Host Based Intrusion Detection Using Machine Learning

Abstract. *Detecting **unknown** malicious code (malcode) is a challenging task. Current common solutions, such as anti-virus tools, rely heavily on prior explicit knowledge of specific instances of a malcode binary code signatures. During the time between the appearance of a new worm and the time that an update is being sent to anti- virus tools a new worm can infect many computers and create significant damage. We present a novel host based intrusion detection approach, based on the behavior of the computer's measurements Machine learning based, classification algorithms learn from previous known malcodes samples, then through exploiting the generalization capability an unknown malcode can be detected. We designed an experiment to test this approach, focusing on worms, in several computer configurations and background applications activity, consisting on 323 monitored features. Four classification algorithms were applied on several feature subsets. Our results indicate that using this approach resulted in an above 90% accuracy in average, and for specific unknown worms accuracy it reaches to above 99%.*

1. Introduction

The detection of malicious codes transmitted over computer networks has been substantially researched during the past years. Commonly, the term 'malicious code' refers to *worms*, which actively propagate, exploiting a vulnerability in the operating system or in a program installed through communication protocols, and *viruses*, which inject their code into an innocent file (a host) and are executed whenever the file is executed. Unlike *worms*, *viruses* require user intervention to propagate. Other malicious codes include *Trojans*, which are computer programs that have a useful functionality, but also have some hidden malicious goal, and *backdoors*, which enable remote access and control with the aim of gaining full or partial access to the infected system.

Excellent technology exists for detecting *known* malicious code, intrusion detection systems based on which are commonly termed *antiviruses*. Typically, *antiviruses* inspect each file (code) that enters the system for any known signs which would identify a malicious code. However, this technology relies heavily on prior explicit knowledge of the executable's binary code. These programs search the executables for known patterns, also called *signatures*. After the appearance of a *worm*, a patch is provided by the operating system provider and the antivirus systems update their signatures accordingly. However, since worms propagate very rapidly, this action is often taken too late, and commonly very expensive damage has already been done by the *worm* [Fosnock, 2005]. While *worms* spread intensively through the internet, *Trojans* and *backdoors* have other malicious motivations, such as *economic*, *terrorist*, or *criminal* [Kienzle and Elder, 2003]. However, they may be installed on a relatively small number of hosts, and thus do not attract the attention of *antivirus* systems and remain undetected. For example, a student can demonstrate a problem he has had with his home work using *flash memory* on his/her professor's computer. During this demonstration the student installs a *backdoor*, and towards the end of the semester can then access the professor's computer and search for a copy of the upcoming examination.

Although security solutions are becoming more and more sophisticated, the designers of malicious codes are still ahead of them, taking advantage of different unknown system

vulnerabilities. As in any *warfare*, it is clear that no ultimate and final solution exists, and new techniques have to be invented to sabotage the attack of every new malicious code.

Generally speaking malicious codes within the same category (e.g., *worms, Trojans, backdoors*,..) share the same characteristics and behavior, which can be reflected in the computer's behavior based on its measurements. For example, worms are known to propagate through the network and thus generate a large volume of communication. Assuming that each type of malicious code category has a common characteristic and shares a common behavior pattern, we suggest that an *unknown* malicious code can be detected based on the computer's behavior using machine learning techniques. In the proposed approach, a classifier is trained based on computer measurements of *known* malicious codes and *non-malicious* behavior; based on the generalization capability of the classification algorithm, it can then detect previously unknown activity. However, this approach may suffer from the variability of computer configurations and the variety of applications and user behavior on each computer. In this study, we investigate whether an unknown malicious activity can be detected at a high level of efficiency and accuracy, given the varying conditions existing in any computer.

Intrusion detection, commonly made at the network level, called *network based intrusion detection* (*NIDS*), was researched substantionally. However, *NIDS* are limited in their detection capabilities (like any detection system). In order to detect malcodes which slipped through the *NIDS* at the network level detection operations are performed locally at the host level, called *Host Based Intrusion Detection* (*HIDS*). *HIDS* are detection systems consist on monitoring activities at a host. *HIDS* compare the states of the computer in several aspects, such as the changes in the file system using checksum comparisons. The main drawback of this approach is the ability of malcodes to disable antiviruses, and other technical llimitations such as the fast change in file system. The main problem is the detection knowledge maintenance, which is acquired commonly manually by the domain expert.

Recent studies have proposed static methods for detecting unknown executables using machine learning techniques. In this approach, based on a training set of static representation of malicious and benign executables, a classifier is trained and learns to identify and classify unknown malicious executables as being malicious [Shultz et al., 2001; Abou-Assaleh et al., 2004; Kolter and Maloof, 2004], however, this approach is not complete since it can detect only files, while for example malcodes entirely located in the memory, such as the Slammer worm [Moore et al., 2003], cannot be detected using this technique.

We propose a novel approach, in which the significant (related to each category of malcodes) features of the host operation system are extracted and monitored constantly through. The logged features are aggregated, thus in each time unit a vector of features presents a snapshot of the host, on which a decision is made whether the host is infected or not. A decision is made using a, machine learning based, classification algorithm. The knowledge for the detection is acquired through inductive learning, and not based on a human expert, given a repository of samples of previous seen malcodes (in this study worms) activity. The generalization property of classification algorithms enable to detect unknown malcodes based on the previously seen malcodes. Moreover, unlike in anomaly based intrusion detection, in which the normal behavior is learned and any deviation from the normal behavior is suspected, here the malicious activity concept is learned, specifically for a malicious code or generally for a family of malicious entities.

In this study, we present the results of applying our approach on *worms*. First, a survey of the relevant background and a related work section is presented. The preparations involved in this study are then described, followed by the classification algorithms, the research questions, and the experimental plan. The results are then presented, followed by a discussion of the insights gained from the study, and finally our conclusions.

2. Background and Related Work

2.1. Malicious Code

The term 'malicious code' (malcode) commonly refers to pieces of code, not necessarily an executable file, which are intended to harm, whether generally or in particular, the specific owner of the host. Malicious codes are classified into four main categories: *Worms*, *viruses*, *Trojans,* and a new one, which is becoming more common, comprising *remote access Trojans* and *backdoors*. While the approach suggested in this study aims to be able to detect any new and unknown malcode activity, since we started our research on worms, in this section we will focus them.

In a recent survey [Kienzle and Elder, 2003], the authors define a worm according to several aspects by which they are distinguished from other types of malcode: 1) *Malicious code* – worms are considered malicious in nature (there are no good worms that break into systems and repair their vulnerability; when a mobile code is used for legitimate purposes it is termed an "agent"). 2) *Network propagation* is also a commonly agreed upon aspect, that is, worms propagate actively over a network, while other types of malicious codes, such as viruses, commonly require human intervention. 3) *Human intervention* is another characteristic which refers to the degree of user activity, such as execution of a file, or sending it through an email, required to propagate a malcode. 4) *Standalone or file infecting* – while viruses infect a file (its host), a worm does not require a host file, and sometimes does not even require an executable file, residing entirely in the memory, as did the *Code Red* [Moore et al., 2002] worm.

Another survey focusing on worms [Weaver et al., 2003] refers to people's motivation to develop a worm, which include: *Experimental curiosity* which can lead any person to create a worm, such as the *ILoveYou* [CERT.4] worm, which was based on a student's thesis project proposal. *Pride and power* which leads programmers to show off their knowledge and skill, which are evidenced by the harm caused by the worm; *commercial advantage*, *extortion* and *criminal gain*, *random* and *political protest*, and *terrorism* and *cyber warfare*. It is a great vehicle to propagate some code for different purposes. The existence of all these types of motivation makes it clear that computer worms are here to stay. Meaningful experience and knowledge can be gained from existing and identified worms, and implemented in generic security systems. An important characteristic of worms is that they are actively propagated through network. This characteristic is very important since, unlike the vulnerability being exploited which may vary between worms, this characteristic exists in any worm and should be considered when attempting to detect any such activity.

2.2. Detecting Malicious code Using Data Mining

Machine learning, a subfield of artificial intelligence and data mining, commonly considered as the application of machine learning to huge data sets, has already been used in efforts to detect and protect against malicious codes.

A recent survey on intrusion detection [Kabiri & Ghorbani, 2005] summarizes recent proposed application of data mining in recognizing malcodes in single computers and in computer networks. Lee et al. proposed a framework consisting of data mining algorithms for the extraction of anomalies of user normal behavior for the use in *anomaly detection* [Lee et al., 1999], in which a normal behavior is learned and any *abnormal* activity is considered as intrusive. The authors suggest several techniques, such as classification, meta-learning, association rules, and frequent episodes, to extract knowledge for further implemented in intrusion detection systems. They evaluated their approach on the DARPA98 [Lippmann et al, 1998] benchmark test collection, which is a standard benchmark of network data for intrusion detection research.

A Naïve Bayesian classifier was suggested in [Kabiri & Ghorbani, 2005] referring to its implementation within the ADAM system developed by Barbara et al. [Barbara & Jajodia, 2001]. The ADAM system had three main parts: (a) monitoring network data through listening to TCP/IP protocol; (b) a data mining engine which enables acquiring the association rules from the network data; and (c) a classification module which classifies the nature of the traffic in two possible classes: normal and abnormal which can later be linked to specific attacks. Other machine learning algorithms techniques proposed are Artificial Neural Networks (*ANN*) [Zanero et al., 2004; Kayacik et al., 2003; Lei et al., 2004], Self Organizing Maps (*SOM*) [Hu et al, 2003] and fuzzy logic [Dickerson et al., 2000; Bridges et al., 2000; Botha et al., 2003].

2.3. Network and Host Based Intrusion Detection Systems

Intrusion Detection Systems (*IDSs*) aim at detecting unwanted manipulation to a machine's user or to a computer network, such as attacks against vulnerable services on the network, unauthorized access to user files, attacks on application and more. *IDS*'s include several major types. *Network based Intrusion Detection System* (*NIDS*) which detects intrusions through monitoring network traffic and multiple hosts which are considered part of the network [Handely et al., 2001; Mukherjee et al., 1994]. Another network based *IDS* is a *Protocol based Intrusion Detection System* (*PIDS*), which is installed at the front end of a server, in order to listen to traffic in a specific protocol (e.g., HTTPS for HTTP servers).
Our suggested approach is related to *Host based Intrusion Detection Systems* (*HIDS*) [Warrender et al., 1999; Wespi., 2000; Tandon & Chan, 2003; Kayacik et al., 2003;]. *HIDS* are installed at the end user computers (hosts) and monitor the dynamic measures on these computers, generally referred as system state, on which the HIDS decides whether the state of the computer is acceptable or an alert of intrusion should be made. *HIDS* are used along with the *NIDS* in order to detect malcodes which slipped through the *NIDS*. The combination of NIDS and HIDS is called commonly a Hybrid IDS. Example of such Hybrid IDS is a famous Prelude IDS[1].

3. Methods

The goal of this study was to assess the feasibility of detecting *unknown* malicious code, in particular computer worms, based on the computer behavior (measurements), using machine learning techniques, and the potential accuracy of such a method. In order to create the datasets we built a local network of computers, which was isolated from the real internet network but enabled to represent a real internet network environment from the point of view of a single computer. This setup enabled us to inject worms into a controlled environment, while monitoring the computer measurements, which were saved in log files. Preliminary results were very encouraging, but an obvious question arose: Is a classifier trained on data from a computer having certain hardware configuration and certain specific background activity able to classify correctly the behavior of a computer having other configurations? Thus, a wider experiment was designed which is presented in this paper. We created eight datasets in two computers, having different configurations, background applications, and user activities. Another goal was to select the minimal subset of features using a feature selection technique. Finally, we applied four classification algorithms on the given datasets in a variety of experiments.

3.1. DataSet Creation

Since there is no benchmark dataset which could be used for this study, we had to create our own dataset. We built a network consisting of a variety of computers (configurations), into

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

which we could inject worms in a controlled environment, and monitor the computer features in log files.

Environment Description

The lab network consisted of 7 computers, which contained heterogenic hardware, and a server computer simulating the internet.

Data monitoring and storage

We used the *windows performance counters*[2], which enable monitoring system features that appear in these main categories (The amount of features in each category appear in parenthesis) : *Internet Control Message Protocol* (27), *Internet Protocol* (17), *Memory* (29), *Network Interface* (17), *Physical Disk* (21), *Process* (27), *Processor* (15), *System* (17), *Transport Control Protocol* (9), *Thread*(12), *User Datagram Protocol* (5). We also used VTrace [Lorch & Smith, 2000]. A software tool which can be installed on a PC running Windows for monitoring purposes. VTrace collects traces of the *file system, the network, the disk drive, processes, threads, interprocess communication, waitable objects, cursor changes, windows, and the keyboard*. The data from the *windows performance* were configured to measure the features every second and store them in a log file as vector. VTrace stored time-stamped events. In order to synchronize the vtrace logged events to the windows performance logs, which were measured in fixed time intervals, the events were aggregated in the same fixed intervals. The events were counted and represented in a vector of values for each time intervals. Eventually the *vtrace* data was merged with the *windows performance* log files, which eventually included a vector of 323 features for every second.

Injected Worms

While selecting worms from the wild for the experiment, our aim was choosing worms that differ in their behavior given the available worms. Some of the worms have a heavy payload of Trojans to install in parallel to the distribution process upon the network, other focus only on distribution. The worms have different strategies for scanning the IP addresses and thus have different communication behavior. Additionally several worms install servers on the infected computer (FTP, IRC, etc.,) which also results in a different network usage. Different worms use different number of threads for scanning the IP addresses, thus each worm has a different CPU consumption, which may results differently in conjunction with heavy user activity. While all the worms are different we wanted to find common characteristics to be able to detect an unknown worm.

We briefly describe here the main characteristics, relevant to this study, of each worm included in this study. The information is based on the virus libraries on the web[3][4][5].

W32.Dabber.A: This worm scans the IP addresses in a random way. It uses W32.Sasser.D worm in order to propagate. It uses the FTP server opened by this worm in order to upload itself to the victim computer, thus it is a self-carried worm. The worm adds itself to the registry in order to 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 on a predefined port, if the port is used it scans for other ports until it finds an unused port. The feature that distinguishes this worm from the others is that it uses an external worm in order to propagate.

W32.Deborm.Y: A self-carried worm, which uses preference for local addresses optimization while scanning the IP addresses. A sophisticated procedure is used in order to determine the base IP address based on the infected computer IP address. The worm scans only local IP addresses based on the base IP address it calculates. This worm registers itself as

[2] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/counter/counters2_lbfc.asp
[3] Symantec – www.symantec.com
[4] Kasparsky www.viruslist.com
[5]Macfee http://vil.nai.com

an MS Windows service and adds itself to the registry in order to be executed on the next user login, thus it uses a human-activity based activation strategy. This worm contains three Trojans as a payload: Backdoor.Sdbot, Backdoor.Litmus, and Trojan.KillAV, and executes all of them. We chose this worm because of the way it chooses the IP addresses and its heavy payload.

W32.Korgo.X: This worm uses a totally random method for scanning IP addresses. It uses the MS Windows LSASS buffer vulnerability in order to connect to the infected computer and download the worm, thus it is a self-carried worm. The worm tries to inject a function to MS Internet Explorer as a new thread. If successful then all worm's subsequent actions will appear to be done be the Internet Explorer, otherwise it continues to run as a stand alone process. Thus this worm belongs to the self-activated worm's class. The worm contains a payload code to connect to the predefined websites in order to receive orders or download newer worm versions. The feature that makes this worm interesting to detect is that this is a self-activated worm.

W32.Sasser.D: This worm uses a preference for local addresses optimization while scanning the network for victim computers. While W32.Slackor.A scans only local addresses this worm scans 48% of the time the addresses of the local type and the other 52% of the totally random addresses. This worm in particular opens 128 threads for scanning the computers, thus it has a heavy impact on computer's CPU usage and it generates significant network traffic. This worm exploits the same vulnerability as the W32.Korgo.X worm does but in a different way. It uses a shell to connect to the infected computer's FTP server and to upload the worm. Thus it is a self-carried worm. This worm adds itself to the registry in order to be run the next time the user logins. Thus it uses human-activity based activation method. This worm contains no additional payload. The uniqueness of this worm is that it uses relatively high number of threads in order to scan the IP addresses.

W32.Slackor.A: This worm uses a preference for local addresses optimization while scanning the network for vulnerable computers. It determines the computer IP address and scans the addresses that have same two first bytes. For example if the computer address have the following form: A.B.C.D the worm scans all the addresses of type A.B.X.Y where X,Y are between 0 and 255. The worm uses MS Windows IPC$ share on the computers it order to propagate, thus it is a self-carried worm. The worm adds itself to the registry in order to be run automatically after the next logon, thus it uses a human-activity based activation strategy. This worm contains a payload in a form of a Trojan that it executes. This Trojan open an IRC server on the victim computer, thus the hacker can connect to and control the Trojan by using simple IRC client software. The fact that this worm opens an IRC server on the infected computer distinguishes it from the other worms and makes it interesting to detect.

Dataset Description

In order to assess the influence of a computer hardware configuration, background running applications, and user activity, we considered three major aspects: *computer hardware configuration*, *constant background application* consuming extreme computational resources, and *user activity*. Each aspect had two options (binary variable).

Computer hardware configuration: Both computers ran on *Windows XP*, since we considered it to be the most used operation system. There were two configurations: an "*old*", having Pentium 3 800Mhz CPU, bus speed 133Mhz and memory 512 Mb, and a "*new*", having Pentium 4 3Ghz CPU, bus speed 800Mhz and memory 1 Gb.

Background application: We ran the an application which mainly affected the following features: Processor object, *Processor Time* (usage of 100%); Processor object, *Page Faults/sec*; and Physical Disk object**,** *Avg Disk Bytes/Transfer*, *Avg Disk Bytes/Write*, *Disk Writes/sec*. The two options were presence or absence of the application.

User activity: several applications, including browsing, downloading and streaming operations through Internet Explorer, Word, Excel, chat through MSN messenger, and

Windows Media Player, were executed to imitate user activity in a scheduled order. The two options were presence or absence of the user activity.

Finally, we had three binary aspects, which resulted in 8 combinations representing a variety of dynamic computer situation and their names, shown in Table 1.

Computer	Background Application	User Activity	Dataset Name
Old	No	No	O
Old	No	Yes	Ou
Old	Yes	No	Oa
Old	Yes	Yes	Oau
New	No	No	N
New	No	Yes	Nu
New	Yes	No	Na
New	Yes	Yes	Nau

Table 1. The three aspects resulting in eight datasets, representing a variety of dynamic situations of a monitored computer. For example, *oau* present a dataset created in an old computer, having a application amd user activity in the background.

Each dataset contained monitored samples of each one of the five injected worms separately, and samples of a *normal* computer behavior, without any injected worm. Each worm was monitored for a period of 20 minutes in resolution of seconds. Thus, each record, containing a vector of measurements and a label, presented a second and labeled by the specific *worm* activity, or *none* activity label. Each dataset contained few thousands (labeled samples) of each worm or none activity.

3.2. Feature Selection

In machine learning application, the large number of features in a lot of domains presents a huge problem. Typically, some of the features do not contribute to the accuracy of the classification task and may even decrease it. Moreover, in our approach reducing the amount of features, while maintaining a high level of detection accuracy, is crucial for several computer performance and resource consumption aspects. Ideally, we would like to have no consumption of computer resources resulting from the monitoring operation of the computer resources (measurements), as well as in the classifier operation. Reducing the amount of features is expected to decrease the classification time and computer resources consumption, as well as increasing the detection accuracy. In order to reduce the amount of required features, we used a feature selection technique, commonly used in data mining. Since this is not the focus of this paper, we will describe the feature selection preprocessing very briefly.

There are two major approaches to *feature selection*: wrapping and filtering. Wrappers wrap the classification algorithm and run it iteratively on each subset of features until finding the best subset of features. Since we wanted to compare the performance of the classification algorithms, we used the *filters* approach, which is a preprocessing stage independent of any classification algorithm. In filters, a measure is used which quantifies the correlation of each feature to the class (in our case, the presence of a particular worm or absence of worm activity). Each feature receives a *rank* representing its expected contribution in the classification task. Eventually, the top ranked features are selected.

We used *Gain Ratio (GR)*, originally presented by Quinlan in the context of decision trees [Quinlan, 1993], which was designed to overcome a bias in the *Information Gain (IG)* measure [Mitchel, 1997], which measures the expected reduction of entropy caused by partitioning the examples according to a chosen feature. Given entropy *E(S)* as a measure of the impurity in a collection of items, it is possible to quantify the effectiveness of a feature in classifying the training data. Equation 2 presents the formula of the entropy of a set of items

S, based on *C* subsets of *S* (for example, classes of the items), presented by S_c. *Information Gain* measure the expected reduction of entropy caused by portioning the examples according to attribute *A*, in which *V* is the set of possible values of *A*, as shown in equation 1. These equations refer to discrete values; however, it is possible to extend it to continuous values attribute.

$$IG(S,A) = E(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} \cdot E(S_v) \quad (1)$$

$$E(S) = \sum_{c \in C} -\frac{|S_c|}{|S|} \cdot \log_2 \frac{|S_c|}{|S|} \quad (2)$$

The *IG* measure favors features having a high variety of values over those with only a few. Gain-Ratio overcomes this problem by considering how the feature splits the data (Equations 3 and 4). S_i are *d* subsets of examples resulting from portioning *S* by the *d-valued* feature *A*.

$$GR(S,A) = \frac{IG(S,A)}{SI(S,A)} \quad (3)$$

$$SI(S,A) = -\sum_{i=1}^{d} \frac{|S_i|}{|S|} \cdot \log_2 \frac{|S_i|}{|S|} \quad (4)$$

We selected the *top 5*, *10*, *20* and *30* ranked features from the resulted list, based on the *GainRatio* measure. We describe later the top 15 ranked features later, and in the appendix, in details. Eventually we had four features subsets and the *full* set of features, which we took as the original baseline for comparison purposes, summed in five forms of sets of datasets. Thus, each one of the eight datasets described earlier was presented in five formats including the five features options.

3.3. Classification Algorithms

One of the goals of this study was to identify the classification algorithm which provides the highest level of accuracy. We employed four commonly used machine learning algorithms: *Decision Trees*, *Naïve Bayes*, *Bayesian Networks* and *Artificial Neural Networks*, in a *supervised learning* approach. Supervised learning includes two phases: the first phase, in which the classification algorithm learns from a provided training set, containing labeled examples, and the second phase in which it classifies the instances given in a testing set. In the testing phase the outputs of the classification algorithm are compared to the actual classes, as they appear in the testing set, represented in an accuracy measure.

While the focus of this paper is not on classification algorithm techniques, but on their application in the task of detecting malicious code activity, we briefly describe the classification algorithms we used in this study.

Decision Trees

Decision tree learners [Quinlan, 1993] are a well-established family of learning algorithms. Initially proposed in the 70s, these algorithms have been continuously developed to include more features and yield a better performance. Classifiers are represented as trees whose internal nodes are tests on individual features and whose leaves are classification decisions. Typically, a greedy heuristic search method is used to find a small decision tree that correctly classifies the training data. The decision tree is induced from the dataset by splitting the variables based on the *expected information gain*, based on Shannon's information measure; thus, the feature appearing at a higher level of the tree is more contributive than the ones in the bottom for the classification task. In order to handle noisy data, they are typically augmented with a "pruning" procedure that prevents overfitting of the training data. In this study we evaluated *J48*, the *Weka* version of the commonly used *C4.5* algorithm [Quinlan, 1993]. Various studies in the past have shown that it is an efficient algorithm that learns

accurate classifiers in many domains. An important characteristic of Decision Trees is the explicit form of their knowledge which can be represented easily as rules.

Naïve Bayes

The *Naïve Bayes* classifier is based on the *Bayes theorem*, which in the context of classification states that the posterior probability of a class is proportional to its prior probability as well as to the conditional likelihood of the features, given this class. If no independent assumptions are made, a Bayesian algorithm must estimate conditional probabilities for an exponential number of feature combinations. "*Naive Bayes*" simplifies this process by making the assumption that features are *conditionally independent*, given the class, and requires that only a linear number of parameters be estimated. The prior probability of each class and the probability of each feature, given each class, is easily estimated from the training data and used to determine the posterior probability of each class, given a set of features. Empirically, *Naive Bayes* has been shown to produce good classification accuracy across a variety of problem domains [Domingos and Pazzani, 1997]. The simplicity of its implementation and its fast (linear) training time has made this algorithm a popular choice in practice. In this study, we evaluated *Naive Bayes*, the standard version that comes with *Weka*.

Bayesian Networks

Bayesian networks or Bayesian belief network or just belief network is a form of the probabilistic graphical model [Pearl, 1986]. Specifically, a Bayesian network is a directed acyclic graph of nodes representing variables and arcs representing dependence relations among the variables. Like Naïve Bayes it is based on the Bayes Theorem, but unlike Naïve Bayes it does not assume that the variables are independent. Actually Bayesian Networks are known for their ability to represent the conditional probabilities which are the relations between variables. The *Bayesian Network* represents the variables as nodes in an acyclic graph of nodes, and their relations, or dependencies, are represented by the arcs. An arc from one node to another means that the first is the parent; thus each node is represented by a table which describes its conditional probabilities based on its parent's variables. This conditional distribution is known as the *posterior* distribution of the subset of the variables given the evidence. The posterior gives a universal sufficient statistic for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically constructing extensions of Bayes' theorem to more complex problems. Bayesian networks were used for modeling knowledge and implemented successfully in different domains. We evaluated the Bayesian Network standard version which comes with *WEKA*.

Artificial Neural Networks

An *Artificial Neural Network* (*ANN*) [Bishop, 1995] is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the structure of the information processing system. It is a network composed of a large number of highly interconnected processing elements, called neurons, working together in order to approximate a specific function. An ANN is configured for a specific application, such as pattern recognition or data classification, through a *learning process* during which the weights of the inputs in each neuron are updated. The weights are updated by a *training algorithm*, such as back-propagation; according to the examples the network receives, in order to reduce the value of *error function*. The power and usefulness of artificial neural networks have been demonstrated in several applications including speech synthesis, medicine, finance and many other problems that fall into the category of pattern recognition. For some application areas, neural models show more promise of achieving human-like performance than do more traditional artificial intelligence techniques. All the ANN manipulations have been performed within a MATLAB(r) environment using Neural Network Toolbox [Demuth, 1998].

3.4. Research Questions

Our main goal was to estimate whether the approach presented here, in which unknown malicious code is detected, based on the computer behavior (measurements), is feasible and enables a high level of accuracy. We defined four hypotheses accordingly:

Hypothesis I: It is possible to detect *known* malicious code, based on a computer's measurements, using machine learning techniques, at an accuracy level above 90%.

Hypothesis II: There is no significant influence on the computer configuration and the computer background activity, from which the training sets were taken, on the detection accuracy.

Hypothesis III: Reducing the amount of features to up to a subset of *30 features* will enable us to maintain the level of accuracy provided by the full set of attributes and even higher.

Hypothesis IV: It is possible to detect an *unknown* malicious code at a level of accuracy above 80%.

In addition to these hypotheses, our aim was to identify the best classification algorithms and the best combination of top ranked features and classification algorithm, based on the accuracy measure.

3.5. Experimental Plan

In order to test the hypotheses described above, we created the eight datasets described in Table 1. We had the eight datasets in five feature subsets: *Top 5*, *10*, *20* and *30* ranked features and the *full* features set. In order to determine the best classification algorithm and top feature selection combination, we ran in each experiment all the *20* combinations (five top selections and four classification algorithms).

Experiment I

To test *hypothesis I* we wanted to learn whether, given a dataset of monitored measurements of a computer behavior, it is possible to classify worms at a high level of accuracy. Thus, we unified all the eight datasets into a single dataset. We will refer to this dataset as *"All"*. We evaluated the four classification algorithms on the *All* dataset with the *full* set of features. Since the training set and test set were identical, we used *10 folded cross validation* [Kohavi, 1995], in which the dataset is portioned randomly into ten equal partitions. Nine partitions are used for the training set, and the remaining partition is used for testing and evaluating the learned model. The process is then repeated ten times, leaving out a partition for testing each time.

Experiment II

To test *hypothesis II,* in which the goal was to estimate the performance variability of the suggested approach given several training sets sampled from a variety of computers, represented by the eight datasets. Actually we wanted to test whether training a classifier on a training set sampled from a given computer will vary significantly given a variety of test sets. For this task we designed two experiments:

The first experiment, called $e2_1$, which was measuring the classification accuracy when training was performed on one of the eight datasets and testing was done on each one of the other eight datasets. Each classifier was trained on a single dataset i and tested on a single dataset j, where i and j are indices of the eight datasets. When $i = j$, we used cross validation since training set (i) and test set (j) were the same dataset. Thus, we had a set of 8 iterations in which a dataset was used for training, and 8 corresponding evaluations were done on each one of the datasets, resulting in 64 evaluation runs.

The second experiment, called $e2_2$, the training set was a unified dataset of seven of the eight datasets, and the test set was the left eighth dataset. We trained each classifier on a given dataset, called *all-i*, containing 7 datasets, where *i* refered to the eighth left dataset, which was the test set, resulting in 8 iterations of evaluation run. Note that both experiments included the

20 combinations of four classification algorithms and five features subsets amounting to 1280 runs for $e2_1$ and 160 runs for the $e2_2$.

In order to test *hypothesis III*, in which the goal was determining the optimal number of features required to achieve the highest accuracy. We used the two experiments, $e2_1$ and $e2_2$, described earlier, results in order to analyze and determine, from the top selection options, *Top 5*, *10*, *20*, *30* or *full*, which one outperforms the others.

Experiment III

In *hypothesis IV*, we wanted to estimate the capability of classifying an unknown worm, which was the main objective of this study. We designed two types of experiments similar to $e2_1$ and $e2_2$, respectively.

In the first experiment, called $e3_1$, the training set was one of the eight datasets, which outperformed in $e2_1$, and the test sets were all the eight datasets representing a variety of computer configurations. However, in this experiment, unlike $e2_1$, the training set included four worms out of the five, and the test set included the excluded worm, presenting an unknown *worm* and the *none* activity.

In the second experiment, called $e3_2$, the training set and the test sets included *all* the eight datasets, but in each iteration the training set included four worms out of the five and the test set included only the *excluded worm* and the *none* activity in order to measure the detection capability of an unknown *worm* and the *none* activity.

The described processes were repeated iteratively for each one of the five worms. Note, that in these experiments, unlike in $e2_1$ and $e2_2$, in which each worm class was defined separately, the training sets had two classes: (generally) *worm* and *none* activity. This experiment was done for each classification algorithm, using the top feature selection that outperformed in the previous experiments (*e2*).

3.6. Evaluation Measures

In order to test the hypotheses, and evaluate the feature selection measures and the classification algorithms, we measured the *True Positive (TP)* measure, which is the number of *positive* instances classified correctly as shown in *Equation 1*, *False Positive (FP)*, which is the number of *positive* instances misclassified, as shown in *Equation 2*, and the *Total Accuracy*, which measures the number of the absolutely correctly classified instances, either positive or negative, divided by the entire number of instances shown in Equation 3. We also measured the *confusion matrix*, which shows how many instances from each class were classified in each one of the classes (ideally all the instances would be in their actual class). The primitive evaluation measures used to define the evaluation measures are presented in Table 2.

	Actual (True) Classification		
Algorithm Classification		**Positive**	**Negative**
	Positive	TP^A	FP^A
	Negative	FN^A	TN^A

Table 2: Primitive evaluation measures description

Note that the letter 'A' in the names of the variables in the table stands for 'Amount' and emphasizes the fact that these measures are true amounts and not fractions.

$$TP = TP^A/(TP^A+FN^A) \quad (1)$$

$$FP = FP^A/(FP^A+TN^A) \quad (2)$$

$$\text{Total Accuracy} = (TP^A+TN^A)/(TP^A+FP^A+TN^A+FN^A) \quad (3)$$

4. Results

In this section we present the outcomes of the experiments performed in order to test the hypotheses, as described above. We first performed a preliminary experiment, in which we tested a few feature selection measures, with the aim of reducing the amount of features used in all the experiments. At the end of this phase, we found that *GainRatio* measure produced the best mean performance. Since the focus of this paper is not the feature selection task, we will therefore refer only to the results achieved using this measure.

4.1. Experiment I

In *experiment I* we unified all the eight datasets into a single dataset which contained the entire set of features, using 10-fold cross validation. While *Decision Trees* and *Bayesian Networks* achieved almost 100% accuracy, the others achieved above 90%. Note that while our final goal is a system which is capable of alerting on a worm (like) activity, in this experiment, the classification algorithm had to classify each sample to the specific worm activity class, within the five given worms, and not to the a generic worm behavior. Although these conditions are more challenging, considering this task as a binary problem, having two classes, *'worm-activity'* and *'none'*, is expected to result in higher performance. The results were very encouraging.

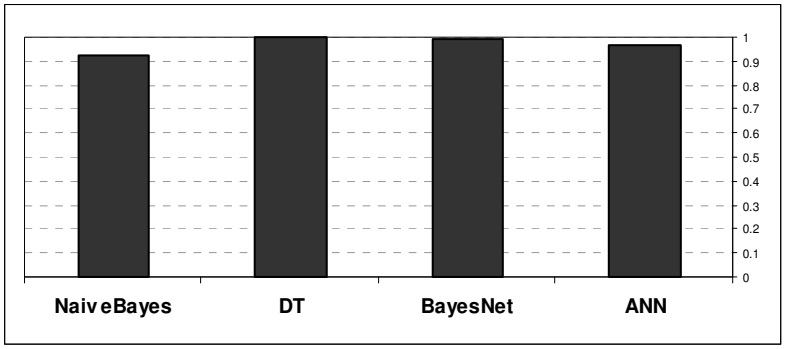

Figure 1: Preliminary results, in which each classification algorithm was evaluated on a unified dataset, consisting of the entire eight datasets and the entire set of features, using 10-fold cross validation. Decision Trees and Bayesian Networks achieved almost 100% accuracy.

4.2. Experiment II

To test the *hypothesis II*, in which we wanted to learn whether the conditions, in which the dataset on which the classifier was trained (e.g., computer configuration, background application and user activity) changes the classification accuracy on each testing set.

Experiment $e2_1$ consisted of twenty experiments using the combinations of five features subsets and four classification algorithms, consisting of 64 iterations each. $e2_1$ gave us the opportunity to measure the variance in the different training sets and in the test sets in terms of accuracy. $e2_2$ also consisted of twenty experiments, but with 8 iterations in each experiment, enabling us to compare the difference in accuracy of each test set. Table 3 shows the mean accuracy achieved for each training set (Data set), consisting of 128 evaluation runs, in the first column. The mean accuracy and the variance are presented. The results of $e2_1$, are presented in two views, when the datasets are used as training set, and when the datasets were the test sets. The third column shows the mean accuracy achieved in $e2_2$ on each test set (presented by the datasets on the left, while the training set were the other seven datasets). In this set of experiments we wanted to estimate whether there was a significant difference

within the datasets, when used as training sets or as test sets in the different experiments. We therefore tested the means of the datasets for homogeneity. The results in the first column, in which we look at $e2_1$ from the perspective of the training sets, were not homogenous. The same occurred with the results in the second column, though we found that the results for the training sets, which were created in the "OLD" computer, were significantly better (alpha = 0.01). The figures in the third column, representing the $e2_2$ results, were found to be statistically significant (alpha = 0.05) homogenous. The classification accuracy in $e2_2$ outperformed the accuracy in $e2_1$, since the training sets in $e2_1$ had a wider representation of the datasets.

Data set			$e2_1$		$E2_2$
Comp Config	**Back App**	**User Activity**	**As training**	**As test**	**As test**
OLD	**NoApp**	**NoUA**	0.68 ± 0.23	0.73 ± 0.23	0.78 ± 0.22
		UA	0.76 ± 0.22	0.73 ± 0.22	0.82 ± 0.18
	App	**NoUA**	0.73 ± 0.21	0.73 ± 0.23	0.81 ± 0.18
		UA	**0.77 ± 0.21**	0.72 ± 0.21	0.81 ± 0.19
NEW	**NoApp**	**NoUA**	0.61 ± 0.24	**0.64 ± 0.22**	**0.71 ± 0.20**
		UA	0.76 ± 0.21	0.72 ± 0.22	0.82 ± 0.22
	App	**NoUA**	0.70 ± 0.21	0.73 ± 0.24	0.86 ± 0.17
		UA	0.73 ± 0.22	0.71 ± 0.23	0.79 ± 0.20
			0.72 ± 0.22	0.72 ± 0.22	**0.80 ± 0.19**

Table 3: The accuracy level achieved when looking from the training set, or test set of $e2_1$. The figures in the third column refer to the level of accuracy achieved in $e2_2$ from the test set, which were found homogenous.

To test *hypothesis III*, we wanted to determine whether the amount of features can be reduced, while maintaining the accuracy level, or even increasing it. For this task we analyzed the performance achieved by the classification algorithms at each one subset of features, resulted at the two experiments, $e2_1$ and $e2_2$.

Table 4 shows the results from $e2_1$. Each cell in the table presents the mean accuracy of 64 evaluation runs, in which a classifier was trained on a training set and tested on the other seven test sets. The *Bayesian Networks* outperformed the other classification algorithms (alpha = 0.01). The *Top20* features subset outperformed the other top selection features subset; while not outperforming the *Top10* at a statistically significant level, it outperformed the others (0.01). *Bayesian Networks* applied to the *Top20* features outperformed all the other combinations.

Feature subset	ANN	BayesNet	DT	NaiveBayes	Average
Top5	0.50 ± 0.10	0.58 ± 0.10	0.59 ± 0.08	0.56 ± 0.08	0.56 ± 0.09
Top10	0.74 ± 0.26	0.90 ± 0.08	0.64 ± 0.25	0.87 ± 0.14	0.79 ± 0.20
Top20	**0.82 ± 0.21**	**0.90 ± 0.09**	**0.70 ± 0.21**	**0.88 ± 0.14**	**0.83 ± 0.17**
Top30	0.75 ± 0.21	0.89 ± 0.08	0.61 ± 0.25	0.85 ± 0.18	0.78 ± 0.19
FULL	0.73 ± 0.16	0.76 ± 0.18	0.51 ± 0.26	0.54 ± 0.26	0.63 ± 0.22
Average	0.71 ± 0.20	**0.80 ± 0.11**	0.61 ± 0.22	0.74 ± 0.17	0.72 ± 0.18

Table 4: Results of $e2_1$. Each cell in the table presents the mean accuracy of 64 evaluation runs. *Bayesian Networks* outperformed the other classification algorithms and *Top20* outperformed the other top selections. The best performance was achieved when *Bayesian Networks* was applied to the *Top20* features subset.

Table 5 shows the results from $e2_2$. Each cell in the table presents the mean accuracy of 8 evaluations, in which each classifier was trained on a training set consisting of seven datasets, and the test set contained only the excluded dataset. In this set of experiments, the mean accuracy performance of most of the algorithms was better than in $e2_1$, as expected, while the *ANN* performance decreased rapidly. Unlike in $e2_1$, the *Decision Trees* outperformed the other algorithms, while *Bayesian Networks* consistently produced good results. On average, the *Top10* features selection outperformed the other top selections. While *Top10* did not outperform the others statistically significantly, the performance of the *Top5* was statistically significantly (alpha = 0.01) lower than the others. Note that while in Table 4 all the algorithms favored the *Top20*, here it varied mainly between *Top20* and *Top10*, though the *ANN* outperformed with the *full* set of features.

	ANN	BayesNet	DT	NaiveBayes	Average
Top5	0.44 ± 0.09	0.62 ± 0.04	0.64 ± 0.01	0.59 ± 0.07	0.57 ± 0.06
Top10	0.74 ± 0.16	0.96 ± 0.04	**0.97 ± 0.04**	0.90 ± 0.06	**0.89 ± 0.09**
Top20	0.55 ± 0.21	**0.96 ± 0.05**	0.94 ± 0.05	**0.94 ± 0.03**	0.85 ± 0.11
Top30	0.62 ± 0.14	0.95 ± 0.05	0.96 ± 0.05	0.93 ± 0.03	0.86 ± 0.08
Full	**0.81 ± 0.23**	0.88 ± 0.09	0.94 ± 0.09	0.88 ± 0.07	0.88 ± 0.14
Average	0.63 ± 0.17	0.88 ± 0.06	**0.89 ± 0.05**	0.85 ± 0.06	0.81 ± 0.10

Table 5: Results of $e2_2$. Each cell presents the mean accuracy of 8 evaluation runs. *Decision Trees* slightly outperformed *Bayesian Networks* and the other classification algorithms. The *Top10* features selection outperformed the other top features selection. The best performance was achieved when *Decision Trees* was applied to the *Top10* features subset.

In order to describe the significant features, resulted from the feature selection process, we present the Top5 features and their definitions. In the category of *ICMP*: (1) *Sent_Echo_sec* - The rate of ICMP Echo messages sent (2) *Messages Sent/sec* - The rate, in incidents per second, at which the server attempted to send. The rate includes those messages sent in error. (3) *Messages/sec* - The total rate, in incidents per second, at which ICMP messages were sent and received by the target entity. The rate includes messages received or sent in error. In the category of *TCP*: (4) *Connections Passive* - The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state. (5) *Connection Failures* - The number of times TCP connections have made a direct transition to the CLOSED state from the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state. The list of the top fifteen ranked features is presented at the Appendix.

Experiment III

To test *hypothesis IV*, we wanted to realize the main objective of this study, i.e., to find out whether it is possible to detect *unknown worms* based on behavior learned from previous worms. The training set consisted of four worms, and the test set contained the fifth excluded worm representing an *unknown worm*. This process was repeated in five iterations. We used each algorithm with the top feature selection which produced the best results in $e2_1$, which was *Top20*.

In $e3_1$ we trained the classifiers on the dataset which outperformed in $e2_1$: *old computer configuration* including constant *background application* and *user activity* (oau). Table 6 shows the results of $e3_1$. Each cell presents the mean accuracy and variance of 5 iterations (for each worm); the algorithm applied to the test set is presented on the left. Unlike in the *e2* experiments, the *ANN* generalized better from the *known* worms presented in the training set, and outperformed the others significantly**.**

Test Set			Classification Algorithm			
Comp	**App**	**UA**	**ANN**	**BayesNet**	**DT**	**NaiveBayes**
OLD	**0**	**0**	0.85 ± 0.31	0.81 ± 0.21	0.91 ± 0.20	0.46 ± 0.06
OLD	**0**	**1**	0.89 ± 0.23	0.84 ± 0.22	**0.95 ± 0.10**	0.88 ± 0.21
OLD	**1**	**0**	0.82 ± 0.22	**0.88 ± 0.12**	0.91 ± 0.20	**0.89 ± 0.13**
OLD	**1**	**1**	0.81 ± 0.25	0.85 ± 0.22	0.93 ± 0.14	0.88 ± 0.22
NEW	**0**	**0**	0.88 ± 0.22	0.84 ± 0.22	0.58 ± 0.14	**0.89 ± 0.22**
NEW	**0**	**1**	0.92 ± 0.14	0.77 ± 0.15	0.45 ± 0.16	**0.89 ± 0.22**
NEW	**1**	**0**	**0.93 ± 0.12**	0.74 ± 0.13	0.48 ± 0.13	0.81 ± 0.23
NEW	**1**	**1**	0.89 ± 0.22	0.79 ± 0.12	0.52 ± 0.03	0.87 ± 0.22
			0.87 ± 0.21	0.81 ± 0.17	0.72 ± 0.14	0.82 ± 0.19

Table 6: The results of $e3_{1,}$ in which the training set contained four worms and the test sets contained the excluded worm, representing an unknown worm. The *ANN* outperformed the other classification algorithms significantly.

Table 7 shows the same results ($e3_1$), but for each type of excluded worm. The detection accuracy varied for different types of worms. Note that the test set consisted only on the *excluded worm* and *none activity* instances. While on average *ANN* outperformed the other classification algorithms, each algorithm classified different types of worms at a higher level of accuracy. We will refer to the variety in each type of worm detection accuracy in the Discussion and Conclusions section.

Worm	ANN	BayesNet	DT	NaiveBayes	Average
1	**0.99 ± 0.01**	0.59 ± 0.11	0.67 ± 0.33	0.51 ± 0.10	**0.69 ± 0.14**
2	0.95 ± 0.03	**0.97 ± 0.04**	0.75 ± 0.27	0.93 ± 0.18	0.90 ± 0.13
3	**0.96 ± 0.10**	0.89 ± 0.11	0.62 ± 0.16	0.88 ± 0.18	0.84 ± 0.14
4	0.95 ± 0.09	0.70 ± 0.04	0.75 ± 0.27	**0.94 ± 0.18**	0.84 ± 0.14
5	0.51 ± 0.14	0.92 ± 0.10	**0.78 ± 0.23**	0.85 ± 0.19	0.76 ± 0.16
Average	**0.87 ± 0.10**	0.81 ± 0.08	0.72 ± 0.25	0.82 ± 0.16	

Table 7: The results of $e3_1$ for each excluded worm and classification algorithm. While the mean accuracy of the *ANN* outperformed the other algorithms, different algorithms classified different type of worms better.

In the $e3_2$ set of experiments we trained the classifiers on the entire eight datasets, containing four worms, and tested on each one of the eight datasets. We present each classification with the top selection features, in which the highest level of accuracy was achieved, which was *Top20* for all the algorithms. Table 8 shows the results achieved in this set of experiments. The results in this experiment were better than in the previous experiment, exceeding a 90% accuracy level of detection for each worm by a different classification algorithm (shown in bold). In average the *Decision Trees* and *Bayesian Networks* outperformed the other algorithms. The table shows also the *true positive* (TP) and *false positive* (FP). *Decision Trees* and *Bayesian Networks* while achieving high level of accuracy maintained a very low level of *false positive* rate.

	ANN_20			BN_20			DT_20			NB_20		
Worm	**Acc**	**TP**	**FP**	**Acc**	**TP**	**FP**	**Acc**	**TP**	**FP**	**Acc**	**TP**	**FP**
1	**0.985**	0.985	0.014	0.554	0.557	0.443	0.936	0.937	0.063	0.497	0.500	0.499
2	0.494	0.499	0.500	0.992	0.992	0.007	0.999	0.999	0.0005	**0.997**	0.997	0.002
3	0.952	0.952	0.047	**0.992**	0.993	0.007	0.680	0.678	0.3215	0.844	0.843	0.156
4	0.994	0.994	0.005	**0.998**	0.998	0.002	0.968	0.968	0.032	**0.998**	0.998	0.002

5	0.636	0.637	0.362	0.990	0.991	0.008	**0.999**	0.999	0.0005	0.975	0.974	0.026
Average	0.81	0.81	0.19	0.91	0.91	0.09	**0.92**	**0.92**	**0.08**	0.86	0.86	0.14
StdDev	0.05	0.05	0.05	0.04	0.04	0.04	0.02	0.02	0.02	0.05	0.05	0.05

Table 8: The results of $e3_2$, in which the training set consisted of all 8 datasets, and the classification algorithm applied to its favored top feature subset. The level of accuracy achieved is higher than in the previous experiment ($e3_1$). There is a difference in the detection accuracy of each classification algorithm for each type of worm. On average *Decision Trees* outperformed the other algorithms, while maintaining a very low level of *false positive* rate.

5. Discussion and Conclusions

We present in this paper an *Host based Intrusion Detection* method for *unknown* malicious code activity, based on monitoring host's various parameters, (as opposed to direct matching of the malcode instance), using various automated classification algorithms. In this study, we started evaluating the proposed method in the detection of computer worms. After having encouraging results, our goal was to check whether a classifier trained on a dataset sampled from a computer, having specific configuration and programs, will be able to classify in a high level of detection accuracy another computer (for example, a newer) instances. We have described the experiments and the dataset created for this purpose. Initially, our aim was to obtain some insight into the influence of the computer configuration and general background application activity on the system's ability to detect a worm. We also wanted to learn which algorithm outperforms the others, and what reasonable, ideally minimal, subset of features maintains a high level of detection accuracy.

In the first experiments, *e1* and *e2*, we trained and tested the classifiers on the entire worm samples in order to estimate the influence of each dataset and the variability in results. We found that the results were statistically significant only when we trained the classifiers on a variety of training sets and not on a single one (as in $e2_1$). This is a very encouraging result, since we assume that, when applying such an approach in the real world, a training set that consists of samples from several types of computer activities is a reasonable requirement. While some variation in the outperforming algorithms and top selection features was observed in $e2_1$ and $e2_2$, in general *Bayesian Networks* outperformed the other algorithms; and using the *Top 20* ranked features from the *GainRatio* measure outperformed using other top feature selections. Being able to monitor only 20 features out of more than 300 is a very encouraging message, since it is very meaningful in terms of the computer's resources consumption and the ability to monitor and classify the computer state in a reasonable or no time delay.

Having obtained these results, we wanted to measure the capability of the classifiers to classify unknown worms. Unlike in the previous experiments, in which the classifiers were presented with a separate class for each worm activity and had to classify them accordingly, which is challenging, in the following experiments there were only two classes in the training set: *worm* activity and *none* activity. The training sets had four worms and the test set consisted only of the excluded *worm* and the *none-activity*. We found that the level of detection accuracy for each worm varies from algorithm to algorithm. In the first experiment ($e3_1$), in which we trained the classifiers only on one dataset, the performance of the *ANN* was significantly better than the others, while in the previous experiments ($e2_1$, $e2_2$) the *ANN* performed at a relatively low level of accuracy.

Finally, in the last experiment, ($e3_2$), in which we trained the classifiers on the entire datasets while excluding a worm, the results were quite good. In this set of experiments, we ran the classification algorithms in their best top feature selection (top 20), for which above 85% accuracy had been achieved. We found that specific algorithms classify specific worms even better, exceeding the 95% level of accuracy. This may lead to the need to use an ensemble of classifiers to achieve a higher level of accuracy for instances of all potential worm classes.

In general *Bayesian Networks* resulted constantly in very good results. We believe that this is a result of the consideration of the dependency within features in the Bayesian networks, which may be very relevant in this task.

To conclude, we have shown that, based on the presented evaluation, it is possible to detect a previously un-encountered instance of a *worm activity* using classification algorithms. In order to attain a high level of accuracy in different types of computers, which is an essential requirement in real life, the training set should include samples taken from several computers (i.e., different configurations) having different types of background user activities. This approach allows classifying previously un-encountered instances of worms at a high level of accuracy; however, the accuracy is shown to vary according to the classification algorithm used and the worm.

6. Future Work

Based on these encouraging results, we plan to perform a wider evaluation using more types of worms and an ensemble of classifiers. We are in the process of broadening the application of this approach to other types of malicious codes, such as Trojans and backdoors, which are expected to present a greater challenge since their activity influence is expected to be less intensive than that of worms. We are also developing a temporal *data mining algorithm* which is expected to enable more sophisticated dynamic behavior and its detection along time in such activities.

7. References

1. Abou-Assaleh, T., Cercone, N., Keselj, V., and Sweidan, R. (2004) N-gram based Detection of New Malicious Code, Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
2. Apap, F., Honig, A., Hershkop, S., Eskin, E., and Stolfo, S.J. (2002) Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses, *In Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002)* Zurich, Switzerland: October 16-18, 2002
3. 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 (SDM 2001), Chicago, USA
4. Bishop, C.(1995) Neural Networks for Pattern Recognition. Clarendon Press, Oxford.
5. Susan M. Bridges and M. Vaughn Rayford, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in Proceedings of the Twenty-third National Information Systems Security Conference. National Institute of Standards and Technology, Oct. 2000.
6. M. Botha and R. von Solms, "Utilising fuzzy logic and trend analysis for effective intrusion detection," Computers & Security, vol. 22, no. 5, pp. 423–434, 2003.
7. CERT. CERT Advisory CA-2000-04, Love Letter Worm, http://www.cert.org/advisories/ca-2000-04.html
8. Demuth, H. and Beale, (1998) M. Neural Network toolbox for use with Matlab. The Mathworks Inc., Natick, MA.
9. John E. Dickerson and Julie A. Dickerson, "Fuzzy network profiling for intrusion detection," in Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, pp. 301–306, Atlanta, USA, July 2000.
10. Domingos, P., and Pazzani, M. (1997) On the optimality of simple Bayesian classifier under zero-one loss, *Machine Learning*, 29:103-130.
11. *Computer Worms: Past, Present and Future.* Craig Fosnock, East Carolina University (2005)

12. Mark Handley, Christian Kreibich and Vern Paxson, Network Intrusion Detection: Evasion, Traffic Normalization, Proc. 10th USENIX Security Symposium, 2001.

13. P. Z. Hu and Malcolm I. Heywood, Predicting intrusions with local linear model, in Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 1780–1785. IEEE, IEEE, July 2003.

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

15. Kienzle, D.M. and Elder, M.C. (2003) Recent worms: a survey and trends. *In Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 1--10. ACM Press, October 27, 2003.

16. Kohavi, R., (1995) A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, *International Joint Conference in Artificial Intelligence*, 1137-1145, 1995

17. H. Gunes Kayacik, A. Nur Zincir-Heywood, and Malcolm I. Heywood, On the capability of an som based intrusion detection system, in Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 1808–1813. IEEE, IEEE, July 2003.

18. Kolter, J.Z. and Maloof, M.A. (2004). Learning to detect malicious executables in the wild. *In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 470–478. New York, NY: ACM Press.

19. Lee, W., Stolfo, S.J. and Mok, K.W. (1999). A data mining framework for building intrusion detection models. *In Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999

20. J. Z. Lei and Ali Ghorbani, "Network intrusion detection using an improved competitive learning neural network," in Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR04), pp. 190–197. IEEE-Computer Society, IEEE, May 2004.

21. Richard P. Lippmann, Isaac Graf, Dan Wyschogrod, Seth E. Webster, Dan J. Weber, and Sam Gorton, "The 1998 DARPA/AFRL Off-Line Intrusion Detection Evaluation," First International Workshop on Recent Advances in Intrusion Detection (RAID), Louvain-la-Neuve, Belgium, 1998.

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

23. Mitchell T. (1997) Machine Learning, *McGraw-Hill*.

24. Moore D., Paxson V., Savage S., and Shannon C., Staniford S., and Weaver N. (2003) Slammer Worm Dissection: Inside the Slammer Worm, *IEEE Security and Privacy, Vol 1 No. 4*, July-August 2003, 33-39.

25. Moore, D., Shannon, C., and Brown, J. (2002) Code Red: a case study on the spread and victims of an internet worm, *Proceedings of the Internet Measurement Workshop 2002*, Marseille, France, November 2002.

26. B. Mukherjee, L. Heberlein, and K. Levitt, "Network Intrusion Detection," IEEE Network, 8(3), pp. 26-41, May/Jun. 1994.

27. Mukkamala, S. and Sung, A. (2003) Identifying Significant Features for Network Forensic Analysis Using Artificial Intelligent Techniques, *International Journal of Digital Evidence*, Winter 2003, Vol. 1, Issue 4.

28. Pearl J., (1986) Fusion, propagation, and structuring in belief networks. *Artificial Intelligence* **29**(3):241–288.

29. Quinlan, J.R. (1986) Induction of decision trees. *Machine Learning*, 1:81-106.

30. Quinlan, J.R. (1993). C4.5: programs for machine learning. *Morgan Kaufmann Publishers Inc.*, San Francisco, CA, USA.

31. Schultz, M., Eskin, E., Zadok, E., and Stolfo, S. (2001) Data Mining Methods for Detection of New Malicious Executables, *Proceedings of the IEEE Symposium on Security and Privacy*, 2001, pp. 178--184.

32. G. Tandon and P. Chan. Learning rules from system call arguments and sequences for anomaly detection. In ICDM Workshop on Data Mining for Computer Security (DMSEC), pages 20–29, 2003.

33. C.Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In IEEE Symposium on Security and Privacy, pages 133–145, 1999.

34. A.Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection (RAID)*, LNCS 1907, pages 110–129, Toulouse, France, October 2000. Springer-Verlag.Weaver, N. Paxson, V. Staniford, and S. Cunningham, R. (2003) A Taxonomy of Computer Worms, *Proceedings of the 2003 ACM workshop on Rapid Malcode*, Washington, DC, October 2003, pages 11-18

35. Witten, I. and Frank E. (1999) Data Mining: Practical Machine Learning Tools and Technique with Java Implementations, *Morgan Kaufman Pub.*, San Francisco.

36. I. Yoo. “Visualizing windows executable viruses using self-organizing maps,” Proc. 2004 ACM Workshop on Visualization and Data Mining for Computer Security. 2004.

37. Ste. Zanero and Sergio M. Savaresi, “Unsupervised learning techniques for an intrusion detection system,” in Proceedings of the 2004 ACM symposium on Applied computing, pp. 412–419, Nicosia, Cyprus, Mar. 2004. ACM Press.

Appendix I – the list of Top 15 ranked features resulted from the *GainRatio* measure.

1. **ICMP: Sent Echo/sec**
   Shows the rate at which Internet Control Message Protocol (ICMP) Echo messages are sent. ICMP is mainly used by networked computer's operating systems to send error messages that indicating, for example, that a requested service is not available or that a host or router could not be reached. Echo message is generally used in order to check the presence of the remote computer and the quality of connection.
2. **ICMP: Messages Sent/sec**
   Shows the rate at which the server attempts to send messages. The rate includes messages sent in error.
3. **ICMP: Messages/sec**
   Shows the total rate at which ICMP messages are sent and received by the entity. The rate includes messages received or sent in error.
4. **TCP: Connections Passive**
   Shows the number of times that TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state. SYN_RCVD state means that the message has been received by the server.
5. **TCP: Connection Failures**
   Shows the number of times that TCP connections have made a direct transition to the CLOSED state from the SYN-SENT or SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state. In other words this counter counts the number of inappropriate jumps from state to state during the session.
6. **TCP: Connections Active**
   Shows the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state i.e., the connection was initiated by the computer.
7. **TCP: Segments Retransmitted/sec**
   Shows the rate at which segments containing one or more previously transmitted bytes are retransmitted.
8. **Memory: Free System Page Table Entries**
   Shows the number of page table entries not currently in used by the system. This counter displays the last observed value only.
9. **ICMP: Received Echo Reply/sec**
   Shows the rate at which ICMP Echo Reply messages are received. Echo Reply message is the message remote computer returns after it receives the Echo message from the server.
10. **ICMP: Messages Received/sec**
    Shows the rate at which ICMP messages are received. The rate includes messages received in error.
11. **ICMP: Received Echo/sec**
    Shows the rate at which ICMP Echo messages are received.
12. **ICMP: Sent Echo Reply/sec**
    Shows the rate at which ICMP Echo Reply messages are sent.
13. **UDP: Datagrams No Port/sec**
    Shows the rate at which UDP data grams are received which there was no application at the destination port.
14. **Process: Total Thread Count**
    Shows the number of threads currently active in this process. An instruction is the basic unit of execution in a processor, and a thread is the object that executes instructions. Every running process has at least one thread.
15. **System: Threads**
    This is the number of threads in the computer at the time of data collection. This is an instantaneous count, not an average over the time interval.