J Comput Virol (2011) 7:95–105
DOI 10.1007/s11416-010-0140-6

ORIGINAL PAPER

Determining malicious executable distinguishing attributes and low-complexity detection

Hassan Khan · Fauzan Mirza · Syed Ali Khayam

Received: 19 July 2009 / Accepted: 7 January 2010 / Published online: 27 January 2010

Abstract Detection of rapidly evolving malware requires classification techniques that can effectively and efficiently detect zero-day attacks. Such detection is based on a robust model of benign behavior and deviations from that model are used to detect malicious behavior. In this paper we propose a low-complexity host-based technique that uses deviations in static file attributes to detect malicious executables. We first develop simple statistical models of static file attributes derived from the empirical data of thousands of benign executables. Deviations among the attribute models of benign and malware executables are then quantified using information-theoretic (Kullback-Leibler-based) divergence measures. This quantification reveals distinguishing attributes that are considerably divergent between benign and malware executables and therefore can be used for detection. We use the benign models of divergent attributes in cross-correlation and log-likelihood frameworks to classify malicious executables. Our results, using over 4,000 malicious file samples, indicate that the proposed detector provides reasonably high detection accuracy, while having significantly lower complexity than existing detectors.

H. Khan (✉) · F. Mirza · S. A. Khayam
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad 44000, Pakistan
e-mail: 43mhassan@seecs.edu.pk; hassankh@usc.edu

F. Mirza
e-mail: fauzan.mirza@seecs.edu.pk

S. A. Khayam
e-mail: ali.khayam@seecs.edu.pk

1 Introduction

Malware is a serious global problem that has repeatedly caused losses in productivity, time, reputation, service availability, and data integrity for both individuals as well as enterprises. Consequently, over the last decade this problem has emerged as one of the biggest challenges for the information security community. The rapidly evolving nature of contemporary malware adds an additional layer of complexity to this already-challenging problem domain. Due to the rapid evolution and unseen natures of emerging malware threats, it is important to develop real-time defenses that can detect zero-day attacks. Moreover, a pragmatic host-based malware detection technique should have low processing and memory requirements in order to cater for the complexity-constrained natures of endhost systems.

In this paper, we propose a host-based approach that leverages static *attributes* of binary executables to detect malicious files. We first identify distinguishing attributes that differ considerably among benign and malicious executables using a diverse dataset containing thousands of benign and malicious executables. We statistically model static attributes observed in benign and malicious files and then quantify the divergence between these attributes in a Kullback-Leibler based information-theoretic framework. This quantification yields distinguishing attributes that are considerably different among malicious and benign executables.

After identifying distinguishing file attributes, we present a simple and low-complexity classifier which employs these attributes for malware detection using cross-correlation and log-likelihood tests. In empirical tests, on a sample of 5,290 previously unseen executables, our classifier provides a detection rate of 86.3, 85.5 and 62.5% for worms, viruses and Trojans, respectively, with an average false positive rate of 6.7%. During empirical tests, we do not distinguish among

compressed (packed) or uncompressed (unpacked) malware. Performance comparison with machine learning and existing host-based malware detection techniques show that the proposed detector, while having significantly lower complexity (less than an order of magnitude in most cases), provides much better accuracy than existing techniques.

The rest of this paper is organized as follows: Sect. 2 describes related work in this area. Section 3 describes the data collection and the diversity of the executables that are used in this work. Section 4 statistically models executable file attributes and then compares the models derived from malicious and benign executables to quantify their attributes' differences. Section 5 evaluates the performance (accuracy and complexity) of statistical and machine learning classification tools and also compares performance with existing techniques. Section 6 discusses possible attacks and countermeasures on the proposed detector. Section 7 summarizes the key findings of this paper.

2 Related work

Pattern-recognition approaches are well-known and conventional approaches to reliable known malicious software detection. A great deal of research has been done in this area [1], including study of techniques for automatic signature extraction [2] and the application of machine learning and data mining algorithms [3,4]. Techniques for specifically analyzing Win32 malware have also been developed [5,6].

High accuracy rates have been achieved by using a generic n-gram based detection approach [7]. Stolfo et al. [8] discuss possibilities to detect embedded malware and malicious executables using 1-gram and 2-gram patterns. Various other heuristic based malware detection approaches which attempt analysis of the machine code [9,10] have been explored. However, except for the pattern recognition approach, these techniques are computationally expensive and unsuitable for real-time analysis and classification. On other hand, the pattern-recognition approach is much less effective at detecting zero-day malware.

While many malware detection techniques have been proposed, in this section we only discuss relevant *Win32 portable executable* malware detection techniques and their limitations. A closely related study to the present work is by Schultz et al. [11] where the authors conducted experiments on viruses and trojan horses and used data mining techniques for detection of malicious code. Three methods proposed by Schultz et al. include binary signature, hexdump and string sequence analysis. For the binary signature they used 38 malicious and 206 benign executables of the PE file format and extracted three types of information: (1) DLLs used (2) function calls from the DLLs, and (3) number of different system calls from each DLL. Their processing resulted in 2,229 binary attributes and 30 integer attributes. The hexdump method worked by extracting two-byte words from an executable which were treated as a binary attribute that were either present or from an executable. Their strings method extracted strings from a binary file and each string was treated as an attribute that was either present or missing from an executable. The hexdump approach could generate up to 65,536 attributes and the possible attributes from the strings method is also very large.

Our approach is similar to the binary signature technique of Schultz et al., because we focus on the static attributes of the file which also contains the signature. The high processing and storage requirements of [11] make it impractical for resource-constrained endhosts. On the other hand, we treat complexity as a fundamental design objective in our detection algorithm. We compare our results with the detection results for binary signature reported in [11] and n-gram based approaches in [8]. It should also be mentioned that details of DLL names and attribute values are not provided in [11], and the URL for the dataset used in that paper is also inaccessible. Therefore, it is not possible to develop the detector of [11] or to reproduce its results. To compare performance with the binary signature approach, we simply borrow the detection and false positive rates from [11]. Performance comparison with [8] is based on our dataset which contains more malware than the dataset used by [8].

3 Dataset description

While our modeling, analysis and detection approaches are quite generic, we design and evaluate our proposed detector on the PE file format, which is the standard executable (EXE) file format used by the Microsoft Windows operating system. This format was chosen because a large number of Windows-based malware samples in this format were available on the Internet. We downloaded various malicious samples from VX Heaven Virus Collection website [12], which included infected executables, virus loaders, worms and trojans horses. VX Heaven Virus Collections is the largest publicly available free malware data set which contains distinct samples or distinct variants of a sample and uses the malware nomenclature. This dataset has already been used by [7,8].

The benign and malicious executables contain both compressed and encrypted samples (packed). In our benign dataset 61.3 and 38.7% of the executables were non-packed and packed respectively. Similarly 47.2 and 52.8% of malicious executables were non-packed and packed respectively. We do not distinguish between packed and non-packed samples. In particular, since we wanted to conduct this work on typical malware and benign executable samples, we did not preprocess (e.g.,unpack) any of the samples prior to the analysis or tests. Although generic unpacking techniques

Table 1 Executable file attributes analyzed in this paper

1. DOS checksum. 2. Number of sections. 3. Time stamp. 4. Number of symbols. 5. Size of code. 6. Size of initialized data section. 7. Size of un-initialized data section. 8. Major OS version. 9. Minor OS version. 10. Major image version. 11. Minor image version. 12. Major subsystem version. 13. Minor subsystem version. 14. Entry point. 15. Image file checksum. 16. Size of exports. 17. Size of imports. 18. Size of resources. 19. Size of exceptions. 20. Size of attribute certificate table. 21. Size of base relocations. 22. Size of copyright. 23. Size of thread local storage. 24. Size of bound imports. 25. Size of import table. 26. Dynamic link libraries used. 27. Registry references made. 28. Cursors. 29. Bitmaps. 30. Icons. 31. Menus. 32. Dialogs. 33. Strings. 34. Font directories. 35. Fonts. 36. Accelerators. 37. RC data. 38. Message tables. 39. Group cursors. 40. Group icons. 41. Version. 42. Entropy of code/ text section.

for executable files exist [13,14], these techniques require simulation of executable files which significantly increases the complexity of proposed approach. In the following section, we identify distinguishing features of malware without discriminating between packed and non-packed executable samples. However, we consider entropy of code section as a discriminating feature.

Throughout this paper, we refer to benign, worms, viruses and trojans as different *genres* of executables. For our experiments we have collected 1410 viruses, 160 worms and 2520 trojans. We also gathered 1,200 benign samples from machines running Microsoft Windows XP and Microsoft Windows 2000 operating systems. The benign samples include executable files bundled with operating system, third party installers, third party executables, and third party uninstallers. Our diverse selection of benign executables ensures soundness of our results in worst case scenarios.

In order to select the attributes that could distinguish among the benign and malicious samples, we analyze the DOS header, COFF header, PE optional header, data directories, import table, sections and resources related information of a PE file. All the attributes that are derived from these file sections are listed in Table 1. These attributes are extracted from our diverse dataset of benign and malicious executables. Some attributes are specific to a malicious class yet most of the attributes hold their properties throughout the malicious (worms, viruses and trojan horses) genres.

4 Statistical analysis and quantification of executable file attributes

Like prior zero-day malware detection studies [7,11], we subdivide the present problem of malware detection using executable file attributes into two sub-problems: *identification of distinguishing attributes* and *classification using the identified attributes*. This section focuses on the former problem where we compare malware and benign files to identify distinguishing or divergent attributes. The main challenge in this context is accurate modeling and quantification of the divergence in file attributes. To this end, we statistically model the executable file attributes as discrete random variables, and then use information divergence measures to quantify the differences between benign and malicious files' attributes.

4.1 Modeling file attributes as discrete random variables

We model each file attribute as a discrete random variable by mapping each plausible value of an attribute to an integer outcome. Using the benign and malware datasets, for each random variable we compute histograms of each outcome. By normalizing these histograms, we obtain the probability mass functions (PMFs) of the attribute random variables. The outcomes to integer mappings for some representative random variables are enumerated in Table 2. The first three attributes in Table 2 are checked for their presence only, that is whether value of these attributes is present (non-zero) or absent (zero). The entropy of code section is the byte distribution entropy.[1] We should emphasize that Table 2 lists all the file attributes that are used for classification in the next section. Using this mapping and our dataset, future studies can compare accuracy of their results with the technique proposed in this work.

4.2 Quantification of attribute dissimilarities using information divergence

After modeling the file attributes as discrete random variables, for each attributed random variable we need to quantify the difference between the PMF derived from the benign samples and the PMFs derived from malware samples. In order to quantify these differences, we use Kullback-Leibler based information divergence measures [15]. A brief description of these measures follows.

For two PMFs $p(X)$ and $q(X)$ of a discrete random variable X, the Kullback-Leibler divergence provides a measure of how statistically different p and q are from each other.

[1] The standard formula for information entropy of a byte-distribution was used: $H(X) = -\sum_{i=0}^{255} p(i) \log_2 p(i)$.

Table 2 Possible values for attribute random variable

Attribute	Possible outcomes	Condition
PE checksum	2	Value is zero OR non-zero
Debug	2	Value is zero OR non-zero
OS and image versions	2	Value is zero OR non-zero
Number of sections	6	<3, 3, 4, 5, 6, >6
Entropy of code section	3	Indeterminable, ≤4, > 4
Resources	5	0, ≤10, ≤20, ≤30, >30
Entry point	3	Invalid, Valid, Obfuscated
Registry keys	3	0, 1, >1
Registry startup keys	2	0, 1
Networking related DLLs	5	{wininet, ws2_32, wsock32, raspi}.dll
High frequency DLLs in benign	7	{winmm, shell32, comctl32, gdi32, version, shellwapi}.dll
Size	6	≤500 KB, ≤1000 KB, ≤2000 KB, ≤3000 KB, ≤4000 KB, >4000 KB

Mathematically it can be expressed as

$$D(p(X)\|q(X)) = \sum_{i\in\Lambda} p(X=i)\log_2\left(\frac{p(X=i)}{q(X=i)}\right) \quad (1)$$

where Λ represents the set of all possible values that the random variable X can take. The Kullback-Leibler (K-L) divergence provides a non-negative information divergence measure which is zero if and only if $p = q$.

As an example, let us use the K-L divergence measure to quantify the difference between the PE Checksum random variable in the benign and worm samples. Let this binary random variable (see Table 2) be represented by X, and denote its benign and worm-based PMFs as $p(X)$ and $q(X)$, respectively. Then the K-L divergence of X between the benign and malicious datasets is:

$$D(p\|q) = p(0)\log_2\frac{p(0)}{q(0)} + p(1)\log_2\frac{p(1)}{q(1)} \quad (2)$$

A closer examination of Eqs. (1) and (2) reveals the two properties of the K-L divergence that are of direct relevance to the present problem. First, the K-L divergence is non-symmetric: In general, $D(p(X)\|q(X)) \neq D(q(X)\|p(X))$, and therefore K-L is not a true distance metric [16]. Thus for accurate feature quantification, we need to evaluate both $D(p(X)\|q(X))$ and $D(q(X)\|p(X))$. Second, the $q(X)$ PMF should be continuous with respect to the $p(X)$ distributions. That is, at any $X = i$, where $i \in \Lambda$, if we have $p(i) \neq 0$, $q(i) = 0$, then $D(p\|q) = \infty$. Since we do not have control over the benign and malware PMFs, this condition is quite stringent and can lead to biased results.

To cater for the above problems of K-L divergence, we use the Resistor Average (RA) information divergence measure [17,18]. The Resistor Average divergence, $\Re$,of two discrete PMFs $p(X)$ and $q(X)$ is defined as:

$$\Re(p(X), q(X)) = \frac{1}{D(p(X)\|q(X))} + \frac{1}{D(q(X)\|p(X))} \quad (3)$$

RA divergence uses two K-L divergence $D(p(X)\|q(X))$ and $D(q(X)\|p(X))$ to generate a symmetric metric. Moreover, the PMFs need not be continuous with respect to the other.

RA divergence provides us with a quantification of the difference between two attributes' PMFs derived from different genres, and hence can be used to identify distinguishing attributes. However, to establish the reliability of the identified attributes, we also need to ensure that these attributes exhibit consistent behavior within each file genre. For instance, let us assume that the PE Checksum attribute's RA divergence is very high between benign and worm samples. While RA divergence will quantify the divergence of this attribute across file genres, if this is a truly distinguishing attribute then its value should be consistent within each file genre. That is, if we randomly subdivide the benign (worm) dataset into smaller subsets, the statistics of this attribute should be similar for all the benign (worm) subsets. RA divergence across file genres does not provide such quantification. Therefore, we propose a new metric, called Differential RA divergence (DiffRA), which quantifies the similarity between samples from the same file genre.

Given two RA divergence values $\Re_A$ and $\Re_B$ generated from distinct pairwise subsets of the same file genre, DiffRA divergence ∂_{Re} is defined as:

$$\partial_{\Re}(\Re_A, \Re_B) = |\Re_A - \Re_B| \quad (4)$$

If an attribute has consistent statistical behavior within a file genre, ∂_{Re} will be close to zero, while for dissimilar $\Re$ distributions, $\partial_{\Re}$ will be large.

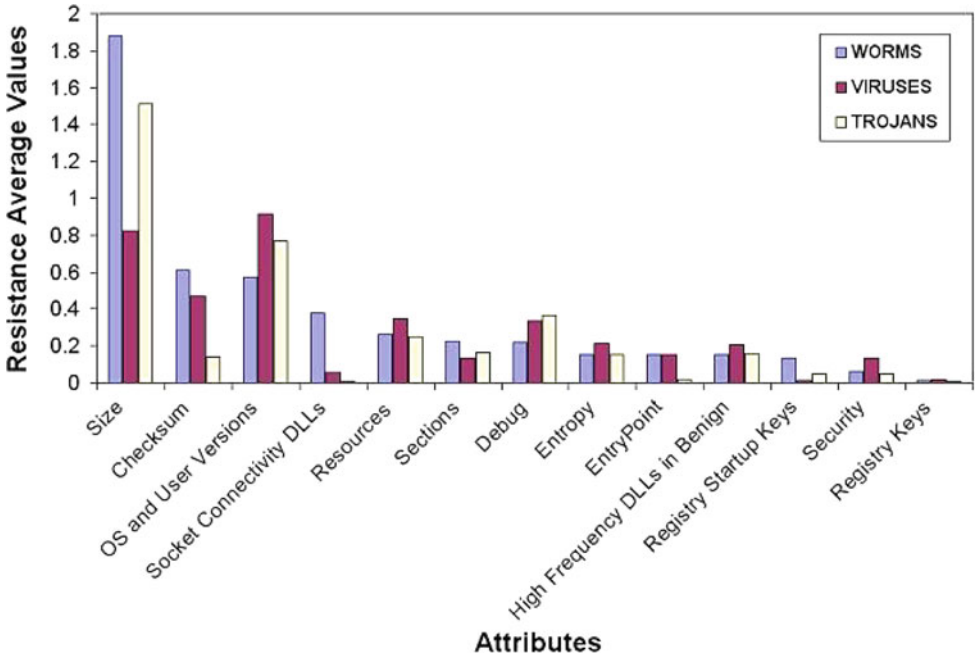

Fig. 1 Resistor Average ($\Re$) divergence values for distinguishing executable attributes: *BEN* benign, *TRJ* trojan horse, *VIR* virus, and *WRM* worm (only 13 attributes having the highest $\Re$ values are shown)

4.3 Identification of distinguishing attributes

We now use the measures described in the last section to identify distinguishing attributes that are considerably different across benign and malware samples. After identifying these attributes, the next section will show how these attributes can be used for malware classification.

Throughout this paper, we use the fivefold cross validation technique [19] to ensure the accuracy and reliability of the results over unseen data. To this end, executable files in each of the four file genres (benign, worm, virus and trojan horses) were subdivided into 10 distinct subsets. For each file genre, we created 5 pairs of these subsets for pair-wise comparison of the statistics of the executables file attributes. Results for each pair in a file genre were compared with the results of other pairs in order to determine the validity of results among a large number of unseen samples.

Using 5-fold cross validation, we first calculate the divergence values among attributes of benign and worm executables, benign and virus executables, and benign and trojan executables. We also calculate the DiffRA divergence (∂_{Re}) among benign executables, among worm executables, among virus executables, and among trojan executables. For a distinguishing feature, we expect that the $\Re$ divergence between different benign and malicious genres will be large, while the DiffRA (∂_{Re}) values within a genre will be small.

Figure 1 shows the results of RA divergence for the 13 attributes that resulted in the highest values. Similarly, Fig. 2 shows the DiffRA (∂_{Re}) values within each file genre. It is apparent from Fig. 1 that the $\Re$ values are large enough to prove that these top-13 attributes differ significantly for the benign and malicious executables. Also, it is apparent from Fig. 2 that ∂_{Re} values of these attributes are quite small within a genre. In order to avoid overfitting, we only use these 13 attributes with highest RA divergence and do not consider the remaining attributes listed in Table 1.

4.4 Discussion

Based on the $\Re$ and ∂_{Re} divergence values, we deduce that the attributes shown in Figs. 1 and 2 are significantly divergent between benign and malware samples, and hence can be used for classification. Classifiers that employ these attributes for malware detection are proposed in the following section. Before we proceed to develop classifiers that rely on the distinguishing attributes, some important questions should be highlighted. For instance, under what conditions is a malware executable classified as benign (missed detections)? And how much does each attribute contribute to the detection accuracy? Also, how difficult or easy is it to launch mimicry attacks on the distinguishing attributes? We discuss these points in detail in Sect. 6.

5 Detection using statistical and machine learning techniques

In this section, we leverage the distinguishing attributes identified in the preceding section to detect zero-day malicious executables. A fundamental constraint that we invoke in this section is to only use benign data to train the attribute models. Malware are then detected using the (trained) benign attribute

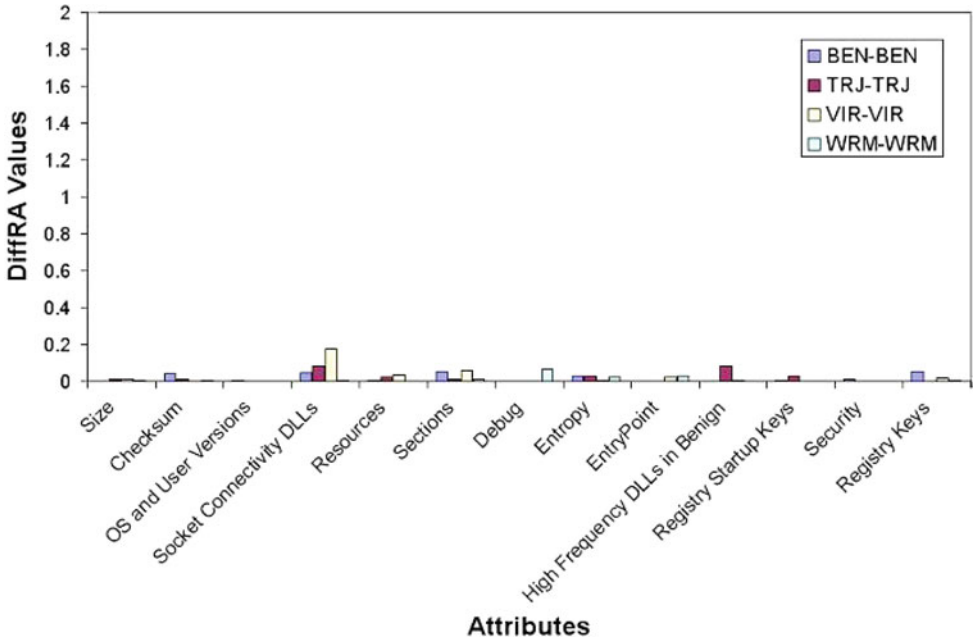

Fig. 2 DiffRA Divergence ($\partial_{\Re}$) between samples of the same file genre

models in a true anomaly detection framework. The detection dataset contains different samples from training dataset and no sample overlaps among these datasets. We also compare the accuracy (in terms of true and false positive rates) and complexity (in terms of CPU cycles) of our proposed detector with existing techniques.

First, we propose a simple and low-complexity statistical detector which compares the learned statistical models of file attributes with the attributes observed in an unknown file using cross-correlation and log-likelihood tests. The output of these tests is used to classify the unknown file. As a baseline for performance evaluation, we compare the performance of the proposed detector with a support vector machine (SVM) based classifier; SVMs have been used extensively for anomaly detection in prior studies [20,21].

In order to evaluate the accuracy of the detectors, we compute the count for *true positives* (TP), *true negatives* (TN), *false positives* (FP) and *false negatives* (FN) as described in [11]. True positives and true negatives occur when our approach correctly classifies a benign or malicious executable. A false positive occurs when our approach mislabels a benign executable as malicious. A false negative is reported when our approach mislabels a malicious executable as benign. Detection rate is defined as $\frac{TP}{TP+FN}$, false positive rate as $\frac{FP}{TN+FP}$, and overall accuracy as $\frac{TP+TN}{TP+TN+FP+FN}$.

5.1 Detection using statistical models of distinguishing attributes

We employ two low-complexity statistical measures to leverage the distinguishing executable file attributes identified in the preceding section. For both the measures, we first learn the benign attribute PMFs using half of the benign dataset, referred to as the "training dataset". The other half is used to ascertain false positive rates and is henceforth referred to as the "benign test dataset". The training dataset and test dataset contain distinct samples and there are no samples that overlap in these datasets. The attributes observed in the unknown file are then compared against the benign PMFs using cross-correlation and log-likelihood tests.

We again emphasize that many of the existing anomaly detectors (including the SVM-based anomaly detector discussed in the following section) leverage malware statistics to train the anomaly detectors [7,21]. Consequently, the accuracy of these detectors is dependent on the malware that are used to train them. We, on the other hand, want to develop a true anomaly detector that does not rely on malicious statistics and flags maliciousness by characterizing deviations from a robust model of benign attributes. Hence, both the statistical tests employed in this section rely only on the benign statistics.

Detection using cross-correlation test

For classification, we first use the cross-correlation measure [22] which quantifies the similarity between two vectors. To use this measure, we treat the ordered list of distinguishing attributes as a discrete random process, $\mathbf{Y}$, where each constituent random variable of the process corresponds to a distinct attribute random variable. From the benign samples, we know the PMFs of each constituent random variable. We treat the distinguishing attribute vector $\mathbf{X}$ from the unknown file as a realization of this random process. To find the cross-correlation of the unknown realization with benign

realizations, we use the benign attributes' PMFs to generate an ensemble of n realizations of the random process, say $Y_i, i = 1, 2, \ldots, n$. We then generate a process realization $\mathbf{Y}$ by taking the ensemble average of each attribute; i.e., for an attribute indexed at k, $Y[K] = \frac{1}{n}\sum_{i=1}^{n} Y_i[k]$. We compute the cross-correlation of $\mathbf{X}$ with each $\mathbf{Y}$ as:

$$O(X, Y) = \sum_{k\in\phi} X[k]Y[k] \tag{5}$$

where ϕ is an ordered set of distinguishing attribute random variables. High values of cross-correlation imply that the unknown file's attributes closely match the statistics observed in the training dataset. Low values of cross-correlation imply that the file is potentially malicious. For our experimental evaluation, we set *n=25* and use different threshold values to classify the malware and benign test dataset files. Based on these varying thresholds, we observe different accuracies using the cross-correlation measure.

Detection using log-likelihood test

Another measure that we use for classification is the log-likelihood [23] of the random process realization observed in the unknown file. Specifically, assuming independence across attributes, the likelihood that the unknown realization $\mathbf{X}$ has been derived from the learned (benign) random process $\bar{Y}$ is

$$\begin{aligned} L(X|\bar{Y}) &= \prod_{k\in\phi} P\{\bar{Y}[k] = X[k]\} \\ &= -\sum_{k\in\phi} \log P\{\bar{Y}[k] = X[k]\} \end{aligned} \tag{6}$$

where ϕ is an ordered set of distinguishing attribute random variables. Like the correlation measure, higher values of this measure imply a potentially benign file, while malicious samples should have low likelihood values.

5.2 Detection using SVM

To compare our results with an alternative technique for classification, we employed a machine learning technique, Support Vector Machines (SVMs), for the classification purpose. Due to their binary classification and non-statistical nature, SVMs have been used quite frequently for anomaly detection in prior studies [20,21]. SVMs are a set of related supervised learning method used for linear classification [24,25]. SVMs classify by mapping input vectors to a higher dimensional space where a maximal separating hyperplane is constructed. The distinguishing characteristic of SVM is that they simultaneously minimize the empirical classification error and maximize the geometric margin.

Given training vectors $x_i \in \Re^n, i = 1, 2, \ldots, l$ in two classes, and a vector $y \in \Re^i$ such that each $y_i \in \{+1, -1\}$, an SVM for non-separable data considers the following primal optimization problem [26]:

$$\min \frac{1}{2}\omega^T\omega + O\sum_{i=1}^{l} \alpha_i y_i(\omega^T K(s_i, x) + b) \tag{7}$$

such that derivatives of the objective function vanish with respect to and subject to the constraint that $\alpha_i \geq 0, i = 1, 2, \ldots, l$. In the objective function ω is a perpendicular to the hyperplane that separates the positive and negative points, O is a parameter that is used to cost the α_i's, $K(s_i, x)$ is a non-linear kernel that maps the input data to another (possibly infinite dimensional) Euclidean space, and s_i's are points called the support vectors that maximize the separation between the positive and negative examples.

For SVM-based classification, we use SVM^{light}, which is an open-source implementation of SVM in the C language [27]. For each file genre, we used half of the samples for training and the remaining samples for testing. The points V, W and T in Fig. 3 show the SVM results for viruses, worms and trojans, respectively.

5.3 ROC-based performance evaluation of the proposed detector

We plot Receiver Operating Characteristics (ROC) curves for our approach and compare it with [11] and [8]. For our technique, we classify a file as malicious if either cross-correlation or log-likelihood classifiers flag it as malicious. For the classifiers, the detection threshold was changed over a range of values, and for each threshold, detection and false positive rates were logged.

Figure 3 shows the ROC curves of our Statistical Models approach, and the Mahalanobis [28] and Exemplars [16] based implementation of [8] for worms, viruses and trojans on our dataset. It can be observed that our approach gives very good detection rates with negligible false positive rates as compared to Mahalanobis and Exemplars based implementations of n-gram. For instance, at a false positive rate of 2%, our approach performs substantially better than the rest of the approaches. The detection rates of Mahalanobis and Exemplars based approaches of [8] commensurate with our results at a false positive rate of approximately 50%, which is unacceptable for any detector. We also show the binary signature approach as it is related to our detection approach. The binary signature method gives a constant detection rate of 30%. As well as having higher accuracy, it will also be shown shortly that in terms of time and memory complexity, our approach is considerably better than the signature based method approach of Schultz et al. At a false positive rate of 6.7%, our proposed detector achieves a detection rate of 86.3,

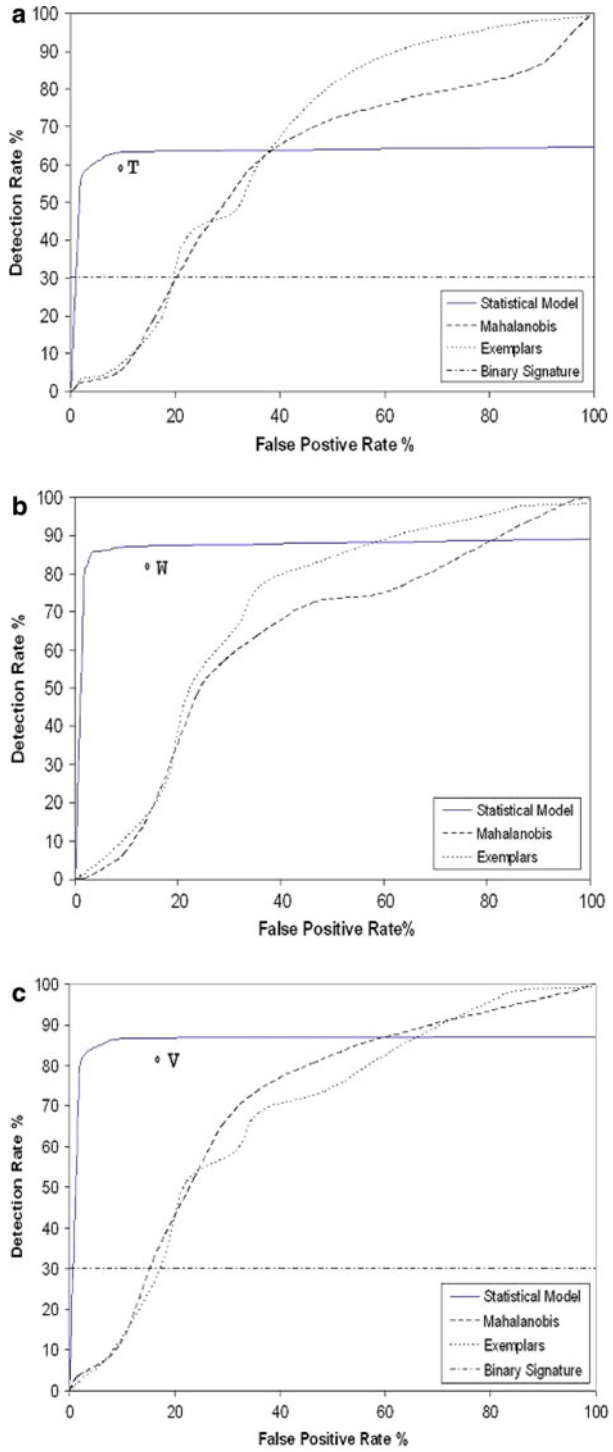

Fig. 3 Statistical Model and n-gram ROC for **a** worms, **b** viruses and **c** trojans. Statistical Models curve shows the results of proposed approach. Mahalanobis [28] and Exemplars [16] show the ROCs of [29]. Binary Signature shows the results of [11]. The points V, W and T in graph show the detection results for viruses, worms and trojans using SVMs, respectively (**a**) Worms, (**b**) Viruses, (**c**) Trojans

85.5 and 62.5% for worms, viruses and trojans, respectively, which is much better than an SVM approach operating on the same attributes.

We note that the detection results for trojans are not as encouraging as for viruses and worms. This is because trojans are typically quite similar to benign executables. We are currently investigating approaches to improve the detection rate for trojans.

5.4 Comparison of algorithm complexity

In addition to its accuracy advantages, CPU and memory requirements of our statistical model approach are far lower than other approaches of static malware detection. An intuitive explanation for this is that our detection algorithm may be considered as two sequential algorithms: (1) extraction of attributes from a given executable file; and (2) evaluation of the cross-correlation and log-likelihood tests. The complexity of the first part is $O(n)$, since the memory required and time taken to extract the attributes from the file is directly proportional to the size of the file. The complexity of the second part is $O(1)$, since both the cross-correlation and the log-likelihood tests are evaluated on constant-size inputs that are independent of the file properties, using only a small amount of memory (between 10–80 bytes), and will take the same amount of time and memory regardless of any of the file properties or values of the attributes (i.e., file contents).

To verify our complexity estimates, we conducted empirical tests to compare the complexity of our approach with the other approaches under consideration. Table 3 shows the complexity comparison of the statistical model based approach with Exemplar and Mahalanobis distance based approaches for different file sizes. It can be observed that for large file sizes the training complexity of the proposed detector is an order of magnitude lower than the other two detectors. Similarly, the run-time complexity of the proposed approach increases linearly with file size, while complexities of the other two approaches increase exponentially. For a file size of 2 KB, our approach is twice as efficient as n-gram based approaches in training and for detection it is 100 times more efficient. For a file size of 15.5 MB, our approach is 43 times more efficient in training and 55 times more efficient in detection.

It should be clear that the training time for the statistical model based detector is negligible as compared to the extensive and lengthy training of SVMs. Similarly, as opposed to the signature based approach which uses 2,229 binary attributes and 30 integer attributes, our proposed detector uses only 4 binary and 7 integer attributes.

6 Attacks and countermeasures

To defeat our approach, a malicious executable must have a sufficient number of attributes that match those of typical benign executables. By studying the divergence mea-

Table 3 CPU cycle count for complexity analysis of the proposed statistical model based detector with and Mahalanobis and Exemplars based approaches

File size	Training			Detection (real-time)		
	Statistical	Mahalanobis	Exemplar	Statistical	Mahalanobis	Exemplar
2 KB	1.0×10^7	2.6×10^7	2.8×10^7	1.0×10^7	1.4×10^9	1.1×10^9
5 KB	2.0×10^7	3.6×10^7	3.6×10^7	1.0×10^7	1.5×10^9	1.8×10^9
50 KB	3.1×10^7	1.7×10^8	1.7×10^8	3.2×10^7	7.4×10^9	9.7×10^9
475 KB	6.54×10^7	1.3×10^9	1.4×10^9	6.86×10^7	9.6×10^9	2.2×10^{10}
15,883 KB	1×10^9	4×10^{10}	4.5×10^{10}	1×10^9	4.8×10^{10}	6.1×10^{10}

Table 4 Modification of existing attributes of W32.Rontokbro@mm to demonstrate robustness against mimicry attacks

Attribute	Mimic-1	Mimic-2	Mimic-3	Mimic-4	Mimic-5	Mimic-6
Modified OS & image versions	N	Y	Y	Y	Y	Y
Modified PE checksum	N	N	Y	Y	Y	Y
Modified debug	N	N	N	Y	Y	Y
Modified eesources	N	N	N	N	Y	Y
Modified size	N	N	N	N	28% inc.	72% inc.
Cross-correlation	21.72	29.82	36.59	38.0	40.7	45.5
Log-likelihood	1.34	1.83	1.95	2.11	3.24	4.45

sures of benign and malicious executables, one can determine which attributes in a malicious executable must be modified to make it appear benign. For example, in Fig. 1, we see that our approach uses file size and number of GUI resources as distinguishing attributes. These attributes stand out because both of these attributes describe features which are generally quite specific to the nature of malware, as opposed to benign software. File size is usually an effective distinguishing attribute because a relatively small file size is useful for malware to spread effectively (i.e., less space for storage and quicker transmission over networks). Most malware executables lack attributes relating to GUIs (e.g., menu or dialog box resources, or GUI-related DLLs), because unlike benign applications, most malware are not designed for user interaction.

In order to demonstrate the robustness of our approach, we first identify those attributes that can be easily defeated by a malware author without compromising the efficiency of malware and then test our technique by exploiting those features for the W32.Rontokbro@mm worm. Table 2 lists all attributes that our approach uses. Although all of these attributes can be defeated, we categorize each attribute on the basis of how easily it can be changed in a malicious executable file and the inefficiency (in terms of execution or network transmission time) which such a change introduces. First, three attributes can be easily modified by making minor changes in the headers of an executable, and consequently they do not affect malware's efficacy in its operations. Resources can be added by using a tool like XN Resource Editor,[2] while Number of Sections can be increased by specifying options during the compilation. Addition of Resources and Sections modifies the Size attribute, but these features reduce the efficacy of malware in replication. Finally, remaining attributes in Table 2, require addition of source code. Fixing entry point is relatively complex task while Imported DLLs and Registry Keys can be added by modifying the code. Due to the unavailability of source code, we only show the results by modifying those attributes which do not require source code for modification. Table 4, shows the results of our experiments and demonstrates robustness of our approach against such mimicry attacks. It can be observed that after forging most of the easily modifiable attributes; the Mimic-1, Mimic-2, Mimic-3, Mimic-4 and Mimic-5 can still be detected for threshold value of 3.25 and 41 for Log-Likelihood and Cross-Correlation, respectively, which has an associated false positive rate of 9.8%.

To gain a better understanding of why our detection approach misclassifies some samples, resulting in false positives/negatives, we present the details of some samples in Table 5, which shows the attribute values for two samples from each executable genre (worm, trojan, virus and benign). For each genre, one sample was correctly identified as malicious or benign and the other sample resulted in a misclassification. This table shows that W32.Blaster.Worm, trojan.KillAv, W32.Segax.Gen and explorer.exe[3] are correctly identified samples from worm, trojan, virus and benign genres, respectively. This table also shows W32.HLLP.Zwqq, MSNpws.trojan, W32.Bacros.A and BurnInTestPro[4] which are misclassified, generating false alarms,from worm, trojan, virus and benign genres, respectively.

[2] http://www.wilsonc.demon.co.uk/d10resourceeditor.htm

[3] Bundled with Microsoft Windows XP SP2.

[4] BurnInTest 3.2: http://www.passmark.com/

Table 5 Attribute values for benign and malicious executables (w = worm, v = virus, t = trojan, b = benign)

Attribute	Positive identification				False alarm			
	w32.blaster.worm(w)	trojan.KillAv(v)	w32.segax.gen(t)	explorer.exe(b)	w32.HLLP.Zwqq(w)	MSNpws.trojan(v)	w32.bacros.A(t)	BurnInTestPro(b)
PE checksum	0	0	0	1	0	0	0	0
Debug	1	0	0	1	1	0	0	0
OS & image versions	0	0	0	1	0	0	0	0
Number of sections	5	8	4	4	6	3	8	8
Entropy of code section	5.20	6.49	4.07	6.28	5.24	7.92	6.10	6.13
Resources	0	29	0	201	2	33	27	10
Entry point	1	2	2	1	1	2	2	2
Registry keys	0	3	0	4	4	1	7	0
Registry startup keys	1	0	0	0	1	0	2	0
Socket-related DLLs	3	1	0	0	5	0	0	0
High frequency DLLs in benign	0	3	0	4	4	3	3	1
Size	176,193	421,376	8,192	1,032,192	2,797,650	181,760	356,352	981,551

It can be observed that many attributes from the false alarm samples differ substantially from their correctly identified counterparts, indicating those attributes and corresponding values that have a significant role in the effectiveness of accurate detection. The W32.HLLP.Zwqq is very similar to a benign application in size and imports many of the high frequency DLLs that are used in benign files. Consequently, its large size reduces its efficacy in replication. The W32.Bacros.A has quite a few resources and registry key references along with imported benign DLLs. The detection rate can be improved in this case by borrowing concepts from [30] to make intelligent decisions about the references to registry. Similarly, MSNpws.trojan contains an unusually high number of resources and registry keys, which is generally true for a large number of trojan samples that we evaluated; this similarity with benign files is the main reason for low trojan detection rates. We are currently investigating how to incorporate more low-complexity attributes to improve the detection rate for trojans. False alarms for benign application typically arise due to the missing attributes in headers related to OS and User versions, missing debug information, and/or very few imported DLLs as compared to the size of file. One conclusion that may be drawn from this table is that the effectiveness of our approach depends largely on the selection of the distinguishing attributes, which in turn are selected from a specific initial set of features. For this study, we chose our initial set of features based on complexity of extraction from executables and wanted to avoid working with attributes that necessitated more complex algorithms (e.g., detailed parsing of the PE file format or disassembling of code); however, study of differences in accurately and falsely classified samples indicates that our approach may give better accuracy rates if either more distinguishing attributes are considered or if the initial set of features also includes features that may involve more complexity to extract (e.g., instruction or API call references).

Due to the simplicity of our approach and the relatively few attributes used, it is easy for someone to implement our approach and, while continually testing its ability for evading detection, modify the attributes of some malware executable until it is no longer detected. However, this may cause the malware to be significantly larger and more complex (e.g., dependent on irrelevant DLLs), thus defeating some key objectives of "effective malware design" that necessitate small size and fast execution.

Our approach does not provide sufficiently high detection rates to replace pattern-recognition-based detection. However, our approach may be implemented in a real-time malware detection/prevention system since the memory and time required for analysis are very low. It is also suitable for implementation in forensic toolkits, where computer investigators have to scan a large number of files on a system to identify potential malware (especially zero-day or custom malware) [29].

We are currently studying techniques to improve the approach described in this paper. One strategy appears to give better detection rates at the cost of slightly higher complexity, and is based combining the approach in this paper with multiple-passes on the executable file to "intelligently" extract and process static distinguishing attributes. In this case, the number and location of features is dependent on a small set of initially extracted features.

7 Conclusion

In this paper, we identified a small number of easily-extracted static attributes of binary executable files that can be used to detect malicious executables. To identify these attributes, we presented a novel application of information-theoretic metrics to the problem of malware detection. A simple and low-complexity statistical classifier was proposed for malware

detection which is substantially more efficient than machine learning-based classifiers and is suitable for deployment in real-time or high-speed malware detection systems.

References

1. Spafford, E.H.: The Internet Worm Program: An Analysis. Tech. Report CSD-TR-823. Department of Computer Science, Purdue University (1988)
2. Kephart, J.O., Arnold, W.C.: Automatic extraction of computer virus signatures. In: 4th Virus Bulletin International Conference, pp. 178–184 (1994)
3. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., White, S.R.: Biologically inspired defenses against computer viruses. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 985–996. Morgan Kaufmann, San Francisco (1995)
4. Lo, R.W., Levitt, K.N., Olsson, R.A.: MCF: a malicious code filter. Comput. Secur. **14**(6), 541–566 (1995)
5. Arnold, W., Tesauro, G.: Automatically generated Win32 heuristic virus detection. In: Proceedings of the 2000 International Virus Bulletin Conference (2000)
6. Bayer, U.: TTAnalyze: A Tool for Analyzing Malware. Distributed System and Automation Groups, Technical University of Vienna (2005)
7. Kolter, J., Maloof, M.: Learning to detect malicious executables in the wild. In: Proceedings of ACM SIGKDD (2004)
8. Stolfo, S.J., Wang, K., Li, W.-J.: Towards stealthy malware detection. In: Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C. (eds.) Malware Detection. Advances in Information Security, vol. 27. Springer, US (2007)
9. Ashcraft, K., Engler, D.: Using programmer-written compiler extensions to catch security holes. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 143–159 (2002)
10. Krugel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: Proceedings of USENIX Security Symposium (2004)
11. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 38–49, Los Alamitos, CA, 2001. IEEE Press, USA (2001)
12. VX heavens. http://vx.netlux.org
13. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: fast, generic, and safe unpacking of malware. In: ACSAC'07: Proceedings of the 23rd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference (2007)
14. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: automating the hidden-code extraction of unpack-executing malware. In: ACSAC'06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference (2006)
15. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**, 79–86 (1951)
16. Yeung, R.W.: A First Course in Information Theory. Kluwer Academic/Plenum Publishers, New York (2002)
17. Lin, J.: Divergence measures based on the shannon entropy. IEEE Trans. Inf. Theory **37**(3), 145–151 (1991)
18. Johnson, D.H., Sinanovic, S.: Symmetrizing the Kullback-Leibler distance. Technical Report (2001)
19. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Mellish, C.S. (ed.) Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1137–1143. Morgan Kaufmann, Menlo Park (1995)
20. Li, K.-L., Haung, H.-K., Tian, S.-F., Xu, W.: Improving one-class SVM for anomaly detection. In: Proceedings of the Second International Conference on Machine Learning and Cybernetics, Wan, 2–5 November 2003
21. Mukkamala, S., Janoski, G.I., Sung, A.H.: Intrusion detection using support vector machines. In: Proceedings of the High Performance Computing Symposium—HPC 2002, pp. 178-183, San Diego, April 2002
22. Brockwell, P., Davis, R.: Introduction to time series and forecasting. Springer, Berlin (1996)
23. Self, S.C., Liang, K.Y.: Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under non-standard conditions. J. Am. Stat. Soc. **82**, 605–610 (1987)
24. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Haussler, D. (ed.) Proceedings of 5th Annual ACM Workshop on COLT, pp. 144–152, Pittsburgh, PA, 1992. ACM Press, New York (1992)
25. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**, 273–297 (1995)
26. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining Knowl. Discov. **2**(2), 121–167 (1998)
27. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning. MIT-Press, Cambridge (1999)
28. Mahalanobis, P.C.: On the generalised distance in statistics. Proc. Natl. Inst. Sci. India **2**(1), 49–55 (1936)
29. Haagman, D., Ghavalas, B.: Trojan defence: a forensic view. Digital Investigation, vol. 2, Issue 1, pp. 23–30 (2005)
30. Stolfo, S.J., Apap, F., Eskin, E., Heller, K., Hershkop, S., Honig, A., Svore, K.: A Comparative evaluation of two algorithms for windows registry anomaly detection. J. Comput. Secur. **13**(4), 659—693 (2005)