Abstract—Computer worm detection is commonly performed by antivirus software tools that rely on prior explicit knowledge of the worm's code (detection based on code signatures). We present an approach for detection of the presence of computer worms based on Artificial Neural Networks (ANN) using the computer's behavioral measures. Identification of significant features, which describe the activity of a worm within a host, is commonly acquired from security experts. We suggest acquiring these features by applying feature selection methods. We compare three different feature selection techniques for the dimensionality reduction and identification of the most prominent features to capture efficiently the computer behavior in the context of worm activity. Additionally, we explore three different temporal representation techniques for the most prominent features. In order to evaluate the different techniques, several computers were infected with five different worms and 323 different features of the infected computers were measured. We evaluated each technique by preprocessing the dataset according to each one and training the ANN model with the preprocessed data. We then evaluated the ability of the model to detect the presence of a new computer worm, in particular, during heavy user activity on the infected computers.

Keywords—Artificial Neural Networks, Feature Selection, Temporal Analysis, Worm Detection.

I. INTRODUCTION

The detection of malicious code transmitted over computer networks has been substantially researched during the past years. Commonly, the term “malicious code” (malcode) refers to different types of codes, such as executables or scripts, which contain some code having a malicious purpose. One type of malcode is worms, which actively propagate, exploiting vulnerability in the operating system, through communication protocols. Other types of malcode are viruses, which inject their code into an innocent file (a host) and are activated whenever the file is executed. Unlike worms, viruses require user intervention to propagate. Other recently disseminated malicious codes include Trojans, which are computer programs that have a useful functionality but also have some hidden malicious goal, and backdoors, which enable remote access and control with the aim of gaining full or partial access to the infected system.

Nowadays, known malcodes are mainly detected and removed by antiviruses. Antiviruses search the executables for known patterns, also called signatures. Antiviruses are helpless when facing an unknown malicious executable. After the appearance of the new worm, a new signature is created by the antivirus system company. The antivirus signature base is periodically updated. However, since worms spread rapidly, the signature update action is often taken too late, and expensive damage may already have been done by the worm. Trojans and backdoors have other malicious motivations, such as economic, terrorist, or criminal. They are commonly installed on a relatively small number of hosts, and thus do not attract the attention of antivirus systems and remain undetected. For example, a student can demonstrate a problem he encountered in his homework using a portable memory device on his/her professor's computer. During this demonstration the student installs a backdoor that allows him to access the professor's computer and search for a copy of an upcoming examination.

A recent survey of intrusion detection [1] suggests using artificial intelligence (AI) techniques to recognize malicious software (malware) in single computers and in computer networks. It describes the research done in developing these AI techniques, and discusses their advantages and limitations. One of the critical requirements of such an AI technique is that it operates efficiently in real-time.

One of the AI techniques mentioned in this survey is Artificial Neural Networks (ANN). Other research studies referenced in the survey used the Self Organizing Map (SOM) method [2]-[4], the main challenge of which as reported by the authors was overcoming the high dimensional inputs. Linear techniques, such as Principle Component Analysis (PCA), Singular Value Decomposition or Support Vector
Machine [5], [6] are used to reduce the input dimensionality and may result in a less accurate modeling.

Other aspects discussed in the survey [1] were the detection of an intrusion or a malware presence by analyzing executable files on local storage devices [7], analyzing the content of the packets sent or received by the computer [8, 2], or looking at the system calls that were invoked by processes running on the system [9].

In this study a different approach is suggested. The detection of the presence of a malware in a computer is performed by analyzing the overall computer behavior. We define the computer behavior by a variety of different values which can be measured in the computer while it is operating.

The main ANN advantages are a high level of accuracy in real-time operation, low CPU resources utilization during the classification phase, and the ability to generalize, in order to detect and identify, any previously unseen classes. The fact that worms propagate very fast on networks makes them one of the most challenging malwares. Fast detection of computers infected with worms is critically important on local networks. For these reasons we propose employing ANN for the detection of worm activity in real time.

It was already shown in our previous work that the detection of new worms using ANN techniques is possible and effective [10]. However, continuous monitoring of large number of measured features may demand a significant part of the computational resources. In this work we reduce the number of features significantly by using various feature selection techniques, while increasing the detection accuracy. We use the Causal Indices (CI) technique to estimate the contribution of each input feature to each classification output. Additionally, we evaluate the effect of several temporal analysis preprocessing techniques on the detection problem.

The structure of this paper is as follows: In section II the classification, feature selection and temporal analysis methods are described. In section III we describe the creation of the datasets we used in the study. In section IV we present the techniques we used to evaluate the different methods, and we describe the CI technique in this section. Section V describes the experimental sequence and the purpose of each particular experiment. In section VI we present the results of the experiments. In section VII we discuss the experiment results and conclude with recommendations for further research.

II. CLASSIFICATION, FEATURE SELECTION AND TEMPORAL ANALYSIS METHODS

A. Classification Method

We used a typical feed forward neural network, together with the Levenberg-Marquardt training method [11]. The number of hidden neurons was set empirically to six.

B. Feature Selection Techniques

Generally, there are two different approaches to feature selection: the wrapper and the filter approach. The wrapper approach searches for the optimal subset of features of a given dataset, for a specific classification algorithm. The main drawback of this approach is the relatively long computation time. The filter approach ranks the features according to a certain measure independent of any classification algorithm. Thus, after the calculation of ranks, one can use any subset of features based on their ranks.

For features selection we used three different filter techniques. These techniques are described below.

1) The relation between the inputs and the hidden neuron’s relative variance

This knowledge extraction and dimensionality reduction technique involves ranking the inputs according to their relevance to the ANN prediction accuracy. It is based on the observation that in a trained ANN model a less relevant input contributes a small proportion of the variance in the hidden layer neuron’s activities. This may be the result of either the small relative variance of the input value or the small final connection weights to all hidden neurons assigned to this input by the trained ANN.

The contribution of an input \( i \) to the total variance of the hidden layer inputs is presented in (1). The \((W_{hi})^T\) is the \( i \)’th row of the transpose of \( W_{hi} \), i.e., the \( i \)’th column of the input-to-hidden connection weights expressed as a row vector; \( R \) is the covariance matrix for the network inputs \( x \), estimated from the training set.

\[
(V_{ij})_i = (W_{hi})^T W_{hi} R_{ii} \tag{1}
\]

The relative contribution of an input \( i \) to the variance of the hidden layer inputs is calculated using (2). In (2) \( j \) is the index of each one of the \( n \) hidden neurons.

\[
(V_{ij})_{rel} = \frac{(V_{ij})_i}{\sum_{j=1}^{n} (V_{ij})} \tag{2}
\]

The less contributing inputs can be discarded and the ANN re-trained with the reduced input set, often yielding better prediction accuracy. This technique is described in detail in [12].

2) The Fisher score ranking

The Fisher score ranking technique calculates the difference, described in terms of mean and standard deviation, between the positive and negative examples relative to a certain feature. Equation (3) defines the Fisher score, in which \( R_i \) is the rank of feature \( i \), describing the proportion of the substitution of the mean of the feature \( i \) values in the positive examples (\( \rho \)) and the negative examples (\( n \)), and the sum of the standard deviation. The bigger the \( R_i \), the bigger the difference between the values of positive and negative examples relative to feature \( i \); thus, this feature is more important for separating the positive and negative examples. This technique is described in detail in [13].
\[ R_i = \frac{\mu_{i,p} - \mu_{i,n}}{\sigma_{i,p} + \sigma_{i,n}} \]  

(3)

3) Gain Ratio Filter
The Gain Ratio measure is based on the Information Gain (IG) measure, which is based on measuring the relative entropy reduction. This method requires discretization to be applied to the continuous data in advance. Equation (4) defines the classic entropy measure, in which \( S \) is the entire dataset, \( C \) is the class attribute, and \( S_c \) is a subset of \( S \) in which the value of \( C \) is \( c \).

\[ E(S) = \sum_{c \in C} \frac{|S_c|}{|S|} \log_2 \frac{|S|}{|S_c|} \]  

(4)

Equation (5) defines the IG rank. IG shows how much information we gain by splitting the dataset relative to attribute \( A \). In this equation, \( V(A) \) is the set of unique values of attribute \( A \), and \( S_v \) is the subset of \( S \) in which the value of attribute \( A \) is \( v \).

\[ IG(S, A) = E(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} E(S_v) \]  

(5)

The problem of the IG method is that it gives higher ranks to attributes with a large number of unique values, i.e., \( V(A) \) is high. Gain Ratio overcomes this bias by using an extra term which represents the way an attribute splits the data. Equation (6) defines this special term and (7) defines the ranking of Gain Ratio.

\[ SL(S, A) = -\sum_{v \in V(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S|}{|S_v|} \]  

(6)

\[ GR(S, A) = \frac{IG(S, A)}{SL(S, A)} \]  

(7)

When \( SL(S,A) \) is zero, it is defined as equal to \( IG(S,A) \). Additional information about this technique can be found in [14].

C. Temporal Analysis Techniques
Since worm propagation involves a temporal characteristic, we thought about improving the detection accuracy by considering the temporal dimension. We present three temporal analysis techniques which use different types of sliding windows. We evaluated these temporal techniques using the reduced datasets that are based on the most-contributive features selected at the feature selection stage. The challenge here is to represent the temporal dimension authentically while using a static classification algorithm. All three techniques are described below.

1) Simple sliding window
This technique proposes constructing a single window from the given dataset. Given a window size \( k \), the new instance will include \( k \) original sequential instances. For example, with a dataset having instances \( [1,2,3,4,5] \) and window size \( k=3 \) we create three instances: \( [1,2,3], [2,3,4], \) and \( [3,4,5] \), respectively. However, note that the current instance will include \( k \) multiplied by \( n \) attributes, where \( n \) is the number of attributes in an original instance.

2) Simple exponential compression
While being simple, the drawback of the simple sliding window representation is that the number of instances in each window grows linearly with the size of the window \( (k) \). Working with many instances could be problematic due to the long computation times. The simple exponential compression (SEC) technique overcomes this problem by averaging the “older” instances. The number of averaged instances will be \( 2^n \) where \( n \) is the place in the window starting with zero. For example, with a dataset \( [1,2,3,4,5,6,7] \) and a window size of 3 the resulting one instance will look as follows: \( [7, \text{avg}(5,6), \text{avg}(1,2,3,4)] \) where 7 is the “youngest” original instance. Generally speaking, each instance represents precisely the current features values, and in a more abstractive and summarized representation, the earlier values. Thus, the number of original instances the window covers grows exponentially with the size of the window \( (k) \).

3) Poisson exponential compression
We propose a technique that we have named Poisson exponential compression (PEC). The PEC technique is similar to SEC in the way that the instances are compressed in an exponential rate. However, this technique does not show precisely the “youngest” instances but the instances with higher Poisson probability density function (PDF). Poisson PDF is presented in (8). \( k \) is the index of the instance starting from the “youngest,” \( \lambda \) is a constant and was 2 in the experiment, and \( e \) is the base of the natural logarithm.

\[ R(k) = \frac{\lambda^k}{k!} e^{-\lambda} \]  

(8)

The motivation for using such a technique was the hypothesis that the most important instance in the window is not necessarily the “youngest.” The amount of averaged instances in place \( n \) is still \( 2^n \). For example, having a dataset \( [1,2,3,4,5,6,7] \) where 7 is the “youngest” instance, we can calculate the array of ranks, using the Poisson PDF, which will be \([0.01,0.04,0.09,0.18,0.27,0.27,0.14]\). Now, given a window size of 3, the resulting one instance will look as follows: \([6, \text{avg}(5,4), \text{avg}(1,2,3,7)]\).
III. DATASET DESCRIPTION

Since no public standard dataset was available for this study, we had to create our own dataset. We created a computer network environment consisting of a variety of computers (configurations). We injected worms into the network environment, and monitored various computer features in each of the infected and non-infected computers.

The computer network environment consisted of seven computers, which contained heterogenic hardware, and a server simulating the Internet.

We used the MS Windows performance tool, which enables monitoring of system features that appear in these main categories: Internet Control Message Protocol (ICMP), Internet Protocol (IP), Memory, Network Interface, Physical Disk, Processes, Processor, System, Transport Control Protocol (TCP), Threads, User Datagram Protocol (UDP). We also used VTrace [15], a software tool which can be installed on a PC running Windows. VTrace collects traces of the file system, the network, the disk drive, processes, threads, inter-process communication, writable objects, cursor changes, windows, and the keyboard. The Windows performance tool was configured to measure the features every second and store them in a log file as vectors. VTrace stored time-stamped events, which were collected into a second file. Both files were merged and contained a vector of 323 features for every second.

In order to perform the evaluation in a realistic environment, we considered three major aspects: computer hardware configuration, constant background application requiring high computational resources, and user activity. For each dataset related to an aspect the data were collected when each one of the five worms was injected separately, for a constant length of time. In addition there was a time period in which no worm was activated. In the rest of the paper we will refer to this clean period as a state named "Clean." The resulting datasets were combined into a single dataset.

**Computer hardware configuration:** Both computers ran MS Windows XP, since we considered it to be the most commonly used operating system. There were two configurations: old, using a PC based on Pentium III 800 MHz CPU, bus speed 133 MHz and memory 512 Mb, and new, using a PC based on Pentium IV 3 GHz CPU, bus speed 800 MHz and memory 1 GB.

**Background application activity:** There were two configurations: with background application activity and without. We ran the WEKA mathematical processing application software [16] which mainly effected the following features: Processor Time (usage of 100%); Page Faults/sec; Avg Disk Bytes/Transfer; Avg Disk Bytes/Write; DiskWrites/sec.

**User activity:** A user opened several applications, including Internet Explorer, Word, Excel MSN messenger, and Windows Media Player in a scheduled order. There were two configurations: with user activity and without. The exact schedule of the user activity is described in Table I.

### TABLE I

<table>
<thead>
<tr>
<th>Time period</th>
<th>User operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5 minutes</td>
<td>- Opening 10 MS Word instances</td>
</tr>
<tr>
<td></td>
<td>- Downloading two files simultaneously</td>
</tr>
<tr>
<td>5-10 minutes</td>
<td>- Opening 5 instances of MS Excel</td>
</tr>
<tr>
<td></td>
<td>- Generating random number in MS Excel</td>
</tr>
<tr>
<td></td>
<td>- One file downloading</td>
</tr>
<tr>
<td></td>
<td>- Listening to internet radio</td>
</tr>
<tr>
<td>10-15 minutes</td>
<td>- Opening 12 instances of MS Word</td>
</tr>
<tr>
<td></td>
<td>- Downloading one file</td>
</tr>
<tr>
<td>15-20 minutes</td>
<td>- Opening 9 instances of MS Excel</td>
</tr>
<tr>
<td></td>
<td>- Generating random numbers in MS Excel</td>
</tr>
<tr>
<td></td>
<td>- Browsing the internet (using MS IE)</td>
</tr>
</tbody>
</table>

In each time period all the user operations were performed simultaneously.

During the evaluation we used five different real worms. The description of each one is presented below.

1. **Dabber.A** (Daber.A)
   - This worm scans networks for random IP addresses, searching for victim machines that have the ftp component of the Sasser worm installed on port 5554.
   - When the worm finds a suitable victim machine, it sends a vulnerability exploit to it to infect the system. It then launches the command shell on port 8967. It also installs a backdoor on port 9898 to receive external commands.

2. **W32.Sasser.D** (Sasser.C)
   - This worm spreads by generating random IP addresses using 128 threads. The IP addresses are generated so that 48% of them should be close to the current computer by using the current computer's IP and 52% of them completely randomly. It connects to the remote computer using TCP port 445 and if the connection is established, a remote shell is opened. The remote shell is used to connect to the infected computer's FTP server and transfer the worm.

3. **W32.Deborm.Y** (DebormY)
   - This worm scans the local network and tries to propagate to other computers on the local network. It attempts to share CS (C drive) using the accounts of the administrator, owner or guest (it succeeds if a certain account does not have a password).

4. **W32.Korgo.X** (PadobotKorgoX)
   - This worm generates random IP addresses and exploits the LSASS Buffer overrun vulnerability using TCP port 445. If it succeeds in taking over a computer, the newly infected computer will send a request to download the worm from the infecting computer by using a random TCP port.

5. **Slackor.A** (Slackor.A)
   - When the Slackor worm is run, it sends a SYN TCP packet to randomly generated IP addresses through port 445 to search for the systems using Server Message Block (SMB). It then attempts to connect to the Windows default shares on these systems by using the username and password pair that it carries. If successful, it tries to copy the worm to the system.
IV. EVALUATION TECHNIQUES

A. Evaluation Measures

In order to perform the comparisons of the methods described, we employed the commonly used evaluation measures: True Positive Rate (TPR), shown in equation (9), False Positive Rate (FPR), shown in equation (10), and Accuracy, shown in equation (11). TP is the number of true positive examples. FP is the number of false positive examples. TN is the number of true negative examples. FN is the number of false negative examples.

\[
\text{TPR} = \frac{TP}{TP + FN} \quad (9)
\]

\[
\text{FPR} = \frac{FP}{FP + TN} \quad (10)
\]

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (11)
\]

Additionally we used Receiver Operating Characteristic (ROC) curves in order to evaluate different methods. An ROC curve is a graphical representation of the tradeoff between the false negative and false positive rates for every possible cut off. Equivalently, the ROC curve is the representation of the tradeoffs between Sensitivity and Specificity.

B. Causal Indexes Analysis

The Causal Indexes (CI) method estimates the influence of each input feature on the classification output. The quasi-quantitative influence of each input on each output is calculated based on the connection weights of a trained ANN [17]. The CI is calculated as the sum of the product of all “pathways” between each input to each output:

\[
CI_{ij} = \sum_{j=1}^{n} w_{ij} \cdot w_{jk} \quad (12)
\]

Equation (12) presents the basic formula of the CI, from input \( i \) to output \( k \). \( w_{ij} \) is the connection weight from input \( i \) to the hidden neuron \( j \). \( w_{jk} \) is the connection weight from hidden neuron \( j \) to output \( k \). The product of these is computed for all the \( n \) hidden neurons. The CI reveals the influence direction (positive or negative) and the relative magnitude of the relationship of any input and any output.

V. EXPERIMENTAL SEQUENCE

We performed two types of experiments. First, we evaluated several feature selection methods, as a preprocessing stage. Second, we evaluated several temporal representations.

A. Feature Selection Experiments

The aim of these experiments was to find the best of the three different feature selection methods: Hidden neurons relative variance based method, Fisher’s score, and Gain Ratio. Gain Ratio improved the performance of ANN as a classification method in [18] more than any other feature selection method. In addition to these three feature selection methods, we used a common feature extraction method known as Principal Component Analysis (PCA) for comparison. For each feature selection method we chose six different subsets of features ranked by this method: top5, top10, top20, top30, top50, and full. We did the same for PCA by choosing the most significant principal components. Each of the sets contained respectively top 5, 10, 20, 30, 50, and 323 features. Thus, the total amount of subsets obtained was \( 4 \times 6 = 24 \). For each subset we performed five different experiments. In each experiment one of the five different worms was removed from the training set. During each experiment we trained the ANN using a dataset constructed from the instances of four worms and 80% of the clean instances. The test set contained only instances of the fifth worm and an additional 20% of clean instances. Thus, the total number of experiments in this section was \( 4 \times 6 \times 5 = 120 \). Note that the test sets on which the ANN was tested contained only a new worm, which did not appear in the training set.

B. Temporal Analysis Experiments

After the best feature selection algorithm was identified, we examined the potential of improving the detection performance by using different temporal representation techniques. We used three techniques: Simple sliding window, Simple exponential compression, and Poisson exponential compression, which were explained earlier in section II.C. Using the best feature selection algorithm, we chose the 20 best attributes. We used the reduced dataset to create three different datasets by applying three different temporal representation techniques. With each one of these datasets we performed five experiments, by moving one worm, out of five, from the training set to the test set. As in the feature selection case, we included 20% of clean examples in the test set. Thus, the total number of experiments was \( 3 \times 5 \times 5 = 75 \). The sliding window size in all experiments was 5; thus the input vector size was constant and equal to \( 20 \times 5 = 100 \).

VI. RESULTS

A. Feature Selection Results

The worm classification results using the feature selection techniques are summarized in Table II. Each cell represents five different experiments, one for each missing worm. The values in the cells are the detection accuracy averages of these five experiments. It can be seen that Fisher’s score method outperformed the other techniques.
TABLE III

<table>
<thead>
<tr>
<th>Att.</th>
<th>Attribute name</th>
<th>Fisher’s score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Perf_MemoryPool_Paged_Allocs</td>
<td>23.80</td>
</tr>
<tr>
<td>2</td>
<td>Perf_MemoryCache_Bites</td>
<td>17.09</td>
</tr>
<tr>
<td>3</td>
<td>Thread_Total_Context_Switches_per_sec</td>
<td>15.24</td>
</tr>
<tr>
<td>4</td>
<td>SystemContext_Switches_per_sec</td>
<td>14.54</td>
</tr>
<tr>
<td>5</td>
<td>Perf_MemorySystem_Driver_Total_Bytes</td>
<td>13.66</td>
</tr>
</tbody>
</table>

Fig. 1 Averaged ROC curves for four different feature selection techniques

The areas under the ROC curves are presented in Table IV. Despite the fact that Fisher’s score attained the best accuracy value, the Relative Variance method of selecting the reduced feature set has the best separation level.

Table V presents the CI values of the top 5 attributes. The attribute numbers are related underlined and defined in Section III.

If a certain CI is positive with a relatively high magnitude, then the related input influences the related output in the same direction, i.e., if the input is high the output will rise also. Alternatively, if the CI is negative with a relatively high magnitude, then the related input influences the related output in the opposite direction, and thus if it is high the output will be relatively low.

From the CI’s presented in Table V the following “rules” can be formulated:

- If 2 is high and 3, 4, 5 are low, then Clean
- If 2, 3 and 4 are high and 1 is low, then DabberA
- If 2 is high and 1, 3, 4, 5 are low, then SasserC
- If 1, 2, 3, 4, 5 are low, then DabormY
- If 1 and 5 are high and 2 is low, then Padobot
- If 2 is low and 1, 3, 4, 5 are average then SlakorA

C. Temporal Analysis Results

The evaluation results of different temporal preprocessing techniques are presented in Table VI. Each cell in this table represents the averaged accuracy of five experiments, one for each missing worm. Simple Window showed the best results in this category, achieving an accuracy of 0.85. As already mentioned, we took 20 best attributes selected by Fisher’s score as raw data and preprocessed them with different preprocessing techniques. Thus, it is interesting to compare the accuracy of Simple Window technique, 0.85, to the accuracy of regular Fisher’s score with 20 attributes, 0.84. Although there is an increase in the accuracy, it is very minor. Such a phenomenon can be explained by the fact that it takes a very short time for a worm to initialize and after the initialization phase it starts to operate in a constant manner. Thus, in this case, the temporal preprocessing does not help much, but only adds attributes, thus making the ANN more
complicated for training.

<table>
<thead>
<tr>
<th>TABLE VI</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPORAL ANALYSIS RESULTS</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Without temp. preprocessing</td>
</tr>
<tr>
<td>Simple Window</td>
</tr>
<tr>
<td>Simple Exponential</td>
</tr>
<tr>
<td>Poisson Exponential</td>
</tr>
</tbody>
</table>

VII. DISCUSSION AND CONCLUSION

In this paper we presented three different feature selection techniques for selecting computer behavior features that can be used for detecting the presence of worms. We showed that the accuracy of worm detection may increase when the detection process is using only the most important features. We presented the five most important attributes and derived different rules, related to these attributes, from the trained ANN by using the Causal Indices method. Additionally, we used three temporal preprocessing techniques to see whether they are able increase the accuracy of the detection.

We showed that Fisher’s score, despite its simplicity, appears to be a good feature selection method for computer behavior data. It demonstrated the best accuracy and minimal standard deviation using only the top five features. Additionally, we showed that preliminary temporal processing does not increase the detection accuracy significantly and may even decrease it. This may happen due to the fact that the initialization period of the worm is very short and after this period of time worm operation is stable and temporal preprocessing becomes irrelevant.

For future work, we propose the evaluation of the system using additional types of malwares such as Viruses, Trojans and so on.

ACKNOWLEDGMENT

This work was done as a part of a Deutsche-Telekom Co. /Ben-Gurion University joint research project.

We would like to thank Clint Feher, for providing the worm software and for creating the large number of security datasets we used in this study.

REFERENCES
