Neural Comput & Applic (2009) 18:663–674
DOI 10.1007/s00521-009-0238-2

ORIGINAL ARTICLE

Using artificial neural networks to detect unknown computer worms

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

Received: 8 May 2007 / Accepted: 29 January 2009 / Published online: 20 February 2009

Abstract Detecting computer worms is a highly challenging task. We present a new approach that uses artificial neural networks (ANN) to detect the presence of computer worms based on measurements of computer behavior. We compare ANN to three other classification methods and show the advantages of ANN for detection of known worms. We then proceed to evaluate ANN's ability to detect the presence of an unknown worm. As the measurement of a large number of system features may require significant computational resources, we evaluate three feature selection techniques. We show that, using only five features, one can detect an unknown worm with an average accuracy of 90%. We use a causal index analysis of our trained ANN to identify rules that explain the relationships between the selected features and the identity of each worm. Finally, we discuss the possible application of our approach to host-based intrusion detection systems.

D. Stopel (✉) · R. Moskovitch · Z. Boger · Y. Shahar · Y. Elovici
Deutsche Telekom Laboratories at Ben-Gurion University, 84105 Beersheba, Israel
e-mail: dima@stopel.net
URL: http://www.stopel.net

R. Moskovitch
e-mail: robertmo@bgu.ac.il

Y. Shahar
e-mail: yshahar@bgu.ac.il

Y. Elovici
e-mail: elovici@bgu.ac.il

Z. Boger
Optimal-Industrial Neural Systems, 84243 Beersheba, Israel
e-mail: zboger@bgu.ac.il

Keywords Artificial neural networks · Worm detection · HIDS · Feature selection

1 Introduction

Modern society's increasing reliance on information and communication technology underscores the importance of coping with malicious software (*malware*) attacks. A single malware program in a single computer connected to a larger network can result in the loss, unauthorized utilization, or modification of large amounts of data and cause users to question the reliability of all of the information on the network.

The detection of malware transmitted over computer networks has been substantially researched over the past several years [1–4]. The term *malware* refers to various types of programs, such as executables or scripts that contain code that has a malicious purpose. One type of malware is a *worm* that actively propagates through communication protocols, in many cases by exploiting vulnerabilities in the operating system [5]. Other types of malware are *viruses* that inject their code into innocent executables files and are activated whenever those infected files are executed. Unlike worms, viruses require user intervention to propagate. Other recently disseminated malware programs 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.

Today, known malware programs are mainly detected and removed by antiviruses that search the executables for known patterns, also called *signatures*. For common antivirus software, the detection of an unknown malicious executable is extremely difficult. In the case of worms, a

new signature is created by the antivirus system company only after the appearance of the new worm, and the antivirus signature base is then updated. However, since worms spread rapidly, the signature update action is often taken too late, after the worm has already had an opportunity to cause expensive damage [6, 7].

In this study, a different approach is proposed. The detection of the presence of malware in a computer host is effected by analyzing the overall computer behavior. We define the behavior according to a variety of different features that can be measured in the computer during its operation. We then apply a trained artificial neural network (ANN) model to detect the appearance of malware based on the values of the measured features. We apply ANN by using the *semi-supervised* approach, which employs a supervised training method and unsupervised detection method. To facilitate the detection of unknown malware based on the generalization of the behavior of known malwares, we propose training supervised ANNs using known malwares, and then extracting the binary patterns of the resulting hidden neurons outputs and using them for classification as in the typical AA-ANN. When a new malware is presented to the trained AA-ANN, it will generate a new binary pattern describing the behavior of the new malware. Such an approach enables us to detect and classify the behavior of malwares that were not included in the training set. In this study, we focus on detection of computer worms.

Rapid detection of worm infections is of critical importance. The main advantages of ANN are its high level of efficiency in real-time operations, low consumption of CPU resources during the classification phase, and its ability to generalize, which is important for detecting any previously unseen worm behaviors. For these reasons, we propose employing ANN models for the detection of worm activity in real time. Such an approach may result in a host-based intrusion detection system (HIDS) based on the analysis of computer behavior.

It would be natural to compare the results of our approach to the existing anti-virus systems. However, it is not suitable in our case, since antiviruses, which commonly are signature-based methods, cannot detect an unknown worm for which a signature is not available. Thus, such a comparison is irrelevant. In this paper, we compare the detection capabilities of our approach to the detection capabilities of three different classification methods, decision trees (DT), *k*-nearest neighbors (*k*NN) and support vector machines (SVM), to detect five real, recently created worms. We regard the capabilities of these alternative classification methods as a base line. In an earlier study, we showed that the detection of new and known worms using ANN techniques is feasible and effective [8]. However, since continuous monitoring of a large number of measured features may be very demanding in terms of computational resources, we reduce the number of features significantly by using various feature selection techniques, while maintaining, and even increasing, the detection accuracy. In addition, we use the causal indices (CI) technique to estimate the influence of each input feature on the classification of each worm [9].

The rest of this paper is structured as follows. In Sect. 2, we present work related to this study. In Sect. 3, we describe the classification methods that we used. In Sect. 4, we describe the feature selection methods that we used during the study. In Sect. 5, we describe the worms that were used to create the dataset illustrated in Sect. 6. Section 7 includes a review of the evaluation measures employed in this study. Section 8 presents the evaluation of the results of the new approach. In Sect. 9, we summarize the study and conclude.

2 Related work

2.1 Malicious software

The term malware commonly refers to pieces of code, not necessarily executable files, which are intended to cause harm, in general or to a particular host owner. Malicious codes are classified into four main categories: worms, viruses, Trojans and a new, rapidly growing category which includes remote access Trojans, and backdoors. While the aim of the approach suggested in this study is to develop the ability to detect activities of any new and unknown malware, our initial research target was worms, and in this section we will focus on them.

In a recent report [5], worms were defined in terms of how they differ from other types of malware. (1) Malicious code—worms are considered malicious in nature. There are no good worms that break into systems to repair their vulnerabilities; when a mobile code is used for a legitimate purpose, it is called an *agent*, (2) network propagation is also a commonly agreed-upon characteristic. Worms propagate actively over networks, while other types of malicious code, such as viruses, commonly require more human intervention, (3) degree of human intervention, which refers to the amount of user activity required to propagate a malware program. Some sort of human intervention is typically required for a worm's propagation, (4) stand-alone or file-infecting—while viruses infect host files, a worm does not necessarily require a host file. Some worms, such as the Code Red [10] worm, do not even require an executable file, residing entirely in the memory.

Another report focusing on worms [11] examined people's motivations in developing worms. According to this study, motives include: (a) experimental curiosity, such as

that which led to the development of the ILoveYou [12] worm, which was based on a student's thesis project proposal; (b) feelings of pride and power that can lead programmers to show off their knowledge and skill through the harm caused by their worm; and (c) commercial advantage, extortion and criminal gain, random and political protest, terrorism, and cyber-warfare. Worms are very good vehicles for propagating code for different purposes. The existence of all of these types of motivation makes it clear that computer worms are here to stay. Meaningful knowledge can be gained from existing, identified worms and applied to generic security systems. A very important characteristic of worms is their active propagation through networks. Unlike the vulnerability being exploited or the payload type, which may vary between worms, this characteristic is shared by all worms and should be considered in efforts to detect unknown worm activity.

2.2 Detecting malicious software using data mining techniques

A recent study of intrusion detection [1] summarized the recently proposed applications of data mining for recognizing malware in single computers and computer networks. Lee et al. [13] proposed a framework of data-mining algorithms for the extraction of unusual instances of user behavior for use in *anomaly detection*, in which normal behaviors are learned and any abnormal activity is considered suspect. The authors suggested several techniques, such as classification, meta-learning, association rules, and frequency of episodes, for the extraction of knowledge for further implementation in intrusion detection systems. They tested their approach on the DARPA98 [14] benchmark test collection, which is a standard network data set for intrusion detection research.

A Naïve Bayesian classifier was mentioned in [1] with reference to its implementation in the ADAM system developed by Barbara et al. [2]. The ADAM system has three main parts: (a) a tool that monitors network data by listening to TCP/IP protocol; (b) a data-mining engine that enables the acquisition of the association rules from the network data; and (c) a classification module that classifies the nature of the traffic into two possible classes, normal and abnormal, that can later be linked to specific attacks. Other proposed machine learning algorithm techniques are ANN [3, 15, 16], self organizing maps (SOM) [17] and fuzzy logic [4, 18, 19].

The techniques used to detect intrusions or the presence of malware include analysis of the executable files on local storage devices [20], analysis of the content of the packets sent or received by the computer [21], and examination of the system calls invoked by processes running on the system [22]. Apap et al. [23] proposed using Windows registry access records to monitor malicious code anomalies. Mukkamala and Sung [24] used ANN and SVM on the DARPA98 data set in order to classify intrusions based on network communication measurements.

All the approaches described above tend to focus on a specific type of data that can be measured in the computer. Our approach is not limited to a specific type of measured data, but instead collects data by measuring as many features as possible and lets the classification or feature selection method define an optimal subset of features. Our experiments show that such a subset generally contains different types of feature from different families.

2.3 Network and host-based intrusion detection systems

Intrusion detection systems (IDS) aim to detect unwanted manipulation of a machine's user or a computer network, such as attacks against vulnerable services on the network, unauthorized access to user files, attacks on applications, etc. There are several major types of IDS. One of them, network-based intrusion detection system (NIDS), detects intrusions by monitoring network traffic and multiple hosts which are considered part of the network [25, 26]. Another type of IDS is a protocol-based intrusion detection system (PIDS), which is installed at the front end of a server, in order to listen to the traffic in a specific protocol (e.g., HTTPS for HTTP servers).

Our study is related to HIDS [15, 27–29]. HIDS are installed in the end user computers (hosts) and monitor the dynamic measures (features) on these computers, generally referred to collectively as system state, according to which the HIDS decides whether the state of the computer is acceptable or an intrusion alert should be declared. HIDS are used alongside NIDS, in order to detect malware which has slipped through the NIDS. The combination of NIDS and HIDS is commonly called a *Hybrid IDS*. One example of a Hybrid IDS is the famous Prelude IDS.[1]

In their 1999 paper on intrusion detection systems, Debar et al. [30] defined the taxonomy of these systems and reviewed several algorithms for learning to identify an intrusion. They recognized the inherent advantages of using ANN modeling for this task, but they qualified their support by their apparent inability to explain its reasoning, and criticized ANN modeling for being "computationally intensive." In this paper, we show that the ANN training and analysis tools we use are capable of overcoming these two limitations. The ANN model training is not computation intensive, as it is done off-line, and the result is not a "black-box" device.

[1] http://www.prelude-ids.org/.

3 Classification methods

In this section, we review the four classification methods used in this study to detect the presence of a worm based on various parameters that were measured in the infected computers.

3.1 Artificial neural networks

3.1.1 General description

An ANN is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. A general description of this paradigm can be found in [31, 32]. The main advantages of ANN are its ability to find patterns in highly non-linear problems and its very fast classification time.

Supervised learning and classification procedures, in which each output unit is told what its desired response to input signals ought to be, have trouble detecting new classes, or, as in our case, an unknown worm. One way to overcome this limitation is to utilize the hidden neurons' rounded outputs as cluster signatures. Thus, each cluster has a binary pattern associated with it.

These binary patterns have been used successfully to form clusters in various ANN applications [33]. Typically, they are formed by using *Auto-Associative* ANN (AA-ANN). AA-ANN systems learn with no external teacher in situations in which the feature vector is presented both as the input and the output. This is also referred to as self-organization, in the sense that the AA-ANN self-organizes the data presented to the network and detects collective properties of the dataset.

Although the main criterion that makes it hard to train AA-ANN, and a regular ANN in general, is the number of hidden neurons; a large number of output neurons may also make the training process difficult. In order to overcome this problem we propose using a semi-supervised approach. Thus, to make the detection of unknown worms based on the generalization of the behavior of known worms possible, we propose training supervised ANNs using known worms, and then extracting the binary patterns of the resulting hidden neuron outputs and using them for classification as in the typical AA-ANN. When a new worm is presented to the trained AA-ANN, it will generate a new binary pattern describing the behavior of the new worm. If the binary pattern of the new worm was not unique, the behavior of this worm would also be similar to that of a known worm (or worms). As long as this pattern is not identical to one of the binary patterns resulting from normal computer operational behaviors, there will be no false negative errors. This method is described in the next subsection.

For our experiments, we used the Levenberg–Marquardt ANN training method [34]. This method is considered to be one of the best algorithms for training ANN, and is available as part of the MATLAB® neural network toolbox [35]. As it uses second-order derivatives, it may require expensive computation resources, but as the training is done off-line, the computing power of a modern, high-speed PC is sufficient for this task. Once trained, the ANN is capable of processing data very quickly and can be used for real-time worm detection.

3.1.2 Classification by clustering

After training ANN in a supervised way, it is possible to classify a specific sample by analyzing the outputs of the hidden neurons instead of the outputs of the output neurons. Such an approach is useful because it enables us to determine the ability of an ANN to identify an unknown worm as an *unknown type*.

For each sample propagated through the trained network, the outputs of hidden neurons are measured and rounded in order to obtain a binary pattern. After the propagation of all of the samples, each sample has its own binary pattern that represents the cluster to which the sample belongs. Now, it is possible to build a *cluster matrix* that represents the obtained clusters. Each row in this matrix represents a cluster and each column represents a known class. Cells in each row represent the number of samples in the cluster that belong to a specific class. The class of each cluster is defined by the label of the majority of samples in the cluster. The samples from other classes that ended up in this cluster are considered to be incorrectly classified. The calculation of various evaluation measures, such as accuracy, from the cluster matrix is described in Sect. 7.

3.2 Decision trees

The DT method is a good choice when the data-mining task involves classification or prediction of outcomes and the goal is the generation of rules that can be easily understood and explained. The DT labels and records data points and assigns them to discrete classes. DT can also provide a measure of confidence that the classification is correct. A DT model is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting them further on each of the branches to achieve homogeneous subsets.

The J48 method used in our experiment is a Java implementation of the C4.5 DT algorithm introduced by Quinlan [36]. The advantages of the DT are its short training and classification times and the fact that simple

rules can be extracted from the tree after the training process. The disadvantage is that typical DT cannot be applied to detect new types of worms.

3.3 *k*-nearest neighbors

In the *k*NN method, the training dataset is used explicitly to classify each sample of a test dataset. When evaluating a new example, the algorithm looks for those existing examples that are 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 similar examples for which the algorithm is searching. The label of the majority of the examples found in a group of k most similar examples is given to a new example [37].

Clearly, the computing time increases with the value of k, but the advantage of higher k values is that they provide smoothing that reduces the 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 *k*NN is its very long classification time. This disadvantage makes it unsuitable for real-time applications.

3.4 Support vector machines

Support vector machines (SVM) were developed in 1995 by Vapnik and his colleagues at AT&T laboratories [38]. This technique was originally developed for linear, binary classification with margin, where margin stands for the minimal distance between the class-separating hyperplane and the closest data point. The general SVM seeks an optimal separating hyperplane that maximizes the margin. An interesting and important feature of the SVM approach is that the solution, which is the separating hyperplane, is based only on the data points that are at the margin. These points are called *support vectors*.

The simple, linear SVM can be extended to a non-linear one when the data of the problem are transformed into a feature space using a set of non-linear functions [38]. In the feature space, even with very high dimensionality, the data points can be separated linearly. One of the important advantages of the SVM is that it does not require this kind of transformation or the calculation of the separating hyperplane in the potentially high dimensional feature space. Instead, a kernel representation can be used, in which the solution is written as a weighted sum of the values of a certain kernel function evaluated at the support vectors. Typical SVM algorithms have two disadvantages. The first is that it may be difficult to explain the obtained model. The second is that SVM algorithms require an important parameter-tuning stage in order to give the desired degree of accuracy.

4 Feature selection methods

There are two different approaches to feature selection: the *wrapper* approach and the *filter* approach [39]. 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 its relatively long computation time. The *filter* approach ranks the features according to a certain measure that is independent of any classification algorithm. Thus, after the calculation of the ranks, one can use any subset of features based on their ranks. We used three different filter techniques. These techniques are described in the following subsections.

4.1 Hidden neurons' relative variance

The hidden neurons' relative variance (HNRV) knowledge extraction and dimensionality reduction technique involves ranking the inputs (features) according to their relevance to the prediction accuracy of the ANN [40]. This technique is based on the observation that, in a trained ANN model, a less relevant input contributes a smaller proportion of the variance in the activities of the hidden layer neurons. This may be the result of either the small relative variance of the input feature values or the small final connection weights of 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). $(W_H)_i^{\mathrm{T}}$ is the ith row of the transpose of W_H (i.e., the ith 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.

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

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

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

4.2 Fisher's score ranking

Fisher's score-ranking technique is used to calculate the difference, described in terms of mean and standard deviation, between positive and negative examples relative to a certain feature [41]. Equation (3) defines the Fisher score, in which R_i is the rank of feature i, which describes the proportions of the substitution of the mean (μ) of the feature i values in the positive examples (p), the negative examples (n) and the sum of their standard deviations (σ). A larger R_i value implies a larger and more significant difference between the values of the positive and negative

examples relative to feature i; thus, a feature with a larger R value is more important for separating the positive and negative examples.

$$R_i = \frac{|\mu_{i,p} - \mu_{i,n}|}{\sigma_{i,p} + \sigma_{i,n}} \tag{3}$$

4.3 Gain ratio filter

The gain ratio (GR) measure is based on the information gain (IG) measure, which is found by measuring the relative entropy reduction [37]. This method requires prior discretization of the continuous data. Equation (4) defines the classic entropy measure, in which S is the entire dataset, C is the class attribute, and S_c is the subset of S in which the value of C is c.

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

Equation (5) defines the IG rank. IG describes 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.

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

The disadvantage of the IG method is that it assigns higher ranks to attributes with large numbers of unique values [i.e., high $V(A)$ values]. The GR method 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 GR rankings.

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

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

If the value of $SI(S,A)$ is zero, then the value of $GR(S,A)$ is equal to the value of $IG(S,A)$.

4.4 Causal indices analysis

Causal index (CI) analysis [42] enables the identification of the relationships between specific inputs and outputs of an ANN model through quasi-qualitative analysis of the trained ANN connection weights. It can be used to identify the effect of the specific computer system measure on the identification of a known worm, thereby providing information about its effects on the host computer behavior. This new knowledge will be useful for detecting new worms, as many new worms are variants of existing ones.

The CI is calculated as the sum of the product of all pathways between each input and each output. Equation (8) presents the basic formula of the CI from input i to output k, where w_{ij} is the connection weight from input i to the hidden neuron j and w_{jk} is the connection weight from hidden neuron j to output k. The product of these weights is computed for all n hidden neurons. The CI describes the influence direction (positive or negative) and the relative magnitude of the relationship of any input to any output.

$$CI_{ik} = \sum_{j=1}^{n} w_{ij} w_{jk} \tag{8}$$

5 The worms used to create the data sets

Worms differ in their behaviors and internal structures. For our experiments, we selected subjects that are representative of different types of worm. This allowed us to analyze the abilities of the different classification methods to find similar patterns in different types of worm representing different groups of worms with potentially different behavioral patterns. We briefly describe here the main characteristics of each of the worms used in this study. The descriptions are based on virus library information available on the Web.[2,3]

W32.Dabber.A This worm randomly scans IP addresses. It uses the FTP server opened by W32.Sasser.D worm in order to upload itself onto the victim computer. Thus, it is a self-carried worm. The worm adds itself to the registry, so that it will be executed the next time the user logs in. Thus, this worm uses a human activity-based activation strategy. The worm contains a backdoor as a payload, which listens in on a predefined port. If the port is in use, it scans other ports until it finds an unused port that it can exploit. This worm is distinguished from the others by the role of a second worm in its propagation.

W32.Deborm.Y This is a self-carried worm that uses local address optimization while scanning the IP addresses. Using a sophisticated procedure, it scans only those IP addresses that are local, relative to the base IP address. This worm registers itself as an MS Windows service and adds itself to the registry, so that it will be executed when the next user logs in. Thus, it uses a human activity-based activation strategy. This worm carries a payload containing three Trojans, Backdoor.Sdbot, Backdoor.Litmus and Trojan.KillAV, and executes all of them. We chose to include this worm in our study because of the way it chooses IP addresses, and its heavy payload.

[2] Symantec threat explorer: http://www.symantec.com/enterprise/security_response/threatexplorer.

[3] Kaspersky virus list:http://www.viruslist.com/.

W32.Korgo.X This worm uses a totally random method for scanning IP addresses. It takes advantage of the MS Windows LSASS buffer vulnerability in order to connect to the infected computer and download itself, making it a self-carried worm. The worm tries to inject a function into MS Internet Explorer as a new thread. If successful, all the worm's subsequent actions will appear to be performed by Internet Explorer. If not, the worm will continue to run as a stand-alone process. Thus, this worm is classified as a self-activated worm. The worm contains a payload code to connect to predefined websites in order to receive orders or download newer worm versions. The feature that makes the detection of this worm interesting is its self-activation.

W32.Sasser.D This worm uses local address optimization while scanning the network for victim computers. While W32.Slackor.A scans only local addresses, this worm scans the local addresses half of the time and totally random addresses the other half of the time. This worm opens 128 threads for scanning the IP addresses, which has a heavy impact on a computer's CPU usage and generates significant network traffic. This worm exploits the same vulnerability that the W32.Korgo.X worm exploits, but in a different way. It uses a shell to connect to the infected computer's FTP server and upload itself, which makes it a self-carried worm. This worm adds itself to the registry, so that it will be run the next time the user logs in. Thus, it uses a human activity-based activation method. This worm contains no additional payload. Its uniqueness lies in its use of a relatively high number of threads to scan the IP addresses.

W32.Slackor.A This worm uses local address optimization while scanning a network for vulnerable computers. It determines the victim computer's IP address and scans the addresses that have the same first two bytes. The worm uses MS Windows IPC's designated position on the infected computer in order to propagate; thus, it is a self-carried worm. The worm adds itself to the registry, so that it will be run automatically after the next login; thus, it uses a human activity-based activation strategy. This worm contains a payload in the form of a Trojan that it executes. This Trojan opens an IRC server on the victim computer, so that a hacker can connect and control the Trojan with simple IRC client software. The fact that this worm opens an IRC server on the infected computer distinguishes it from other worms and makes its detection particularly interesting.

W32.HLLW.Doomjuice.B This worm randomly generates IP addresses and attempts to propagate to other computers through the backdoor opened by the worm W32.Mydoom.A@mm. It tries to connect to other computers using a specific TCP port and, if a connection is established, it uses the backdoor to infect the new computer. It is programmed to add itself to the registry, so that it will be loaded upon start-up. This worm runs a continuous denial of service (DoS) attack on the Microsoft website (microsoft.com). The fact that it runs a DoS attack makes its detection particularly interesting.

W32.HLLW.Raleka.H This worm uses local address optimization while scanning the network for vulnerable computers. It takes over the computer by exploiting the Microsoft DCOM RPC vulnerability. It opens a random TCP port for remote connections and may also receive commands from its chat site. This worm opens a chat server on the computer, and this makes its detection interesting.

6 Description of data sets

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. The computer network environment consisted of seven computers containing heterogenic hardware and a server simulating the Internet. We injected worms into the network environment, and then monitored various computer features in each of the infected and uninfected computers.

We used the MS Windows performance tool[4], which enables the 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 and user datagram protocol (UDP). We also used VTrace [43], 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 as vectors in a log file. VTrace stored time-stamped events, which were collected in a second file.

In our preliminary study [8], we used only the Windows performance tool, which provided us with a dataset consisting of 68 features. For that study, we built the datasets in the absence of any user activity. We will refer to this dataset as *ds1*. While creating *ds1*, each worm was injected separately into a clean computer and the features were gathered from the computer during that time. In addition, there was a time period during which no worms were activated. In the rest of the paper, we will refer to the state of the computers during this period as "clean." The worms used in *ds1* were: Deborm.Y, DoomJuice.B, Padobot.KorgoX, Raleka.H and Sasser.C.

[4] http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-s/sag_mpmonperf_02a.mspx.

In our next study [9], we employed the VTrace tool, as well. Thus, we generated two files, one using the Windows performance tool and the other using the VTrace tool. Both files were merged to generate a vector of 323 features for every second. We will refer to this dataset as *ds2*. During the creation of *ds2*, each worm was injected separately into a clean computer for 20 min. As in *ds1,* the features were also gathered for a clean-state during the same period of time. The samples in both datasets contained multi-class labels, identifying each worm and the clean state.

In the case of *ds2*, we considered three major aspects: computer hardware configuration, constant background application, which demands high computational resources, and user activity, in order to perform the evaluation in a realistic environment:

a. *Computer hardware configuration*: both computers operated using MS Windows XP, since we considered it to be the most commonly used operating system. The two configurations we considered were *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 a memory of 1 GB.
b. *Background application activity*: there were two options, with background application activity and without. We ran the *WEKA* mathematical processing application software [44] which mainly affected the following features: *Processor Time* (usage of 100%), *Page Faults/sec*, *Average Disk Bytes/Transfer*, *Average Disk Bytes/Write,* and *Disk Writes/sec*.
c. *User activity*: the two options here are the presence and absence of user activity. The detailed description of user activity can be found in Table 1. In each time period, all the user operations were performed simultaneously.

Thus, we had two options for each one of the three aspects, resulting in a total of eight possible subsets (each subset is a combination of three aspects). All eight subsets were combined into one dataset, which we named *ds2*. During the creation of *ds2*, we were forced to replace the worms Raleka.H and DoomJuice.B with Dabber.A and Slackor.A worms, as the heavy CPU usage of these worms made any user activity impossible.

Table 1 User activity schedule

Time period (min)	User operations
0–5	Opening ten instances of MS Word
	Downloading two files simultaneously
5–10	Opening five instances of MS Excel
	Generating random numbers in MS Excel
	Downloading one file
	Listening to Internet radio
10–15	Opening 12 instances of MS Word
	Downloading one file
15–20	Opening nine instances of MS Excel
	Generating random numbers in MS Excel
	Browsing the Internet (using MS I.E)

7 Evaluation measures

7.1 General evaluation measures

In order to compare the described methods, we employed the commonly used evaluation measures: true positive rate (TP), shown in (9), false positive rate (FP), shown in (10), and accuracy, shown in Eq. (11). TP^N is the number of positive examples classified correctly. TN^N is the number of negative examples classified correctly. FN^N is the number of positive examples misclassified. FP^N is the number of negative examples misclassified.

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

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

$$\text{accuracy} = \frac{TP^N + TN^N}{TP^N + TN^N + FP^N + FN^N} \tag{11}$$

Additionally, we plotted receiver operating characteristic (ROC) curves in order to evaluate the different methods. An ROC curve is a graphical representation of the trade-off between the true positive and false positive rates for every possible cut-off value. That is, the ROC curve is the representation of the trade-offs between sensitivity and specificity.

7.2 Calculating the evaluation measures from the cluster matrix

It is possible to calculate accuracies and TP from the clustering matrix described in Sect. 3.1 for each of the classes using (12) and (13). In these equations, M is the cluster matrix, l is the index of the class whose accuracy or TP we want to calculate, and $\sigma(i)$ is the ith index (of k total) of the cluster in which class l is dominant (σ is the group of all such indices).

$$\text{accuracy}(l) = \frac{\sum_{i=1}^{k} M_{\sigma(i),l} + \sum_{i=\{\{1..n\}/\sigma\}} \sum_{j=\{1..l-1,l+1..n\}} M_{i,j}}{\sum_{i=1}^{n} \sum_{j=1}^{n} M_{i,j}} \tag{12}$$

$$TP(l) = \sum_{i=1}^{k} M_{\sigma(i),l} \Big/ \sum_{i=1}^{n} M_{i,l} \tag{13}$$

8 Evaluating the new approach

8.1 Experimental plan

We performed two major experiments. The purpose of the first experiment, which we refer to as *ex1*, was to compare the detection abilities of our approach with the detection abilities of the base line classification methods. For this experiment, we used *ds1*. Since the number of samples was large (151,200), we used only 1% of randomly (uniformly) chosen instances as the training set. The rest of the dataset was used as the test set. We performed multiple classifications of known worms using our approach and the base line classifiers: *k*NN, DT, and SVM.

The purpose of the second experiment, *ex2*, was to investigate the capabilities of our approach in detecting unknown worms and to determine the best feature selection technique for this task. For this experiment, we used *ds2*. As stated above, the purpose of this experiment was to evaluate different feature selection techniques for the detection of an unknown worm using ANN modeling. First, we ranked the features according to each feature selection method. Then, for each feature selection method, we used the top 5, top 10, top 20, top 30, top 50, and full set of features as inputs to the ANN model. For comparison, we used the principal component analysis (PCA) algorithm to generate a composite input to the ANN, taking the most significant principal components for use as features. Thus, we had $4 \times 5=20$ different datasets, in addition to the full dataset, resulting in a total of 21 datasets. For each of these datasets, we performed five classification runs, one run for each worm. During each run, we designated one of the worms as the unknown worm and built the following training and test sets. For the training set, we took 10% of randomly chosen instances, excluding the instances of the worm we chose and also excluding 20% of the clean instances. For the test set for each worm, we took the instances of the chosen worm and the 20% of the clean instances that were excluded from the training set.

8.2 Experimental results

Table 2 contains the results of *ex1*. Each cell describes the error rate of each one of the four classification methods evaluated in *ds1* and the elapsed time, which includes the training time for the dataset and the test time (training time/test time). All the error rates are related to the test set only. While the DT method outperformed the other techniques, with a 0.03% error rate, the ANN had an average error rate of only 0.04%. ANN showed the best results for detection of the clean state. That is, it had the lowest false positive rate.

We continued the experiments with the ANN method, since this was the only method that could identify a new worm in a multi-class detection mode and it had the best speed in the classification phase (as can be seen in the last row of Table 2), thus making its application in HIDS feasible.

Table 2 Summary of error rates for *ds1*

Class	ANN	*k*NN	DT	SVM
Deborm.Y	0.00%	0.20%	0.02%	0.15%
DoomJuice.B	0.00%	0.09%	0.07%	0.00%
Padobot.KorgoX	0.08%	0.02%	0.00%	0.02%
Raleka.H	0.08%	0.18%	0.00%	0.02%
Sasser.C	0.10%	0.02%	0.02%	0.05%
Clean	0.01%	0.18%	0.08%	0.17%
Average	0.04%	0.11%	0.03%	0.07%
Elapsed time train/classify	86 min/0.1 s	3 s/4 h	9 s/0.3 s	20 s/2 s

Table 3 summarizes the results of *ex2*. This table contains the results of the unknown worm classification performed using the feature selection techniques applied to *ds2*. The values in the cells are the detection accuracy averages of five different experiments, one for each missing worm.

The results presented in Table 3 suggest that the Fisher's score method outperformed the other techniques. Surprisingly, the maximum accuracy was achieved by using only five of the attributes selected by the Fisher's score method. The average accuracy of unknown worm detection using these attributes was 0.90. These five attributes are presented in Table 4. They are related to memory management and the number of system context switches.

In order to analyze the differences between different feature selection techniques, we calculated the averaged ROC curves for each of them. First, the ROC curve was calculated for each experiment. Then, the values of the ROC curve were averaged for each feature selection technique. The averaged ROC curves are presented in Fig. 1.

The separation level for the GR technique was significantly lower than those of the other three techniques. However, the separation levels of the HNRV technique and Fisher's score technique were very close to the separation level of PCA. The areas under the ROC curves are presented in Table 5. Despite the fact that Fisher's score was the most accurate feature selection method, the HNRV method achieved the best separation level. For this reason, the HNRV feature selection method may be more suitable for typical HIDS applications.

CI values were calculated from the ANN model that used the five best features, as selected using the Fisher's score method, as inputs. The results are presented in Table 6, in which CIs higher than 5 are marked in bold and

Table 3 The results of the classification with different feature selection techniques

	Top5	Top10	Top20	Top30	Top50	Full	Average
HNRV	0.67 ± 0.35	0.71 ± 0.35	0.85 ± 0.14	0.56 ± 0.31	0.77 ± 0.23	**0.61 ± 0.35**	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	0.61 ± 0.35	0.76 ± 0.26
PCA	0.57 ± 0.34	0.71 ± 0.37	0.56 ± 0.37	0.72 ± 0.28	0.74 ± 0.32	0.61 ± 0.35	0.65 ± 0.34

Table 4 Five best features, as selected by Fisher's score

Attribute no.	Attribute name	Fisher's score
1	PerfMemoryPoolPagedAllocs	23.80
2	PerfMemoryCacheBytes	17.09
3	ThreadTotalContextSwitches/sec	15.24
4	SystemContextSwitches/sec	14.54
5	PerfMemorySystemDriverTotal	13.66

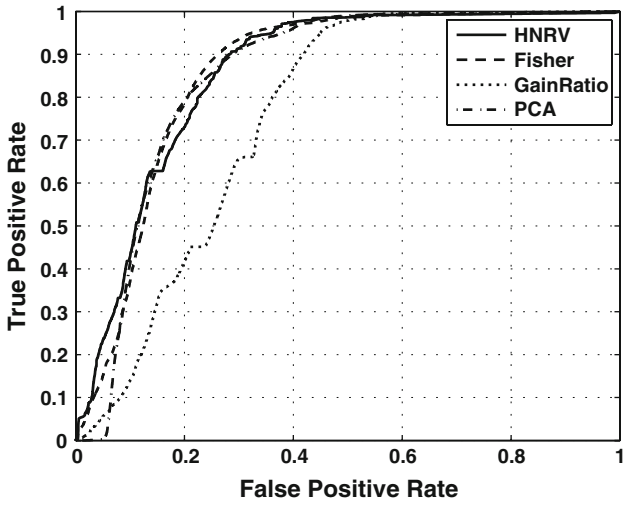

Fig. 1 Averaged ROC curves for four different feature selection techniques

Table 5 The areas under the ROC curves

HNRV	Fisher's score	Gain ratio	PCA
0.90	0.86	0.77	0.86

defined as *high*, and those lower than −5 are marked in italics and defined as *low*. If a certain CI is positive with a relatively high magnitude, then the related input influences the related output in the same direction. That is, if the input is high, the output will also be high. Alternatively, if the CI is negative with a relatively high magnitude, then the related input influences the related output in the opposite direction. That is, if the input is high, the output will be relatively low.

From the CI data presented in Table 6, the following rules can be formulated:

- If feature 2 is *high* and features 3, 4, 5 are *low*, then Clean
- If features 2, 3 and 4 are *high* and feature 1 is *low*, then DabberA
- If feature 2 is *high* and features 1, 3, 4, 5 are *low*, then SasserC
- If features 1, 2, 3 and 4 are *low*, then DabormY

Table 6 CI values of the top five attributes

No.	Attribute name	Clean	Dabber.A	Sasser.C	Deborm.Y	KorgoX	Slackor.A
1	PerfMemoryPoolPagedAllocs	2.8	*−7.2*	*−12.8*	*−7.4*	**9.7**	−4.3
2	PerfMemoryCacheBytes	**6.4**	**6.9**	**11.5**	*−9.0*	*−6.6*	*−6.2*
3	ThreadTotalContextSwitches/sec	*−17.2*	**18.8**	*−7.6*	*−7.7*	−3.1	−1.9
4	SystemContextSwitches/sec	*−15.2*	**15.6**	*−16.4*	*−8.3*	1.1	−1.4
5	PerfMemorySystemDriverTotal	*−10.8*	−0.4	*−14.5*	2.6	**6.7**	−3.8

- If features 1 and 5 are *high* and feature 2 is *low*, then Padobot
- If feature 2 is *low* and features 1, 3, 4, 5 are *average*, then SlackorA

9 Summary and discussion

In this paper, we presented a new approach based on ANN for analyzing computer behavior data and detecting the presence of computer worms. The advantages of our approach over the other three methods are its ability to classify correctly a worm not used in the training, very good detection of new behavior of a known worm, and its short classification time. In addition, we presented our evaluation of 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 uses only the most important features. We presented the five most important attributes and used the CI method to derive different rules related to these features from the trained ANN. We show that the Fisher's score, despite its simplicity, appears to be a good feature selection method for computer behavior data. This selection method demonstrated the maximum accuracy and the lowest standard deviation when only the top five features were used.

The capabilities of the modern ANN training algorithms, coupled with the increased speed of today's computers, have reduced the time needed for off-line training. The resulting ANN models can be used on-line, as their execution speed is compatible with HIDS needs. The process of measuring features, however, may still consume a significant amount of the host's overall computing power. For this reason, there is an advantage in identifying the optimal set of sample features to be used as inputs to the ANN model.

Using ANN models in HIDS detection procedures allows the possibility of easy, periodic updating. New suspected intrusion attempts, initially identified by the AA-ANN as causing abnormal behavior, can be added to the original training data as either new normal behavior or new known threats. The re-training is quick, uses the existing ANN connection weights as a starting point and can be done on-line when needed.

For future work, we propose the evaluation of this approach for detection of additional types of malware, such as viruses and Trojans.

Acknowledgments This study 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 data sets used in this study.

References

1. Kabiri P, Ghorbani A (2005) Research on intrusion detection and response: a survey. Int J Netw Secur 1(2):84–102
2. Barbara D, Wu N, Jajodia S (2001) Detecting novel network intrusions using Bayes estimators. In: Proceedings of the first SIAM international conference on data mining
3. Zanero S, Savaresi S (2004) Unsupervised learning techniques for an intrusion detection system. In: Proceedings of the ACM symposium on applied computing
4. Botha M, Solms R (2003) Utilising fuzzy logic and trend analysis for effective intrusion detection. Comput Secur 22(5):423–434. doi:10.1016/S0167-4048(03)00511-X
5. Kienzle D, Elder M (2003) Recent worms: a survey and trends. In: Proceedings of the ACM workshop on rapid malcode
6. Fosnock C (2005) Computer worms: past, present, and future. Infosec
7. Henry P (2003) A brief look at the evolution of killer worms. A CyberGuard Corporation White Paper
8. Stopel D, Boger Z, Moskovitch R, Shahar Y, Elovici Y (2006) Application of artificial neural networks techniques to computer worm detection. In: Proceedings of the international joint conference on neural networks
9. Stopel D, Boger Z, Moskovitch R, Shahar Y, Elovici Y (2006) Improving worm detection with artificial neural networks through feature selection and temporal analysis techniques. Int J Comput Sci Eng 15:202–209
10. Moore D, Shannon C, Brown J (2002) Code Red: a case study on the spread and victims of an internet worm. In: Proceedings of the internet measurement workshop
11. Weaver N, Paxson V, Staniford S, Cunningham R (2003) A taxonomy of computer worms. In: Proceedings of the ACM workshop on rapid malcode
12. CERT CERT Advisory CA-2000-04. Love letter worm. http://www.cert.org/advisories/ca-2000-04.html
13. Lee W, Stolfo S, Mok K (1999) A data mining framework for building intrusion detection models. In: Proceedings of the IEEE symposium on security and privacy
14. Lippmann R, Graf I, Wyschogrod D, Webster S, Weber D, Gorton S (1998) The 1998 DARPA/AFRL off-line intrusion

detection evaluation. In: Proceedings of the first international workshop on recent advances in intrusion detection
15. Gunes H, Kayacik A, Zincir-Heywood N, Heywood M (2003) On the capability of an SOM based intrusion detection system. In: Proceedings of the international joint conference on neural networks
16. Lei J, Ghorbani A (2004) Network intrusion detection using an improved competitive learning neural network. In: Proceedings of the second annual conference on communication networks and services research
17. Hu P, Heywood M (2003) Predicting intrusions with local linear model. In: Proceedings of the international joint conference on neural networks
18. Dickerson J, Dickerson J (2000) Fuzzy network profiling for intrusion detection. In: Proceedings of the 19th international conference of the North American Fuzzy Information Processing Society (NAFIPS)
19. Bridges S, Vaughn Rayford M (2000) Fuzzy data mining and genetic algorithms applied to intrusion detection. In: Proceedings of the 23rd national information systems security conference
20. Yoo I (2004) Visualizing windows executable viruses using self-organizing maps. In: Proceedings of the ACM workshop on visualization and data mining for computer security
21. Ultes-Nitsche U, Yoo I (2002) An integrated network security approach: pairing detecting malicious patterns with anomaly detection. In: Proceedings of the second conference on information security for South Africa
22. Liu Z, Bridges S, Vaughn R (2003) Classification of anomalous traces of privileged and parallel programs by neural networks. In: Proceedings of the IEEE international conference on fuzzy systems
23. Apap F, Honig A, Hershkop S, Eskin E, Stolfo S (2002) Detecting malicious software by monitoring anomalous windows registry accesses. In: Proceedings of the fifth international symposium on recent advances in intrusion detection
24. Mukkamala S, Sung A (2003) Identifying significant features for network forensic analysis using artificial intelligent techniques. Int J Digit Evidence 1(4):1–17
25. Handley M, Kreibich C, Paxson V (2001) Network intrusion detection: evasion, traffic normalization. In: Proceedings of the 10th USENIX security symposium
26. Mukherjee B, Heberlein L, Levitt K (1994) Network intrusion detection. IEEE Netw 8(3):26–41. doi:10.1109/65.283931
27. Warrender C, Forrest S, Pearlmutter B (1999) Detecting intrusions using system calls: alternative data models. In: Proceedings of the IEEE symposium on security and privacy
28. Wespi A, Dacier M, Debar H (2000) Intrusion detection using variable-length audit trail patterns. In: Proceedings of the international workshop on recent advances in intrusion detection
29. Tandon G, Chan P (2003) Learning rules from system call arguments and sequences for anomaly detection. In: Proceedings of the ICDM workshop on data mining for computer security
30. Debar H, Dacier M, Wespi A (1999) Towards a taxonomy of intrusion–detection systems. Comput Netw 31:805–822. doi:10.1016/S1389-1286(98)00017-6
31. Sarle W (2002) Neural Network FAQ, part 1 of 7: Introduction. Periodic posting to the Usenet newsgroup comp.ai.neural-nets. ftp://ftp.sas.com/pub/neural/FAQ.html
32. Bishop C (1995) Neural networks for pattern recognition. Clarendon Press, Oxford
33. Boger Z (2003) Finding patient's cluster's attributes by auto-associative ANN modeling. In: Proceedings of the international joint conference on neural networks
34. Hagan M, Menhaj M (1994) Training feed forward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993. doi:10.1109/72.329697
35. Demuth H, Beale M (1993) Neural network toolbox for use with Matlab. The Mathworks Inc., MA
36. Quinlan J (1993) C4.5: Programs for machine learning. Morgan Kaufmann, San Francisco
37. Mitchell T (1997) Machine learning. McGraw-Hill, New York
38. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
39. Liu H, Motorda H (1998) Feature selection for knowledge discovery and data mining. Kluwer Academic Publishers. Norwell, MA
40. Boger Z (2003) Selection of the quasi-optimal inputs in chemometric modeling by artificial neural network analysis. Anal Chim Acta 490(1–2):31–40. doi:10.1016/S0003-2670(03)00349-0
41. Golub T, Slonim D, Tamaya P, Huard C, Gaasenbeek M, Mesirov J, Coller H, Loh M, Downing J, Caligiuri M, Bloomfield C, Lander E (1997) Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 286:531–537. doi:10.1126/science.286.5439.531
42. Baba K, Enbutu I, Yoda M (1990) Explicit representation of knowledge acquired from plant historical data using neural network. In: Proceedings of the international joint conference on neural networks
43. Lorch J, Smith A (2000) The VTrace tool: building a system tracer for Windows NT, Windows 2000. MSDN Mag 15(10):86–102
44. Witten I, Frank E (2005) Data Mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco