Detection of Unknown Computer Worms Activity Based on Computer Behavior using Data Mining

Robert Moskovitch, Ido Gus, Shay Pluderman, Dima Stopel, Chanan Glezer, Yuval Shahar and Yuval Elovici

Abstract—**Detecting unknown worms is a challenging task. Extant solutions, such as anti-virus tools, rely mainly on prior explicit knowledge of specific worm signatures. As a result, after the appearance of a new worm on the Web there is a significant delay until an update carrying the worm's signature is distributed to anti-virus tools. During this time interval a new worm can infect many computers and create significant damage. We propose an innovative technique for detecting the presence of an *unknown* worm, not necessarily by recognizing specific instances of the worm, but rather based on the computer measurements. We designed an experiment to test the new technique employing several computer configurations and background applications activity. During the experiments 323 computer features were monitored. Four feature selection techniques were used to reduce the amount of features and four classification algorithms were applied on the resulting feature subsets. Our results indicate that using this approach resulted, in above 90% average accuracy, and for specific unknown worms accuracy reached above 99%, using just 20 features while maintaining a low level of false positive rate.**

I. Introduction

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

Manuscript received October 31, 2006. This work was supported by Deutsche Telekom Co.

Robert Moskovitch (corresponding author) is a PhD student, Deutsche Telekom Laboratories at Ben-Gurion University, Be'er Sheva, 84105 Israel. Phone: +972-52-2668071; email: robertmo@bgu.ac.il.

Ido Gus and Shay Pluderman were undergraduate students at the Deutsche Telekom Laboratories at Ben-Gurion University, Ben-Gurion University of the Negev, Be'er Sheva, 84105 Israel; email: gus@bgu..ac.il, shaipl@gbgu.ac.il.

Dima Stopel is an M.Sc student, Deutsche Telekom Laboratories at Ben-Gurion University, Be'er Sheva, 84105 Israel. email: stopel@cs.bgu.ac.il.

Chanan Glezer is with Deutsche Telekom Laboratories at Ben-Gurion University, Ben-Gurion University of the Negev, Be'er Sheva, 84105 Israel; email: chanan@bgu.ac.il.

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

Yuval Elovici, is the Head of the the Deutsche Telekom Laboratories at Ben-Gurion University, Ben-Gurion University of the Negev, Be'er Sheva, 84105 Israel; email: elovici@bgu.ac.il.

Nowadays, excellent technology (i.e., antivirus software packages) exists for detecting *known* malicious code. Typically, *antivirus software packages* inspect each file that enters the system, looking for known signs (signatures) uniquely identifying an instance of 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* instance, a patch is provided by the operating system provider and the antivirus vendors update their signatures-base accordingly. This solution is not perfect, however, since worms propagate very rapidly. By the time the antivirus software has been notified about the new worm, a very expensive damage has already been inflicted [1].

Intrusion detection, commonly made at the network level, called *network based intrusion detection* (*NIDS*), was researched substantionally [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, called *Host Based Intrusion Detection* (*HIDS*). *HIDS* are detection systems consist on monitoring activities at a host. *HIDS*, commonly, consist on comparing 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 limitations such as the fast change in the file system. The main problem is the detection knowledge maintenance, which is acquired commonly manually by the domain expert.

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 and learns to identify and classify unknown malicious executables as being malicious [3,4,5]. While this approach is a potentially good solution, it is not complete since it can detect only executable files, and malcodes located entirely in the memory, such as the *Slammer* worm [6], cannot be detected using this technique. Moreover, any technique can be sabotaged by a malcode.

Our suggested approach can be classified under *HIDS*, but the novelty here is that it is based on the computer measurements and that the knowledge is acquired automatically using inductive learning, given a dataset of known worms. This approach avoids the need in manual knowledge acquisition, which is sometimes unavailable, especially in such approach, as well as the knowledge maintenance. While this approach doesn't prevent the infection it enables a fast detection of the infection which may result in alert, which can be further reasoned at the network level. Further reasoning based on the network-topology can be performed by a network administration function, and relevant decisions and policies, such as disconnecting a single computer or a cluster, can be further implemented. In this study, we focus on a proposed technique that enables the detection of unknown worms based on a single computer's (host) behavior.

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 as represented by its measurements. Based on these common characteristics, we suggest that an unknown worm can be detected based on the computer's behavior using Data Mining techniques. In the proposed approach, a classifier is trained on computer measurements of known *worm* and *non-worm* behaviors. 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 research questions and the corresponding experimental plan in section 4. Finally, results are presented, followed by a discussion and conclusions.

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 we originated our research on worms, we will focus on them in this section.

Kienzle and Elder [7], define a *worm* by several aspects through which 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 worms developers [9] including: *Experimental curiosity* which can lead any person to create a worm, such as the *ILoveYou* worm [10]. *Pride and power* leading programmers to show-off their knowledge and skill exhibited 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 effectively address the challenge posed by worms, meaningful experience and knowledge should extracted by analyzing known worms. Current days 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 to learn and generalize from previous see worms, in order to classify effectively unknown worms, as a last detection method.

B. Detecting Malicious Code Using Data Mining

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 [2] 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* [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 further implemented in intrusion detection systems. They evaluated their approach on the DARPA98 [12] benchmark test collection, which is a standard benchmark of network data for intrusion detection research.

A Naïve Bayesian classifier was suggested in [2] referring to its implementation within the ADAM system developed by Barbara et al. [13]. 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*) [14,15,16], Self Organizing Maps (*SOM*) [17] and fuzzy logic [18,19,20].

III. Methods

The general goal of this study is to assess the viability of employing Data Mining techniques in detecting the existence of unknown worms in an individual computer host based on it's behavior (measurements). In order to create a testing environment, we have built a local network of computers, which enabled us to inject worms into a controlled environment, while monitoring the computers and collecting measurements. Preliminary results were very encouraging, but we wanted to estimate the influence of the environment in which the training set was produced on the detection accuracy in another environment. In an extensive experiment we had showed elsewhere [21] that there is no significant influence. Moreover, when a classifier trained on an old computer it had better detection accuracy then the opposite. In this study we want to further investigate the capability of detecting unknown malicious code.

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, into which we could inject worms. The network was a controlled environment, in which we could monitor the computer features and document the measurements into log files.

1) Environment Description

The lab network consisted of 7 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 (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). In addition we used *VTrace* [22]. 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 choosing worms that differ in their behavior, within 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. 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 log in (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,. This worm registers itself as an MS Windows service and is being executed upon user login (human based activation). 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 its heavy payload.

(3) *W32.Korgo.X* is a self carrying worm which uses 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 to connect 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. It scans about half of the time local addresses and in the other half random addresses. In particular it opens 128 threads for scanning the netwrok, which requires a heavy CPU consumption, as well as significant network traffic. It is a self carried worm uses a shell to connect to the infected computer's FTP server and to upload itself.

(5) *W32.Slackor.A* is 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

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

[2] Symantec – www.symantec.com
[3] Kasparsky www.viruslist.com
[4]Macfee http://vil.nai.com
[5] Symantec – www.symantec.com
[6] Kasparsky www.viruslist.com
[7]Macfee http://vil.nai.com

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. The two options in the *Background Application* and *User Activity* were presence or absence of the user activity.

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. We therefore had three binary aspects, which resulted in 8 possible combinations 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. Moreover, in our approach reducing the amount of features, while maintaining a high level of detection accuracy, is crucial for meeting computer performance and resource consumption. 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 classification algorithms, we used the *filters* approach, which is applied on the dataset and is independent of any classification algorithm (unlike wrappers, in which the best subset is chosen upon an iterative evaluation experiment). Under *filters*, a measure is calculated to quantify the correlation of each feature with the class (in our case, the presence or absence of a worm activity). Each feature receives a *rank* representing its expected contribution in the classification task. Eventually, the top ranked features were selected.

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 [23] and their ensemble, based on a simple average of the three ranks. We took the highest ranked (top) features 5, 10, 20 and 30 from each feature selection measure ranked list. Finally we had four subsets and the *full* features set, for whom we had eight datasets each resulting in 17 datasets. While the feature selection isn't the focus of this study, but its application we briefly describe the measures we used.

Chi-Square measures the lack of independence between a feature f and a class c_i and can be compared to the chi-square distribution with one degree of freedom to judge extremeness. Equation 1 shows how the chi-square measure is defined and computed, where N is the total number of documents and f refers to the presence of the feature (and $\bar{f}$ its absence), and c_i refers to its membership in c_i.

$$\chi^2(f,c_i)=\frac{N[P(f,c_i)P(\bar{f}_i,\bar{c}_i)-P(f,\bar{c}_i)P(\bar{f},c_i)]^2}{P(f)P(\bar{f})P(c_i)P(\bar{c}_i)} \quad (1)$$

Gain Ratio originally presented by Quinlan in the context of decision trees [24], which was designed to overcome a bias in the *Information Gain (IG)* measure [25], 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 3 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 2. These equations refer to discrete values; however, it is possible to extend it to continuous values attribute.

$$IG(S,A)=E(S)-\sum_{v\in V(A)}\frac{|S_v|}{|S|}\cdot E(S_v) \quad (2)$$

$$E(S)=\sum_{c\in C}-\frac{|S_c|}{|S|}\cdot\log_2\frac{|S_c|}{|S|} \quad (3)$$

The *IG* measure favors features having a high variety of values over those with only a few. *GR* overcomes this problem by considering how the feature splits the data (Equations 4 and 5). S_i are *d* subsets of examples resulting from portioning *S* by the *d-valued* feature *A*.

$$GR(S,A)=\frac{IG(S,A)}{SI(S,A)} \tag{4}$$

$$SI(S,A)=-\sum_{i=1}^{d}\frac{|S_i|}{|S|}\cdot\log_2\frac{|S_i|}{|S|} \tag{5}$$

ReslifF [26] estimates the quality of the features according to how well their values distinguish between instances that are near each other. Given a randomly selected instance *x*, from a dataset *s* with *k* features, Relief searches the data set for its two nearest neighbors. From the same class, called nearest hit *H* and from different class, called nearest miss *M*. The quality estimation *W*[*Ai*] is stored in a vector of the features *Ai*, based on the values of a difference function *diff() given x, H and M* as shown in equation 6..

$$diff(A_i,x_{1i},x_{2i})=\begin{cases}|x_{1i}-x_{2i}| & \text{if } \mathrm{A_i} \text{ is numeric,}\\ 0 & \text{if } \mathrm{A_i} \text{ is nominal} \,\&\, x_{1i}=x_{2i},\\ 1 & \text{if } \mathrm{A_i} \text{ is nominal} \,\&\, x_{1i}\neq x_{2i},\end{cases} \tag{6}$$

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 [24] 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 [24]. 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 [27]. In this study, we evaluated Naive Bayes, the standard version that comes with Weka.

3) Bayesian Networks

Bayesian networks is a form of the probabilistic graphical model [28]. 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) [29] is an information processing paradigm that is inspired by the way biological nervous systems (i.e., the brain) are modeled with regards 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 [30].

IV. Experimental Design

In the first part of the study, we wanted to identify the best feature selection measure, the best classification algorithm and the minimal features required to maintain a high level of accuracy. In the second part we wanted to measure the ability to classify unknown worms using a training set of known worms. In order to answer these questions we designed two experimental plans, based on *seventeen* sets of subsets resulted from the four feature selection measures, from which we extracted the *Top 5*, *10*, *20* and *30*, and the *full* feature set, in which each set appeared in eight created datasets (described earlier), for the evaluation. After evaluating all the classification algorithms on the sets of datasets, we selected the best feature selection and the top features to evaluate the unknown worms' detection.

A. Experiment I – Best feature selection

To determine which feature selection measure, top feature selection and classification algorithm are the best, we had a wide set of experiments, in which we evaluated each classification algorithm, feature selection and top selection combination. In this experiment, called *e1*, we trained each classifier on a single dataset *i* and tested on each one (*j*) of the *eight* datasets. Thus, we had a set of 8 iterations in which a dataset was used for training, and 8 corresponding evaluations which were done on each one of the datasets, resulting in 64 evaluation runs, for each one of the combination of classification algorithm, feature selection measure and top feature selection. When $i = j$, we used *10 folded cross validation* [31], in which the dataset is partitioned into 10 partitions and repeatedly the classifier is trained on 9 partitions and tested on the 10^{th}. Note, that the task was to specifically classify the exact worm out of the five or a none (worm) activity, and not generally to a binary classification of "worm" or a "none" activity, which was our final goal in the context of an unknown worm detection. Such conditions while being more challenging were expected to bring more insights.

B. Experiment II – Unknown worms detection

To estimate the potential of the suggested approach in classifying an *unknown worm* activity, which was the main objective of this study, we designed an additional experiment, called *e2*, in which we trained classifiers based on *part* of the (five) *worms* and the *none* activity, and tested on the *excluded worms* (from the training set) and the *none* activity, in order to measure the detection capability of an *unknown worm* and the *none* activity.

In the first experiment ($e2_1$) the training set consisted on *four* worms and the testing set contained the fifth excluded worm, while the *none* activity appeared in both datasets. This process repeated five times for each worm.

To better estimate the generalization based on the amount of worm activity examples in the training set, we extended $e2_1$ to $e2_2$, in which we decreased the amount of worms in the training set and increased the amount of (unknown) worms in the test set. We did it for three options of one to three worms in the training set in addition to $e2_1$, in which there were four worms. The test set included *only* the excluded worms and not the worms presented in the training set since we wanted to measure the detection rate of the unknown specifically. Note, that in these experiments, unlike in *e1*, there were two classes: (generally) *worm,* for any type of worm, and *none* activity. This experiment was evaluated on each classification algorithm, using the outperforming top selected features from *e1*.

C. 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*).

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

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

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. Results

A. Experiment I

Our objective in *e1* was to determine the best feature selection measure, top feature subset size, and classification algorithms. We ran 68 (4 classification algorithms applied to 17 data sets) evaluations (each includes 64 runs), summing up to 4352 evaluation runs. Figure 1 shows the mean performance achieved for each feature selection measure in each top selection. Based on the mean performance of the four classification algorithms *GainRatio* outperformed the other measures in most of the top features selection, while the ensemble outperformed at the *Top5*. Unlike the independent measures in which there was a monotonic growth when features added, in the ensemble a monotonic slight decrease was observed as more features used. The

Top20 features outperformed in general (by averaging) and in *GainRatio* in particular.

Figure 2 shows the same results but presenting the mean performance of the classification algorithms and the top feature subset size. *Bayesian Networks* outperforms for any amount of top selection, and in average the *Top20* outperformed the other top selections. To emphasize the significant features types the *Top5* of the *GainRatio* included: A_1ICMP: *Sent_Echo_sec*, *Messages_Sent_sec*, *Messages_sec*, and A_1TCP: *Connections_Passive* and *Connection_Failures*, which are windows performance counters, related to ICMP and TCP, describing general communication properties.

B. Experiment II

Based on the results achieved in *e1*, in which the *Top20* from *GainRatio* outperformed on average, we used only this features subset in *e2*. Table 1 presents a detailed report on the results from $e2_1$. Each row presents the results achieved when worm *i* (see section III.A.2) was in the test set and the columns refer to the classification algorithms. On average, while the *Decision Trees* outperformed the other classification algorithms, each classification algorithm outperformed in detecting different unknown worms and the *false positive* rates in all the classification algorithms were low.

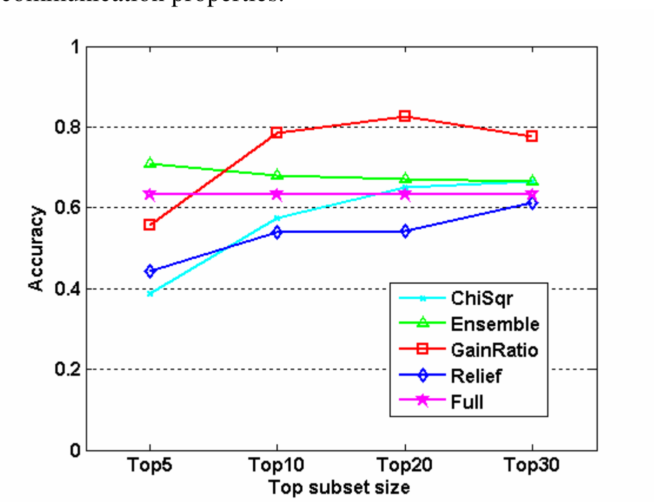

Fig. 1. The mean performance achieved by each feature selection measure, and the top ranked features. While *Top20* outperforms for most of the measures, *Top5* outperforms for the *Ensemble*.

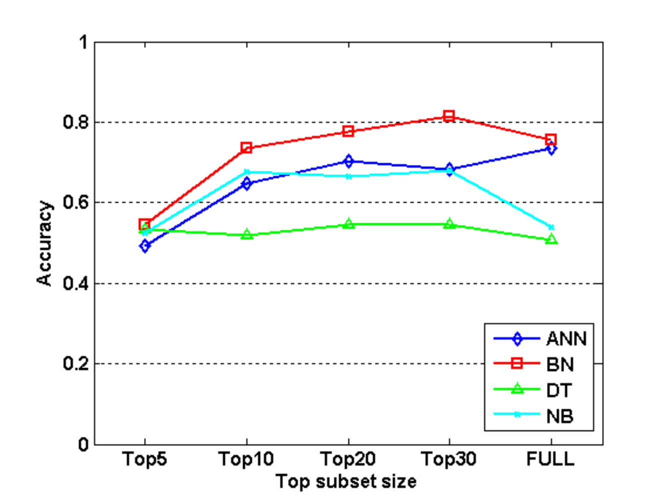

Fig 2. The performance achieved by each classification algorithm and the top ranked selection. *Bayesian Networks* outperformed across all categories. While for most of the algorithms *Top30* and *Top20* achieved similar performance, in the *Bayesian Networks* the *Top30* outperformed.

TABLE I

THE RESULTS OF $E2_1$, SHOW 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.

	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
Avg.	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

Figure 3 presents the results of $e2_2$, in which a monotonic increase in the accuracy is shown. As more worms are in the training set the accuracy is higher. Note that the number of worms in the x axis refer to the number of excluded worms, which were in the test set.

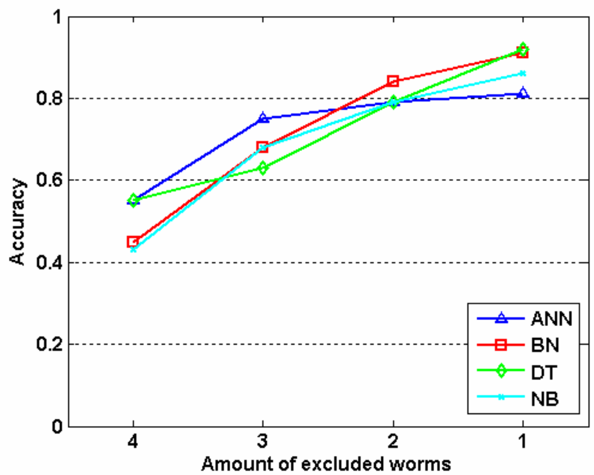

Fig 3. The performance monotonically increases as fewer worms are excluded (and more worms appear in the training set)

I. Conclusions and Future Work

We presented the concept of detecting *unknown* computer worms based on a host behavior, using Data Mining algorithms. Based on the results shown in this study using Data Mining concepts, such as *feature selection* and *classification algorithms*, it is possible to identify the most important computer features in order to detect unknown worm activity, currently performed by human experts. Based on the initial experiment (*e1*), the *GainRatio* feature selection measure was most suitable in this task. In average the *Top20* features produced the highest results. *Bayesian Networks* commonly outperformed other classification algorithms. In the detection of unknown worms (*e2*), the results shown that it is possible to achieve a high level of accuracy (above 90% in average), however, the detection of each worm varied and each classification algorithm seemed to classify better other unknown worms exceeding 99% accuracy. Thus, we are considering applying an ensemble of classifiers to achieve a unified level of accuracy in detecting a variety of worm instances. These results are highly encouraging and show that worms, which commonly spread intensively can be stopped from propagating in real time. The advantage of the suggested approach is the automatic acquisition and maintenance of knowledge, based on inductive learning. This avoids the need in a human expert who is not always available and familiar with the general rules. This is possible these days based on the existing amount of known worms, as well as the generalization capabilities of classification algorithms.

We are currently in the process of extending the amount of worms in the dataset, as well as extending the suggested approach to other types of malicious code using temporal data mining.

Acknowledgment

We would like to thank our undergraduate student, Clint Feher, for providing the worm software and creating the large number of data sets we used in this study.

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. (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.

[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] (133/2006) Robert Moskovitch, Ido Gus, Shay Pluderman, Dima Stopel, Yisrael Fermat, Yuval Shahar and Yuval Elovici, Host Based Intrusion Detection Using Machine Learning, Faculty of Engineering, Ben Gurion Unversity, Israel (2006)..

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

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

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

[26] H Liu, H Motoda and L Yu, A Selective Sampling Approach to Active Selection, *Artificial Intelligence*, 159 (2004) 49-74.

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

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

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

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

[31] 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