Available online at www.sciencedirect.com
ScienceDirect
Computational Statistics and Data Analysis 52 (2008) 4544–4566
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Detection of unknown computer worms based on behavioral classification of the host

Robert Moskovitch, Yuval Elovici, Lior Rokach*

Deutsche Telekom Laboratories at Ben-Gurion University, Ben Gurion University, Be'er Sheva, 84105, Israel

Received 6 January 2008; accepted 30 January 2008
Available online 17 February 2008

Abstract

Machine learning techniques are widely used in many fields. One of the applications of machine learning in the field of information security is classification of a computer behavior into malicious and benign. Antiviruses consisting of signature-based methods are helpless against new (unknown) computer worms. This paper focuses on the feasibility of accurately detecting unknown worm activity in individual computers while minimizing the required set of features collected from the monitored computer. A comprehensive experiment for testing the feasibility of detecting unknown computer worms, employing several computer configurations, background applications, and user activity, was performed. During the experiments 323 computer features were monitored by an agent that was developed. Four feature selection methods were used to reduce the number of features and four learning algorithms were applied on the resulting feature subsets. The evaluation results suggest that by using classification algorithms applied on only 20 features the mean detection accuracy exceeded 90%, and for specific unknown worms accuracy reached above 99%, while maintaining a low level of false positive rate.

1. Introduction

Malicious code (malcode) detection, transmitted over computer networks has been researched intensively in recent years (Kabiri and Ghorbani, 2005). One type of abundant malcode is *worms*, which proactively propagate across networks while exploiting vulnerabilities in operating systems and programs. Other types of malcodes 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 signature-base accordingly. This solution is not perfect since worms propagate very rapidly and by the

* Corresponding author.
E-mail addresses: robertmo@bgu.ac.il (R. Moskovitch), elovici@bgu.ac.il (Y. Elovici), liorrk@bgu.ac.il (L. Rokach).

0167-9473/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2008.01.028

time the local antivirus software tools have been updated, very expensive damage has already been inflicted by the worm (Fosnock, 2005).

Intrusion detection, commonly at the network level, called *network-based intrusion detection* (*NIDS*), was researched substantially (Kabiri and Ghorbani, 2005). 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 executable binary code, a classifier is trained to identify and classify unknown malicious executables as being malicious (Schultz et al., 2001; Abou-Assaleh et al., 2004; Kolter and Maloof, 2006; Caia et al., 2007).

Existing methods rely on the analysis of the binary for the detection of unknown malcode. Some less typical worms are left undetectable. Therefore an additional detection layer at runtime is required. The proposed approach assumes that the malicious actions are reflected in the general behavior of the host. Thus, by monitoring the host, one can inexplicitly identify malcodes. This property can be used as an additional protection layer.

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 in 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 learning algorithm, we argue that a classifier can further detect previously unknown worm activity. Nevertheless, this approach may be affected by the variance in computer and application configurations as well as user activity (running and using various applications) 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 monitored features.

In this paper we introduce three main contributions: We show that current machine learning techniques are capable of detecting and classifying worms solely by monitoring the host activity. Using feature selection techniques we show that a relatively small set of features are sufficient for solving the problem without sacrifice accuracy. We present empirical results from an extensive study of various machine configurations suggesting that the proposed methods achieve high detection rates on previously unseen worms.

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

2. Background and related work

2.1. 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 (2003) define a *worm* by several aspects through which it can be distinguished from other types of malcodes: (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* (Moore et al., 2002) worm. Different purposes and motivations stand behind worm developers (Weaver et al., 2003) including: *Experimental curiosity* (*ILoveYou* worm,; (CERT, 2000); *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 supervised learning methods can be very useful in learning and generalizing from previously encountered worms, in order to classify unknown worms effectively.

2.2. Detecting malicious code using supervised learning techniques

Supervised and unsupervised learning has already been used for detecting and protecting against malicious codes. A recent survey on intrusion detection systems (Kabiri and Ghorbani, 2005; Rokach and Elovici, 2007) summarizes recently proposed applications for recognizing malcodes in single computers and in computer networks. Lee et al. (1999) proposed a framework consisting of a set of algorithms for the extraction of anomalies of user normal behavior for use in anomaly detection, 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 (Lippmann et al., 1998) benchmark.

A Naïve Bayesian classifier was suggested in Kabiri and Ghorbani (2005), referring to its implementation within the ADAM system, developed by Barbara et al. (2001), which had three main parts: (a) a network data monitor listening to TCP/IP protocol; (b) a learning engine which enables acquisition of the association rules from the network data; and (c) a classification module which classifies the nature of the traffic into two possible classes, normal and abnormal, which can later be linked to specific attacks. Other soft computing algorithms were proposed for detecting malicious code: Artificial Neural Networks (ANN) (Zanero and Savaresi, 2004; Kayacik et al., 2003; Lei and Ghorbani, 2004), Self-Organizing Maps (SOM) (Hu and Heywood, 2003) and fuzzy logic (Dickerson and Dickerson, 2000; Bridges and Vaughn Rayford, 2000; Botha and von Solms, 2003).

3. Methods

The goal of this study was to assess the feasibility of detecting *unknown* malicious code, in particular computer worms, based on the computer's behavior (measurements), using machine learning techniques, and the potential accuracy of such methods. In order to create the datasets we built an isolated local network of computers, simulating a real Internet network which allows worms to propagate. This setup enabled us to inject worms into a controlled environment, while monitoring the computer behavior. The monitoring is performed by an agent, developed specifically for this purpose, that measures various parameters and save their values in log files.

In this study we examine whether a classifier, trained on data collected from a computer having a certain hardware configuration and certain specific background activity, is capable of correctly classifying the behavior of a computer having other configurations. In order to answer this question we designed several experiments. We created eight datasets having different configurations, different background applications, and different user activities. Another goal was to select the minimal subset of features which are required to correctly classify new cases. Reducing the number of features used in the model, implies that less monitoring efforts are needed when the proposed approach is served as the basis for an operational system. Finally, we applied four classification algorithms on the given datasets in a varied series of experiments, starting with detecting known worms in different environments and later detecting completely new, previously unseen worms.

Fig. 1 specifies the process that was used in order to perform this study. The upper part refers to the training phase. We collected a set of worms and used them to infect the hosts in the controlled environment. Then an agent, which

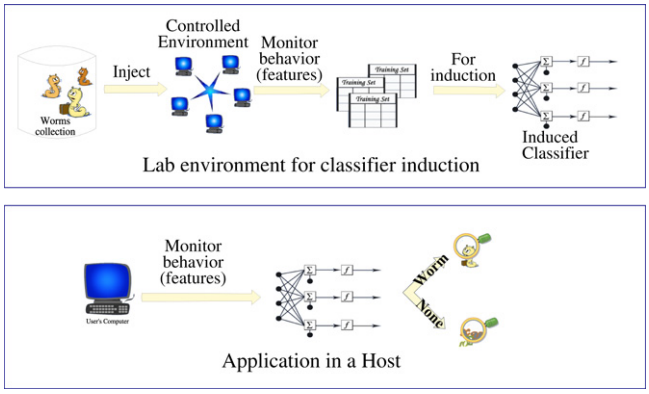

Fig. 1. Outline of the train phase and the test phase.

was installed on each host, recorded the behavior of the host. Based on the collected dataset, we trained the classifiers. The bottom part in Fig. 1 refers to the test phase. In this phase we examine if the induced classifier can be used to identify the existence of unknown worm.

3.1. 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.

Environment description

The lab network consisted of 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 the number of features 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), *and User Datagram Protocol* (5). In addition we used *VTrace* (Lorch and Smith, 2000), a software tool which can be installed on a PC running Windows for monitoring purposes. *VTrace* collects traces of the *file system, the network, the disk drive, processes, threads, interprocess communication, waitable objects, cursor changes, windows, and the keyboard*. The data from the *windows performance counter* were configured to measure the features every second and store them in a log file as vectors. *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.

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 that they have 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 by the presence of which it would be possible 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.[2,3,4] We briefly describe the five worms we used:

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

(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.KillA V, 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 address 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 address 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.
(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 found on the web.

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 operating system, having two hardware configuration types: an "*old*," having Pentium 3 800 MHz CPU, bus speed 133 MHz and memory 512 Mb, and a "*new*," having Pentium 4 3 GHz CPU, bus speed 800 MHz and memory 1 Gb. (2) *Background application activity*: 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. Appendix A specifies the set of features that were examined in this research.

We created eight datasets (see Table 1). 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 min. 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 worm or clean computer. We therefore had three binary aspects, which resulted in eight possible combinations, shown in Table 1, 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.

3.2. Feature selection methods

In many 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. Feature selection is the process of identifying relevant features in the dataset and discarding everything else as irrelevant and redundant. Since feature selection reduces the dimensionality of the data, it enables the classification algorithms to operate more effectively and rapidly. In some cases, classification performance can be improved; in other instances, the obtained

Table 1
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

classifier is more compact and can be easily interpreted. In host-based detection applications there is an additional motivation. Ideally, we would like to minimize the self-consumption of computer resources required for the monitoring operations (measurements), i.e. minimizing the collection of the features.

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 receives a rank which represents its expected contribution in the classification task.

3.2.1. Feature selection methods

We used three feature selection methods, which resulted in a list of ranked features for each feature selection method and an ensemble incorporating all three of them. We used Chi-Square (CS), Gain Ratio (GR) and ReliefF implemented in the WEKA environment (Witten and Frank, 2005) and their ensemble.

Chi-Square

Chi-Square measures the lack of independence between a feature f and a class c_i (such as W32.Dabber.A) and can be compared to the chi-square distribution with one degree of freedom to judge extremeness. Eq. (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.

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

Gain Ratio

Gain Ratio was originally presented by Quinlan in the context of *Decision Trees* (Mitchell, 1997), which was designed to overcome a bias in the *Information Gain* (IG) measure, 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 in classifying the training data. Eq. (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* 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 Eq. (2). These equations refer to discrete values; however, it is possible to extend them to continuous values attribute.

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

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

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 (Eqs. (4) and (5)). S_i are d subsets of examples resulting from

portioning S by the *d-valued* feature A.

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

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

Relief

ReslifF (Pearl, 1986) 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 dataset for its two nearest neighbors from the same class, called nearest hit H, and from a 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 Eq. (6).

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

Features' ensembles

Instead of selecting features based on of the feature selection methods, one can use the ensemble strategy (see for instance Rokach et al. (2007)) which combines the feature subsets that are obtained from several feature selection methods. Specifically, we combine several methods by averaging the features' ranks as shown in Eq. (7):

$$\mathrm{Rank}(fi) = \frac{\sum_{j=1}^{k} \mathrm{rank}^j(fi)}{k} \tag{7}$$

where *fi* is a feature, *filter* is one of the k filtering (feature selection) methods. Specifically in our case $k = 3$.

3.2.2. *Consolidating features from different environments: Averaged versus unified consolidation*

Often when applying a feature selection method, such as the filters approach, the method is applied on the entire dataset aiming to rank the features based on their measured correlation to the class. However, unlike in the common datasets, our dataset consisted of eight datasets coming from different environments, as explained earlier (see Table 1). Since some features might be more important in specific environments and less in others it is not clear how this has to be considered. In this study we propose two approaches to considering and integrating the aspects of the datasets. In the first approach, termed *unified* dataset, we unified all the eight datasets into a single dataset and applied the filter on the unified dataset.

Alternatively, we examined the approach termed *averaged* in which we applied the filter on each one of the eight datasets and computed the average rank for each feature. Note that for averaging the features' ranks obtained from the different environments, we used Eq. (7) again.

Fig. 2 illustrates both approaches, in which the top part refers to the unified approach, where the feature selection (FS) is applied on the unified dataset '*all*' and the averaged approach, at the bottom, in which the feature selection is applied on each dataset and averaged into a rank list.

After applying both approaches we extracted the top ranked features. We took the highest ranked (top) features 5, 10, 20 and 30 from the output of each feature selection method. Finally, we had four feature sets (Top 5, 10, 20, 30) for each of the four filters (fs1, fs2, fs3, ensemble), for each feature consolidation (unified, averaged). On top of that we also examined the *full* feature set (with no feature selection). This totally results in 33 feature sets ($4 \times 4 \times 2 + 1$) as shown in Fig. 3.

3.3. *Classification algorithms*

One of the goals of this study was to pinpoint the classification algorithm that provides the highest level of detection accuracy. We employed four commonly used Machine Learning algorithms: *Decision Trees*, *Naïve Bayes*, *Bayesian*

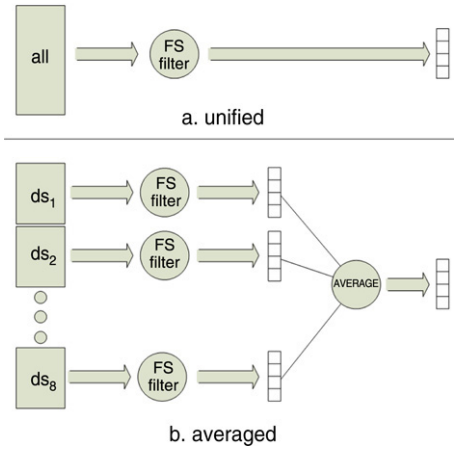

Fig. 2. Unified versus the averaged approach for environment feature consolidation.

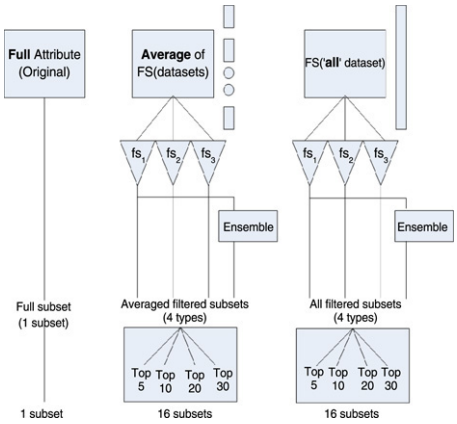

Fig. 3. The creation of 33 feature sets.

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.

Decision trees

Decision tree learners (Quinlan, 1993) 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 top–down 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 (Quinlan, 1993). An important characteristic of Decision Trees is the explicit form of their knowledge which can be represented as a set of if–then rules. This set of rules can be then easily embedded in any existing IDS.

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. Naïve 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, are easily estimated from the training data and used to determine the posterior probability of each class, given a set of features. Naïve Bayes has been shown empirically to produce good classification accuracy across a variety of problem domains (Domingos and Pazzani, 1997). In this study, we evaluated Naïve Bayes, the standard version that comes with WEKA.

Bayesian networks

Bayesian networks are a form of the probabilistic graphical model (Pearl, 1986). 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 as a mechanism for automatically constructing extensions of Bayes Theorem to more complex problems. Bayesian networks were used for modeling knowledge and have been implemented successfully in different domains. We evaluated the Bayesian Network standard version which comes with WEKA.

Artificial Neural Networks

An Artificial Neural Network (ANN) (Bishop, 1995) is an information processing paradigm that is inspired by the way biological nervous systems (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 (Demuth and Beale, 1998).

4. Experimental design

Our main goal in this study was to investigate 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 when applied to a variety of computers. We defined four research questions accordingly:

Q_1: In the detection of *known* malicious code, based on a computer's measurements, using machine learning techniques, what is the achievable level of accuracy?

Q_2: Is it possible to reduce the number of features to below 30, while maintaining a high level of accuracy (compared to the full set of features). Which feature consolidation approach (unified versus averaged) and feature selection method are superior?

Q_3: Will the *computer configuration* and the *computer background activity*, from which the training sets were taken, have a significant influence on the detection accuracy?

Q_4: Is the detection of *unknown* worms possible, based on a training set of known worms?

In addition to these research questions, 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.

4.1. Evaluation measures

For evaluation purposes, we measured the *True Positive Rate* (TPR) measure, which is the number of *positive* instances classified correctly, as shown in Eq. (8), *False Positive Rate* (FPR), which is the number of *negative* instances misclassified (Eq. (8)), and the *Total Accuracy*, which measures the number of absolutely correctly classified instances,

either positive or negative, divided by the entire number of instances shown in Eq. (9). Additionally, we calculated the ROC curves, but we do not present them here because of lack of space.

$$\text{TPR} = \frac{|TP|}{|TP| + |FN|}; \qquad \text{FPR} = \frac{|FP|}{|FP| + |TN|} \tag{8}$$

$$\text{Total accuracy } = \frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|}. \tag{9}$$

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).

In the first part of the study, we wanted to identify the best feature consolidation approach (unified or averaged) and feature selection method, 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 capability of classifying unknown worms based on a training set of known worms. In order to answer these questions we designed two experimental plans, based on 33 datasets, as will be described later. After evaluating all the classification algorithms on the 33 datasets, we selected the best feature selection and the top features to evaluate the unknown worm detection.

4.2. Experiment I

To determine the best combination of feature selection method, number of features, and classification algorithm, we performed a wide set of experiments, in which we evaluated all the combinations of feature selection method, classification algorithm, and number of top features.

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 eight iterations in which a dataset was used for training, and eight corresponding evaluations which were done on each one of the datasets, resulting in 64 evaluation runs. When $i = j$, we used *10 fold cross validation*, in which the dataset is randomly partitioned into ten partitions and repeatedly the classifier is trained on nine partitions and tested on the tenth. Each evaluation run (out of the 64) was repeated for each one of the combinations of feature selection method, classification algorithm, and number of top features. Thus, each evaluation run was repeated for the 33 feature set described earlier in Fig. 3 (in each repetition different features are extracted from the datasets). Note that the task was to classify specifically the exact worm out of the five or a none (worm) activity, and not to generate a general 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 insight.

4.3. Experiment II

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 capability to detect an *unknown worm* and the *none* activity accurately.

In this experiment the training set consisted of $5 - k$ worms and the testing set contained the k excluded worms, while the *none* activity appeared in both datasets. This process repeated for all the possible combinations of the k worms ($k = 1$–4). In each combination a classifier was trained on the training set and tested on all the remaining seven datasets. The test set included *only* the excluded worms and not the worms presented in the training set since we wanted to measure specifically the detection rate of the unknown. 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 found in $e1$.

5. Results

5.1. Experiment I

Our objective in $e1$ was to determine the best: feature selection approach, feature selection method, number of top features, and classification algorithms. We ran 132 (four classification algorithms applied to 33 feature sets) evaluations (each comprises 64 runs), summing up to 8448 evaluation runs.

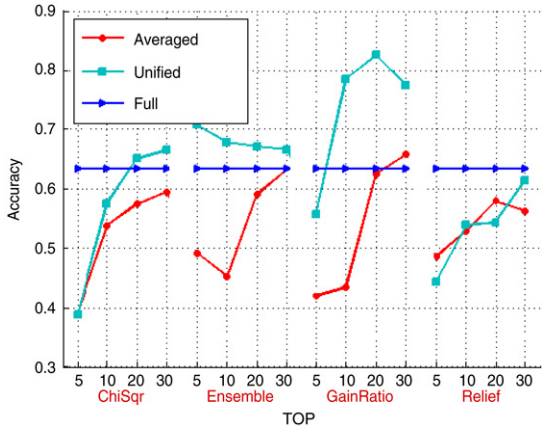

Fig. 4. The mean accuracy for various feature selection methods and the number of top features. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 4 shows the mean accuracy (of all the classification algorithms) achieved for each environment feature consolidation (unified or averaged), each feature selection method, top 5, 10, 20, 30 features and for the full set of features as a baseline. While the feature selection method and number of top features are not relevant for the FULL feature set curve (blue lines in Fig. 4) we presented the curve for comparison purposes. In general, all the feature subsets having less than 30 features achieved a mean performance quite similar to the full set of features (including 323 features). The *unified* consolidation approach outperforms the *averaged* consolidation approach for most of the cases, especially when the Gain Ratio feature selection method is used. Additionally, unlike the averaged consolidation approach, which in most of the cases are below the full set performance, the unified consolidation approach for most of the cases is above it. Additionally, the *Top20* features delivered the best in most of the cases.

Based on the mean accuracy of the four classification algorithms GainRatio feature selection method outperformed the other feature selection methods for most of the top number of features, while the ensemble feature selection method outperformed for *Top5*. Unlike the independent measures, in which there was a monotonic growth when features were added, in the ensemble a monotonic slight decrease was observed as more features were used. The *Top20* features outperformed in general (by averaging) and when using GainRatio feature selection method in particular.

Fig. 5 shows the same results, but presents the mean accuracy of the classification algorithms for several numbers of top features. *Bayesian Networks* outperforms for any number of top features, and on average the 20 top features *Top20* outperformed the other number of top features.

For example, *Top5* features for *GainRatio* feature selection method included: 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 made a direct transition to the SYN-RCVD state from the LISTEN state; (5) *Connection Failures*—the number of times TCP connections 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 made a direct transition to the LISTEN state from the SYN-RCVD state. The list of the top twenty ranked features is presented in Appendix B. Based on the results achieved in $e1$, in which the unified consolidation approach, *Top20* features, *GainRatio* feature selection method, outperformed on average, we used only this feature subset in the second part of the evaluation ($e2$).

In Q_3 we wanted to estimate the performance sensitivity 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. To perform this test we designed two experiments ($e1_1$, $e1_2$).

Table 2 presents the mean accuracy and the resulting standard deviation in each experiment, including 128 evaluation runs. In the first main column the results of $e1_1$ are presented in two columns, when the dataset *di* was used as a training set (left column), and when used as test sets (right column). The results of $e1_1$ when *di* was the

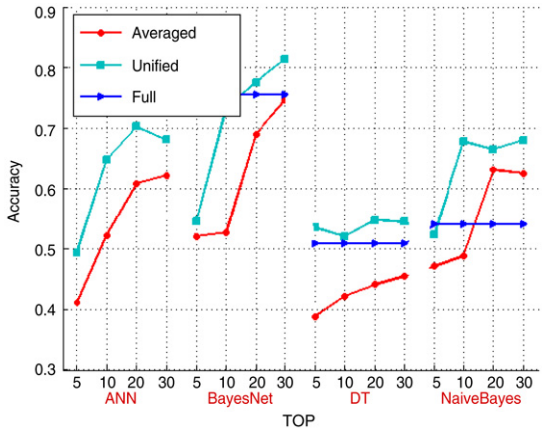

Fig. 5. The mean accuracy for various classification algorithms and number of top features. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
The results achieved in $e1_1$ and $e1_2$

Experiment dataset *di*	$e1_1$		$e1_2$
	Training: *di*	Testset: *di*	Training: all datasets except *di* Testset: *di*
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

training set were not homogeneous. The results of $e1_1$ when *di* was the testset were not homogeneous too, though we found that the training on datasets created in the 'old' computer was significantly better ($\alpha = 0.01$). In $e1_2$, in which all datasets except di were training set and di was the testset, the results were statistically significant ($\alpha = 0.05$) homogeneous. Note that $e1_1$ and $e1_2$ experiments are based on all the learning algorithms, and not specifically on each algorithm. The classification accuracy in $e1_2$ outperformed the accuracy in $e1_1$, since the training sets in $e1_2$ included several datasets.

5.2. Experiment II

In Q_4 we wanted to estimate the possibility of classifying an unknown worm when training on data collected from a single computer. In this set of experiments we used only the *Top20* features, which outperformed in $e1$. The training set included four worms out of the five and the none activity samples, 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 $e1$, in which each worm class was defined separately, there were two classes: (generally) *worm* and *none* activity.

Table 3 presents the results of $e2$. On average the *Decision Trees* and *Bayesian Networks* outperformed the other classification algorithms. The table also shows the *true positive* (TP) and *false positive* (FP). *Decision Trees* and *Bayesian Networks* achieved high level of accuracy and maintained a low *false positive* rate.

Fig. 6 presents the results of $e2$, in which a monotonic increase in the accuracy is shown, as more worms are included in the training set. Note that the number of worms in the x axis refers to the number of worms excluded from the training set, and were included in the test set. In general the *ANN* outperformed all the other algorithms, while the *BN* kept on showing very good results. Note that testing on the seven datasets separately decreased the mean accuracy

Table 3
The results of *e*2

Worm	*ANN* 20			*BN* 20			*DT* 20			*NB* 20		
	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

There is a difference in the detection accuracy of each classifier for each type of worm. On average, Decision Trees outperformed the other classifiers, while maintaining a low false positive rate.

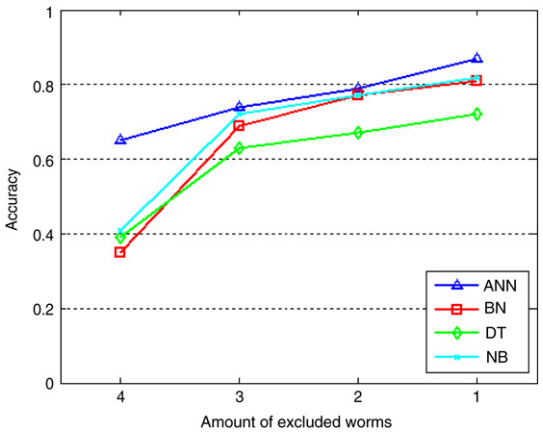

Fig. 6. The performance monotonically increases as fewer worms are excluded from the training set. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

slightly. In addition, when only one worm was excluded, in specific worms we achieved 99% accuracy and a very low false positive rate of 0.005.

6. Discussion and conclusions

In this paper we explored the feasibility of detecting unknown worm activity in individual computers, at a high level of accuracy, given the variation in hardware and software environmental conditions, while minimizing the set of features collected from the monitored computer. Four research questions were investigated, referring to the feasibility of the approach, the best settings, and the level of achieved accuracy, 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 a very high level of accuracy To reduce the computational resources in the classification task we wanted to reduce the number of features. Two consolidation approaches to integrating the eight datasets for the task of feature selection were proposed: unified, in which all the datasets were unified into a single dataset, and averaged, in which we first applied the feature selection method on each dataset and averaged the ranked features into a single rank. Our results showed that the mean performance of the unified approach outperformed the averaged approach. Based on the evaluation results, in general *Bayesian Networks* outperformed the other algorithms; and using the *Top 20* ranked features from the *GainRatio* was the best. The reduction in the number of features and the improvement in accuracy agreed well with the baseline of above 300 features in the full set, since it reduces the computer's resource consumption needed for monitoring its behavior. We investigated the influence of the variance in the training phase and detection phase on 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 homogeneous ($\alpha = 0.01$) based on a

homogeneity test, unlike training on a single dataset (as in $e1_1$). 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 activity in several environments is a reasonable requirement.

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' type. The training sets had four worms and the 'none' activity 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 $e2$ 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. Later we reduced the number of worms in the training set and increased the number of unknown worms in the test set. We found an increase in accuracy as more worms were presented in the training set.

The limitations of this study are the number of worms and the variety of 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 it is possible to detect previously un-encountered computer worms using our novel approach, which is based on monitoring 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 types of computers (i.e., different configurations).

Acknowledgments

This work was supported by Deutsche Telekom Co. We would like to thank the undergraduate students Shai and Ido and Clint Feher who contributed to the preparation of the dataset.

Appendix A. Operating system measurements

The following table includes all feature mapping in the data set. For further information about the objects and their meaning in the windows counters tools, please refer to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/counter/counters2_lbfc.asp, for more information about VTrace please refer to http://msdn.microsoft.com/msdnmag/issues/1000/VTrace/.

ID	Feature name
1	_A_1ICMPMessages_sec_
2	_A_1ICMPMessages_Received_sec_
3	_A_1ICMPMessages_Received_Errors_
4	_A_1ICMPReceived_Dest__Unreachable_
5	_A_1ICMPReceived_Time_Exceeded_
6	_A_1ICMPReceived_Parameter_Problem_
7	_A_1ICMPReceived_Source_Quench_
8	_A_1ICMPReceived_Redirect_sec_
9	_A_1ICMPReceived_Echo_sec_
10	_A_1ICMPReceived_Echo_Reply_sec_
11	_A_1ICMPReceived_Timestamp_sec_
12	_A_1ICMPReceived_Timestamp_Reply_sec_
13	_A_1ICMPReceived_Address_Mask_
14	_A_1ICMPReceived_Address_Mask_Reply_

(*continued on next page*)

ID	Feature name
15	_A_1ICMPMessages_Sent_sec_
16	_A_1ICMPMessages_Outbound_Errors_
17	_A_1ICMPSent_Destination_Unreachable_
18	_A_1ICMPSent_Time_Exceeded_
19	_A_1ICMPSent_Parameter_Problem_
20	_A_1ICMPSent_Source_Quench_
21	_A_1ICMPSent_Redirect_sec_
22	_A_1ICMPSent_Echo_sec_
23	_A_1ICMPSent_Echo_Reply_sec_
24	_A_1ICMPSent_Timestamp_sec_
25	_A_1ICMPSent_Timestamp_Reply_sec_
26	_A_1ICMPSent_Address_Mask_
27	_A_1ICMPSent_Address_Mask_Reply_
28	_A_1IPDatagrams_sec_
29	_A_1IPDatagrams_Received_sec_
30	_A_1IPDatagrams_Received_Header_Errors_
31	_A_1IPDatagrams_Received_Address_Errors_
32	_A_1IPDatagrams_Forwarded_sec_
33	_A_1IPDatagrams_Received_Unknown_Protocol_
34	_A_1IPDatagrams_Received_Discarded_
35	_A_1IPDatagrams_Received_Delivered_sec_
36	_A_1IPDatagrams_Sent_sec_
37	_A_1IPDatagrams_Outbound_Discarded_
38	_A_1IPDatagrams_Outbound_No_Route_
39	_A_1IPFragments_Received_sec_
40	_A_1IPFragments_Re_assembled_sec_
41	_A_1IPFragment_Re_assembly_Failures_
42	_A_1IPFragmented_Datagrams_sec_
43	_A_1IPFragmentation_Failures_
44	_A_1IPFragments_Created_sec_
45	_A_1MemoryPage_Faults_sec_
46	_A_1MemoryAvailable_Bytes_
47	_A_1MemoryCommitted_Bytes_
48	_A_1MemoryCommit_Limit_
49	_A_1MemoryWrite_Copies_sec_
50	_A_1MemoryTransition_Faults_sec_
51	_A_1MemoryCache_Faults_sec_
52	_A_1MemoryDemand_Zero_Faults_sec_
53	_A_1MemoryPages_sec_
54	_A_1MemoryPages_Input_sec_
55	_A_1MemoryPage_Reads_sec_
56	_A_1MemoryPages_Output_sec_
57	_A_1MemoryPool_Paged_Bytes_
58	_A_1MemoryPool_Nonpaged_Bytes_
59	_A_1MemoryPage_Writes_sec_
60	_A_1MemoryPool_Paged_Allocs_
61	_A_1MemoryPool_Nonpaged_Allocs_
62	_A_1MemoryFree_System_Page_Table_Entries_
63	_A_1MemoryCache_Bytes_

ID	Feature name
64	_A_1MemoryCache_Bytes_Peak_
65	_A_1MemoryPool_Paged_Resident_Bytes_
66	_A_1MemorySystem_Code_Total_Bytes_
67	_A_1MemorySystem_Code_Resident_Bytes_
68	_A_1MemorySystem_Driver_Total_Bytes_
69	_A_1MemorySystem_Driver_Resident_Bytes_
70	_A_1MemorySystem_Cache_Resident_Bytes_
71	_A_1Memory__Committed_Bytes_In_Use_
72	_A_1MemoryAvailable_KBytes_
73	_A_1MemoryAvailable_MBytes_
74	_A_1Network_Interfac__Packet_Scheduler_Miniport_Bytes_Total_sec_
75	_A_1Network_InterfacTX____Packet_Scheduler_Miniport_Packets_sec_
76	_A_1Network_Interfacket_Scheduler_Miniport_Packets_Received_sec_
77	_A_1Network_Interfac_Packet_Scheduler_Miniport_Packets_Sent_sec_
78	_A_1Network_InterfacPacket_Scheduler_Miniport_Current_Bandwidth_
79	_A_1Network_Interfacacket_Scheduler_Miniport_Bytes_Received_sec_
80	_A_1Network_Interfacduler_Miniport_Packets_Received_Unicast_sec_
81	_A_1Network_Interfacr_Miniport_Packets_Received_Non_Unicast_sec_
82	_A_1Network_Interfacheduler_Miniport_Packets_Received_Discarded_
83	_A_1Network_Interfac_Scheduler_Miniport_Packets_Received_Errors_
84	_A_1Network_InterfacScheduler_Miniport_Packets_Received_Unknown_
85	_A_1Network_Interfac___Packet_Scheduler_Miniport_Bytes_Sent_sec_
86	_A_1Network_InterfacScheduler_Miniport_Packets_Sent_Unicast_sec_
87	_A_1Network_Interfacduler_Miniport_Packets_Sent_Non_Unicast_sec_
88	_A_1Network_Interfacheduler_Miniport_Packets_Outbound_Discarded_
89	_A_1Network_Interfac_Scheduler_Miniport_Packets_Outbound_Errors_
90	_A_1Network_Interfaccket_Scheduler_Miniport_Output_Queue_Length_
91	_A_1Network_Interface_MS_TCP_Loopback_interface_Bytes_Total_sec_
92	_A_1Network_Interface_MS_TCP_Loopback_interface_Packets_sec_
93	_A_1Network_InterfacTCP_Loopback_interface_Packets_Received_sec_
94	_A_1Network_Interfac_MS_TCP_Loopback_interface_Packets_Sent_sec_
95	_A_1Network_InterfacMS_TCP_Loopback_interface_Current_Bandwidth_
96	_A_1Network_InterfacS_TCP_Loopback_interface_Bytes_Received_sec_
97	_A_1Network_Interfacback_interface_Packets_Received_Unicast_sec_
98	_A_1Network_Interfac_interface_Packets_Received_Non_Unicast_sec_
99	_A_1Network_Interfacopback_interface_Packets_Received_Discarded_
100	_A_1Network_Interfac_Loopback_interface_Packets_Received_Errors_
101	_A_1Network_InterfacLoopback_interface_Packets_Received_Unknown_
102	_A_1Network_Interface_MS_TCP_Loopback_interface_Bytes_Sent_sec_
103	_A_1Network_InterfacLoopback_interface_Packets_Sent_Unicast_sec_
104	_A_1Network_Interfacback_interface_Packets_Sent_Non_Unicast_sec_
105	_A_1Network_Interfacopback_interface_Packets_Outbound_Discarded_
106	_A_1Network_Interfac_Loopback_interface_Packets_Outbound_Errors_
107	_A_1Network_Interfac_TCP_Loopback_interface_Output_Queue_Length_
108	_A_1PhysicalDisk__Total___Disk_Read_Time_
109	_A_1PhysicalDisk__Total___Disk_Time_
110	_A_1PhysicalDisk__Total___Disk_Write_Time_
111	_A_1PhysicalDisk__Total___Idle_Time_
112	_A_1PhysicalDisk__Total_Avg__Disk_Bytes_Read_

(*continued on next page*)

ID	Feature name
113	_A_1PhysicalDisk__Total_Avg__Disk_Bytes_Transfer_
114	_A_1PhysicalDisk__Total_Avg__Disk_Bytes_Write_
115	_A_1PhysicalDisk__Total_Avg__Disk_Queue_Length_
116	_A_1PhysicalDisk__Total_Avg__Disk_Read_Queue_Length_
117	_A_1PhysicalDisk__Total_Avg__Disk_sec_Read_
118	_A_1PhysicalDisk__Total_Avg__Disk_sec_Transfer_
119	_A_1PhysicalDisk__Total_Avg__Disk_sec_Write_
120	_A_1PhysicalDisk__Total_Avg__Disk_Write_Queue_Length_
121	_A_1PhysicalDisk__Total_Current_Disk_Queue_Length_
122	_A_1PhysicalDisk__Total_Disk_Bytes_sec_
123	_A_1PhysicalDisk__Total_Disk_Read_Bytes_sec_
124	_A_1PhysicalDisk__Total_Disk_Reads_sec_
125	_A_1PhysicalDisk__Total_Disk_Transfers_sec_
126	_A_1PhysicalDisk__Total_Disk_Write_Bytes_sec_
127	_A_1PhysicalDisk__Total_Disk_Writes_sec_
128	_A_1PhysicalDisk__Total_Split_IO_Sec_
129	_A_1Process__Total___Privileged_Time_
130	_A_1Process__Total___Processor_Time_
131	_A_1Process__Total___User_Time_
132	_A_1Process__Total_Creating_Process_ID_
133	_A_1Process__Total_Elapsed_Time_
134	_A_1Process__Total_Handle_Count_
135	_A_1Process__Total_ID_Process_
136	_A_1Process__Total_IO_Data_Bytes_sec_
137	_A_1Process__Total_IO_Data_Operations_sec_
138	_A_1Process__Total_IO_Other_Bytes_sec_
139	_A_1Process__Total_IO_Other_Operations_sec_
140	_A_1Process__Total_IO_Read_Bytes_sec_
141	_A_1Process__Total_IO_Read_Operations_sec_
142	_A_1Process__Total_IO_Write_Bytes_sec_
143	_A_1Process__Total_IO_Write_Operations_sec_
144	_A_1Process__Total_Page_Faults_sec_
145	_A_1Process__Total_Page_File_Bytes_
146	_A_1Process__Total_Page_File_Bytes_Peak_
147	_A_1Process__Total_Pool_Nonpaged_Bytes_
148	_A_1Process__Total_Pool_Paged_Bytes_
149	_A_1Process__Total_Priority_Base_
150	_A_1Process__Total_Private_Bytes_
151	_A_1Process__Total_Thread_Count_
152	_A_1Process__Total_Virtual_Bytes_
153	_A_1Process__Total_Virtual_Bytes_Peak_
154	_A_1Process__Total_Working_Set_
155	_A_1Process__Total_Working_Set_Peak_
156	_A_1Processor__Total___C1_Time_
157	_A_1Processor__Total___C2_Time_
158	_A_1Processor__Total___C3_Time_
159	_A_1Processor__Total___DPC_Time_
160	_A_1Processor__Total___Idle_Time_
161	_A_1Processor__Total___Interrupt_Time_

ID	Feature name
162	_A_1Processor__Total___Privileged_Time_
163	_A_1Processor__Total___Processor_Time_
164	_A_1Processor__Total___User_Time_
165	_A_1Processor__Total_C1_Transitions_sec_
166	_A_1Processor__Total_C2_Transitions_sec_
167	_A_1Processor__Total_C3_Transitions_sec_
168	_A_1Processor__Total_DPC_Rate_
169	_A_1Processor__Total_DPCs_Queued_sec_
170	_A_1Processor__Total_Interrupts_sec_
171	_A_1SystemFile_Read_Operations_sec_
172	_A_1SystemFile_Write_Operations_sec_
173	_A_1SystemFile_Control_Operations_sec_
174	_A_1SystemFile_Read_Bytes_sec_
175	_A_1SystemFile_Write_Bytes_sec_
176	_A_1SystemFile_Control_Bytes_sec_
177	_A_1SystemContext_Switches_sec_
178	_A_1SystemSystem_Calls_sec_
179	_A_1SystemFile_Data_Operations_sec_
180	_A_1SystemSystem_Up_Time_
181	_A_1SystemProcessor_Queue_Length_
182	_A_1SystemProcesses_
183	_A_1SystemThreads_
184	_A_1SystemAlignment_Fixups_sec_
185	_A_1SystemException_Dispatches_sec_
186	_A_1SystemFloating_Emulations_sec_
187	_A_1System__Registry_Quota_In_Use_
188	_A_1TCPSegments_sec_
189	_A_1TCPConnections_Established_
190	_A_1TCPConnections_Active_
191	_A_1TCPConnections_Passive_
192	_A_1TCPConnection_Failures_
193	_A_1TCPConnections_Reset_
194	_A_1TCPSegments_Received_sec_
195	_A_1TCPSegments_Sent_sec_
196	_A_1TCPSegments_Retransmitted_sec_
197	_A_1Thread__Total__Total___Privileged_Time_
98	_A_1Thread__Total__Total___Processor_Time_
199	_A_1Thread__Total__Total___User_Time_
200	_A_1Thread__Total__Total_Context_Switches_sec_
201	_A_1Thread__Total__Total_Elapsed_Time_
202	_A_1Thread__Total__Total_ID_Process_
203	_A_1Thread__Total__Total_ID_Thread_
204	_A_1Thread__Total__Total_Priority_Base_
205	_A_1Thread__Total__Total_Priority_Current_
206	_A_1Thread__Total__Total_Start_Address_
207	_A_1Thread__Total__Total_Thread_State_
208	_A_1Thread__Total__Total_Thread_Wait_Reason_
209	_A_1UDPDatagrams_sec_
210	_A_1UDPDatagrams_Received_sec_

(*continued on next page*)

ID	Feature name
211	_A_1UDPDatagrams_No_Port_sec_
212	_A_1UDPDatagrams_Received_Errors_
213	_A_1UDPDatagrams_Sent_sec_
VTRACE features	
214	Process_create
215	Process_destroy
216	Thread_create
217	Thread_destroy
218	Thread_switch
219	Process_set_priority
220	Thread_set_priority
221	Message_get_nf_call
222	Message_get_f_call
223	Message_get_return
224	Message_peek_r_nf_call
225	Message_peek_r_f_call
226	Message_peek_nr_nf_call
227	Message_peek_nr_f_call
228	Message_peek_return
229	Message_dispatch_call
230	Message_dispatch_return
231	Message_trans_accel_call
232	Message_trans_accel_ret
233	Message_translate_call
234	Message_translate_return
235	Message_trans_mdi_call
236	Message_trans_mdi_return
237	Message_set_timer
238	Message_cancel_timer
239	Message_wait_call
240	Message_wait_return
241	Message_get_input_state
242	Message_get_queue_status
243	Key_press
244	Cursor_load
245	Cursor_set
246	Window_create
247	Dialog_create
248	File_complete_operation
249	File_open
250	File_read
251	File_write
252	File_close
253	File_query_info
254	File_set_info
255	File_directory_control
256	File_name
257	File_information
258	File_open_query_close

ID	Feature name
259	File_rename
260	File_delete
261	File_flush
262	File_lock
263	File_unlock
264	File_set_position
265	File_link
266	File_complete_mdl_op
267	File_fcb_information
268	Message_post_window
269	Message_post_thread
270	Message_send
271	Netobj_complete_op
272	Netobj_open
273	Netobj_close
274	Netobj_connect
275	Netobj_disconnect
276	Netobj_send
277	Netobj_send_datagram
278	Netobj_receive
279	Netobj_receive_datagram
280	Netobj_listen
281	Netobj_accept
282	Netobj_associate_address
283	Netobj_disassociate_addr
284	Netobj_notify_connect
285	Netobj_notify_disconnect
286	Netobj_notify_receive
287	Netobj_notify_rcv_dgram
288	Netobj_notify_rcv_exped
289	VTrace_action
290	Device_hook
291	VTrace_version
292	Local_time
293	VTrace_set_mask
294	Beat
295	Device_unhook
296	High_timestamp
297	Ignore_activity
298	Set_cpu_speed
299	User_changed
300	Waitobj_signal
301	Waitobj_set_timer
302	Waitobj_cancel_timer
303	Waitobj_wait_call
304	Waitobj_wait_return
305	Waitobj_wait_gen_call
306	Waitobj_wait_gen_return
307	Msg___waitobj_wait_call

(*continued on next page*)

ID	Feature name
308	Msg___waitobj_wait_ret
309	Rawdisk_read
310	Rawdisk_write
311	Rawdisk_complete_op
312	Section_create
313	Section_open
314	Section_map_view
315	Section_unmap_view
316	Section_get_mdl
317	Testing
318	Flush_icache
319	Flush_write_buffer
320	Terminate_process
321	Terminate_thread
322	Write_request_data
323	Write_VM

Appendix B. Top twenty features selected by each feature selection method

All (V_All)	
ChiSqr	ReliefF
_A_1MemoryCache_Bytes_Peak_	_A_1ICMPReceived_Dest__Unreachable_
_A_1Process__Total_Virtual_Bytes_Peak_	_A_1ICMPSent_Destination_Unreachable_
_A_1MemoryFree_System_Page_Table_Entries_	_A_1SystemFile_Control_Bytes_sec_
_A_1Process__Total_Virtual_Bytes_	_A_1Process__Total_IO_Other_Bytes_sec_
_A_1Process__Total_Pool_Nonpaged_Bytes_	_A_1ICMPMessages_Outbound_Errors_
_A_1MemoryPool_Nonpaged_Bytes_	_A_1MemorySystem_Code_Total_Bytes_
_A_1Process__Total_Thread_Count_	Netobj_disconnect
_A_1SystemThreads_	_A_1ICMPSent_Echo_sec_
_A_1Process__Total_Pool_Paged_Bytes_	_A_1ICMPMessages_Sent_sec_
_A_1TCPConnections_Active_	_A_1Process__Total_Handle_Count_
_A_1Network_Interfac___Packet_Scheduler_Miniport _Bytes_Sent_sec_	_A_1ICMPMessages_sec_
_A_1TCPConnection_Failures_	_A_1Processor__Total___Processor_Time_
_A_1MemoryPool_Nonpaged_Allocs_	_A_1SystemException_Dispatches_sec_
_A_1Process__Total_Handle_Count_	_A_1TCPConnections_Reset_
_A_1Network_InterfacTX____Packet_Scheduler _Miniport_Packets_sec_	_A_1Processor__Total___Idle_Time_
_A_1Network_Interfac__Packet_Scheduler_Miniport _Bytes_Total_sec_	_A_1Processor__Total___User_Time_
_A_1Process__Total_Page_File_Bytes_Peak_	_A_1Process__Total___User_Time_
_A_1IPDatagrams_sec_	_A_1Thread__Total__Total___User_Time_
_A_1SystemFile_Control_Bytes_sec_	_A_1Processor__Total_Interrupts_sec_
_A_1Process__Total_IO_Other_Bytes_sec_	_A_1Memory__Committed_Bytes_In_Use_

All (V_All)	
GainRatio	Ensemble
_A_1ICMPSent_Echo_sec_	_A_1ICMPSent_Echo_sec_
_A_1ICMPMessages_Sent_sec_	_A_1ICMPMessages_Sent_sec_
_A_1ICMPMessages_sec_	_A_1Process__Total_IO_Other_Bytes_sec_
_A_1TCPConnections_Passive_	_A_1SystemFile_Control_Bytes_sec_
Netobj_disconnect	_A_1ICMPMessages_sec_
_A_1TCPConnection_Failures_	_A_1MemoryCache_Bytes_Peak_
_A_1TCPConnections_Active_	_A_1TCPConnection_Failures_
_A_1TCPSegments_Retransmitted_sec_	_A_1TCPConnections_Active_
Write_request_data	_A_1MemoryFree_System_Page_Table_Entries_
_A_1MemoryFree_System_Page_Table_Entries_	_A_1Process__Total_Virtual_Bytes_Peak_
_A_1ICMPReceived_Echo_Reply_sec_	_A_1TCPConnections_Passive_
_A_1ICMPMessages_Received_sec_	_A_1Process__Total_Handle_Count_
_A_1ICMPReceived_Echo_sec_	_A_1MemoryPool_Nonpaged_Bytes_
_A_1ICMPSent_Echo_Reply_sec_	_A_1ICMPReceived_Dest__Unreachable_
_A_1UDPDatagrams_No_Port_sec_	_A_1Process__Total_Thread_Count_
_A_1Process__Total_Thread_Count_	_A_1SystemThreads_
_A_1SystemThreads_	_A_1MemoryPool_Nonpaged_Allocs_
_A_1Network_Interfac___Packet_Scheduler_Miniport_Bytes_Sent_sec_	_A_1Process__Total_Pool_Nonpaged_Bytes_
_A_1Network_Interfac_Scheduler_Miniport_Packets_Outbound_Errors_	Netobj_disconnect
_A_1Network_InterfacTX____Packet_Scheduler_Miniport_Packets_sec_	_A_1Process__Total_Virtual_Bytes_

References

Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R., 2004. N-gram based detection of new malicious code. In: Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC'04.

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.

Bishop, C., 1995. Neural Networks for Pattern Recognition. Clarendon Press, Oxford.

Botha, M., von Solms, R., 2003. Utilising fuzzy logic and trend analysis for effective intrusion detection. Computers & Security 22 (5), 423–434.

Bridges, S.M., Vaughn Rayford, M., 2000. 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.

Caia, D.M., Gokhaleb, M., Theilerc, J., 2007. Comparison of feature selection and classification algorithms in identifying malicious executables. Computational Statistics & Data Analysis 51, 3156–3172.

CERT. CERT Advisory CA-2000-04, Love letter worm. http://www.cert.org/advisories/ca-2000-04.html.

Demuth, H., Beale, M., 1998. Neural Network Toolbox for Use with Matlab. The Mathworks Inc., Natick, MA.

Dickerson, J.E., Dickerson, J.A., 2000. Fuzzy network profiling for intrusion detection. In: Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, Atlanta, USA, July 2000, pp. 301–306.

Domingos, P., Pazzani, M., 1997. On the optimality of simple Bayesian classifier under zero-one loss. Machine Learning 29, 103–130.

Fosnock, C., 2005. Computer worms: Past, present and future. East Carolina University.

Hu, P.Z., Heywood, M.I., 2003. Predicting Intrusions with Local Linear Model. In: Proceedings of the International Joint Conference on Neural Networks, vol. 3. IEEE, pp. 1780–1785.

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

Kayacik, H.G., Zincir-Heywood, A.N., Heywood, M.I., 2003. On the capability of a Som based intrusion detection system. In: Proceedings of the International Joint Conference on Neural Networks, vol. 3. IEEE, pp. 1808–1813.

Kienzle, D.M., Elder, M.C., 2003. Recent worms: A survey and trends. In: Proceedings of the 2003 ACM Workshop on Rapid Malcode. ACM Press, pp. 1–10.

Kolter, J.Z., Maloof, M.A., 2006. Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 7, 2721–2744.

Lee, W., Stolfo, S.J., 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.

Lei, J.Z., Ghorbani, A., 2004. Network intrusion detection using an improved competitive learning neural network. In: Proceedings of the Second Annual Conference on Communication Networks and Services Research, CNSR04. IEEE-Computer Society, IEEE, pp. 190–197.

Lippmann, R.P., Graf, I., Wyschogrod, D., Webster, S.E., Weber, D.J., Gorton, S., 1998. The 1998 DARPA/AFRL off-line intrusion detection evaluation. In: First International Workshop on Recent Advances in Intrusion Detection, RAID, Louvain-la-Neuve, Belgium, 1998.

Lorch, J., Smith, A.J., 2000. The VTrace tool: Building a system tracer for windows NT and windows 2000. MSDN Magazine 15 (10), 86–102.

Mitchell, T., 1997. Machine Learning. McGraw-Hill.

Moore, D., Shannon, C., Brown, J., 2002. Code red: A case study on the spread and victims of an internet worm. In: Proceedings of the Internet Measurement Workshop 2002. Marseille, France, November 2002.

Pearl, J., 1986. Fusion, propagation, and structuring in belief networks. Artificial Intelligence 29 (2), 241–288.

Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rokach, L., Chizi, B., Maimon, O, 2007. A methodology for improving the performance of non-ranker feature selection filters. International Journal of Pattern Recognition and Artificial Intelligence 21 (5), 1–22.

Rokach, L., Elovici, Y., 2007. Data mining for intrusion detection. In: Janczewski, L.J., Colarik, A.M. (Eds.), Encyclopaedia of Cyber Terrorism and Cyber Warfare. In: Idea Group Reference, Winter.

Schultz, M., Eskin, E., Zadok, E., Stolfo, S., 2001. Data mining methods for detection of new malicious executables. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 178–184.

Weaver, N., Paxson, V., Staniford, S., Cunningham, R., 2003. A taxonomy of computer worms. In: Proceedings of the 2003 ACM Workshop on Rapid Malcode. Washington, DC, October 2003, pp. 11–18.

Witten, I.H., Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco.

Zanero, S., Savaresi, S.M., 2004. Unsupervised learning techniques for an intrusion detection system. In: Proceedings of the 2004 ACM Symposium on Applied Computing. ACM Press, Nicosia, Cyprus, pp. 412–419.