represents a cluster, each column represents a class. Cells in each row represent the number of samples, in certain cluster, which belong to a certain class (e.g., "WormA" or "Clean"). The class of the cluster is defined as the class of the majority of the samples in the cluster. The samples from other classes in such cluster are considered to be *incorrectly* classified samples. The ways of calculating evaluation measures, such as accuracy, from such table are described in subsection IV-B.

C. Decision Trees

Decision tree (DT) method is a good choice when the data-mining task is classification or prediction of outcomes and the goal is to generate rules that can be easily understood and explained. DT labels records and assigns them to discrete classes. DT can also provide the measure of confidence that the classification is correct.

DT is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches to achieve homogeneous partitions. The J48 method, which was used in our experiment, is a java implementation of the C4.5 DT algorithm introduced by Quinlan [25].

The advantages of the DT are the fast training and classification times and the availability to easily extract simple rules from the tree after the training process. The disadvantage is that DT cannot be applied in an *unsupervised* setup. Thus it is impossible to identify new types of patterns.

D. k- Nearest-Neighbor

In *k*-Nearest-Neighbor (KNN) prediction, the training data set is used to predict the value of a variable of interest for each member of a test data set. When receiving a new example the algorithm looks for existing examples most similar to the new one. Similarity may be based on feature values (Euclidean distance) or on some different similarity measure. *k* defines the number of such similar examples the algorithm is looking for. The label of the majority of the examples found in this group of *k* most similar examples is given to a new example.

Of course the computing time goes up as *k* goes up, but the advantage is that higher values of *k* provide smoothing that reduces vulnerability to noise in the training data. In practical applications, *k* is typically in units or tens rather than in hundreds or thousands. One of the main disadvantages of KNN is the very long classification time. This disadvantage makes it problematic to use KNN in real time systems.

III. Data Set description

In order to evaluate the new approach, several computers were infected with seven *real*, different worms. Sixty-eight different parameters of the infected computers were measured (see the full list of 68 features in the Appendix). In subsection A the main characteristics of the seven worms are described. In subsection B data gathering process is described. In subsection C specific properties of each data set are illustrated.

A. Worms description

1. **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).

2. **W32.HLLW.Doomjuice.B** (DoomJuiceB)

This worm randomly generates IP addresses and attempts to propagate to computers by using the backdoor opened by the worm W32.Mydoom.A@mm. It tries to connect to computers using TCP port 3127, and if the connection is established it uses the backdoor to infect the computer. It is programmed to add itself to the registry so that it is loaded on startup.

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

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

4. **W32.HLLW.Raleka** (Raleka.H)

This worm launches 200 threads that generate random IP addresses. The random IP addresses are chosen so that some of them are similar to the current IP and the others are totally random. The similar IP addresses are more likely to be in the same local network. It takes over the computer by exploiting the Microsoft DCOM RPC vulnerability. It opens a random TCP port above 32767 for remote connections and may also get commands from the chat site.

5. **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 are generated completely at 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.

6. **Daber.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.

7. **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.

B. Data gathering process description

Each worm was injected separately into a clean computer for a specified amount of time. During this time various features were gathered from the computer. These features represented the behavior of the computer infected with a certain worm. Conceptually, the reason to choose the specified subset of features that used in this study was to combine most of the possible measures related to networking, and on the other hand to combine measures that may give general picture about the processes and threads currently running on the computer. All other measures, such as graphics related etc., seemed to be irrelevant and were not included. It is possible to divide all chosen features conceptually into six general subgroups:

1. Processor features
2. TCP layer features
3. UDP layer features
4. IP layer features
5. Low Network Interface features
6. Objects features

In addition to the infected cases, the same features were measured in a clean, uninfected computer. In the following pages we will refer to this *clean* case as a virtual worm with name "Clean". In order to gather the data, Microsoft's Performance tool was used.

There were two data-gathering processes consisting of six different sub-processes. In each such sub-process one worm (including the "Clean" one) was allowed to operate for a constant time period, and during this time the behavior of this computer was recorded with a specified resolution, i.e., all the features were measured in specified time interval. Each row (called *sample*), representing the behavior of the computer in the specified time interval, was labeled with the worm name that was related to a certain sub-process. The difference between these two processes is that the first one (DS1) is longer (about seven hours for each worm) without user activity whereas the second one (DS2) is shorter (about twenty minutes for each worm) and included heavy user activity during the data gathering process. The schedule of 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 slice all the activities were performed simultaneously.

C. Data Set specific description

The exact specifications of the two data sets including resolution, number of features etc., are presented in Table II. As described, in each process, the resolution and total execution time were common and constant for all the sub-processes.

Due to the unusually large amount of samples available, the training percentage of the two of the data sets was chosen to be 1% and 10% respectively (and not 70% as usual) in order to check in a better way the ability of the ANN to *generalize*.

As it is seen from Table II, five worms were used in each data set but the worms were not the same. While creating DS2 we have been forced to replace two of the worms as they took so much computer resources that it was impossible to perform any user activity on the machine while measuring its behavior. Total number of different worms in all experiment was seven as stated.

TABLE II
TWO DATA SETS DESCRIPTION

Specification	Data Set 1 (DS1)	Data Set 2 (DS2)
Resolution for all sub-processes	1 second	1 second
Execution time for each sub-process	7 hours	20 minutes
Worms used	1-5	1,3,5-7
Features amount	68 + 6 CAs	68 + 6 CAs
Total samples amount	150789	9371
Training set percentage	1%	10%
Training samples amount	1508	938
Test set percentage	99%	90%
Test samples amount	149281	8433
User activity	No	Yes

CA stands for Class Attribute.

In order to check even further the ability of ANN to generalize and detect new worms, one additional data set has been constructed for each data set (DS1 and DS2). During this construction the samples of one of the worms have been transferred completely from the training data set to the test data set, i.e., the training set did not contained samples related to this worm at all. The references to those two additional data sets will be Data Set 1.1 (DS1.1) and Data Set 2.1 (DS2.1) respectively. After training the ANN with the remaining samples in the training set, the accuracy of the

prediction was measured using the hidden-neurons-based clustering method as described in subsection II-B.

IV. Evaluation

A. General evaluation measures

In order to perform the comparisons of the tested methods, we employed the commonly used evaluation measures: True Positive Rate (TPR) (1), False Positive Rate (FPR) (2), Precision (3), F-Measure (4) and accuracy (5). The character 'A' in the end of measure name stands for "*number*" (amount), and should emphasize the fact that these measures are not fractions but true numbers.

$$\text{TPR} = TPA/(TPA+FNA) \tag{1}$$

$$\text{FPR} = FPA/(FPA+TNA) \tag{2}$$

$$\text{Precision} = TPA/(TPA+FPA) \tag{3}$$

$$\text{F-Measure} = \frac{2\times(TPR\times precision)}{TPR+precision} \tag{4}$$

$$\text{accuracy} = \frac{TPA+TNA}{TPA+TNA+FPA+FNA} \tag{5}$$

Generally we divide all the experiments into two categories. The aim of the first category of experiments was to compare the detection abilities of the ANN to the DT and KNN methods. In order to evaluate this we applied all three methods to DS1 and DS2. The common measure between ANN and other two was TPR per particular class. The second category of experiments goal was to test the quality of the detection using clustering analysis based on the supervised trained network and to check the ability of the ANN to detect worms that were not present in training set. TPR and accuracies were calculated for the networks trained on DS1 and DS2 using (6) and (7) respectively, as described in subsection IV-B. This was done for comparison with the TPR and accuracies from typical supervised analysis (first category). Additionally TPR and accuracies were calculated for networks trained on DS1.1 and DS2.1 in order to show the ability of those networks to detect new worms. Later on we also compare the elapsed time for the training and evaluation procedures for each method.

B. Ways for calculating evaluation measures from clusters matrix

It is possible to calculate Accuracies and TPR from clustering matrix described in subsection II-B, for each one of the classes using (6) and (7) respectively. In these equations M is the clusters matrix, l is the class index its accuracy or TPR we want to calculate and $\sigma(i)$ is the i^{th} index (of k total) of the cluster where class l is dominant (σ is the group of all such indices).

$$\text{accuracy}(l) = \frac{\sum_{i=1}^{k} M_{\sigma(i),l} + \sum_{i=\{\{1..n\}/\sigma\}} \sum_{j=\{1..l-1,l+1..n\}} M_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{n} M_{i,j}} \tag{6}$$

$$\text{TPR}(l) = \sum_{i=1}^{k} M_{\sigma(i),l} \Big/ \sum_{i=1}^{n} M_{i,l} \tag{7}$$

V. Results

A. Artificial Neural Networks

In Tables III (no user activity) and IV (with user activities) the False Negative Rate (FNR = 1-TPR) and accuracy results for supervised analysis using the ANN method are presented for each class. The average accuracy achieved for ANN was *99.96%* and *99.79%*, respectively. For clarity, 1-TPR and 1-accuracy values are given in percents.

TABLE III
Results summary: ANN, no user activity

1-TPR	1- accuracy	Class
0.00%	0.02%	Deborm.Y
0.00%	0.00%	DoomJuice.B
0.08%	0.01%	Padobot.KorgoX
0.08%	0.03%	Raleka.H
0.10%	0.02%	Sasser.C
0.01%	0.01%	Clean

TABLE IV
Results summary: ANN, with user activity,

1 - TPR	1- accuracy	Class
0.99%	0.31%	Clean
0.79%	0.16%	Daber.A
0.39%	0.15%	Deborm.Y
0.32%	0.05%	Padobot.KorgoX
0.71%	0.39%	Sasser.C
0.58%	0.19%	Slackor.A

It is clear that generally the results are quite good. It is possible to see from Fig. 1 that the majority of the errors were concentrated around the points where change to a new worm in the data set took place, i.e., when worm only began to function.

As mentioned in section II, a clustering classification was also used in order to determine the quality of detection. Table V presents the summary of such evaluation technique on all four data sets. For clarity, 1-accuracy values are given in percents. Values labeled with *italic* stands for worms that were missing in the training set.

It was found that in some cases, examples of the same worm were clustered by different binary patterns. All the accuracies, except one, in Table V are above 95% so it is clear that the ANN could distinguish the missing worms from the "Clean" class, and from other worms. In the DS2.1 tests, some of the worms were misidentified, but the exact

identification is not as important as the distinction between "clean" and "worm" states.

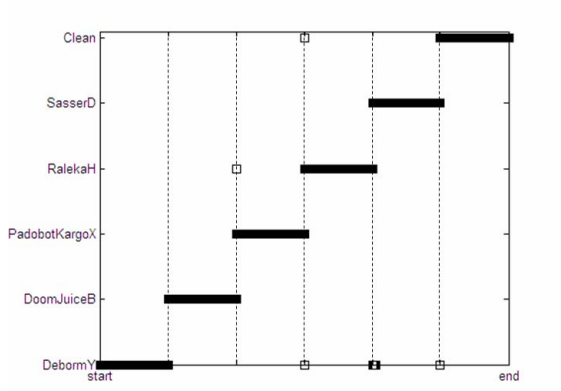

Fig. 1. The distribution of the predicted classes relative to their placement in the data set. The x-axis is the number of examples. The broken lines are the time points at which a new worm began its activity.

TABLE V
CLUSTERING CLASSIFICATION OF THE DATA SETS

DS1	DS2	DS1.1	DS2.1	Class
0.02%	0.11%	2.19%	11.34%	Deborm.Y
0.00%	-	*1.56%*	-	DoomJuice.B
0.00%	0.15%	0.15%	1.49%	Padobot.KorgoX
0.02%	-	0.96%	-	Raleka.H
0.01%	0.28%	1.12%	1.55%	Sasser.C
0.01%	0.85%	0.40%	0.50%	Clean
-	0.02%	-	*1.44%*	Daber.A
-	0.64%	-	10.24%	Slackor.A

All numbers represent 1-accuracy values in percents

In malware detection in general and worms detection in particular, the False Negative (FN) parameter relative to the infected state is very important. Such false negatives are those examples when a *worm* was detected as *clean* case. Such examples are the worst mistake the detection system may make. Table VI presents the FNR values for *clustering evaluation* of each of the datasets. As it can be observed these FNR values are very low.

TABLE VI
FALSE NEGATIVE VALUES FOR EACH DATA SET

DS1	DS2	DS1.1	DS2.1
0.01%	0.39%	0.35%	0.26%

B. Decision Trees

The results for DT evaluated on DS1 are summarized in Table VII. The average accuracy achieved for the DS1 was: *99.89%*. The accuracy that was achieved in DS2 by DT was almost perfect i.e., the DT method has not made even one mistake in the detection.

In Fig. 2 the distribution of the predicted classes among the samples is presented. As in the ANN case, the majority of the mistakes are close to the points where a change between worms took place.

TABLE VII
RESULTS SUMMARY: DECISION TREE

1-TPR	FPR	Precision	F-Measure	Class
0.10%	0.00%	1.000	0.999	Deborm.Y
0.40%	0.00%	1.000	0.998	DoomJuice.B
0.00%	0.00%	1.000	1.000	Padobot.KorgoX
0.00%	0.00%	1.000	1.000	Raleka.H
0.10%	0.00%	1.000	1.000	Sasser.C
0.00%	0.10%	0.994	0.997	Clean

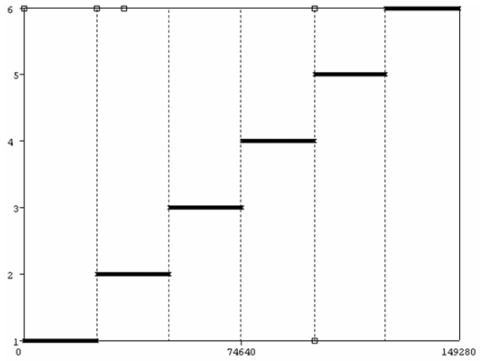

Fig. 2. The distribution of the predicted classes relative to their placement in the data set. The x-axis is the number of examples. The broken lines are the time points at which a new worm began its activity. The 1-6 numerals on the y-axis are, respectively, Deborm.Y, DoomJuice.B, Padobot.KorgoX, Raleka.H, Sasser.C, and Clean.

C. k-Nearest-Neighbor

In Table VIII the results for *k*-Nearest-Neighbor algorithm are presented. The average accuracy achieved for this algorithm on DS1 was *99.66%*. The average accuracy for DS2 is similar to one of DS1. As it is seen in Fig. 3, in this method we do not have the phenomenon we had in the ANN and DT concerning the distribution of the predicted classes relative to the placement. In this method the *errors* are distributed more or less evenly upon the scale. Although in this case it can be seen that the majority of mistakes took the Raleka.H class.

TABLE VIII
RESULTS SUMMARY: *K*-NEAREST-NEIGHBOR

1-TPR	FPR	Precision	F-Measure	Class
0.70%	0.10%	0.996	0.995	Deborm.Y
0.40%	0.00%	1.000	0.998	DoomJuice.B
0.10%	0.00%	1.000	1.000	Padobot.KorgoX
0.10%	0.20%	0.989	0.994	Raleka.H
1.10%	0.00%	1.000	1.000	Sasser.C
0.60%	0.10%	0.995	0.994	Clean

D. Time Comparison

In Table IX the elapsed time periods of all methods, for training and classifying the DS1, are presented. All the experiments were performed on a PC with Intel® Pentium® 4 3.00 GHz CPU and 1GB of RAM. All the ANN manipulations have been performed in a MATLAB® environment. Calculation of the DT and KNN were performed using the WEKA software [26].

The ANN model has the *longest* time for the training phase, while *fastest* in the classification phase. Due to the

fact that training phase has to be performed once off-line, and can be performed on a fast computer, the classification time period is the one that important here. Thus it is clear that in such applications the ANN model has computational advantages over other methods.

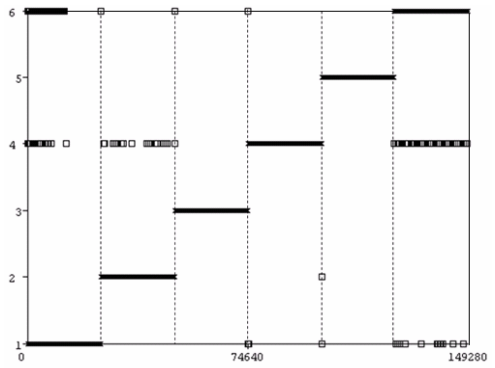

Fig. 3. The distribution of the predicted classes relative to their placement in the data set. The x-axis is the number of examples. The broken lines are the time points at which a new worm began its activity. The 1-6 numerals on the y-axis are, respectively, Deborm.Y, DoomJuice.B, Padobot.KorgoX, Raleka.H, Sasser.C, and Clean.

TABLE IX
TIME PERIODS DESCRIPTION

Method	Training Time	Classification Time
ANN	86 minutes	*0.1 seconds*
k-Nearest-Neighbor	*0.01 seconds*	4 hours
Decision Tree	0.02 seconds	8 seconds

Numbers marked with *italic* emphasize the minimum in each column.

E. Feature Ranking Analysis

One way of evaluating the relative relevance of the inputs is to calculate the contribution of each input feature to the variance of the hidden neurons, based on the trained ANN structure [19]. If the relative contribution of an input feature is small, either this feature does not vary much in the training examples, or the ANN training has assigned low connection weights from it to each of the hidden neurons.

We performed a feature ranking analysis and top five features along their ranks are presented in Table X below.

TABLE X
TOP FIVE FEATURES RELATIVE RELEVANCE

Feature	Relative Relevance
TCP\Connections Active	0.145
System\Threads	0.143
Objects\Threads	0.130
IP\Datagrams Received Address Errors	0.101
System\Threads	0.088

VI. DISCUSSION

The performance of the three methods is different for the evaluated datasets. Without a user activity, the supervised ANN has less FN errors than the DT and the KNN. However, when presented with the DS2 data having user activity, the DT gave no errors, out-performing the ANN and KNN. The advantage of the ANN method over the other two methods is its ability to classify correctly a worm not used in the training. The ANN clustering based on the rounded hidden neurons patterns appears to give better classifications than the traditional threshold analysis of the ANN outputs for DS1 dataset, without user activity. For DS2, with user activity, clustering analysis gives mixed results – some worm classifications are more accurate in the clustering method, while some are more accurate in the traditional ANN classification method.

While it is a preliminary report the results are very encouraging and it leads to the exploration of several directions. The first one is to find the more significant features for worm detection. This will be done by training the ANN models with all possible measured parameters, and then, by analyzing the trained ANN, recursively discard the less relevant parameters and retrain the ANN with the reduced set until the optimal set is found, a wrapper feature selection approach. The second direction is to use aggregation of *k-length* sequences of samples that may help to see different temporal patterns in worm activity. This could be especially important in detection of the worms in most early stages, the problem we have encountered in this study.

In conclusion it can be seen from our study that it is *possible* to detect malicious activity of *worms* by looking at the attributes derived from the computer operation parameters such as memory usage, CPU usage, traffic activity etc. On the other hand the place of the misclassifications suggests that there are still difficulties related to the detection of the worms in the beginning of their activity.

APPENDIX

Following is the full list of 68 features that were measured. These features represent the computer behavior. Each line shows the sub-group and the name of the feature.

1. IP\Datagrams Forwarded/sec
2. IP\Datagrams Outbound Discarded
3. IP\Datagrams Outbound No Route
4. IP\Datagrams Received Address Errors
5. IP\Datagrams Received Delivered/sec
6. IP\Datagrams Received Discarded
7. IP\Datagrams Received Header Errors
8. IP\Datagrams Received Unknown Protocol
9. IP\Datagrams Received/sec
10. IP\Datagrams Sent/sec
11. IP\Datagrams/sec
12. IP\Fragment Re-assembly Failures
13. IP\Fragmentation Failures
14. IP\Fragmented Datagrams/sec
15. IP\Fragments Created/sec
16. IP\Fragments Re-assembled/sec
17. IP\Fragments Received/sec
18. Network Interface\Bytes Received/sec
19. Network Interface\Bytes Sent/sec
20. Network Interface\Bytes Total/sec
21. Network Interface\Current Bandwidth
22. Network Interface\Output Queue Length

23. Network Interface\Packets Outbound Discarded
24. Network Interface\Packets Outbound Errors
25. Network Interface\Packets Received Discarded
26. Network Interface\Packets Received Errors
27. Network Interface\Packets Received Non-Unicast/sec
28. Network Interface\Packets Received Unicast/sec
29. Network Interface\Packets Received Unknown
30. Network Interface\Packets Received/sec
31. Network Interface\Packets Sent Non-Unicast/sec
32. Network Interface\Packets Sent Unicast/sec
33. Network Interface\Packets Sent/sec
34. Network Interface\Packets/sec
35. Objects\Events
36. Objects\Mutexes
37. Objects\Processes
38. Objects\Sections
39. Objects\Semaphores
40. Objects\Threads
41. Process(_Total)\Thread Count
42. Processor(_Total)\% C1 Time
43. Processor(_Total)\% C2 Time
44. Processor(_Total)\% C3 Time
45. Processor(_Total)\% DPC Time
46. Processor(_Total)\% Idle Time
47. Processor(_Total)\% Interrupt Time
48. Processor(_Total)\% Privileged Time
49. Processor(_Total)\% Processor Time
50. Processor(_Total)\% User Time
51. Processor(_Total)\Interrupts/sec
52. System\System Calls/sec
53. System\Threads
54. TCP\Connection Failures
55. TCP\Connections Active
56. TCP\Connections Established
57. TCP\Connections Passive
58. TCP\Connections Reset
59. TCP\Segments Received/sec
60. TCP\Segments Retransmitted/sec
61. TCP\Segments Sent/sec
62. TCP\Segments/sec
63. Thread(_Total/_Total)\Context Switches/sec
64. UDP\Datagrams No Port/sec
65. UDP\Datagrams Received Errors
66. UDP\Datagrams Received/sec
67. UDP\Datagrams Sent/sec
68. UDP\Datagrams/sec

ACKNOWLEDGMENTS

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 worms software and for creating the large number of security data sets 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," in *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," in *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," in *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] Yoo, I. 2004. Visualizing windows executable viruses using self-organizing maps. *In Proceedings of the 2004 ACM Workshop on Visualization and Data Mining For Computer Security*.

[8] InSeon and Ulrich Ultes-Nitsche, 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] Sarle, W.S., ed., *Neural Network FAQ, part 1 of 7: Introduction,* periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: ftp://ftp.sas.com/pub/neural/FAQ.html

[11] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[12] H. Guterman, "Application of principal component analysis to the design of neural network," *Neural, Parallel and Scientific Computing* Vol. 2, 1994, pp. 43-54.

[13] Z. Boger, "Who is afraid of the BIG bad ANN?" *Proc. Int. Joint Conf. Neural Networks*, 2002, pp. 2000-2005.

[14] Z. Boger and H. Guterman, "Knowledge extraction from artificial neural networks models," *Proc. IEEE Int. Conf. Systems Man and Cybernetics*, 1997, pp. 3030-3035.

[15] Z. Boger, "Artificial neural networks methods for identification of the most relevant genes from gene expression array data," *Proc. Int. Joint Conf. Neural Networks*, Vol. 4, 2003, pp. 3095 – 3100.

[16] Z. Boger, "Finding patients' clusters' attributes by auto-associative ANN modeling," *Proc. Int. Joint Conf. Neural Networks*, 2003, pp. 2643-2648.

[17] K. Baba, I. Enbutu and M. Yoda, "Explicit representation of knowledge acquired from plant historical data using neural networks," in: *Proc. Int. Joint Conf. on Neural Networks,* 1990, pp. 155-160.

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

[19] T. Kuflik, Z. Boger and P. Shoval, "Filtering search results using an optimal set of terms identified by an artificial neural network," *Information Processing & Management*, Vol. 42, 2005, pp. 469-483.

[20] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997. pp. 55-83, 239-257.

[21] L. Bochereau and P. Bourgine, "Extraction of semantic features logical rules from a multi-layer neural network," *Proc. Int. Joint Conf. Neural Networks*, Vol. 2, 1990, pp. 579-583.

[22] R. Kamimura and S. Nakanishi, "Hidden information maximization for feature detection and rule discovery," *Network Computation in Neural Systems*, Vol. 6, 1995, pp. 577-602.

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

[24] Demuth, H. and Beale, M. Neural Network toolbox for use with Matlab. The Mathworks Inc., Natick, MA,

[25] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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