Active Learning to Improve the Detection of Unknown Computer Worms Activity

Robert Moskovitch, Nir Nissim, Roman Englert, Yuval Elovici
Deutche Telekom Laboratories at Ben Gurion University
Be'er Sheva, 84105, Israel.
{robertmo, nirni, englert, elovici}@bgu.ac.il

Abstract - Detecting unknown worms is a challenging task. We propose an innovative technique for detecting the presence of an unknown worm based on the computer measurements extracted from the operating system. 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 measures were used to reduce the number of features. We applied support vector machines on the resulting feature subsets. In addition, we used active learning as a selective sampling method to increase the performance of the classifier and improve its robustness in noisy data. Our results indicate that using the proposed approach resulted in a mean accuracy in excess of 90%, and for specific unknown worms accuracy reached above 94%, using just 20 features while maintaining a low false positive rate.

Keywords: Classification, Active Learning, Support Vector Machines, Malcode Detection.

1 Introduction

The detection of malicious code (malcode) transmitted over computer networks has been researched intensively in recent years. One type of abundant malcode is *worms*, which proactively propagate across networks while exploiting vulnerabilities in operating systems or in installed programs. Other types of malcode 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.

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 has obvious demerits, however, since worms propagate very rapidly. By the time the antivirus software has been notified about the new worm, very expensive damage has already been inflicted [1].

Intrusion detection, commonly done at the network level, called *network-based intrusion detection* (*NIDS*), was researched substantially. 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 which monitor activities at a host. *HIDS* usually compare the states of the computer in several aspects, such as the changes in the file system using checksum comparisons. The main drawback of this approach is the ability of malcodes to disable antiviruses; other technical limiting factors include the fast changes in the file system. The main problem is detection knowledge maintenance, which is usually performed 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 [2,3,4]. While this approach is potentially a good solution, it is not complete. It can detect only executable files, but not malcodes located entirely in the memory, such as the *Slammer* worm [5], which cannot be detected using this technique. Moreover, any technique can be sabotaged by a malcode.

In this paper we suggest a new approach that can be classified under *HIDS*, but the novelty here is that it is based on the computer operating system measurements and that the knowledge is acquired automatically using inductive learning, given a dataset of known worms. While this approach does not prevent infection, it enables fast detection of an infection which may result in an alert, which can be further reasoned when monitoring the network level, and relevant decisions and policies, such as disconnecting a single computer or a cluster, can be further implemented.

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 support vector machines. In a previous study we performed this task using several machine learning algorithms, including Decision Trees, Naïve Bayes, Byaesian Networks and Neural networks [6]. Support Vector Machines are known for their outperforming capabilities in binary classification tasks, especially in conjunction with Active Learning.

Active Learning is commonly used to reduce the amount of labeling required from an expert (a time consuming task).The classifier is actively asking to label specific examples in the dataset, which it considers as the most informative, based on its current knowledge. Here all the examples are labeled and we use the Active Learning approach, as a selective sampling method. We argue that SVM will achieve better results when using Active Learning as a selective sampling method.

In the proposed approach, a classifier is trained by a dataset containing computer measurements collected from computers infected by a known *worm* and from computers that were not infected. Based on the generalization capability of the classification algorithm, we hypothesized that a classifier can further detect previously unknown worm activity. This approach may be affected by the variation of computer and application configurations as well as user behavior on each computer. 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 that are collected from the monitored computers.

The rest of the article is structured as follows. Section 2 surveys the relevant background for this work. A description of support vector machines, relevant kernel functions and active learning, used in this study, is given in section 3, followed by the research questions, corresponding experimental plan and evaluation results presented section 4. Finally, in section 5 we conclude the paper with a discussion on the evaluation results, conclusions and future work.

2 Background and Related Work

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 original research was on worms, we will focus on them in this section.

Kienzle and Elder [7], define a *worm* by several aspects through which it can be distinguished from other types of malcode: 1) *Malicious code* – worms are considered malicious in nature; 2) *Network propagation or Human intervention* – a commonly agreed upon aspect, that is, worms propagate actively over a network, while other types of malicious codes, such as viruses, commonly require human activity to propagate; 3) *Standalone or file infecting* – while viruses infect a file (its host), a worm does not require a host file, and sometimes does not even require an executable file, residing entirely in the memory, as did the *Code Red* [8] worm. Different purposes and motivations drive 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 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 hypothesize that support vector machines can be a very useful way to learn and generalize from previously seen worms, in order to classify unknown worms effectively.

Machine Learning has already been used in efforts to detect and protect against malicious codes. A recent survey of intrusion detection methods [2] summarizes recent proposed applications of machine learning in recognizing malcodes in single computers and in computer networks. Lee et al. [11] proposed a framework consisting of data mining algorithms for the extraction of anomalies of user normal behavior for the 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 further implementation 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], with reference to its implementation within the ADAM system developed by Barbara et al. [13]. The ADAM system had three main parts: (a) a network data monitor which listens to TCP/IP protocol; (b) a data mining engine which enables acquisition of the association rules from the network data; and (c) a classification module which classifies the nature of the traffic in two possible classes, normal and abnormal, which can later be linked to specific attacks. Other machine learning algorithm techniques proposed are Artificial Neural Networks (*ANN*) [14,15,16], Self Organizing Maps (*SOM*) [17] and fuzzy logic [18,19,20].

3 Methods

The general goal of this study is to assess the viability of employing Support Vector Machines (SVM) in detecting

the existence of *unknown* worms in an individual computer host, based on its behavior (measurements), and the option of improving the performance using active learning. To create a testing environment, we built a local network of computers, which enabled us to inject worms into a controlled environment, while monitoring the computers and collecting measurements.

3.1 DataSet Creation

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

3.1.1 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 enabled us to monitor system features that appear in these main categories (the amount of features in each category appears in parentheses): *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* [21], a software tool which can be installed on a PC running Windows for monitoring purposes. *VTrace* collects traces of the *file system, the network, the disk drive, processes, threads, inter-process 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 a vector. *VTrace* stored time-stamped events, which were aggregated into the same fixed intervals, and merged with the *windows performance* log files. These eventually consisted of a vector of 323 features collected every second.

3.1.2 Injected Worms

When selecting worms from the wild, our goal was to choose worms that differ in their behavior from the available worms. Some of the worms have a heavy payload of Trojans to install in parallel to the distribution process upon the network; others focus only on distribution. Another aspect is having different strategies for IP scanning which results in varying communication behavior, CPU consumption, and network usage. While all the worms are different, we wanted to find common characteristics so as to be able to detect an unknown worm. The worms we used were: **(1) *W32.Dabber.A***, **(2) *W32.Deborm.Y*, (4) *W32.Sasser.D***, **(5) *W32.Slackor.A*.** All the worms perform port scanning and possess different characteristics. We provide further information in [6]; in addition further information can be accessed through libraries on the web[2][3][4].

3.1.3 Dataset Description

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 is 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 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 activity labeled by the specific *worm*, or *none* activity label. Each dataset contained a few thousands of (labeled samples) each worm or none activity. We therefore had three binary aspects, which resulted in eight 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. Having such a variability of sampling environments and features created a noisy dataset, and thus we used a selective (active) sampling approach to reduce its influence and to boost the performance.

[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] Macafee http://vil.nai.com

3.2 Feature Selection

In Machine Learning 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 requirements for the monitoring operations (measurements) and the classifier computations. This can be achieved 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 different kernels in the SVM, 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 using an iterative evaluation experiment). Using filters, a measure is calculated to quantify the correlation of each feature with the class (in our case, the presence or absence of worm activity). Each feature receives a rank representing its expected contribution in the classification task. Finally, the top ranked features were selected.

We used three feature-selection measures: Chi-Square (CS), Gain Ratio [22,23] (GR), ReliefF [24] implemented in the Weka environment [25] and their ensemble, which resulted in a list of ranks for each feature 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 which we had eight datasets each resulting in 17 datasets.

3.3 Support Vector Machines

We employed the SVM classification algorithm [26] using three different kernel functions, in a supervised learning approach. We briefly introduce the SVM classification algorithm and the principles and implementation of Active Learning we used in this study. Naturally, SVM is a binary classifier which finds a linear hyperplane that separates the given examples into the two given classes. An extension which enables handling multiple classes' classification was developed. SVM is known for its capability to handle large amount of features, such as text. We used the SVM-light implementation [26], given a training set, in which an example is a vector $x_i = \langle f_1, f_2 \ldots f_n \rangle$, where f_i' is a feature, and labeled by $y_i = \{-1, +1\}$. The SVM attempts to specify a linear hyperplane that has the maximal margin, defined by the maximal (perpendicular) distance between the examples of the two classes. Figure 1 illustrates a two dimensional space, in which the examples are located according to their features and the hyperplane splits them according to their label.

The examples lying closest to the hyperplane are the "supporting vectors". *W*, the Normal of the hyperplane is a linear combination of the most important examples (supporting vectors), multiplied by LaGrange multipliers (alphas).

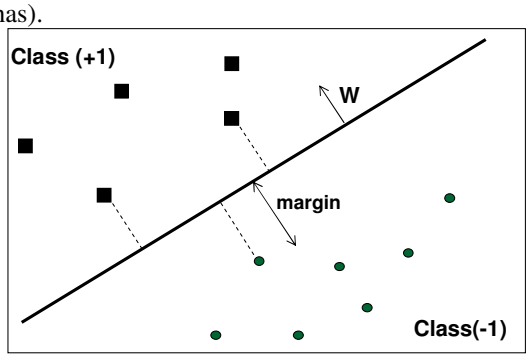

Figure 1: SVM that separates the training set into two classes with maximal margin.

Since the dataset in the original space often cannot be linearly separated, a kernel function *K* is used, and SVM actually projects the examples into a higher dimensional space in order to create linear separation of the examples. Note that kernel functions must satisfy Mercer's condition [27] as shown in formula 1. For the general case, the SVM classifier will be in the form shown in formula 2, where *w* is defined in formula 3.

$$K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) \quad (1)$$

$$f(x) = w \cdot \Phi(x) = \sum_{1}^{n} \alpha_i y_i K(x_i\, x) \quad (2)$$

$$w = \sum_{1}^{n} \alpha_i y_i \Phi(x_i) \quad (3)$$

Two commonly used *kernel* functions were used: *Polynomial kernel*, as shown in formula 4, creates polynomial values of degree *p*, where the output depends on the direction of the two vectors, examples x_1, x_2 shown in formula 4, in the original problem space. Note that a private case of polynomial kernel, having $p=1$, is actually the *Linear kernel*. The second is a *Radial Basis Function* (RBF), as shown in formula 5, in which a Gaussian is used as the RBF and the output of the kernel depends on the Euclidean distance of examples x_1, x_2

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^P \quad (4)$$

$$K(\mathbf{x}_1, \mathbf{x}_2) = e\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}\right) \quad (5)$$

3.4 Active Learning

Active Learning (AL) is usually used to reduce the effort expended on labeling examples, which is commonly a time-consuming and costly task, while maintaining a high accuracy rate. In *AL* the learner actively asks that the most informative examples be labeled so it can learn from them, which commonly results in using fewer examples.

In our study all the examples are labeled, since we know which worm was active during the sampling; however, since the data were noisy we had to apply a selective sampling method to increase the accuracy. We used AL as

a selective sampling technique which selects only the examples which lead to a better classification model.

In this study we implemented an active learner which aims at reducing the expected *generalization error*, presented in [28], named Error-Reduction. Through the estimation of the expected error reduction achieved through labeling an example, the example which has the maximal reduction will be labeled. Since the real future error rates are unknown, the learner utilizes its current classifier in order to estimate those errors. In the beginning of an *AL* trial, an initial classifier $P_{\hat{D}}(y \mid x)$ is trained over a randomly selected initial set D. For every optional label $y \in Y$ (of every example x in the pool P) the algorithm induces a new classifier $P_{\hat{D}'}(y \mid x)$ trained on the extended training set $D' = D + (x, y)$, and the future expected generalization error of the classifier is estimated using a log-loss function, shown in formula 6. The log-loss function measures the error degree of the current classifier, where this classifier represents the induced classifier as a result of selecting a specific example from the pool and adding it to the training set, having a specific label. The log-loss for each example (see formula 6) will be calculated several times for each optional label (in our case binary). Then for every example $x \in P$ the self-estimated average error is calculated, based on formula 7, over each one of the optional labels $y \in Y$, which is actually a weighted average of the error for all the specific examples over all its optional labels, as shown in formula 6.

$$E_{P_{\hat{D}'}} = \frac{1}{|P|} \sum_{x \in P} \sum_{y \in Y} P_{\hat{D}'}(y \mid x) \cdot \left| \log\left(P_{\hat{D}'}(y \mid x) \right) \right| \quad (6)$$

$$\sum_{y \in Y} P_{\hat{D}}(y \mid x) \cdot E_{P_{\hat{D}'}} \quad (7)$$

Finally the example x with the lowest expected self-estimated error is chosen and added to the training set. In a nutshell, an example will be chosen from the pool only if it dramatically improves the confidence of the current classifier over all the examples in the pool (means lower estimated error)

3.5 Evaluation Measures

For the purpose of evaluation we used the *True Positive (TP)* measure presenting the rate of instances classified as *positive* correctly (true), *False Positive (FP)* presenting the rate of *positive* instances misclassified, as shown in equation 8, 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 9. 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*).

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

$$Total\ Accuracy = \frac{TP^A + TN^A}{TP^A + FP^A + TN^A + FN^A} \quad (9)$$

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).

4 Experiments and Results

In the first part of the study, we wanted to identify the best feature selection measure, the best kernel function and the minimal features required to maintain a high level of accuracy. In the second part we wanted to measure the possibility of classifying unknown worms using a training set of known worms, and the possibility of increasing the performance using selective sampling. In order to answer these questions we designed three experimental plans, based on the *seventeen* sets of subsets which 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, for the evaluation [6]. For clarity and readability we describe the experiments and their corresponding results, instead of separating them into two sections.

4.1 Experiment I – Best Feature Selection

We performed a wide set of experiments, called *e1*, in which we evaluated each kernel function, feature selection and top selection combination to determine the outperforming combination. 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, for each combination. When *i* = *j*, we used *10- fold cross validation* Note, that the task was to classify specifically the exact worm out of the five or none (worm) activity, and not 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.

Figure 2 shows the mean performance for each feature selection measure and top selection, in which we used the SVM with the polynomial and linear kernels. Most of the time the FULL[5] features outperformed the measures, while the *GainRatio* outperformed the Full in the Top10 and 20.

[5] Note that FULL does not have Top selections, but we are presenting it thus for comparison purposes and better illustration.

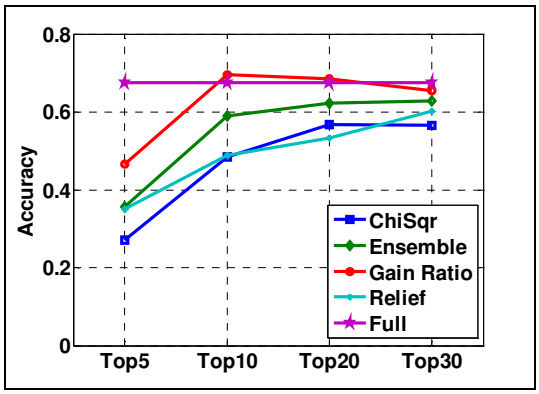

Figure 2. The mean performance of the Full feature set outperforms most of the time, while the GainRatio outperformed in Top10 and Top20.

Figure 3 presents the mean performance of the kernels based on the same results. The polynomial outperforms until using the *Top20*, whereas from *Top30* the *Linear kernel* outperformed. In general the use of the Full features outperformed the top selections in the linear kernel; however, for the Polynomial kernel using Top20 achieved the maximal accuracy, which was the reason why we used the Top20 of GainRatio for the rest of the experiments. Generally, it can be seen that using above 20 features didn't improve the performance, probably because of adding features which correlate less with the classes.

The *Top5* significant features when using *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.

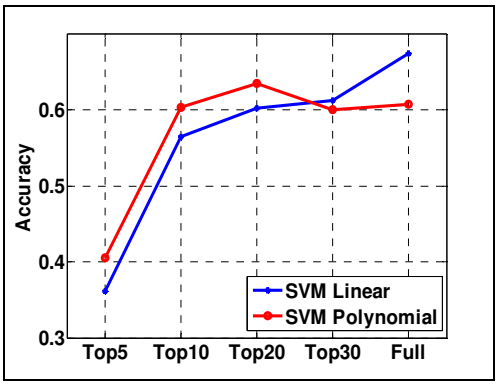

Figure 3. The Polynomial kernel until Top20 outperformed and from Top30 the Linear kernel outperformed

4.2 Experiment II – Unknown Worms Detection

To evaluate the capability of the suggested approach to classify an *unknown worm* activity, which was our main objective, an additional experiment, called *e2*, was performed. 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, for *k = 1* to *4*. Note that in these experiments, unlike in *e1*, there were two classes: (generally) *worm,* for any type of worm, and *none* activity and we used the Top20 features of the GainRatio.

Figure 4 presents the results of *e2*. A monotonic improvement was observed, as more worms were included in the training set. Note that the number of worms in the *x* axis refer to the *number of excluded worms*, which were *in the test set*. The *RBF* outperformed all the other kernels, while the *polynomial* performed worse than the *linear*. For specific worms, when a single worm was excluded an outperforming accuracy of 95% was observed, which we do not show for lack of space.

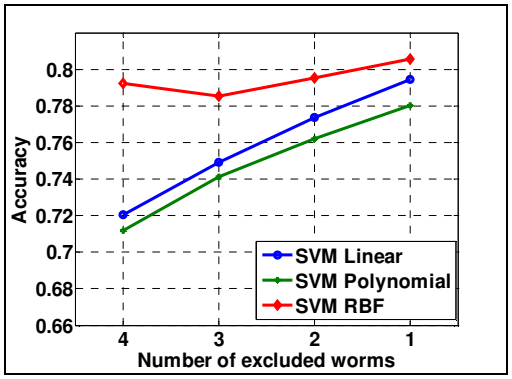

Figure 4. The performance monotonically increases as fewer worms are excluded (and more worms appear in the training set). The RBF kernel presents a different behavior, in which even when learning from a single worm a high level of accuracy is achieved.

In this set of experiments, since we trained on the dataset which outperformed in *e1*, which was extracted from the *old* computer configuration, the classification accuracy when testing on the *new* configuration datasets was relatively low, which decreased the overall accuracy. However, while the RBF outperformed the other kernels it also maintained a high level of accuracy even when training on a single worm. Based on these results we assumed that better results will be achieved by using a selective sampling approach to reduce the noise in the training set by which the RBF is challenged. Thus, we employed a selective sampling approach, based on active learning, as was described in the section about active learning, and is presented in the next section about experiment three.

4.3 Experiment 3 – Using Selective Sampling

In this set of experiments we wanted to maximize the performance achieved by the RBF kernel function. First we trained the classifier on a dataset containing samples from the entire set of eight datasets, unlike in *e2* in which the training set consisted of a single dataset, and tested on all the datasets. The evaluation was made in the same setup of e2, in which worms were excluded from the

training set appeared in the test set. We argued that by using active learning as a selective sampling method, in an Error Reduction criterion we will improve the baseline results. Thus, we repeated the same experiment with the entire set of examples, as a baseline, and with the selective sampling method, in which we evaluated the performance after selecting 50, 100 and 150 samples. For the selective sampling process, initially six examples from each type of class (worms and none) were selected randomly, and then in each AL iteration and additional example was selected.
Figure 5 presents the results of experiment 3. Two main outcomes can be observed. The baseline performance, in which the classifier was trained on the entire training set, was quite the same as in experiment two (see Figure 4), which means that training on a single dataset (within the eight) results in a similar accuracy. The selective sampling had improved the baseline performance significantly even when 50 samples were selected. We don't present the results of selecting 100 and 150 since no improvement was observed in addition to the 50 samples. Generally, an improvement of more than 10% was achieved, when selective sampling was used, and above 40% when training on four worms which exceeded 94% accuracy.

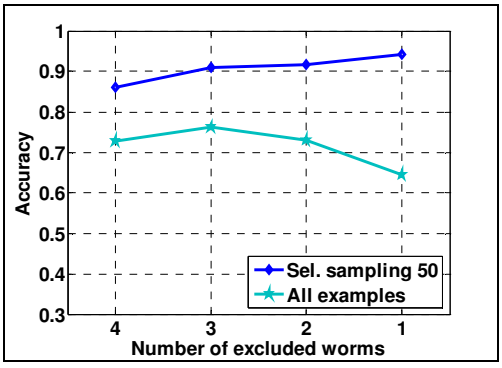

Figure 5. The selective sampling approach improves the accuracy results significantly. Generally, an improvement of above 10% in accuracy was achieved, while on four worms above 40% improvement was achieved.

5 Conclusions and Future Work

We presented the concept of detecting *unknown* computer worms based on a host behavior, using the SVM classification algorithm based on several kernels. Based on the results shown in this study, the use of support vector machines in the task of detecting unknown computer worms is possible. We used a feature-selection method which enabled 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 to this task. On average the *Top20* features produced the highest results and the *RBF kernel* commonly outperformed other kernels. In the detection of unknown worms (*e2*), the results show that it is possible to achieve a high level of accuracy (exceeding 80% on average); as more worms were included in the training set the accuracy improved. To reduce the noise in the training set and improve the learning we argued that the use of the active learning approach as a selective method would improve the performance, which actually happened, increasing the accuracy after selecting 50 examples to above 90% accuracy and 94% when the training set contained four worms. When we selected 100 and 150 examples no improvement was observed above the performance after selecting 50 examples.
These results are highly encouraging and show that unknown 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 for 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 on 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.

Acknowledgments We would like to thank the anonymous reviewers for the helpful comments.

References

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

[2] 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.

[3] 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)

[4] 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.

[5] 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.

[6] Moskovitch, R., Rokach, L. and Elovici, Y. Detection of Unknown Computer Worms based on Behavioral Classification of the Host, Computational Statistics and Data Analysis, In Press.

[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] Lippmann, R.P., Graf, I., Wyschogrod, D., Webster, S.E., Weber, D.J. and Gorton, S. "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., and 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] Zanero, S. and Savaresi, S.M. "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] Kayacik, H.G. Zincir-Heywood, A.N. and Heywood, M.I. 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] Lei, J. Z. and Ghorbani, A. "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] Hu, P.Z. and Heywood, M.I. 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] Dickerson, J.E. and Dickerson, J.A. "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] Bridges, S.M. and Rayford, M.V. "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] Botha, M. and von Solms, R. (2003) "Utilising fuzzy logic and trend analysis for effective intrusion detection," Computers & Security, vol. 22, no. 5, pp. 423–434.

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

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

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

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

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

[26] Joachims, T. (1999). Making large-scale svm learning practical. In B. Sch¨olkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning. MIT Press.

[27] Burges, C.J.C. (1998) A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2:121–167.

[28] Roy, N. and McCallum, A. (2001) Toward optimal active learning through sampling estimation of error reduction. In Proceedings of ICML-2001, 18th International Conference on Machine Learning, pages 441–448.