Computing and Informatics, Vol. 37, 2018, 759–780, doi: 10.4149/cai_2018_3_759

MALWARE DETECTION USING A HETEROGENEOUS DISTANCE FUNCTION

Martin Jureček, Róbert Lórencz

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague, Czech Republic
e-mail: {martin.jurecek, lorencz}@fit.cvut.cz

Abstract. Classification of automatically generated malware is an active research area. The amount of new malware is growing exponentially and since manual investigation is not possible, automated malware classification is necessary. In this paper, we present a static malware detection system for the detection of unknown malicious programs which is based on combination of the weighted k-nearest neighbors classifier and the statistical scoring technique from [12]. We have extracted the most relevant features from portable executable (PE) file format using gain ratio and have designed a heterogeneous distance function that can handle both linear and nominal features. Our proposed detection method was evaluated on a dataset with tens of thousands of malicious and benign samples and the experimental results show that the accuracy of our classifier is 98.80 %. In addition, preliminary results indicate that the proposed similarity metric on our feature space could be used for clustering malware into families.

Keywords: Malware detection system, feature selection, similarity measure, k-nearest neighbors classifier, partitioning around medoids

1 INTRODUCTION

The problem of automated malware detection presents challenges for antivirus vendors (AV). Most AV rely primarily on a signature detection technique which is relatively simple and efficient rule-based method for detecting known malware [10]. Signature (unique hex code strings) of the malware is extracted and added to the database. The antivirus engine compares the contents of a file with all malware signatures in its database and if a match is found, the file is reported as malware. A good signature must capture malware with a minimal false positive probability.

The major weakness of signature detection is its inability to detect obfuscated and zero-day malware. A number of non-signature based malware detection techniques have been proposed [19, 11, 20]. These techniques are used in an effort to detect new or unknown malware and can be grouped into two main approaches: static and dynamic heuristic methods. Static methods can be based on an analysis of the file format without actually running the program. Dynamic analysis aims to examine a program which is executed in a real or virtual environment. Non-signature based malware detection techniques suffer from two main problems: high false positive rate and large processing overhead.

In this paper, we present a static malware detection system based on combination of the statistical classifier and the k-nearest neighbors (KNN) classifier. Experimental results indicate that the combination of the classifiers may provide a potential benefit for detecting samples not detected by KNN.

In our work we propose the following four main contributions:

- We present a feature space extracted from PE file format by using feature selection method based on information gain.
- We design a new distance function that can handle both nominal and linear attributes.
- We present a malware detection system for detecting previously unknown malicious PE files. In order to achieve a higher detection rate, the system uses a combination of two different kinds of classifiers.
- We evaluate the effectiveness of our detection system and distance function on a real-world malware collection.

The rest of the paper is organized in the following way. Section 2 provides an overview of previous work on malware classification. In Section 3 we present the feature space and the distance function used in KNN classifier. Section 4 discusses our proposed detection technique, while Section 5 covers our experimental results. Finally, conclusions are given in Section 6.

2 RELATED WORK

In this section, we survey some relevant previous work in the area of classification schemes for malware detection. To maintain the focus, we mainly discuss the work using static detection based on machine learning techniques. Then we briefly discuss various existing statistical-based scores and also several methods that rely on dynamic analysis.

Schultz et al. [19] introduced the concept of data mining for detecting previously unknown malware. In their research they presented three different static feature sources for malware classification: information from the portable executable

(PE) header and strings and byte sequences extracted from binaries. These features were used in three different kinds of algorithms: an inductive rule-based learner, a probabilistic method, and a multi-classifier system. A rule induction algorithm called Ripper [4] was applied to find patterns in the dynamic-link library (DLL) data (such as the list of DLLs used by the binary, the list of DLL function calls, and the number of different system calls used within each DLL). The well-known probabilistic method, learning algorithm Naive Bayes, was used to find patterns in the string data and n-grams of byte sequences. Multinomial Naive Bayes algorithm that combined the output of several classifiers reached the highest detection rate of 97.76 %. The authors tested the data mining methods against standard signatures and their results indicate that the data mining detection rate of a previously unknown malware was twice as high in comparison to the signature-based methods.

Kolter and Maloof [11] improved the Schulz's third technique by using overlapping byte sequences instead of non-overlapping sequences. They used different kinds of classifiers: naive Bayes, instance-based learner, similarity-based classifier called TFIDF, Support Vector Machine (SVM), Decision Trees (DT) and boosted variants of SVM, DT and TFIDF. Authors evaluated their classifiers performance by computing the area under a receiver operating characteristic curve. Boosted Decision tree model (J48) achieved the best accuracy, an area under the ROC curve of 0.996 and outperformed the rest of the classifiers.

In other studies, operational code (opcode) has been used as static information for malware detection. Common techniques are based on the frequency of appearance of opcode-sequences [18], examination of opcode frequency distribution difference between malicious and benign code [2], or identification of critical instruction sequences [22]. Other techniques use similarity of executables based on opcode graphs [17]. However, some executable files cannot be disassembled properly, therefore the opcode approach is not always feasible [2].

The more recent work [23] contains three statistical-based scoring techniques, namely Hidden Markov models, Simple substitution distance, and Opcode graph-based detection. Authors showed that a combination of these scoring techniques with a Support Vector Machine yields significantly more robust results than those obtained using any of the individual scores.

We also briefly mention a few existing detection methods that rely on dynamic analysis. Examples of the information we can obtain from dynamic analysis include application programming interface (API) and system calls, instruction traces, memory writes, registry changes, and so on. In [24], an artificial neural network was employed to detect previously unknown worms based on the computer's behavioral measures. Eskandari et al. [25] extracted a set of program API calls and combined them with control flow graph. Qiao et al. [26] proposed a new malware analysis method based on frequency analysis of API call sequences. Note that dynamic analysis is time-consuming as each malware sample must be executed for a certain time period.

3 FEATURE SPACE AND METRIC

We design our proposed detection system for the portable executable (PE) file format [5], which is the most widely used file format for malware samples. In order to classify an executable file in the PE format, we extract static format information and translate it into a feature vector suitable for classification.

3.1 Feature Space

Before presenting attributes used in our feature vector, let us firstly look at the general outline of the PE file format. A simplified overview of the PE file format is illustrated in Figure 1.

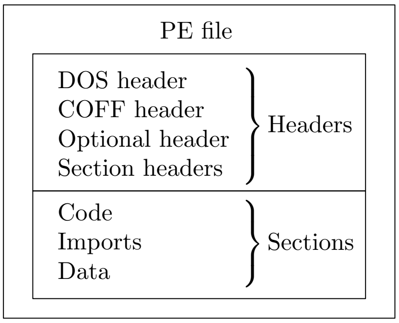


Figure 1. PE file structure

A PE file consists of headers and sections that encapsulate the information necessary to manage the executable code. The PE file header provides all the descriptive information concerning the locations and sizes of structures in the PE file to the loader process. The header of a PE file consists of the DOS header, the PE signature, the COFF file header, the optional header, and the section headers. The optional file header is followed immediately by the section headers which provide information about sections, including locations, sizes, and characteristics. Sections divide the file content into code, resources, and various types of data.

Based on our empirical studies and analysis of the PE format, we selected a set of static features that are helpful in distinguishing malware and benign files and used gain ratio for selection the most relevant features.

3.1.1 Features Selection

In order to determine which attribute in a given training set is the most useful for discriminating between the classes, we use entropy-based measure, information gain (IG) [13]. The information gain is the expected reduction in entropy caused by knowing the value of attribute a. $\mathrm{IG}(\mathcal{T}, a)$ of an attribute a relative to training dataset $\mathcal{T}$ and is defined as

$$\mathrm{IG}(\mathcal{T}, a) = \mathrm{Entropy}(\mathcal{T}) - \sum_{v \in V(a)} \frac{|\mathcal{T}_v|}{|\mathcal{T}|} \mathrm{Entropy}(\mathcal{T}_v) \tag{1}$$

where $V(a)$ denotes the set of all possible values for attribute a, and $\mathcal{T}_v$ denotes the subset of $\mathcal{T}$ for which attribute a has value v. Note that the entropy of the training dataset $\mathcal{T}$ is given by:

$$\mathrm{Entropy}(\mathcal{T}) = -\sum_{c \in C} p_c \log_2 p_c \tag{2}$$

where p_c is the proportion of $\mathcal{T}$ belonging to class c.

The information gain measure is biased towards attributes with many values. One way of avoiding this difficulty is to use a modification of the measure called the gain ratio (GR) [15]. The gain ratio measure penalizes attributes with large numbers of possible values by incorporating a term called split information (SI):

$$\mathrm{SI}(\mathcal{T}, a) = -\sum_{i=1}^{d} \frac{|\mathcal{T}_i|}{|\mathcal{T}|} \log_2 \frac{|\mathcal{T}_i|}{|\mathcal{T}|} \tag{3}$$

where $\mathcal{T}_i$ are the d subsets of training dataset $\mathcal{T}$ resulting from partitioning $\mathcal{T}$ by the d-valued attribute a. Split information $\mathrm{SI}(\mathcal{T}, a)$ is the entropy of $\mathcal{T}$ with respect to the values of attribute a. The gain ratio is then defined as

$$\mathrm{GR}(\mathcal{T}, a) = \frac{\mathrm{IG}(\mathcal{T}, a)}{\mathrm{SI}(\mathcal{T}, a)} \tag{4}$$

and we select only features with the highest values of gain ratio.

3.1.2 Our Proposed Feature Space

The following feature set was extracted using gain ratio and used in our work:

- Many fields from the PE headers, such as the number of sections, date/time stamp, major or minor versions of linker, operating system, image, subsystem; sizes and addresses of data directories; DLL characteristics, etc. Table 1 lists all features that are derived from the PE headers. For detailed description of these features, see Chapter 3 in [5].
- Features from sections and their headers: VirtualSize, VirtualAddress, SizeOfRawData, PointerToRawData, Section Flags (see Chapter 4 in [5]), and features not contained within the PE structure, including entropies and checksums of sections.
- Resources of the PE file are used to provide supporting content, such as icons, fonts, strings and other elements. In case of malicious files, resources are often

used to store code and configuration data. For example, the number of resources and the number of types of resources were used in our work.

- Overlay is a data that is appended at the end of the executable file. We considered the size of the overlay.
- Other features: the size of all imports, the number of DLLs referred, the number of APIs referred.

Feature	*Feature*
NumberOfSections	MajorOperatingSystemVersion
TimeDateStamp	MajorImageVersion
SizeOfOptionalHeader	MajorSubsystemVersion
Characteristics	MinorSubsystemVersion
MajorLinkerVersion	SizeOfImage
MinorLinkerVersion	CheckSum
AddressOfEntryPoint	Subsystem
ImageBase	DllCharacteristics
SectionAlignment	NumberOfRvaAndSizes
FileAlignment	Addresses and sizes of data directories

Table 1. List of features from the PE headers

Note that our feature set is similar to that in the existing works [12, 21, 1].

3.2 Distance Function

Many classifiers require some measure of dissimilarity or distance between feature vectors, and its performance depends upon a good choice of distance function. Especially the KNN classifier depends significantly on the metric used to compute distances between two feature vectors.

In this section, we propose a similarity metric on our feature space. We used this metric to compute distances to find k nearest neighbors used in KNN classifier. Note that the features used in our work are of various types and sizes. These features can be divided into three types: numbers, bit fields, and strings. For example, number of sections or various addresses can be represented by numbers, section flags or characteristics by bit fields, and checksums by strings. Furthermore, some features have different ranges. For example, the number of sections is considerably smaller than the total number of called API functions. The proposed distance function can handle these types of features and also takes into account their different ranges.

The most commonly used metric is the Euclidean distance which works well for numerical attributes. However, it does not appropriately handle nominal attributes. The Value Difference Metric (VDM) [27] was proposed to define a suitable distance function for nominal attributes. A version of the VDM without a weighting scheme is defined for values x and y of an attribute a as:

$$\mathrm{VDM}_a(x,y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right|^q \tag{5}$$

where

- C is the number of classes,
- $n_{a,x,c}$ is the number of instances in the training set $\mathcal{T}$ which have value x for attribute a and the instance belongs to class c,
- $n_{a,x}$ is the number of instances in $\mathcal{T}$ that have value x for attribute a.

Since the Euclidean distance is not suitable for nominal attributes, and VDM is inappropriate for numeric attributes, heterogeneous metric can be used to handle our feature space. Wilson and Martinez introduced Heterogeneous Value Difference Metric (HVDM) [29] which is defined for feature vectors $\mathbf{x}$ and $\mathbf{y}$ as:

$$\mathrm{HVDM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{6}$$

where m is the number of attributes of the feature vector and the definition of distance function $d_a(x,y)$ depends on the type of attribute a as follows:

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown,} \\ \mathrm{NORM\_VDM}_a(x,y), & \text{if } a \text{ is nominal,} \\ \mathrm{NORM\_DIFF}_a(x,y), & \text{if } a \text{ is linear.} \end{cases} \tag{7}$$

Functions $\mathrm{NORM\_VDM}_a(x,y)$ and $\mathrm{NORM\_DIFF}_a(x,y)$ are defined as:

$$\mathrm{NORM\_VDM}_a(x,y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right|, \tag{8}$$

$$\mathrm{NORM\_DIFF}_a(x,y) = \frac{|x-y|}{4\sigma_a} \tag{9}$$

where σ_a is the standard deviation of the values of numeric attribute a.

3.2.1 Our Proposed Distance Function

Since the feature vector described in Section 3.1.2 contains more types of nominal attributes we propose the following distance function:

$$\mathcal{D}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{10}$$

where

$$d_a(x,y) = \begin{cases} \mathcal{H}(x,y), & \text{if } a \text{ is an array of bits,} \\ \delta(x,y), & \text{if } a \text{ is a checksum,} \\ \text{NORM\_DIFF}_a(x,y), & \text{if } a \text{ is numeric,} \\ \text{NORM\_VDM}_a(x,y), & \text{otherwise } (a \text{ is a string}). \end{cases} \tag{11}$$

$\mathcal{H}(x,y)$ denotes Hamming distance defined for binary $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n)$ as

$$\mathcal{H}(x,y) = |\{i \mid x_i \neq y_i, i = 1, \dots, n\}| \tag{12}$$

and $\delta(x,y)$ is the characteristic function defined as

$$\delta(x,y) = \begin{cases} 0, & \text{if x = y,} \\ 1, & \text{otherwise.} \end{cases} \tag{13}$$

Since the distance functions d_a are metrics, $\mathcal{D}$ is also a metric. Properties of a metric, especially the triangle inequality, can be used in effective finding of nearest neighbors in a metric space.

Note that we distinguish between a checksum and a string attribute that is not a checksum. For the demonstration of distance function $\mathcal{D}$ on our feature space, we present the examples of attributes of each type:

- array of bits: Section flags, Characteristics, DllCharacteristics,
- numeric attribute: number of sections, number of DLLs, size of all imports,
- checksum: checksums of various pieces of the file content,
- string: major/minor version of linker, operating system, subsystem.

4 PROPOSED SYSTEM FOR DETECTING MALWARE

In this section, we present a system for detecting malware which is composed of a KNN classifier and a statistical scoring technique.

4.1 The k-Nearest Neighbors Classifier

The k-nearest neighbors (KNN) classifier is one of the most popular supervised learning methods introduced by Fix and Hodges [8]. It is one of the simplest and best-known nonparametric algorithms in pattern classification.

Let $T = \{(x_1, c_1), \dots, (x_m, c_m)\}$ be the training set, where x_i is training vector and c_i is the corresponding class label. Given a query point x_q, its unknown class c_q is determined as follows. First, select the set $T' = \{(x_1, c_1), \dots, (x_k, c_k)\}$ of k nearest neighbors to the query point x_q. Then assign the class label to the query point x_q by majority vote of its nearest neighbors:

$$c_q = \arg\max_c \sum_{(x_i,c_i)\in T'} \delta(c, c_i) \tag{14}$$

where c is a class label, c_i is the class label for i^{th} neighbor among k nearest neighbors of the query point, and $\delta(c, c_i)$ takes a value of one if $c = c_i$ and zero otherwise. Cover and Hart [6] found that if the number of samples approaches infinity, the nearest-neighbor error rate is bounded from above by twice the Bayes error rate.

Distance-weighted k-nearest neighbor procedure (WKNN) was first introduced in [7] as an improvement to KNN. This extension is based on the idea that closer neighbors are weighted more heavily than such neighbors that are far away from the query point. KNN implicitly assumes that all k nearest neighbors are equally important in making a classification decision, regardless of their distances to the query point. In WKNN, nearest neighbors are weighted according to their distances to the query point as follows. Let $x_1, \ldots, x_k$ be k nearest neighbors of the query object and $d_1, \ldots, d_k$ the corresponding distances arranged in increasing order. The weight w_i for i-th nearest neighbor is defined as:

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1}, & \text{if } d_k \neq d_1, \\ 1, & \text{otherwise.} \end{cases} \tag{15}$$

The resulting class of the query point is then defined by the majority weighted vote as follows:

$$c_q = \arg\max_c \sum_{(x_i,c_i)\in T'} w_i \cdot \delta(c, c_i). \tag{16}$$

Note that finding the nearest neighbors is a very expensive operation due to the enormous size of our dataset. The nearest neighbors can be found more efficiently by representing the training dataset as a tree.

4.2 The Statistical-Based Classifiers

In this section, we present the scoring techniques that we used in our research. In the case of the statistical-based classifier, we ignore the positions of points in our metric space and we focus on statistical properties of attribute values, in contrast to the KNN classifier.

4.2.1 Naive Bayes

This section introduces the Naive Bayes classifier [28] for binary (two-class) classification problems. A Naive Bayes classifier is a probabilistic algorithm based on Bayes' Theorem that predicts the class with the highest *a posteriori* probability. Assume a set of two classes $\{\mathcal{C}, \mathcal{M}\}$, where $\mathcal{C}$ denotes the class of benign samples and $\mathcal{M}$ denotes the class of malware. Training datasets are provided and a new (unknown) sample, which is represented by a feature vector $x = (x_1, \dots, x_n)$, is presented. Let $P(\mathcal{M}|x)$ denote the probability that a sample is malicious given the feature vector x that describes the sample. Similarly, $P(\mathcal{C}|x)$ denotes the probability that a sample is benign given the feature vector x that represents the sample. The Naive Bayes classification rule is stated as

$$\begin{aligned} &\text{If } P(\mathcal{M}|x) < P(\mathcal{C}|x), x \text{ is classified as benign sample,} \\ &\text{If } P(\mathcal{M}|x) > P(\mathcal{C}|x), x \text{ is classified as malware.} \end{aligned} \tag{17}$$

The *a posteriori* probabilities $P(C|x)$ may be expressed in terms of the *a priori* probabilities and the $P(x|C)$ probabilities using Bayes' theorem as

$$P(C|x) = \frac{P(x|C)\ P(C)}{P(x)}. \tag{18}$$

Assuming that the values of the attributes (features) are conditionally independent on one another, Equation (18) may be expressed as

$$P(C|x) = \frac{\prod_{i=1}^{n} P(x_i|C)\ P(C)}{P(x)}. \tag{19}$$

Probabilities $P(x_i|C)$ can be estimated from the training set by counting the attribute values for each class. More precisely, the probability $P(x_i = h|C)$ is represented as the number of samples of class C in the training set having the value h for attribute x_i, divided by the number of samples of class C in the training set. The output of the classifier is the highest probability class C':

$$C' = \arg\max_{C} \left(P(C) \prod_{i=1}^{n} P(x_i|C) \right). \tag{20}$$

4.2.2 Statistical Classifier – STATS

The following statistical classifier was introduced in [12]. Let $x = (x_1, \dots, x_n)$ be a vector from our feature space and $\mathcal{M}$ a class of malware. Then probability

$$P(x \in \mathcal{M}|x_i = h) = \frac{n_{x_i,h,\mathcal{M}}}{n_{x_i,h}} \tag{21}$$

is the conditional probability that the output class of x is malware given that attribute x_i has the value h. Denote this probability by p_i, $i = 1, \dots, n$. Note that the notations $n_{x_i,h,\mathcal{M}}$ and $n_{x_i,h}$ were used in the definition of VDM discussed in Section 3.2. Define a function f with two parameters p_i and S_c as

$$f(p_i, S_c) = \max\{0, p_i - S_c\} \tag{22}$$

where S_c is an empirical constant. For each file x we define a score as

$$\text{score} = \sum_{i=1}^{n} f(p_i, S_c). \tag{23}$$

From this score, we can determine a threshold S_s, above which we will classify a file as malware. The decision rule is then defined as follows:

$$x \text{ is classified as } \begin{cases} \text{malware,} & \text{if score} > S_s, \\ \text{benign file,} & \text{otherwise.} \end{cases} \tag{24}$$

The pseudocode of the statistical-based classifier is described in Algorithm 1.

Algorithm 1 Statistical classifier – STATS

```
Input: original training set, query point $x$, distance metric $\mathcal{D}$
Output: label of $x$
1: score = 0
2: Compute probability vector $(p_1, \dots, p_n)$
3: for $i = 1$ to $n$ do
4:   if $p_i > S_c$ then
5:     score $+= p_i - S_c$
6:   end if
7: end for
8: if score $> S_s$ then
9:   return malware
10: else
11:   return benign file
12: end if
```

In the rest of this paper, the statistical-based classifier is denoted as STATS.

4.3 Our Approach

We propose a malware detection approach based on a combination of the well-known KNN classifier and the chosen statistical motivated classifier. In order to achieve higher detection rates, there should be some kind of diversity between the classifiers. KNN is a geometric-based classifier which uses labels of the nearest neighbors in some metric space to classify an unlabeled point. On the other hand, statistical-based approaches like Naive Bayes or the STATS classifier mentioned above use conditional probabilities of attributes of sample point and do not use information about its position in feature space.

The proposed detection method works as follows. First, set the threshold to some sufficiently high value. Then compute score using the chosen statistical scoring technique. If the score of the unknown file exceeds the threshold, then the resulting class will be malware, otherwise apply distance-weighted KNN. The pseudocode for the classification scheme is shown in Algorithm 2.

Algorithm 2 Our detection system

```
Input: original training set, query point $x$, distance metric $\mathcal{D}$
Output: label of $x$
1: compute score from the statistical scoring technique
2: if score > threshold then
3:     return malware
4: else
5:     apply WKNN
6: end if
```

The reason why we chose KNN is that it is a relatively accurate classifier for large datasets and the results of our experiments demonstrate that the statistical classifier is able to correctly classify samples lying in the area of feature space where the accuracy of KNN is low. The statistical classifier uses information from the training dataset in a different way than the KNN classifier. It checks whether a feature vector contains values typical for malware, in contrast to KNN that considers only differences between feature vectors.

For example, consider a feature vector $x = (x_1, \dots, x_n)$ containing only a few values (typically checksums), for which there is a high probability that x belongs to malware. Many other attributes could have previously unseen values or ones with a low prevalence. Therefore, malicious nearest neighbors could not be closer than benign nearest neighbors and in this case, KNN classifier would not be an appropriate method.

4.3.1 The System Architecture

The system consists of three major components: a PE parser, a database of conditional probabilities and a classification module, as illustrated in Figure 2.

The functionality of the PE parser is to extract all PE format file's header information, DLLs, and API functions from programs in the dataset and store all the extracted information in a database. Recall that our system is applied only to Windows PE files and the PE parser extracts only the most useful features for discriminating between benign and malicious files. These features were determined by the feature selection algorithm mentioned in Section 3.

During the training phase, once the structural information of the PE files is extracted, the conditional probabilities $P(x \text{ is malware} | x_i = h)$ are computed for each PE attribute x_i and for each possible value h of attribute x_i. Note that only the PE features extracted from labeled samples of the training dataset are used in the computation of the conditional probabilities.

After extracting PE features and computing the conditional probabilities, feature vectors are created for every known PE file. The set of these feature vectors

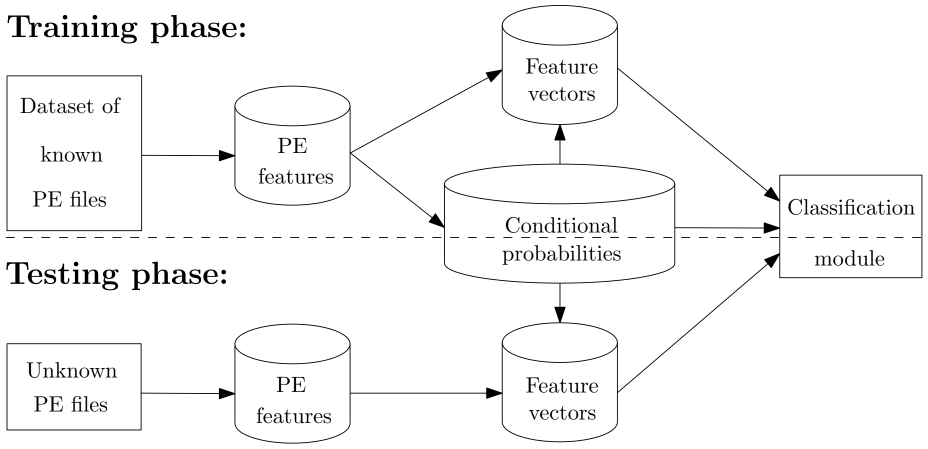


Figure 2. Architecture of the classification model

called training set will be used in the classification module where the classification algorithm is applied to feature vectors of unknown PE files.

5 EVALUATION RESULTS AND ANALYSIS

In this section, we introduce the performance metrics and present the results of our experiments. We compare our approach with several other machine learning methods for malware detection.

5.1 Performance Metrics

We present the evaluation metric we used to measure the accuracy of our proposed approach for the detection of unknown malicious codes. For evaluation purposes, the following classical quantities are employed:

- True Positive (TP) represents the number of malicious samples classified as malware,
- True Negative (TN) represents the number of benign samples classified as benign,
- False Positive (FP) represents the number of benign samples classified as malware,
- False Negative (FN) represents the number of malicious samples classified as benign.

The performance of our classifier on the test set is measured using three standard parameters. The most intuitive and commonly used evaluation measure in Machine

Learning is the Accuracy (ACC):

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{25}$$

It is defined on a given test set as the percentage of correctly classified instances. However, since our dataset is not well-balanced, the accuracy measure could be an inappropriate measure of performance. If we use a classifier which labels every sample as benign, then TN will be very high and TP will be very low. As a result, the accuracy obtained on our dataset will be very high.

The second parameter, True Positive Rate (TPR) (or detection rate), is defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{26}$$

TPR is the percentage of truly malicious samples that were classified as malware. The third parameter is False Positive Rate (FPR) and is defined as follow:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}. \tag{27}$$

FPR is the percentage of benign samples that were wrongly classified as malware.

We also evaluate our classifier using Receiver Operating Characteristic (ROC) analysis [3]. ROC curve is represented as a two-dimensional plot, in which true positive rate is plotted against false positive rate at various threshold settings. The area under the ROC curve (AUC) serves as the performance measure of our detection techniques. An AUC of 1.0 represents the ideal case where both false positive and false negative equal zero. On the other hand, AUC of 0.5 means that the classifier's performance is no better than flipping a coin.

5.2 Dataset

The dataset used in this research consists of a total of 101,604 Windows programs in the PE file format, out of which 21,087 are malicious and 80,517 are legitimate or benign programs. There were no duplicate programs in our dataset. The malicious and benign programs were obtained from the laboratory of the industrial partner.

In order to expose any biases in the data, we used the 5-fold cross-validation procedure. Generally in k-fold cross-validation [14], the dataset is randomly divided into k subsets of equal size, where k-1 subsets are used for training and 1 subset is used for testing. In each of the k folds a different subset is reserved for testing and the accuracies obtained for each fold are averaged to produce a single cross validation estimate.

In the cluster analysis we used five prevalent malware families that have appeared during the year 2016. Specifically, we have used the following malware families:

- Allaple – a polymorphic network worm that spreads to other computers and performs denial-of-service (DoS) attacks.

- Dinwod – a trojan horse that silently downloads and installs other malware on the compromised computer.
- Virlock – a ransomware that locks victims' computer and demands a payment in order to unlock it.
- Virut – a virus with backdoor functionality that operates over an IRC-based communications protocol.
- Vundo – a trojan horse that displays pop-up advertisements and also injects JavaScript into HTML pages.

5.3 Classification Results

We implemented the classifiers as described in Section 4. The feature space and the distance function proposed in Section 3 were used in the KNN and the WKNN classifiers. The combination of the WKNN and the statistical scoring technique from [12] is denoted as WKNN_STATS and the combination of the WKNN and the Naive Bayes classifier is denoted as WKNN_NB. For each experiment, we performed 5-fold cross-validation that gives approximately unbiased estimate of a classifier's accuracy.

In our first experiment, we attempt to distinguish between benign and malicious PE files. We used accuracy, discussed in Section 5.1, as a comparison criterion for comparing classifiers. The accuracies obtained after applying the WKNN and the KNN classifiers for various numbers of nearest neighbors are depicted in Figure 3.

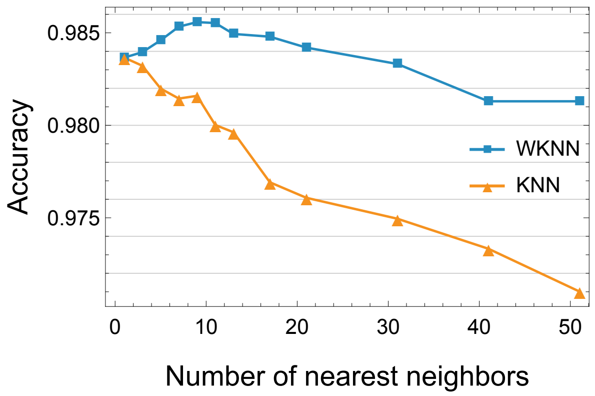


Figure 3. Classification accuracies of the WKNN and the KNN classifiers, for various numbers of nearest neighbors

The WKNN classifier achieved the highest accuracy using nine nearest neighbors, while the KNN classifier achieved the highest accuracy using only three nearest neighbors.

The classification results of the classifiers implemented in this research are listed in Table 2.

Classifier	*TPR*	*FPR*	*Accuracy*
KNN	96.23 %	1.07 %	98.37 %
WKNN	96.82 %	0.99 %	98.56 %
NB	82.78 %	1.17 %	95.50 %
STATS	90.08 %	**0.76** %	97.34 %
WKNN_NB	97.37 %	1.05 %	98.62 %
WKNN_STATS	**98.08** %	1.01 %	**98.80** %

Table 2. Classification results of six approaches implemented in this work

Among these classifiers, the WKNN_STATS outperformed others with the highest accuracy of 98.8 %. Note that the WKNN_STATS classifier was tested for various threshold values, and the best result was achieved with the following parameters:

- the number of nearest neighbors $k = 9$ used in the WKNN classifier,
- the thresholds $S_c = 0.8$ and $S_s = 0.53$ used in the STATS classifier.

In addition to that, we constructed the ROC curves which are shown in Figure 4 for three chosen classifiers.

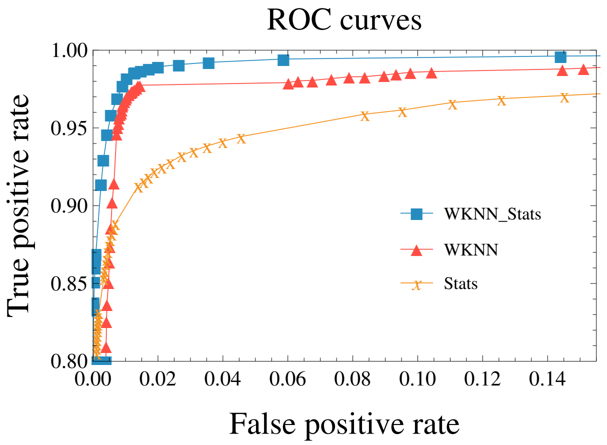


Figure 4. ROC curves for the WKNN, STATS and WKNN_STATS classifiers

We can conclude from Figure 4 that a combination of the classifiers outperforms both individual classifiers.

Table 3 reports the AUC for three classifiers discussed in Section 4 and two related static methods: KM [11] and PE Miner [21]. As the table illustrates, WKNN_STATS classifier provides the best AUC value with 0.998. The ROC curve and AUC values confirm that our experiment provides excellent results regarding malware detection.

Classifier	*AUC*
WKNN	0.993
STATS	0.983
WKNN_STATS	**0.998**
KM	0.996
PE Miner	0.992

Table 3. Comparison of the AUC value for five static methods

5.4 Clustering Results

In the second experiment we apply cluster analysis to five prevalent malware families described in Section 5.2. First, we present the clustering algorithm used in this experiment and then describe the evaluation measures and show the results.

5.4.1 Partitioning Around Medoids

Partitioning around medoids (PAM) proposed by Kaufman and Rousseeuw [9] is a well-known technique for performing non-hierarchical clustering. The reason why we have decided to use the PAM algorithm is that it allows clustering with respect to any distance metric. The pseudocode of the PAM algorithm is described in Algorithm 3.

Algorithm 3 PAM algorithm

```
Input: Number of clusters $k$, set of data points $T$
Output: $k$ clusters
1:  Initialize: randomly select $k$ data points from $T$ to become the medoids
2:  Assign each data point to its closest medoid
3:  for all cluster do
4:      identify the observation that would yield the lowest average distance if it were
        to be re-assigned as the medoid
5:      if the observation is not current medoid then
6:          make this observation the new medoid
7:      end if
8:  end for
9:  if at least one medoid has changed then
10:     go to step 2
11: else
12:     end the algorithm.
13: end if
```

5.4.2 Evaluation Measures

We evaluated the quality of clusters through the measures of purity and silhouette coefficient (SC). Let n_{ij} be the number of samples of class i in cluster C_j and let $p_{ij} = \frac{n_{ij}}{|C_j|}$. The probability p_{ij} is the probability that a randomly selected sample from cluster C_j belongs to class i. The purity of cluster C_j is defined as $\text{Purity}(C_j) = \max_i p_{ij}$.

The overall purity value is defined as the weighted sum of individual purities for each cluster, taking into account the size of each cluster:

$$\text{Purity} = \frac{1}{n}\sum_{j=1}^{k} |C_j|\text{Purity}(C_j). \tag{28}$$

To measure the quality of clusters, we compute the average silhouette coefficient [16] for each cluster. Suppose there are n samples $x_1, \ldots, x_n$ that have been divided into k clusters $C_1, \ldots, C_k$. Consider a sample $x_i \in C_j$, and define the average distance between x_i to all other samples in cluster C_j:

$$a(x_i) = \frac{1}{|C_j| - 1}\sum_{\substack{y \in C_j \\ y \neq x_i}} d(x_i, y). \tag{29}$$

Let $b_k(x_i)$ be the average distance from sample $x_i \in C_j$ to all samples in cluster C_k not containing x_i:

$$b_k(x_i) = \frac{1}{|C_k|}\sum_{y \in C_k} d(x_i, y). \tag{30}$$

After computing $b_k(x_i)$ for all clusters C_k, where $k \neq j$, we select the minimum of those numbers:

$$b(x_i) = \min_{k \neq j} b_k(x_i). \tag{31}$$

The silhouette coefficient of x_i is obtained by combining $a(x_i)$ and $b(x_i)$ as follows:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}. \tag{32}$$

The value of $s(x_i)$ in Equation (32) can vary between -1 and 1. It is desirable to have the value $s(x_i)$ as close to 1 as possible, since then the clusters are well-separated. The average silhouette coefficient for a given cluster is defined as the average value of $s(x_i)$ over all samples in the cluster.

5.4.3 Experimental Results

We computed the silhouette coefficient, as discussed above. For computing the silhouette coefficient, we used our proposed distance function on the feature space

Majority Class	*Size*	*Purity*	*SC*
Allaple	424	0.9343	0.3298
Dinwod	285	0.7429	0.7172
Virlock	452	0.9771	0.5635
Virut	337	0.68	0.2389
Vundo	252	0.6886	0.1921
Overall	1 750	0.8298	0.3883

Table 4. The purity and the silhouette coefficient for clusters

discussed in Section 3. Table 4 summarizes the results of silhouette coefficient based experiments using the PAM algorithm.

According to the experiences of authors of SC [16], silhouette coefficient values between 0.7 and 1.0 indicate excellent clustering results. SC values between 0.5 and 0.7 indicate a reasonable structure of cluster. SC values below 0.25 indicate that no substantial structure has been found.

Regarding the clustering malware into families, our results show that the quality of clusters varies widely, depending on the particular family. From the results in Table 4, we see that the PAM algorithm can correctly classify the malware family with an accuracy of about 68 % to over 97 %, depending on the particular family.

Note that such accuracies are lower than those obtained with classifiers presented in Section 4. The reason is that distinguishing between malware families is a more challenging problem than a binary classification of malware and benign files.

6 CONCLUSION

In this paper, we proposed a new detection system using a combination of the k-nearest neighbors classifier and the statistical-based classifier. The system can automatically detect unknown malware samples. The feature set used in our work was a collection of properties extracted from the PE file format. We designed a new distance function that is capable of handling various types of features.

Experimental results indicate that the combination of the classifiers may provide a potential benefit to detect samples not detected by KNN. We compared the different classification methods and concluded that the combination of the weighted k-nearest neighbors classifier and the statistical-based classifier achieves the highest accuracy, 98.8 %. The results also indicate that the proposed heterogeneous distance function and the feature space are appropriate for malware detection and could be also used for clustering malware into families.

The proposed static malware detection system is relatively easy to implement, and can be utilized to support commercial antivirus systems. For future work, it would be interesting to experiment with additional statistical scoring techniques in the context of malware classification.

Acknowledgements

The authors acknowledge the support of the OP VVV funded project CZ.02.1.01/ 0.0/0.0/16_019/0000765 "Research Center for Informatics".

REFERENCES

[1] Asquith, M.: Extremely Scalable Storage and Clustering of Malware Metadata. Journal of Computer Virology and Hacking Techniques, Vol. 12, 2016, No. 2, pp. 49–58, doi: 10.1007/s11416-015-0241-3.

[2] Bilar, D.: Opcodes as Predictor for Malware. International Journal of Electronic Security and Digital Forensics, Vol. 1, 2007, No. 2, pp. 156–168, doi: 10.1504/IJESDF.2007.016865.

[3] Bradley, A. P.: The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. Pattern Recognition, Vol. 30, 1997, No. 7, pp. 1145–1159, doi: 10.1016/S0031-3203(96)00142-2.

[4] Cohen, W. W.: Learning Trees and Rules with Set-Valued Features. Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI/IAAI), Vol. 1, 1996, pp. 709–716.

[5] Microsoft Corporation: Visual Studio, Microsoft Portable Executable and Common Object File Format Specification, Revision 9.3, 2015.

[6] Cover, T.—Hart, P.: Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory, Vol. 13, 1967, No. 1, pp. 21–27, doi: 10.1109/TIT.1967.1053964.

[7] Dudani, S. A.: The Distance-Weighted k-Nearest-Neighbor Rule. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, 1976, No. 4, pp. 325–327, doi: 10.1109/TSMC.1976.5408784.

[8] Fix, E.—Hodges Jr., J. L.: Discriminatory Analysis – Nonparametric Discrimination: Consistency Properties. Technical Report, DTIC Document, 1951.

[9] Kaufman, L.—Rousseeuw, P. J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, Wiley Series in Probability and Statistics, Vol. 334, 2009.

[10] Kephart, J. O.—Arnold, W. C.: Automatic Extraction of Computer Virus Signatures. 4th Virus Bulletin International Conference, 1994, pp. 178–194.

[11] Kolter, J. Z.—Maloof, M. A.: Learning to Detect and Classify Malicious Executables in the Wild. The Journal of Machine Learning Research, Vol. 7, 2006, pp. 2721–2744.

[12] Merkel, R.—Hoppe, T.—Kraetzer, C.—Dittmann, J.: Statistical Detection of Malicious PE-Executables for Fast Offline Analysis. In: De Decker, B., Schaumüller-Bichl, I. (Eds.): Communications and Multimedia Security (CMS 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6109, 2010, pp. 93–105.

[13] Mitchell, T. M.: Machine Learning. New York, 1997.

[14] Picard, R. R.—Cook, R. D.: Cross-Validation of Regression Models. Journal of the American Statistical Association, Vol. 79, 1984, No. 387, pp. 575–583, doi: 10.1080/01621459.1984.10478083.

[15] Quinlan, J. R.: Induction of Decision Trees. Machine Learning, Vol. 1, 1986, No. 1, pp. 81–106, doi: 10.1007/BF00116251.

[16] Rousseeuw, P. J.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Journal of Computational and Applied Mathematics, Vol. 20, 1987, pp. 53–65, doi: 10.1016/0377-0427(87)90125-7.

[17] Runwal, N.—Low, R. M.—Stamp, M.: Opcode Graph Similarity and Metamorphic Detection. Journal in Computer Virology, Vol. 8, 2012, No. 1–2, pp. 37–52, doi: 10.1007/s11416-012-0160-5.

[18] Santos, I.—Brezo, F.—Nieves, J.—Penya, Y. K.—Sanz, B.—Laorden, C.—Bringas, P. G.: Idea: Opcode-Sequence-Based Malware Detection. In: Massacci, F., Wallach, D., Zannone, N. (Eds.): Engineering Secure Software and Systems (ESSoS 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5965, 2010, pp. 35–43.

[19] Schultz, M. G.—Eskin, E.—Zadok, F.—Stolfo, S. J.: Data Mining Methods for Detection of New Malicious Executables. Proceedings of the 2001 IEEE Symposium on Security and Privacy (S & P 2001), IEEE Computer Society, 2001, pp. 38–49, doi: 10.1109/SECPRI.2001.924286.

[20] Shabtai, A.—Moskovitch, R.—Elovici, Y.—Glezer, C.: Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey. Information Security Technical Report, Vol. 14, 2009, No. 1, pp. 16–29, doi: 10.1016/j.istr.2009.03.003.

[21] Shafiq, M. Z.—Tabish, S. M.—Mirza, F.—Farooq, M.: PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In: Kirda, E., Jha, S., Balzarotti, D. (Eds.): Recent Advances in Intrusion Detection (RAID 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5758, 2009, pp. 121–141.

[22] Siddiqui, M.—Wang, M. C.—Lee, J.: Data Mining Methods for Malware Detection Using Instruction Sequences. Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications (AIA '08), 2008, pp. 358–363.

[23] Singh, T.—Di Troia, F.—Corrado, V. A.—Austin, T. H.—Stamp, M.: Support Vector Machines and Malware Detection. Journal of Computer Virology and Hacking Techniques, Vol. 12, 2016, No. 4, pp. 203–212.

[24] Stopel, D.—Boger, Z.—Moskovitch, R.—Shahar, Y.—Elovici, Y.: Application of Artificial Neural Networks Techniques to Computer Worm Detection. Proceedings of the 2006 IEEE International Joint Conference on Neural Networks (IJCNN '06), 2006, pp. 2362–2369.

[25] Eskandari, M.—Hashemi, S.: A Graph Mining Approach for Detecting Unknown Malwares. Journal of Visual Languages and Computing, Vol. 23, 2012, No. 3, pp. 154–162, doi: 10.1016/j.jvlc.2012.02.002.

[26] Qiao, Y.—Yang, Y.—Ji, L.—He, J.: Analyzing Malware by Abstracting the Frequent Itemsets in API Call Sequences. 2013 12th IEEE International Conference

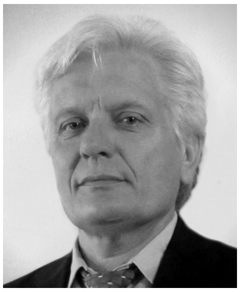
on Trust, Security and Privacy in Computing and Communications (TrustCom), 2013, pp. 265–270.

[27] Stanfill, C.—Waltz, D.: Toward Memory-Based Reasoning. Communications of the ACM, Vol. 29, 1986, No. 12, pp. 1213–1228, doi: 10.1145/7902.7906.

[28] Webb, A. R.—Copsey, K. D.: Statistical Pattern Recognition. Third Edition. Wiley, 2011.

[29] Wilson, D. R.—Martinez, T. R.: Improved Heterogeneous Distance Functions. Journal of Artificial Intelligence Research, Vol. 6, 1997, No. 1, pp. 1–34.

Martin Jureček graduated from the Charles University in Prague, Faculty of Mathematics and Physics, with the specialization in mathematical methods of information security. He is now a Ph.D. student at the Faculty of Information Technology of the Czech Technical University in Prague. His main research interests focus on the application of machine learning and artificial intelligence approaches to malware detection. Another area of his interest is cryptography and information security.



Róbert Lórencz graduated from the Faculty of Electrical Engineering of the Czech Technical University in Prague in 1981. He received his Ph.D. degree in 1990 from the Institute of Measurement and Measuring Methods, Slovak Academy of Sciences in Bratislava. Currently he is Full Professor at the Faculty of Information Technology of the Czech Technical University in Prague. His research interests are cryptography and arithmetic units for cryptography primitives, various cryptoanalysis methods of block and stream ciphers. Another topic of his interest is alternative arithmetic for numerical computation.