Heuristic Malware Detection Mechanism Based on Executable Files Static Analysis

A.V. Kozachok[1], M.V. Bochkov[2], E.V. Kochetkov[1]

[1]*Academy of the Federal Guard Service, 35, Priborostroitelnaya Street, Oryol, 302034, Russia*
[2]*Business risk educational center, 27, Professor Popov Street, Saint-Petersburg, 197022, Russia*

Abstract

To ensure the protection of information processed by computer systems is currently the most important task in the construction and operation of the automated systems. The paper presents the application justification of a new set of features distinguished at the stage of the static analysis of the executable files to address the problem of malicious code detection. In the course of study, following problems were solved: development of the executable files classifier in the absence of a priori data concerning their functionality; designing class models of uninfected files and malware during the learning process; development of malicious code detection procedure using the neural networks mathematical apparatus and decision tree composition relating to the set of features specified on the basis of the executable files static analysis. The paper also describes the functional model of malware detection system using the executable files static analysis. The conclusion contains the results of experimental evaluation of the developed detection mechanism efficiency on the basis of neural networks and decision tree composition. The obtained data confirmed the hypothesis about the possibility of constructing the heuristic malware analyzer on the basis of features distinguished during the static analysis of the executable files. However, the approach based on the decision tree composition enables to obtain a significantly lower false negative rate probability with the specified initial data and classifier parameter values relating to neural networks.

Keywords: Anti-virus protection, Malware, Neural networks, Decision trees, Heuristic analysis, Machine learning

1. Introduction

Security of information processed by computer systems poses the most important task for building and operating of automated systems today. Along with that, one of the most dangerous threat is computer malware that can modify (delete) user data, steal confidential information, slowdown or disable operating system. This research substantiates the possibility of using feature space based on the static analysis of executable files for solving the task of heuristic malware detection using the neural networks and decision trees composition mathematical apparatus.

Today there exist different malware detection techniques. The most widely known techniques are the following ones [1]:

– signature-based search (malicious code detection based on byte sequence which definitely characterizes it);

– heuristic search (code detection based on indirect attributes which characterize it as being malicious);

– behavioral mechanisms that affect executing forbidden operations by different processes (e.g., access to critical memory areas or executable code injection into other processes).

All above-listed techniques have essential weak points, i.e. they possess limited capabilities for detection of modified and new viruses, or require the user to be involved in the decision-making process with respect to file belonging to a certain class.

Today the antivirus software cannot guarantee 100% malware protection. The results of tests performed by AV-Comparatives in March 2017 show that heuristic detection rate of new malware strains amounts to approximately 95-98% for most of the modern antivirus software [2].

To detect new malware, heuristic methods or more generally statistical approaches are the most promising research trends nowadays. Some of them based on structural analysis and executable file features [3, 4, 5, 6]. One of the solutions for increasing the effectiveness of heuristic malware detection process is the development of new tools and techniques for malware detection. The purpose of this study is to substantiate the possibility to build a heuristic technique for malware detection based on static analysis of executable files. The distinctive feature of the approach suggested consists in the use of new feature space for building a heuristic detector based on the well-known machine learning techniques, i.e. neural networks and decision trees composition. In this context, a decision on the malware presence will be taken according to a certain law based on availability or absence of totality of features from criteria array defined at the stage of executable file static analysis.

2. Forming feature space based on static executable files analysis

In order to substantiate the possibility of using suggested feature space for solving the task of heuristic malware detection, the neural networks and decision trees composition technique has been applied in this study. Let us consider the totality of the features being studied.

The whole feature space may be divided into eight conditional feature groups. Group 1 comprises the features based on the results of characteristics evaluation for the following executable files parts: file header size, optional header, MS-DOS header, digital certificate. Since the structure of the headers has been defined, in case of their size change relevant attribute will be detected. Group 2 comprises the features associated with the use of packing, archiving and encrypting utilities for executable files such as UPX, MPRESS, PeCompact etc. Group 3 comprises information about dynamic libraries, as well as functions exported and imported by the executable file. The rate of certain API-functions and dynamic libraries usage by malware has been precomputed. As a result, two classes have been identified. The first class comprises API-functions by means of which malicious actions can be performed. The second class comprises the rest of the functions. In this context, belonging to a certain class of API-functions is to be regarded as a feature. Group 4 comprises data on digital certificates, namely, whether they are available in the file, whether data are out-of-date or have been recalled. Group 5 comprises the features based on the information about PE-file structure, namely, availability of anonymous section, whether overlay technique is applied, whether the first section is available for writing, whether the control function is transferred by the file to other files, entry point address is out of the file section boundaries, in the first or other sections, whether the last and other sections are of executable type. Feature group 6 is formed based on the manifest, its availability, correspondence of the manifest structure to standard format, whether the administration privilege is requested by the manifest etc. Group 7 comprises information about executable file interface with the operating system, namely, whether the use of Structured Exception Handling (SEH), Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR) is ignored, whether the application is executed in Visual Basic virtual machine, whether Thread Local Storage (TLS) is used. Group 8 comprises information not included in previous seven groups; for example, whether the file contains rigidly fixed IP-addresses, whether direct cookie links are present, whether databases are in use etc.

Each analyzed executable file is described in the form of a Boolean vector, where one means the feature is available, zero means no feature is available.

3. Static heuristic malware detection mechanisms

3.1. Machine learning detection mechanism based on static executable files analysis

To increase the effectiveness of heuristic analysis of executable files we suggest using the malware detection technique based on neural networks. Utilization of the neural network mathematical apparatus together with the created feature space will enable us to solve the following tasks:

1. generating class models in the course of learning (uninfected files and malware);
2. developing malware detection procedure through the use of feature vector based on static analysis of executable files;
3. classifying executable files without priori data on their infection with malicious code.

Solution of the task for developing the neural network malware detection technique based on static analysis of executable files comprises two main stages as follows:

1. learning the neural network which defines malware and uninfected file classes (learning subsystem);
2. calculating output values of neural network based on the sequence of features singled out of the files analyzed, and decision-making on files belonging to a certain class (classification subsystem).

Let us consider the learning process in detail. It consists of two main stages:

1. supervised learning the neural networks;
2. adjusting hyper parameter values for decision-making on the file infection status.

At the first stage, the neural network is learnt in a supervised mode. The network is provided with values of both input and priori known output signals, whereas weight coefficients are subject to corrective adjustment in order to increase the accuracy of the neural network learning.

The result of the first learning stage is a weighting coefficients matrix (model) of the learnt neural network.

At the second learning stage for the given learning set it is necessary to evaluate mistake probability values of the first and second grade depending on the hyper parameter values, and adjust them in accordance with the requirements to the first and second grade mistake criticality for further effective functioning of the neural network as a malware detection tool.

As a result of the second learning stage hyper parameter values to be used for executable file classification muat be chosen. The learning procedure result is a weighting coefficients matrix and hyper parameter values.

The detection procedure consists of two main stages [7]. During the first stage the detection system input receives the feature sequence singled out of the file analyzed. Then the neural network output values are calculated using weighting coefficients matrices entered into the database.

Depending on output value classifier makes a decision on the analyzed file belonging to a certain software class.

3.2. Heuristic malware detection mechanism based on decision trees composition

As an alternative approach, in order to substantiate the possibility to build heuristic malware detection tool based on static analysis of executable files this study provides the results of classifier effectiveness evaluation based on the decision trees composition.

As a rule, composition of algorithms is regarded as a combination of N algorithms $d_1(x), \ldots, d_N(x)$ in a single one. The idea consists in learning the algorithms and averaging of the obtained responses:

$$a(x) = \frac{1}{N} \sum_{n=1}^{N} d_n(x). \tag{1}$$

This formula directly answers the regression problem. For the case of binary classification $d_1(x), \ldots, d_N(x)$ it is necessary to take a sign from the resulting formula:

$$a(x) = sign \frac{1}{N} \sum_{n=1}^{N} d_n(x). \tag{2}$$

To build a decision trees composition [8] first it is necessary to learn the basic N algorithms on different subsets singled out from the learning set. To single out random sets, an approach based on the random sets generation from the learning set through removal followed by return procedure (bootstrap) has been applied in this study. At the same time, the size of each subset amounts to $0,632L$, where L is the learning set size.

Additionally, random subspace technique [9] has been applied. The technique consists in choosing the random subset of features for learning each basic algorithm. The number of the features chosen is a hyperparameter of the given technique.

4. Results and Discussion

At the learning stage for both approaches described above it is necessary to create two representative learning sets: uninfected files and malware. A test set is to be created using the files that are not included in the learning file set. During the pre-processing stage, a totality of features is singled out in the form of a Boolean vector from the executable files sent to the system input.

For learning block it is required to single out a totality of feature sequences from the whole totality of files of the representative learning set. For detection block it is required to single out a feature sequence from one file which was received at the classification system input.

Initial data used in the study:

– 1862 uninfected files (system and program files collected from different Windows operating systems);

– 1910 malware files (authors private collection);

– 353 features singled out based on static analysis of executable files.

As a result of performed static analysis of provided file sets, a feature vector has been generated for each executable file of both classes. Groups of performed experiments have confirmed the hypothesis that building a malicious code heuristic analyzer based on the static analysis of executable files is possible.

To evaluate the effectiveness of the neural network technique the following initial data have been used:

– learning rate factor (constant) 0.001;

– accuracy 0.0001;

– number of iterations for reaching the required learning accuracy has been limited by 200.

Hyper parameters to adjust were [10]:

– activation function selection:

  – *relu*, the rectified linear unit function, returns $f(x) = \max(0, x)$;
  – *tanh*, the hyperbolic tan function, returns $f(x) = \tanh(x)$;
  – *logistic*, the logistic sigmoid function, returns $f(x) = 1/(1 + \exp(-x))$;
  – *identity*, no-op activation, useful to implement linear bottleneck, returns $f(x) = x$;

– solver function selection:

  – *adam* refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba [11];
  – *sgd* refers to stochastic gradient descent;
  – *lbfgs* is an optimizer in the family of quasi-Newton methods;

– number of hidden layers and neurons.

The first group of experiments allowed us to select appropriate activation function and solver values. Figure 1 shows the results of experimental evaluation with crossvalidation based on our dataset divided in two sets train (0.7) and test (0.3) and combination of various parameter values.We used one hidden layer with one hundred neurons. As a score value we selected F-measure.

A box plot (box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the F-measure values while the whiskers extend to show the rest of the distribution, except for points that are determined to be ”outliers” using a method that is a function of the inter-quartile range. The mean value is shown by line inside box. Figure 1 shows us approximately equal three possible combination variants: logistic_lbfgs, tanh_lbfgs, relu_adam.

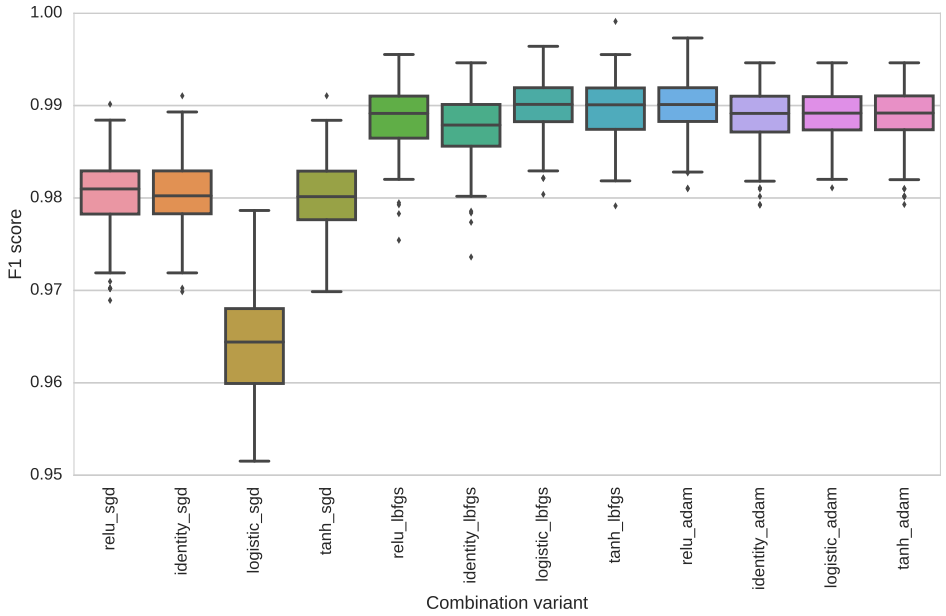

Figure 1: F-measure depending on activation function and solver combination variant.

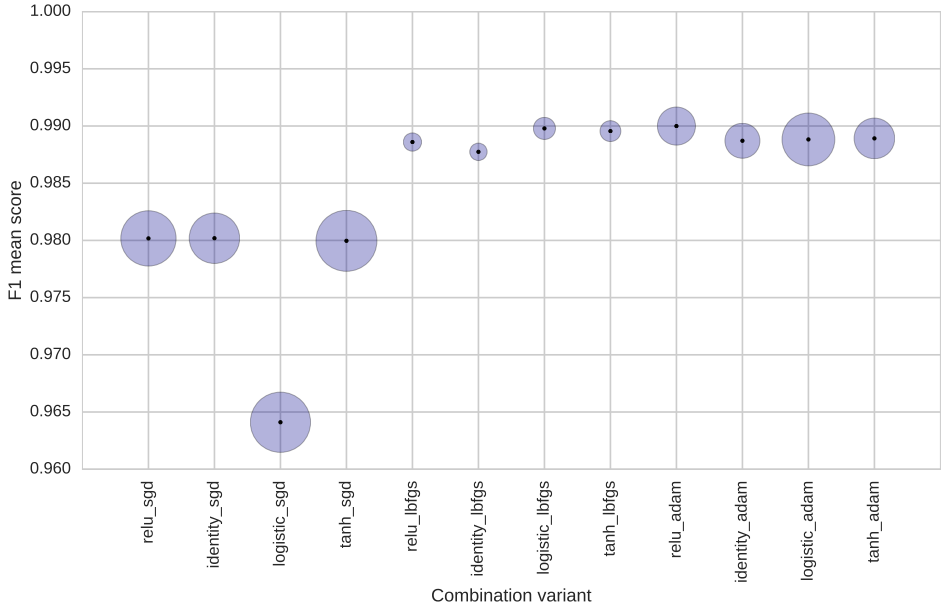

Figure 2: F-measure and crossvalidation time depending on activation function and solver combination variant.

Another group of experiments allowed us to select exactly one combination: tanh_lbfgs, because it is the fastest one. Figure 2 shows mean F-measure value for all variants (black dots) and blue rounds show relative crossvalidation time.

The third group of experiments allowed us to select a number of hidden layers and neurons. The initial state was with one hidden layer with 10 neurons. We used maximum three hidden layers with 100 neurons. The step was equal to 10 neurons.

Table 1 shows 15 best F-measure values got during experimental evaluation sorted from max to min.

Table 1: F-measure mean crossvalidation score depending on number of hidden layers and neurons

Hidden layer neurons			F-measure mean score
1st	2nd	3rd	
20	*70*	*80*	0.99110
20	*20*	*0*	0.99109
50	10	100	0.99092
20	10	70	0.99083
20	40	40	0.99082
10	20	60	0.99074
20	30	70	0.99074
70	20	30	0.99074
20	40	0	0.99074
20	70	60	0.99073
20	50	100	0.99047
10	100	90	0.99047
20	50	50	0.99047
20	60	0	0.99047
20	10	50	0.99047

First two rows have closed values, but the first neural network configuration consists of three hidden layers, in spite of the second one with two layers. The first one consumes for about 60% much more time to classify a test set than the second.

The performed groups of experiments have enabled us to substantiate this hyper parameter values:

– activation function – *tanh*:

– solver function – *lbfgs*:

– 2 hidden layers with 20 neurons.

As a result of the developed malware detection technique based on neural network application the following values have been obtained:

– Accuracy = 0.99125;

– F-measure = 0.99110;

– Sensitivity = 0.99425;

– Specificity = 0.99409.

Then the classifier has been learnt based on the decision trees composition, and the obtained result has been cross validated. Figure 3 shows the relation between detection accuracy and number of decision trees. Along with that, the sets of both classes have been divided with the proportion of 0.7 (learning), 0.3 (test).

The performed experimental evaluation of the developed solution allows us to substantiate the following hyper parameters of the classifier:

– number of decision trees is 85;

– number of valuable features is 55 (figure 4).

The following evaluation values have been obtained when choosing the given parameters of the decision trees composition classifier:

– Accuracy = 0.99240;

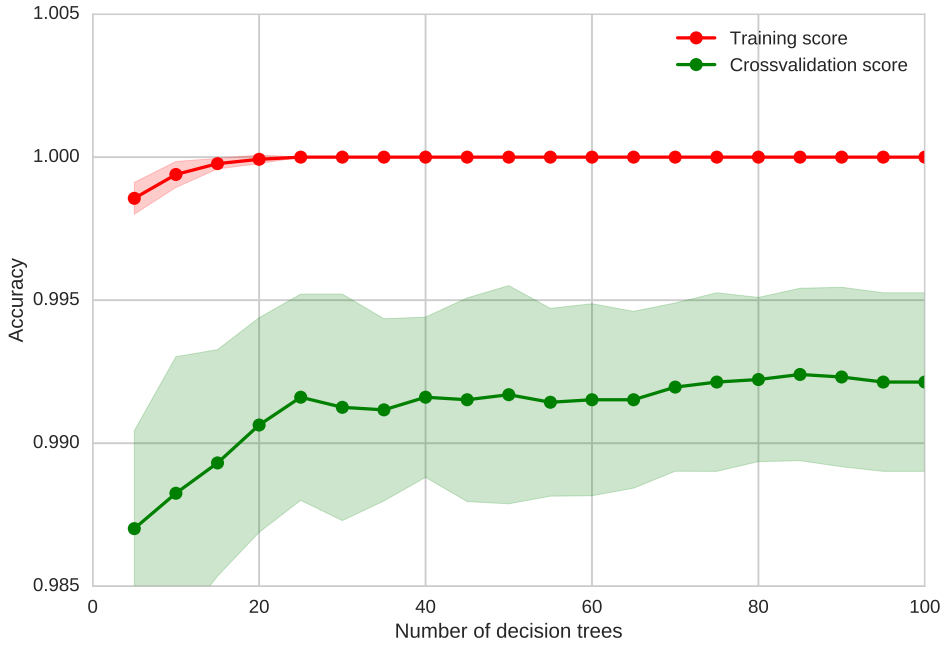

Figure 3: Accurarcy value depending on number of trees.

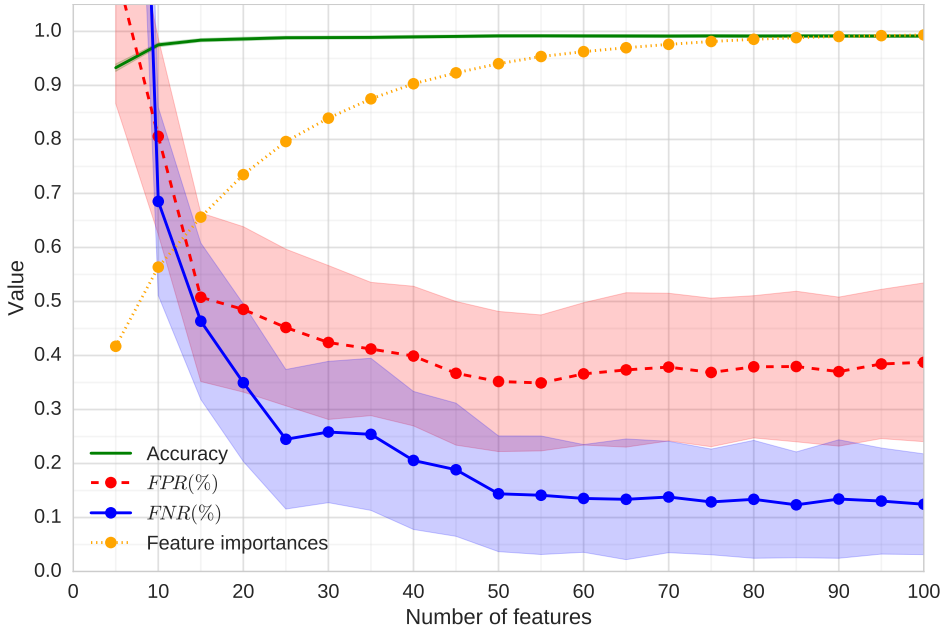

Figure 4: Accurarcy value depending on number of features.

– F-measure = 0.99226;
– Sensitivity = 0.99616;
– Specificity = 0.99605.

5. Conclusion

It should be noted that both approaches have confirmed the hypothesis that building a malicious code heuristic analyzer based on features singled out during static analysis of executable files is possible. However, the approach based on the decision trees composition allows obtaining better accuracy value relative to neural network tool with the above-mentioned initial data and classifier hyper parameter values.

The obtained values of mistake probabilities for developed prototypes comply with the requirements of the Federal Service for Technical and Export Control [12] imposed to antivirus software.

In conclusion it should be noted that the suggested approach consisting in the application of the space of features singled out from the executable files at the stage of static analysis and well known machine learning techniques enables us to implement a new mechanism for heuristic detection of malicious code which provides the possibility to reveal new and modified malware samples.

References

[1] Kozachok, A. V. Mathematical model of destructive software recognition tools based on hidden markov models [Text] / A. V. Kozachok // "Vestnik SibGUTI". — 2012. — Vol. 3. — P. 29–39. — (in Russian).

[2] Anti-Virus Comparative — malware protection test [Electronic resource]. — 2017. — URL: https://www.av-comparatives.org/wp-content/uploads/2017/04/avc_mpt_201703_en.pdf.

[3] Siddiqui, M. A survey of data mining techniques for malware detection using file features [Text] / Muazzam Siddiqui, Morgan C. Wang, Joohan Lee // Proceedings of the 46th Annual Southeast Regional Conference on XX. — ACM-SE 46. — New York, NY, USA : ACM, 2008. — P. 509–510. — URL: http://doi.acm.org/10.1145/1593105.1593239.

[4] Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey [Text] / Asaf Shabtai, Robert Moskovitch, Yuval Elovici, Chanan Glezer // Information Security Technical Report. — 2009. — Vol. 14, no. 1. — P. 16–29. — Malware. URL: http://www.sciencedirect.com/science/article/pii/S1363412709000041.

[5] Opem: A static-dynamic approach for machine-learning-based malware detection [Text] / Igor Santos, Jaime Devesa, Félix Brezo [et al.] // International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions / Springer. — Ostrava, Czech Republic : Springer-Verlag Berlin Heidelberg, 2013. — P. 271–280.

[6] David, B. Structural analysis of binary executable headers for malware detection optimization [Text] / Baptiste David, Eric Filiol, Kévin Gallienne // Journal of Computer Virology and Hacking Techniques. — 2017. — Vol. 13, no. 2. — P. 87–93. — URL: http://dx.doi.org/10.1007/s11416-016-0274-2.

[7] Dropout: A simple way to prevent neural networks from overfitting [Text] / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky [et al.] // J. Mach. Learn. Res. — 2014. —jan. — Vol. 15, no. 1. — P. 1929–1958. — URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[8] Schmid, H. Probabilistic Part-of-Speech Tagging Using Decision Trees [Text] / Helmut Schmid. — Manchester, UK : UMIST, 1994.

[9] Shi, T. Unsupervised learning with random forest predictors [Text] / Tao Shi, Steve Horvath // Journal of Computational and Graphical Statistics. — 2006. — Vol. 15, no. 1. — P. 118–138.

[10] API design for machine learning software: experiences from the scikit-learn project [Text] / Lars Buitinck, Gilles Louppe, Mathieu Blondel [et al.] // ECML PKDD Workshop: Languages for Data Mining and Machine Learning. — [S. l. : s. n.], 2013. — P. 108–122.

[11] Kingma, D. Adam: A method for stochastic optimization [Text] / Diederik Kingma, Jimmy Ba // arXiv preprint arXiv:1412.6980. — 2014.

[12] Federal Service for Technology and Export Control. Informational report on antivirus software requirements approval [Text]. — 2012. — (in Russian).