Host Based Intrusion Detection using Machine Learning

Robert Moskovitch, Shay Pluderman, Ido Gus, Dima Stopel, Clint Feher, Yisrael Parmet, Yuval Shahar and Yuval Elovici
Deutsche Telekom Laboratories at Ben-Gurion University
Ben Gurion University, Be'er Sheva, 84105, Israel
{robertmo,shaipl,gus,stopel,clint,iparmet,yshahar, elovici}@bgu.ac.il

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 malcode binary code signatures. During the time between its appearance and an update being sent to anti-virus tools, a new worm can infect many computers and cause significant damage. We present a new host-based intrusion detection approach, based on analyzing the behavior of the computer to detect the presence of unknown malicious code. The new approach consists on classification algorithms that learn from previous known malcode samples which enable the detection of an unknown malcode. We performed several experiments to evaluate our approach, focusing on computer worms being activated on several computer configurations while running several programs in order to simulate background activity. We collected 323 features in order to measure the computer behavior. Four classification algorithms were applied on several feature subsets. The average detection accuracy that we achieved was above 90% and for specific unknown worms even above 99%.**

Keywords-component; Malicious code detection; worms;

I. INTRODUCTION

The detection of malicious code (malcode) transmitted over computer networks have been researched intensively in recent years. One type of abundant malcode is *worms*, which proactively propagate across networks while exploiting vulnerabilities in operating systems and programs. Other types of malcode include computer *viruses*, *Trojan horses, spyware, and adware*. In this study we focus on worms, though we plan to extend the proposed approach to other types of malcodes.

Nowadays, excellent technology (i.e., antivirus software packages) exists for detecting and eliminating *known* malicious code. Typically, *antivirus software packages* inspect each file that enters the system, looking for known signs (signatures) which uniquely identify an instance of known malcode. Nevertheless, antivirus technology is based on prior explicit knowledge of malcode signatures and cannot be used for detecting unknown malcode. Following the appearance of a new *worm*, a patch is provided by the operating system provider (if needed) and the antivirus vendors update their signatures-base accordingly. This solution is not perfect since worms propagate very rapidly and by the time the local antivirus software tools have been updated, very expensive damage has already been inflicted by the worm [1].

Intrusion detection, commonly at the network level, called *network based intrusion detection* (*NIDS*), was researched substantially [2]. 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. Detection systems at the host level, called *Host-based Intrusion Detection* (*HIDS*), are currently very limited in their ability to detect unknown malcode.

Recent studies have proposed methods for detecting unknown malcode using Machine Learning techniques. Given a training set of malicious and benign executables binary code, a classifier is trained to identify and classify unknown malicious executables as being malicious [3,4,5].

In this study, we focus on detecting the presence of a worm based on the computer's (host) behavior. Our suggested approach can be classified under *HIDS*. The main contribution of our approach is that the knowledge is acquired automatically using inductive learning, given a dataset of known worms (avoids the need for *manual* acquisition of knowledge). While the new approach does not prevent infection, it enables a fast detection of an infection which may result in an alert, which can be further reasoned by the system administrator. Further reasoning based on the network-topology can be performed by a network and system administration function, and relevant decisions and policies, such as disconnecting a single computer or a cluster, can be applied.

Generally speaking, malcode within the same category (e.g., *worms, Trojans, spyware, adware*) share similar characteristics and behavior patterns. These patterns are reflected by the infected computer's behavior. Thus, we hypothesize that it is feasible to learn the computer behavior having the presence of a certain type of malcode, which can be measured through the collection of various parameters along time (CPU, Memory, etc..). In the proposed approach, a classifier is trained with computer measurements from infected and not infected computers. Based on the generalization capability of the classification algorithm, we argue that a classifier can further detect previously unknown worm activity. Nevertheless, this approach may be affected by the variation of computer and application configurations as well as user behavior on each computer. In this study, we investigate whether an unknown worm activity can be detected, at a high level of accuracy, given the variation in hardware and software environmental conditions on individual computers, while minimizing the set of features required.

The rest of the article is structured as follows: in section 2, a survey of the relevant background for this work is presented. The methods used in this study are described in section 3, followed by the description of the experiments design in section 4. In section 5 we present the results and conclude with section 6.

II. Background and Related Work

A. Malicious Code and Worms

The term 'malicious code' (malcode) refers to a piece of code, not necessarily an executable file, intended to harm, whether generally or in particular, a specific owner (host). The approach suggested in this study aims at detecting any malcode activity, whether known or unknown. However, since our preliminary research is on worms, we will focus on them in this section.

Kienzle and Elder [7] define a *worm* by several aspects through which it can be distinguished from other types of malcode: 1) *Malicious code* – worms are considered malicious in nature; 2) *network propagation or Human intervention* – 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 activity to propagate; 3) *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* [8] worm. Different purposes and motivations stand behind worm developers [9] including: *Experimental curiosity* (*ILoveYou* worm [10]); *pride and power* leading programmers to show off their knowledge and skill through the harm caused by the worm; *commercial advantage*, *extortion* and *criminal gain*, *random* and *political protest*, and *terrorism* and *cyber warfare*. The existence of all these types of motivation indicates that computer worms are here to stay as a network vehicle serving different purposes and implemented in different ways. To address the challenge posed by worms effectively, meaningful experience and knowledge should be extracted by analyzing known worms. Today, given the known worms, we have a great opportunity to learn from these examples in order to generalize. We argue that data mining methods can be a very useful in learning and generalizing from previously encountered worms, in order to classify unknown worms effectively.

B. Detecting Malicious Code Using Data Mining

Data mining has already been used for detecting and protecting against malicious codes. A recent survey on intrusion detection systems [2] summarizes recently proposed applications of data mining for 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 use in *anomaly detection* [11], 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 implementation in intrusion detection systems, evaluating their approach on the DARPA98 [12] benchmark.

A Naïve Bayesian classifier was suggested in [2], referring to its implementation within the ADAM system, developed by Barbara et al. [13], which had three main parts: (a) a network data monitor listening to TCP/IP protocol; (b) a data mining engine which enables acquisition of 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 were proposed for detecting malicious code (Artificial Neural Networks (*ANN*) [14,15,16], Self Organizing Maps (*SOM*) [17] and fuzzy logic [18,19,20])

III. Methods

The goal of this study was to assess the feasibility of detecting *unknown* malicious code, in particular computer worms, based on the computer's 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 (simulated a real internet network in order to allow worms to try to propagate). 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 collected from a computer having certain hardware configuration and certain specific background activity, able to classify correctly the behavior of a computer having other configurations? In order to answer this question we designed several experiments. We created eight datasets using 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.

A. DataSet Creation

Since there is no benchmark dataset which could be used for this study, we created our own dataset. A network with various computers (configurations) was deployed, enabling us to inject worms, and monitor the computer behavior and log the measurements.

1) Environment Description

The lab network consisted on seven computers, which contained heterogenic hardware, and a server computer simulating the internet. We used the *windows performance counters*[1], which enable monitoring system features that appear in these main categories (including amount of features in parenthesis): *Internet Control Message Protocol* (27), *Internet Protocol* (17), *Memory* (29), *Network Interface* (17), *Physical*

[1]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/counter/counters2_lbfc.asp

This work was supported by Deutsche Telekom Co.

Disk (21), *Process* (27), *Processor* (15), *System* (17), *Transport Control Protocol* (9), *Thread* (12), *and User Datagram Protocol* (5). In addition we used *VTrace* [21], 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, which were aggregated into the same fixed intervals, and merged with the *windows performance* log files. These eventually included a vector of 323 features for every second.

2) Injected Worms

While selecting worms from the wild, our goal was to choose worms that differ in their behavior, from among the available worms. Some of the worms have a heavy payload of Trojans to install in parallel to the distribution process upon the network; others focus only on distribution. Another aspect is having different strategies for IP scanning which results in varying communication behavior, CPU consumption and network usage. 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[234]. We briefly describe the five worms we used:

(1) ***W32.Dabber.A*** scans IP addresses randomly. It uses the W32.Sasser.D worm to propagate and opens the FTP server to upload itself to the victim computer. Registering itself enables its execution on the next user login (human based activation). It drops a backdoor, which listens on a predefined port. This worm is distinguished by its use of an external worm in order to propagate.

(2) ***W32.Deborm.Y*** is a self-carried worm, which prefers local IP addresses. It registers itself as an MS Windows service and is executed upon user login (human based activation). This worm contains three Trojans as a payload: Backdoor.Sdbot, Backdoor.Litmus, and Trojan.KillAV, and executes them all. We chose this worm because of its heavy payload.

(3) ***W32.Korgo.X*** is a self carrying worm which uses a totally random method for IP addresses scanning. It is self-activated and tries to inject itself as a function to MS Internet Explorer as a new thread. It contains a payload code which enables it to connect to predefined websites in order to receive orders or download newer worm versions.

(4) ***W32.Sasser.D*** uses a preference for local addresses optimization while scanning the network. About half the time it scans local addresses, and the other half random addresses. In particular it opens 128 threads for scanning the network, which requires a heavy CPU consumption, as well as significant network traffic. It is a self-carried worm and uses a shell to connect to the infected computer's FTP server and to upload itself.

[2] Symantec – www.symantec.com
[3] Kaspersky www.viruslist.com
[4]Macfee http://vil.nai.com

(5) ***W32.Slackor.A*****,** a self-carried worm, exploits MS Windows sharing vulnerability to propagate. The worm registers itself to be executed upon user login. It contains a Trojan payload and opens an IRC server on the infected computer in order to receive orders.

All the worms perform port scanning and possess different characteristics. Further information about these worms can be accessed through libraries on the web[567].

3) Dataset Description

In order to examine 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*, being binary variables. *(1) Computer hardware configuration*: Both computers ran on *Windows XP*, which considered the most widely used operation system, having two configuration types: 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. *(2) Background application*: We ran an application affecting mainly the following features: *Processor object*, *Processor Time* **(**usage of 100%); *Page Faults/sec*; *Physical Disk object***,** *Avg Disk Bytes/Transfer*, *Avg Disk Bytes/Write*, and *Disk Writes/sec*. *(3) 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. Thus, two options were for the *Background Application* and the *User Activity*: presence or absence of each.

TABLE I. THE THREE ASPECTS RESULTING IN EIGHT DATASETS, REPRESENTING A VARIETY OF SITUATIONS OF A MONITORED COMPUTER.

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

We created eight datasets (see table I). 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. We collected the values of the features every second. Thus, each record, containing a vector of measurements and a label, presented an activity along a second labeled by a specific *worm*, or a *none* activity label. Each dataset contained a few thousand (labeled) samples of each

[5] Symantec – www.symantec.com
[6] Kaspersky www.viruslist.com
[7]Macfee http://vil.nai.com

worm or none activity. We therefore had three binary aspects, which resulted in eight possible combinations, shown in Table I, representing a variety of dynamic computer configurations and usage patterns. Each dataset contained monitored samples for each of the five worms injected separately, and samples of a normal computer behavior without any injected worm. Each sample (record) was labeled with the relevant worm (class), or 'none' for "clean" samples.

B. Feature Selection

In Data Mining applications, the large number of features in many domains presents a huge challenge. Typically, some of the features do not contribute to the accuracy of the classification task and may even hamper it. Ideally, we would like to minimize the self-consumption of computer resources required for the monitoring operations (measurements) and the classifier computations. This can be achieved through reduction of the classified features using the feature selection technique. Since this is not the focus of this paper, we will describe the feature selection preprocessing very briefly. In order to compare the performance of the various classification algorithms, we used the filters approach, which is applied on the dataset and is independent of any classification algorithm, in which a measure is calculated to quantify the correlation of each feature with the class (the presence or absence of worm activity). Each feature is ranked which represents its expected contribution in the classification task.

We used three feature-selection measures, which resulted in a list of ranks for each feature-selection measure and an ensemble incorporating all three of them. We used Chi-Square (CS), Gain Ratio (GR), ReliefF implemented in the Weka environment [22] and their ensemble, based on a simple average of the three ranks. In a recent publication we have shown that the best performance was achieved when GainRatio was used, thus here we focus in details on the results achieved using GainRatio and further experiments using these settings. While the feature selection is not the focus of this study, but rather its application, we briefly describe the GainRatio measure.

Gain Ratio (GR), originally presented by Quinlan in the context of decision trees [23], which was designed to overcome a bias in the Information Gain (IG) measure [24], and 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 at 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 Sc. Information Gain measures 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.

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

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

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

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

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

We selected the top 5, 10, 20 and 30 ranked features from the GainRatio measure. We describe the top 5 ranked features later. Eventually we had four feature 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 subsets of optional features.

C. Classification Algorithms

One of the goals of this study was to pinpoint the classification algorithm which provides the highest level of detection accuracy. We employed four commonly used Machine Learning algorithms: *Decision Trees*, *Naïve Bayes*, *Bayesian Networks* and *Artificial Neural Networks*, in a *supervised learning* approach, in which the classification algorithm learns from a provided training set, containing labeled examples.

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

1) Decision Trees

Decision tree learners [23] are a well-established family of learning algorithms. Classifiers are represented as trees whose internal nodes are tests on individual features, and 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*. Modern implementations include pruning, which avoids over-fitting. In this study we evaluated J48, the Weka version of the commonly used C4.5 algorithm [23]. An important characteristic of Decision Trees is the explicit form of their knowledge which can be easily represented as a set of rules.

2) 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. Naive Bayes has been shown empirically to produce good classification accuracy across a variety of problem domains [25]. In this study, we evaluated Naive Bayes, the standard version that comes with Weka.

3) *Bayesian Networks*

Bayesian networks are a form of the probabilistic graphical model [26]. Specifically, a Bayesian network is a directed acyclic graph of nodes with variables and arcs representing dependence among the variables. Like Naïve Bayes, Bayesian networks are based on the Bayes Theorem; however, unlike Naïve Bayes they do not assume that the variables are independent. Actually Bayesian Networks are known for their ability to represent conditional probabilities which are the relations between variables. 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.

4) *Artificial Neural Networks*

An Artificial Neural Network (ANN) [27] is an information processing paradigm that is inspired by the way biological nervous systems (i.e., the brain) are modeled with regard to information processing. 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 ANN have been demonstrated in numerous applications including speech synthesis, medicine, finance and many other pattern recognition problems. For some application domains, neural models show more promise in achieving human-like performance than do more traditional artificial intelligence techniques. All ANN manipulations in this study have been performed within a MATLAB(r) environment using Neural Network Toolbox [28].

IV. Experimental Design

Our main goal in this study 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 in the application of a variety of computers. We defined four hypotheses accordingly:

Hypothesis I: Detection of *known* malicious code, based on a computer's measurements, using machine learning techniques can reach an accuracy level above 90%.

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

Hypothesis III: Reducing the amount of features below *30 features* will enable maintenance of the accuracy provided by the full set of attributes or above.

Hypothesis IV: Detecting *unknown* worms is possible at an accuracy level above 80%.

In addition to these hypotheses, we wanted to identify the best classification algorithms and the best combination of top ranked features and classification algorithm. We start with the definition of the evaluation measures and continue with the experiments we designed for this study.

A. *Evaluation Measures*

For the purpose of evaluation, we used the *True Positive (TP)* measure presenting the rate of instances classified as *positive* correctly, *False Positive (FP)* presenting the rate of *positive* instances misclassified (Equation 7), and the *Total Accuracy* – the rate of the entire *correctly* classified instances, either positive or negative, divided by the entire number of instances, as shown in Equation 8. The actual (^{A}) amount of classifications are represented by XY^A, where Y presents the classification (*positive* or *negative*) and X presents the classification correctness (*true* or *false*).

$$TP=\frac{TP^A}{TP^A+FN^A}; \qquad FP=\frac{FP^A}{FP^A+TN^A}; \qquad (7)$$

$$Total\ Accuracy=\frac{TP^A+TN^A}{TP^A+FP^A+TN^A+FN^A}; \ (8)$$

We also measured a *confusion matrix*, which depicts the number of instances from each class which were classified in each one of the classes (ideally all the instances would be in their actual class).

V. Experiments and Results

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

A. *Experiment I*

To test *hypothesis I*, in which we wanted to test whether based on monitored measurements of a computer behavior a high level of accuracy can be achieved, we unified all the eight datasets into a single dataset, which we called '*All*.' The four classification algorithms were evaluated on the *All* dataset with the *full* set of features. Since the training set and test set were identical, we used *10-fold cross validation* [29], in which the dataset was portioned randomly into *ten* equal partitions. Nine partitions were used for the training set, and the remaining partition used to evaluate the learned model. The process then repeated ten times, leaving out a partition for testing each time.

While *Decision Trees* and *Bayesian Networks* achieved 99% accuracy, ANN achieved 96%, and Baive Bayes achieved 92%. Note that classifying each sample to a specific worm (type) activity, within the five given worms is more challenging than classifying to a generic worm behavior, alerting for an unknown worm which was our final goal. Thus, the results could be improved when evaluating this method as a binary problem, which was encouraging.

B. *Experiment II*

In *hypothesis II* we wanted to estimate the performance variability of the suggested approach given several training sets sampled from a variety of computers, represented by the eight datasets. Thus, we tested whether the accuracy obtained by training a classifier on a training set sampled from a given computer will vary significantly when evaluated on a variety of test sets, for which we designed two experiments:

In the first experiment, called $e2_1$, 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 the training set (i) and test set (j) were the same dataset. Thus, we had a set of eight iterations in which each dataset used for training and evaluated on the eight corresponding datasets, resulting in 64 evaluation runs.

In the second experiment, called $e2_2$, the training set was a unified dataset of seven of the eight datasets, called *all-i* in which *i* refers to the eighth left dataset, which was the test set, resulting in eight iterations of evaluation runs. 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 $e2_2$. To test whether a significant difference exists among the datasets, when used as training sets or as test sets, we performed a statistical homogeneity test.

Table II presents the mean accuracy and the resulting standard deviation in each experiment, including 128 evaluation runs. In the first main column the results of $e2_1$ are presented in two views, when the dataset (on the left) was used as a training set, and when used as test sets. The results of $e2_1$ when focusing on the training sets were not homogenous. The same occurred with the results when grouping by the testing sets, though we found that the training on datasets created in the 'old' computer was significantly better ($\alpha = 0.01$). In $e2_2$, in which the datasets were considered as test sets, the results were statistically significant ($\alpha = 0.05$) homogenous. Note that these tests were performed on the entire experiments of all the classification algorithms, and not specifically on each algorithm. 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.

TABLE II. THE RESULTS ACHIEVED IN $E2_1$ AND $E2_2$. IN EACH COLUMN THE DATASETS FUNCTION DIFFERENTLY, AS TRAINING SET OR TEST SETS. IN E22 THE RESULTS WERE SIGNIFICANTLY HOMOGENOUS (A=0.01).

Experiment	**$e2_1$**		**$e2_2$**
Dataset	***As training***	***As test***	***As test***
o	0.68 ± 0.23	0.73 ± 0.23	0.78 ± 0.22
ou	0.76 ± 0.22	0.73 ± 0.22	0.82 ± 0.18
oa	0.73 ± 0.21	0.73 ± 0.23	0.81 ± 0.18
oau	0.77 ± 0.21	0.72 ± 0.21	0.81 ± 0.19
n	0.61 ± 0.24	0.64 ± 0.22	0.71 ± 0.20
nu	0.76 ± 0.21	0.72 ± 0.22	0.82 ± 0.22
na	0.70 ± 0.21	0.73 ± 0.24	0.86 ± 0.17
nau	0.73 ± 0.22	0.71 ± 0.23	0.79 ± 0.20
average	0.72 ± 0.22	0.72 ± 0.22	0.80 ± 0.19

To test hypothesis III, in which we wanted to determine the optimal number of features required to achieve the highest accuracy, we used the results of $e2_1$ and $e2_2$, calculating the mean and variance of the top selection options, Top 5, 10, 20, 30 or full, and the classification algorithms.

Table III 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 significantly ($\alpha = 0.01$). On average the Top20 features subset outperformed the other top selection features subset significantly ($\alpha = 0.01$), out of the Top10. Bayesian Networks applied to the Top20 features outperformed all the other combinations.

TABLE III. THE RESULTS OF E21FOR EACH CLASSIFIER AND TOP FEATURES. BAYESIAN NETWORKS OUTPERFORMED THE OTHER CLASSIFIERS AND THE TOP20 WAS THE BEST FEATURES SUBSET.

	ANN	**BN**	**DT**	**NB**	**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
Avg	0.71±0.20	**0.80±0.11**	0.61±0.22	0.74±0.17	0.72±0.18

Table IV shows the results of $e2_2$. Each cell in the table presents the mean accuracy of eight evaluations, in which each classifier was trained on a training set consisting of seven datasets, and the test set contained only the excluded dataset. Generally, the mean accuracy 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, but not significantly, the performance of the *Top5* was significantly ($\alpha = 0.01$) lower than the others. Note that, while in $e2_1$ all the algorithms favored the *Top20*, here it varied mainly between *Top20* and *Top10*, and *ANN* outperformed with the *full* set of features.

TABLE IV. E22 RESULTS FOR EACH CLASSIFIER AND TOP FEATURES. DECISION TREES SLIGHTLY OUTPERFORMED BAYESIAN NETWORKS AND THE OTHER CLASSIFIERS AND THE TOP10 FEATURES OUTPERFORMED THE OTHER.

	ANN	**BN**	**DT**	**NB**	**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

We present here the Top5 features (because of the limited space of the paper) 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 in the Appendix.

C. *Experiment III*

In *hypothesis IV* we wanted to estimate the possibility of classifying an unknown worm when training on data based on a single computer, which was the main and final objective of this study. In this set of experiments we used only the Top20 features, which outperformed in *e2*. The training set included four worms out of the five and the test set included the excluded worm and the none activity samples. This process was done for each worm repeating in five iterations. Note, that in these experiments, unlike in *e2*, in which each worm class was defined separately, there were two classes: (generally) *worm* and *none* activity.

Table V presents the results of *e3* exceeding a 90% accuracy level of detection for each worm by a different classification algorithm (shown in bold). On average the *Decision Trees* and *Bayesian Networks* outperformed the others. 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 low *false positive* rate.

TABLE V. THE RESULTS OF E3. THERE IS A DIFFERENCE IN THE DETECTION ACCURACY OF EACH CLASSIFIER FOR EACH TYPE OF WORM. IN AVERAGE DECISION TREES OUTPERFORMED THE OTHER CLASSIFIERS, WHILE MAINTAINING A 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

VI. DISCUSSION AND CONCLUSIONS

We presented in this paper a *Host-based Intrusion Detection* method for *unknown* malicious code activity. The novelty of the proposed approach is that it is based on monitoring a host's various parameters (as opposed to direct matching of the malcode signature), using various automated classification algorithms. Four hypotheses were investigated, for which a dataset was created and several corresponding experiments were designed. In the first experiment we showed that the detection of *known* worms is feasible at almost 100% accuracy. In the second experiment we wanted to examine the influence of the variance in the training phase and detection phase, of the configuration of a computer and its programs, to determine whether this method can be generalized. We found that training on seven unified datasets was significantly homogenous (α=0.01) based on a homogeneity test, unlike training on a single dataset (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. Based on the results in $e2_1$ and $e2_2$, in general *Bayesian Networks* outperformed the other algorithms; and using the *Top 20* ranked features from the *GainRatio* was the best, which is very encouraging, especially when comparing to more than 300 features in the full set, since it is very meaningful in terms of the computer's resources consumption.

To examine the possibility of classifying unknown worms, unlike in previous experiments, two classes were defined in the dataset, a worm type consisting of the worms' samples and none. 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. Finally, in *e3* above 85% accuracy was achieved in general; Decision Trees achieved

92%, while specific algorithms exceeded the 95% level of accuracy for specific worms. We noticed that the detection of each worm varied within each algorithm, while being different among algorithms, and thus we suggest using 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, which might be explained by the consideration of the dependency within features, unlike other classifiers.

The limitations of this study are the amount of worms and the computer configurations. Note that the worms were selected to provide a reasonable variety and the computers which were used were dramatically different. However, this was enough to achieve statistically significant results.

To conclude, we have shown that, based on the presented evaluation, it is possible to detect previously un-encountered worms using our novel approach, which is based on the computer "behavior" (features). 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).

VI. 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. In addition, 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.

References

[1] Craig Fosnock, Computer Worms: Past, Present and Future. East Carolina University (2005)

[2] 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.

[3] 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.

[4] 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)

[5] Kolter, J.Z. and Maloof, M.A., Learning to detect and classify malicious executables in the wild, *Journal of Machine Learning Research*, 7 (2006) 2721-2744.

[6] 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.

[7] 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.

[8] 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.

[9] 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

[10] CERT. CERT Advisory CA-2000-04, Love Letter Worm, http://www.cert.org/advisories/ca-2000-04.html

[11] 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

[12] 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.

[13] 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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

[24] Mitchell T. (1997) Machine Learning, *McGraw-Hill.*

[25] Domingos, P., and Pazzani, M. (1997) On the optimality of simple Bayesian classifier under zero-one loss, *Machine Learning*, 29:103-130.

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

[27] Bishop, C.(1995) Neural Networks for Pattern Recognition. Clarendon Press, Oxford.

[28] Demuth, H. and Beale, (1998) M. Neural Network toolbox for use with Matlab. The Mathworks Inc., Natick, MA.

[29] 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.