Using Machine Learning Algorithms to Detect Malware by Applying Static and Dynamic Analysis Methods

Jakub Palša, Ján Hurtuk, Martin Chovanec, Eva Chovancová

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
email: {jakub.palsa, jan.hurtuk, martin.chovanec, eva.chovancova}@tuke.sk

Abstract: This paper focuses on malware analysis and detection using machine learning methods. The aim of the authors was to perform static and dynamic analysis of programs designed for Windows and then to present the results of the analysis as a dataset. We analysed and implemented different classification methods, such as decision trees, random forests, support vectors and naive Bayes methods. We verified their ability to distinguish malicious and harmless samples and evaluated their success rate using classification accuracy metrics. Then, we compared the results obtained by prediction over the dataset generated by static and dynamic analysis. Classification was more successful on the data gained using the dynamic analysis method. The best malware detection algorithms have been found to be decision tree-based algorithms, in particular the random forest algorithm, which achieves excellent malware detection accuracy of up to 95.95% with a standard deviation of only 0.58%.

Keywords: malware; static analysis; dynamic analysis; dataset; classification

1 Introduction

As the number of every day users using computers/IT systems increases, so does the desire of attackers to obtain and exploit sensitive user information through malware. The number of threats and their severity is constantly increasing, and while in some cases, the damage caused by malware may be imperceptible to the user, in other cases it can lead to severe losses.

As malware evolves over time, the creators of security solutions, aimed to protect systems from malware are seeking and developing new ways to detect it. The traditional method of malware detection using signature recognition is becoming less and less effective as attackers often use various obfuscation techniques to modify malware code to evade this detection method and this method allows only the detection of known malware with known symptoms. Even a minor delay in the response of security solution providers upon the arrival of a new type of malware can cause irreparable damage, which motivates researchers to find more sophisticated ways to detect malicious samples, especially new malicious samples that have not been analysed before.

The drawbacks of traditional malware detection methods are sought to be addressed by machine learning techniques, capable of detecting malware with a high degree of accuracy. Machine learning allows a program to learn from available samples and then to react to new samples, using the learned information. The efficiency of using machine learning to detect malware is boosted by the availability of labeled malicious samples freely available not only to security experts, but now also to the research community. Another factor of this is also the rapid growth of ever cheaper computing power, which allows researchers to speed up machine learning training and to use large quantities of samples.

And that is why in our research we take the path of detecting malicious software through machine learning methods. In the first place in chapter no. 2, we analyse the expertise of researchers who deal with this issue. Next, we describe the sequence of steps of our research. In chapter no. 3 we point out security and its security work with malicious software. Chapter no. 4 describes how we prepared test samples, which were to perform the analysis, which we discuss in Chapter no. 5. In chapter no. 6 we will create a method of creating a data set and in chapter no. 7 we point out the classification methods of machine learning, which were trained, tested and subsequently evaluated on the basis of success. In chapter no. 8 we evaluate and interpret the results of individual machine learning models. In the chapter no. 9 we compare our best results with the results of researchers from chapter no. 2.

2 Related Works

The problem of malware detection using machine learning is not new and has been addressed in many other works. Two key phases have emerged in using machine learning for malware detection: feature extraction from the input data; and selection the most relevant ones that best represent the set of samples and classification. Extracting the features of potentially malicious samples can be done by analysis – static or dynamic [1]. The goal of the analysis is to understand the capabilities of the particular piece of malware, the system parts and files it can attack, its structure, etc. Static analysis is performed without running the analysed sample, as an examination of the structure and code of the analysed sample using various tools [2]. Dynamic analysis focuses on the behaviour of the analysed sample at runtime, observing the interactions with the system and its impact on the system. Both types of analysis have their advantages and limitations and complement each other. After feature extraction, each sample is represented as a feature vector, used by a classification algorithm to train a machine learning model.

2.1 Experimental Results

This section provides an overview of several studies in malware detection and also describes some shortcomings of each approach.

Bai et al. [3] focused on malware detection in a dataset of 19113 executables – 8592 harmless and 10521 malicious samples. As features, they used the information obtained by static analysis of the headers of the executable files. They claimed to have found a total of 197 features, allowing them to distinguish harmless samples from harmful ones; they also used filtering and wrapper methods to select the most appropriate features. In their study, they evaluated the use of classification algorithms – J48 (decision trees) and random forests. To improve the performance of the J48 algorithm, they used combinations of multiple trained models in bagging and boosting techniques. The aforementioned authors performed a total of 3 experiments, differing in the way the data was classified and the choice of features. They concluded that this approach could detect unknown malware with a high accuracy, while maintaining a low false positive rate. The detection accuracies achieved in the experiments ranged from 94.6% to 99.1%. As they concluded, the random forests algorithm and the bagging and boosting techniques significantly increased the classification accuracy, compared to the case when they used the J48 algorithm without using these techniques.

The comparison of various classification algorithms was also discussed by Kumar et al. in [4]. They compared decision trees, random forests, K-nearest neighbors, logistic regression, linear discriminant analysis and naive Bayes algorithms on a dataset of 5210 (2488 harmless and 2722 harmful) samples. As features (68), they used the information obtained from the headers of the analysed executable files. The aforementioned authors achieved the best overall accuracy (98.78%) using the Random Forests algorithm. The worst detection success rate (56.04%) was found when using the Naive Bayes algorithm. An interesting feature of the work was the use of the so-called *integrated feature set*, which was used to increase the overall accuracy of the classification algorithms.

In [5], Moser et al. point out the problems of static analysis in malware detection. They demonstrate obfuscation techniques and point out that static analysis alone may no longer be sufficient for malware detection. In their paper, they conclude that dynamic analysis should be a necessary complement to static analysis, as it is significantly less vulnerable to obfuscating code transformations.

In [6], Firdausi et al. used the K-nearest neighbours, Naive Bayes, Support Vector Machine, J48, and Multilayer Perceptron algorithms on the features obtained by dynamic analysis using Anubis, a freely available dynamic analysis tool. The dataset consisted of executable samples, including a total of 220 unique malware samples. In several different experiments, they achieved the best accuracy (96.8%) using the decision tree-based J48 algorithm. On the contrary, they achieved the worst results (only 62.8%) using the Naive Bayes algorithm. As the authors conclude, malware detection using machine learning combined with dynamic analysis is a fairly effective method.

Shijo & Salim [7] also focused on this approach and took advantage of the benefits of static and dynamic analysis. They used a combination of both types of analysis to detect malicious samples using machine learning on a dataset of 1487 (997 malicious and 490 benign) samples. Static features were strings extracted from executables. They performed dynamic analysis using the Cuckoo Sandbox tool in a secure, virtual environment, outputting a report on the execution behaviour of each sample, listing API calls and registry changes. The authors used 2 algorithms in their work, namely the support vector method and the random forest algorithm. As the authors of the aforementioned paper reported, they obtained best results using the support vector method, namely a detection accuracy of 95.88% for static analysis, 97.1% for dynamic analysis and 98.7% by combining the two. Thus, the achieved results showed that the combination of static and dynamic analysis increased the detection accuracy compared to the use of static and dynamic analysis alone. However, a disadvantage of the study was the smaller number of samples used for training.

2.2 Evaluation of the Experiments

The related works and existing solutions make it clear that using machine learning to detect malicious samples is advantageous and brings a number of benefits over the traditional malware detection approaches. Using a combination of static and dynamic analysis to extract symptoms from individual samples seems to be the most advantageous, as using only one of the methods is no longer sufficient. Research shows that when selecting an appropriate machine learning algorithm, using decision trees to detect malware seems to be a suitable approach – due to the accuracy of detecting malicious samples using decision-tree-based algorithms. Detection accuracy can also be increased by combining multiple trained models, as it is evident in the case of using the Random Forests algorithm. Knowing this, one may design a system capable of classifying a sample as harmful or harmless with high accuracy, based on performing static and dynamic analysis of the particular sample.

Based on the data obtained from previous research experiments, we performed further research. This focused on the combination of static and dynamic analysis and on the combination of several trained malware detection models.

3 Secure Test Environment

An important element of static and dynamic software analysis is the environment, in which the analysis itself takes place. The goal is to create an environment providing no obstacles to the particular piece of malware, allowing its observation in its full beauty. However, it is necessary to prevent malware from breaking out from this environment and causing real damage.

3.1 The Virtual System

For our research, we chose to use *Oracle VirtualBox 6.1* virtualization software. It should be noted that keeping virtualization software up-to-date is key, as many types of malwares attempt to detect execution in a virtual environment and exploit its security flaws to infect the host system.

As the guest virtual operating system, we chose *Windows 7*. At the time, this version of the operating system was widely used and widely deployed. As a result, a large amount of malware targeting this system appeared. Compared to Windows 10, Windows 7 can be modified to execute malicious code more easily, as Windows 10 incorporates a number of automated security features that are laborious to disable and keep disabled.

We installed *Dependency Walker* (a static analysis tool) and *Cuckoo Sandbox* (a dynamic analysis tool, necessary for the execution and to uncover the intent of the particular piece of malware) into the virtual environment. The installation of third-party software also helped to reduce the sterility of the operating system. The latter could cause the malware to detect the execution of the virtual environment and lead to a failure of the analysis.

In order to pretend that the environment is that of a device used daily, for some time, we used the virtual operating system to perform common activities such as browsing web pages, downloading documents from the Internet, playing audiovisual media, etc. This regular use of the system led to the creation of temporary files and registry entries, which also help to mask the fact that it is a virtual system.

To increase the likelihood of successful malware execution, we used older versions of the respective programs.

A very important step in the preparation of the virtual test environment was the modification of the security settings of the Windows operating system. The modifications consisted of disabling the following:

1) *Windows Defender*, a security program that would actively prevent malware execution,
2) the *Windows Update* service, which could install security patches and passively prevent malware from being executed; and
3) *Windows Firewall*, a security program that would monitor the flow of data between networks.

A snapshot of the system was taken after the system configuration was completed. This provides continuous access to the desired virtual operating system configuration, which can be restored any time, preventing lengthy reconfiguration.

Creating a system snapshot is a very important step in performing dynamic analysis. Restoring it allows to negate any impact of the analysed code on the system. It also ensures the same starting conditions for the analysis of each sample. The system snapshot is an important element, also used by the *Cuckoo Sandbox* tool when automating the analysis.

3.2 The Host System

As the host operating system of the workstation, we used *Ubuntu 18.04*. This Linux-based system was chosen because *Cuckoo Sandbox* works best on Linux-based systems. Version 18.04 was necessary because it is the last version of the Ubuntu operating system that both natively supports and includes the *Python 2.7* programming environment. Python 2.7 is required to properly install Cuckoo Sandbox software and its supporting programs, as currently, newer versions (3.x) are not supported by the Cuckoo Sandbox project. The employed version of Cuckoo Sandbox was version 2.0.7.

For added security when working with malware, *virtuaenv*, a Python virtual environment has been established on the host system, without administrative (*sudo*) privileges. With this, every time Cuckoo Sandbox needed to perform an operation requiring such a privilege, the user had to confirm the operation.

4 Preparation of Test Samples

For the purposes hereof, malware samples were obtained from *virusshare.com*, an online malware sample repository [9]. This provides real malware samples for people such as security researchers, forensic analysts, etc. It is maintained by the users themselves, contributing verified malware samples to it.

We downloaded the *VirusShare_00164.zip* package. We chose to use it for its relatively small size (11.88 GB), compared to other packages. Moreover, the more significant reason for its selection was the date it was added – 15 September 2015. This paper focuses on the analysis of malware infecting Windows devices, as this system is the target for the largest amount of available malware.

For this reason, 15 September 2015 is potentially the most appropriate date, as:

- the most widely used version of the Windows operating system in 2015 was Windows 7, with a 62.31% share [10] of all Windows versions.
- Windows 10 was released on 29 July 2015. Thus, back then, a large amount of malware was uploaded to VirusShare.com by the users of Windows 7, the target system for this work.

The healthy samples used in this work are executable programs such as web browsers, audio and video players, UI customization tools, etc. These were obtained from *portablefreeware.com* [11] and *portableapps.com* [12], hosting a large number of downloadable executables.

The downloaded malware sample package in the zip archive contains 65536 malware samples. For the purposes hereof, 3000 executable samples with the .exe extension were selected. Healthy samples are represented by 838 executable programs. Thus, there were approximately 3.6 malware samples for each healthy sample.

A total of 3838 samples were analysed – see Table 1. These were analysed to create the dataset needed for the machine learning process.

Table 1
Number of Samples Prepared for Analysis

sample class	**sample count**
malware	3000
healthy	838
malware + healthy	**3838**

5 Analysis Execution

After successfully preparing the test samples, we produced the final dataset, which we then used to perform static and dynamic analysis.

5.1 Comparison of Static and Dynamic Analysis

Unlike static analysis, dynamic analysis does not require malware source code, as it can be performed on any application [13] [14]. Unlike static analysis, dynamic analysis can track the actual malware functionality [15], since certain parts of the code, such as an imported library, do not mean active execution of particular library functions.

However, static analysis has several advantages over dynamic analysis (where the examined sample is executed), the most significant of which being speed, security and low requirements [16].

The fact that static analysis does not monitor the behaviour of the programs, comes with certain disadvantages, which are, on the other hand, the advantages of dynamic analysis [17] [18]:

- it is impossible to observe the real behaviour of the particular program;
- it is hard to detect functions actually used;
- it is hard to classify programs with hardly accessible code;
- it is impossible to identify unknown malware.

5.2 Static Analysis

To generate the outputs of the static analysis, we used Microsoft's *Dependency Walker* tool to analyse executable files. It displays all the modules of the monitored file in a hierarchical tree structure. It is freely available for 32 and 64-bit Windows systems.

Using *Dependency Walker*, we analysed the file headers and functions. Then, we saved the obtained information in a text file. Given that over 3000 samples had to be analysed, doing this manually was not an option. Therefore, we used the *Robotask* tool to run *Dependency Walker* and then send instructions to it. It looped through all the samples in the folder and sent the following instructions:

- **CTRL+O** – open dialog box to open the file;
- **absolute path** – the path to the file to be analysed;
- **ENTER** – confirm the selected file.

Then, after the analysis, Robotask sent further instructions:

- **CTRL + S** – save the retrieved files;
- **absolute path** – where to save the data;
- **3 x TAB** – select the format of the file to be saved – we chose „Text with list of imported/exported functions"
- **ENTER** – confirm saving;

The obtained text files contained a huge amount of sample data. For the purposes hereof, it sufficed to focus on the essential details to distinguish a malicious sample from a clean sample. The publicly available Windows API puts enormous power in the hands of malware creators [19], and this is what our research focuses on. In [20], the authors describe the features most commonly used by malware.

5.3 Dynamic Analysis

The first step of dynamic analysis was to configure the *Cuckoo Sandbo*x environment. The option to create a memory dump was disabled after the sample analysis was completed. Creating memory dumps of the virtual operating system for each analysed sample would quickly fill the storage space of the workstation used to perform the analysis. Moreover, this analysis information is not relevant to the purposes hereof.

The analysis mode was set to *headless*, so the analysis would be performed in the background, without allowing on-screen observation of the behaviour of the virtual system. This saved system resources. Moreover, with such a large amount of analysis it would have been impossible to monitor the graphical output of the virtual system. For the same reason, we also disabled the virtual system tool producing screenshots upon any change on the screen during the analysis.

We enabled the possibility to simulate mouse movements in the virtual system during the analysis, to create a more credible impression of the real system running the malicious code.

After finishing the configuration of the *Cuckoo Sandbox* tool, we could actually execute the analysis. The analysis can be initialized either using a command line interface or through a graphical web interface using the *localhost* server on port 8080. Since the web interface was more user-friendly, we chose this for batch execution of the analyses.

The following items can be configured before running the analysis:

• *Network routing*	how the sample to be analysed will access the Internet.
• *Package*	the file type according to which the appropriate analysis procedure will be selected.
• *Priority*	the priority of the analysis of the sample.
• *Timeout*	an important parameter of the analysis – this determines how long the analysis will run (in seconds).

6 Creating the Dataset

The dataset is a set of data used to train, test or otherwise work with algorithms. However, it has to have the appropriate form to allow any further operation.

6.1 Processing the Outputs of Static Analysis

After analysing all datasets, it was necessary to consolidate the results into the final dataset form. As the dataset we used the occurrences of importing the 112 selected functions.

Then, we saved the dataset in *comma-separated values (csv)* format. For this purpose, we wrote a Python script to sequentially scan through the obtained text files containing information about the samples, to find the aforementioned features. We were only interested in those occurrences where the function was actually used, i.e. we only searched for imported functions. The search results were written to another text file. At the beginning of this file, there was a header with the "TARGET" entries (i.e. whether the sample was malicious or not), the "filename" (name of the sample) and then the names of all selected functions. Two different versions were created. The first contained the number of occurrences of each of the selected functions, while the second contained only binary information about whether the function was used at least once. One line was created for each sample, with the following format: **clean file – 0, malicious file – 1**, filename, then the number of calls and/or binary information for each selected function. This data was separated by commas, as it is common for *.cvs* files, well-known by the libraries used in machine learning.

After performing the static analysis and converting the results to csv format, the dataset was ready for use. In its final form, it had 3584 records. Its layout is shown in Table 2.

Table 2
Structure of the Dataset after Static Analysis

sample class	sample count
malware	2747
healthy	837
malware + healthy	**3584**

6.2 Processing the Outputs of Dynamic Analysis

After successful analysis of the malware sample by the Cuckoo Sandbox automation software, an analysis report is generated. This provides a summary of the results of all the processes that were performed on the sample. To create a dataset to train and test the machine learning model, all obtained reports have to be processed. To process the data and create the dataset, we used the Python programming language, specifically version 2.7.

The standard data package obtained after analysing the sample is a directory with many subdirectories. However, we were only interested in the *reports* subdirectory, containing the resulting analysis report (called *report*, stored in *json* format).

In order to work with the analysis reports as efficiently and quickly as possible, a script was created – this goes through all the files and deletes the ones that are not named *report.json*. The unnecessary files include various temporary files created by *Cuckoo Sandbox*. This reduces the number of files scanned during the later operations and increases the speed of those operations.

After cleaning-up the working directory containing the analysis reports, we could start processing the reports themselves. The processing consisted of the following steps:

1) List the names of all functions called from the *Windows API* library. The function names will form the attributes of the respective samples. A separate script was created to retrieve all unique function names from all messages.
2) Create the dataset structure. The number of analysed samples indicates the number of rows in the dataset. The header consists of attributes whose count is equal to the number of unique system calls obtained in the previous step.
3) Rescan all reports. The first pass was necessary to determine the number of samples and their total number of attributes to create the dataset structure. The second pass is handled by a similar script, though this time the script adds each function call found in the message to the appropriate column labelled with the name of that function for each single sample found in the dataset.
4) Add a binary identifier as the first attribute of the sample to indicate whether it is malicious or benign. Malware samples had the malware attribute set to 1, while healthy samples had this attribute set to 0.
5) Save the dataset to a file in comma-separated-values (*csv*) format.

The structure of the dataset gained by processing the reports is shown in Table 3.

Table 3

Structure of the Dataset after Dynamic Analysis

sample class	**sample count**
malware	2937
healthy	828
malware + healthy	**3765**

7 Machine Learning Classification Methods

In this work, four supervised machine learning classification methods ([21], [22], [23], [24]) have been investigated:

1) the Decision Tree method,
2) the Random Forest method,
3) the Support Vector method, and
4) the Naive Bayes method.

For each classification method, a classifier was selected from the *Scikit-Learn* library [25], this was then trained and tested. The data were normalized by scaling using the *StandardScaler()* function. For the support vector method classifier, unlike the other classification methods used, up to 3 models were created, depending on the type of kernel used.

A custom function with nested *for ()* loops was used to tune the hyperparameters, where all combinations of hyperparameter values were tested. Instead of cross-validation, a custom implementation was created. In this, all combinations of hyperparameters were tested 30 times, but at each of the 30 iterations, the dataset was re-segmented into training and test datasets in order to obtain diverse input data. This ensured that the success of the classifier in prediction was verified by using a particular combination of its hyperparameters.

8 Evaluation and Interpretation of Results

The evaluation phase of the prediction model is where the ability of the applied algorithm to correctly classify a sample from the test dataset is verified. Unlike in the learning phase (where, in addition to the training data, the algorithm uses also the attributes of classes of these data), in this phase, when working with the test data it has no information about what group the sample belongs to. In this work, all classifiers were trained on 75% of the input dataset and tested on the remaining 25%. The algorithm assigns a class attribute to the test data – according to its best knowledge – and then its performance is evaluated by the evaluation metric used.

Many evaluation metrics will use *true positive* (TP), *false positive* (FP), *true negative* (TN) and *false negative* (FN) values in their calculations. These values are expressed in a confusion matrix.

The confusion matrix, as shown in Table 4, divides the data into four groups according to their actual and predicted class:

- *true positive* – correctly classified positive samples, i.e. samples correctly classified as malware,

- *false positives* – misclassified negative samples, i.e. harmless samples classified as malware,
- *true negative* – correctly classified negative samples, i.e. samples correctly classified as harmless,
- *false negative* – misclassified positive samples, i.e. malware samples predicted to be harmless.

Table 4
Confusion Matrix

		PREDICTED CLASS	
		positive	*negative*
TRUE CLASS	*positive*	**true positive**	**true negative**
	negative	**false positive**	**false negative**

It is very important to choose an appropriate evaluation metric because not every metric is suitable for all cases. It depends on whether we desire to achieve a high overall success rate for the delivered data or whether the focus is on a particular class. For the purposes hereof, we used the following evaluation metrics [26]:

- classification accuracy and
- sensitivity.

Classification accuracy

The metric of classification accuracy as shown in Equation 1, indicates the proportion of correctly classified samples to all samples. In measuring classification accuracy, the size of the classes is not taken into account and hence no weights are assigned to the classes.

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{1}$$

Sensitivity

Sensitivity as shown in Equation 2 is the ratio of correctly classified positive samples to the total number of positive samples. It represents the percentage of correctly identified malware files.

$$Sensitivity = \frac{TP}{TP + FN} \tag{2}$$

8.1 Decision Tree

Table 4 shows the results obtained using the decision tree algorithm. Using static analysis, the decision tree method achieved the highest classification accuracy values of almost 90%. When using dynamic analysis, the values exceeded 94%.

Table 4

Success Rate of the Decision Tree Model

Static analysis			Dynamic analysis		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
DT_S_1	89,74 ± 0,98	95,76 ± 1,08	DT_D_1	94,53 ± 0,74	96,37 ± 0,73
DT_S_2	89,70 ± 0,99	95,75 ± 1,03	DT_D_2	94,38 ± 0,65	96,27 ± 0,69
DT_S_3	89,63 ± 1,11	95,73 ± 1,22	DT_D_3	94,30 ± 0,80	96,10 ± 0,93
DT_S_4	89,61 ± 0,98	95,06 ± 0,77	DT_D_4	94,22 ± 0,78	96,22 ± 0,79
DT_S_5	89,59 ± 1,18	95,10 ± 1,07	DT_D_5	94,18 ± 0,73	96,23 ± 0,71

8.2 Random Forest Method

Table 5 shows the prediction success rate of the Random Forest method. The latter achieved the highest classification accuracy values for both static and dynamic analysis, reaching values exceeding 91% for static analysis and almost 96% for dynamic analysis. This proves that the composite random forest method is more efficient than the decision tree method alone.

Table 5

Success Rate of the Random Forest Model

Static analysis			Dynamic analysis		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
RF_S_1	91,32 ± 0,92	96,94 ± 0,60	RF_D_1	95,95 ± 0,58	98,08 ± 0,49
RF_S_2	91,26 ± 0,88	96,91 ± 0,54	RF_D_2	95,93 ± 0,62	98,06 ± 0,51
RF_S_3	91,25 ± 0,90	97,00 ± 0,61	RF_D_3	95,91 ± 0,70	98,12 ± 0,53
RF_S_4	91,24 ± 0,85	96,84 ± 0,61	RF_D_4	95,90 ± 0,68	98,10 ± 0,53
RF_S_5	91,23 ± 0,88	96,80 ± 0,63	RF_D_5	95,88 ± 0,72	98,10 ± 0,56

8.3 Support Vector Method with a Linear Kernel

The Support Vector method achieved the highest classification accuracy amounting to 87.94% when using static analysis and 95.95% when using dynamic analysis, as shown in Table 6. In dynamic analysis using the Support Vector method, the linear kernel yielded the highest value of classification accuracy of all kernels.

Table 6

Success Rate of the Support Vector Model Using a Linear Kernel

Static analysis			Dynamic analysis		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
SVCL_S_1	87,94 ± 1,02	95,75 ± 0,86	SVCL _D_1	92,38 ± 0,83	97,82 ± 0,62
SVCL_S_2	87,92 ± 0,84	95,44 ± 0,87	SVCL _D_2	92,37 ± 0,84	97,83 ± 0,63
SVCL_S_3	87,01 ± 1,19	96,76 ± 0,70	SVCL _D_3	92,36 ± 0,82	97,81 ± 0,62
SVCL_S_4	84,10 ± 1,10	88,42 ± 1,32	SVCL _D_4	92,25 ± 0,77	97,52 ± 0,63
SVCL_S_5	83,93 ± 1,09	88,22 ± 1,49	SVCL _D_5	92,25 ± 0,75	97,49 ± 0,62

8.4 Support Vector Method with a Radial Kernel

The values obtained using the Support Vector method and a radial kernel are shown in Table 7. The highest classification accuracy was 87.94% in case of static analysis and 92.38% in case of dynamic analysis.

Table 7
Success Rate of the Support Vector Model Using a Radial (rbf) Kernel

Static analysis			**Dynamic analysis**		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
SVCR_S_1	88,11 ± 0,61	96,69 ± 0,65	SVCR_D_1	91,93 ± 0,84	98,68 ± 0,50
SVCR_S_2	87,84 ± 0,67	96,61 ± 0,67	SVCR_D_2	91,84 ± 0,90	98,46 ± 0,65
SVCR_S_3	87,76 ± 0,73	96,53 ± 0,73	SVCR_D_3	91,71 ± 0,73	98,61 ± 0,57
SVCR_S_4	87,68 ± 0,78	96,94 ± 0,63	SVCR_D_4	91,66 ± 0,82	98,90 ± 0,55
SVCR_S_5	87,60 ± 0,75	96,39 ± 0,74	SVCR_D_5	91,65 ± 0,82	98,88 ± 0,51

8.5 Support Vector Method with a Polynomial Kernel

As far as static analysis is concerned, the highest achieved classification accuracy of the polynomial function kernel of the support vector method amounted to 88.48%. When using dynamic analysis, this value changed to 92.17%. The results are shown in Table 8.

Table 8
Success Rate of the Support Vector Model Using a Polynomial Kernel

Static analysis			**Dynamic analysis**		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
SVCP_S_1	88,48 ± 0,78	95,46 ± 0,66	SVCP_D_1	92,17 ± 0,79	97,60 ± 0,69
SVCP_S_2	88,38 ± 0,75	95,31 ± 0,70	SVCP_D_2	92,17 ± 0,80	97,59 ± 0,70
SVCP_S_3	88,33 ± 0,80	95,40 ± 0,81	SVCP_D_3	92,16 ± 0,84	97,33 ± 0,77
SVCP_S_4	88,31 ± 0,93	95,39 ± 0,77	SVCP_D_4	92,15 ± 0,83	97,32 ± 0,76
SVCP_S_5	88,23 ± 0,81	95,16 ± 0,69	SVCP_D_5	92,14 ± 0,76	97,60 ± 0,70

8.6 Naive Bayes Method

The naive Bayes method produced the most drastically different results, comparing static and dynamic analysis. However, neither method was able to correctly predict the occurrence of malware samples, as it is evident in Table 9. Also, in case of both analysis types, this method achieved the largest standard deviation values when using the naive Bayes classifier.

Table 9
Success Rate of the Naive Bayes Method Model

Static analysis			**Dynamic analysis**		
Parameters	**Accuracy (%)**	**Sensitivity (%)**	**Parameters**	**Accuracy (%)**	**Sensitivity (%)**
NB_S_1	42,27 ± 1,28	26,61 ± 1,66	NB_D_1	59,53 ± 1,76	48,58 ± 2,42
NB_S_2	42,22 ± 1,26	26,55 ± 1,63	NB_D_2	59,12 ± 1,51	48,00 ± 2,10
NB_S_3	42,20 ± 1,28	26,51 ± 1,67	NB_D_3	58,43 ± 1,99	47,27 ± 2,67
NB_S_4	42,08 ± 1,26	26,33 ± 1,64	NB_D_4	54,45 ± 2,38	42,40 ± 3,14
NB_S_5	42,05 ± 1,26	26,29 ± 1,64	NB_D_5	48,58 ± 2,00	34,63 ± 2,57

8.7 Summary

Table 10 compares the highest values achieved for each algorithm. Using both types of analysis (static analysis-Figure 1, dynamic analysis-Figure 2), the random forests method, the decision tree method and the support vector method achieved good results.

On the other hand, the naive Bayesian methods could not cope with the particular problem. The most efficient model herein was the random forests model in dynamic analysis, where it achieved a classification accuracy value of 95.95% with a standard deviation of only 0.58%.

Table 10
Comparison of the Best Predictions of the Algorithms Used

Static analysis			Dynamic analysis		
Algorithm	**Accuracy (%)**	**Sensitivity (%)**	**Algorithm**	**Accuracy (%)**	**Sensitivity (%)**
RF	91,32 ± 0,92	96,94 ± 0,60	RF	95,95 ± 0,58	98,08 ± 0,49
DT	89,74 ± 0,98	95,76 ± 1,08	DT	94,53 ± 0,74	96,37 ± 0,73
SVC_P	88,48 ± 0,78	95,46 ± 0,66	SVC_L	92,38 ± 0,83	97,82 ± 0,62
SVC_R	88,11 ± 0,61	96,69 ± 0,65	SVC_P	92,17 ± 0,79	97,60 ± 0,69
SVC_L	87,94 ± 1,02	95,75 ± 0,86	SVC_R	91,93 ± 0,84	98,68 ± 0,50
NB	42,27 ± 1,28	26,61 ± 1,66	NB	59,53 ± 1,76	48,58 ± 2,42

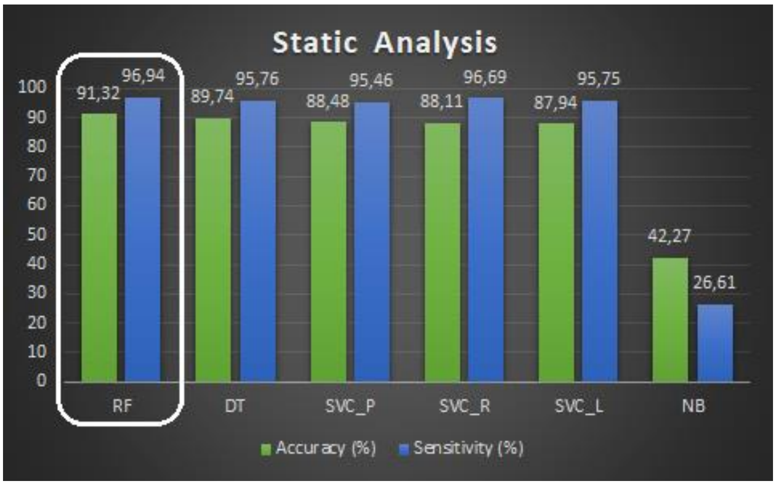

Figure 1
The Highest Values Achieved for Each Algorithm in Static Analysis

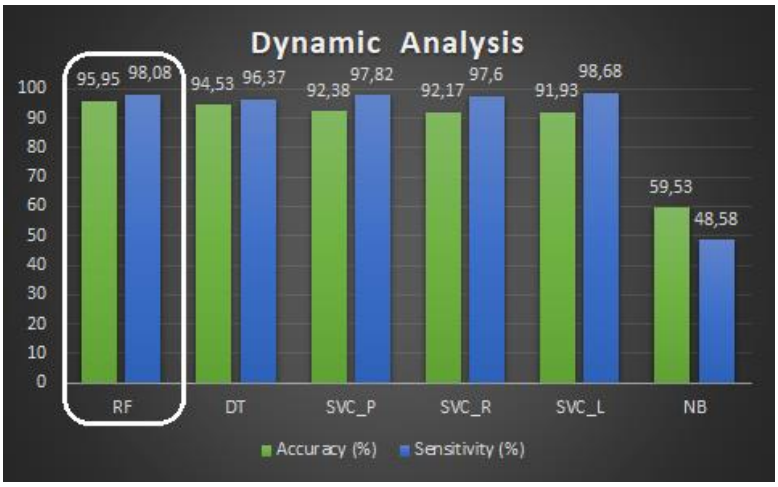

Figure 2
The Highest Values Achieved for Each Algorithm in Dynamic Analysis

9 Comparison of Studies

A comparison of the results of the best algorithms mentioned in the analysed papers and the present study are shown in Table 11. The best malware detection algorithms were found to be decision-tree-based algorithms, especially the random forest algorithm, which achieves excellent malware detection accuracy by aggregating the results of multiple decision tree classifiers for a more accurate result. The algorithm performed well on the features obtained by both static and dynamic analysis, but also in hybrid analysis, where it achieved only 1% less accuracy and sensitivity than the best algorithm, the Support Vector method. The differences in the results of the different papers may be due to the different features and also to their count, as the best accuracies were achieved in the papers sporting fewer features. This was achieved by using methods to select the most relevant flags and removing irrelevant flags that may be useless for the model. The size of the dataset and the different types of malwares in the malicious samples could also have an impact on the results.

Table 11
Comparison of the Best Algorithms from the Analysed Papers

Paper	Analysis	Data	Algorithm	Accuracy (%)	Sensitivity (%)
Bai et al.	static analysis	harmless – 8592 harmful – 10521	Random forests	99,9	99,1
Kumar et al.	static analysis	harmless – 2488 harmful – 2722	Random forests	98,78	99,0
this study	static analysis	harmless – 837 harmful – 2747	Random forests	91,32	96,94
this study	dynamic analysis	harmless – 828 harmful – 2937	Random forests	95,95	98,08
Firdausi et al.	dynamic analysis	harmless – 250 harmful – 220	J48	96,8	95,9
Shijo & Salim	hybrid analysis	harmless – 490 harmful – 997	Support vector method	98,71	98,7

Data comparison from the Table 11 represented in graph Figure 3.

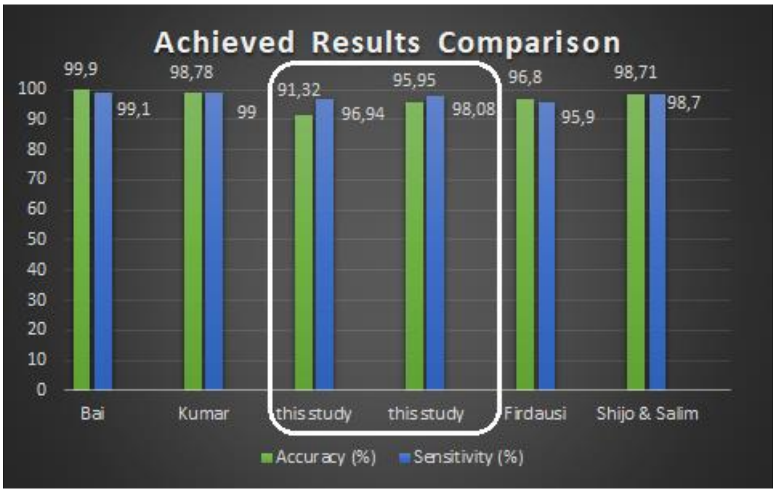

Figure 3
Comparison of the Best Algorithms from the Analysed Papers in Graph Form

Conclusion

Of all machine learning classification methods used, random forests achieved the best results in both types of analyses.

In addition to random forests, decision trees and support vector methods also achieved solid results.

The Naive Bayes methods did not prove to be able to correctly detect malware samples, compared to the other machine learning methods used.

In general, all algorithms achieved higher values of both classification accuracy and sensitivity when using the dataset created by dynamic analysis of the samples. However, by combining static and dynamic analysis and also by combining multiple trained models, the accuracy of malware detection improved.

Acknowledgement

This work was supported by the Faculty of Electrical Engineering and Informatics at the Technical University of Košice under contract No. FEI-2021-76: A modern interpreter of predicate linear logic formulas and by KEGA 002TUKE-4/2021 Implementation of Modern Methods and Education Forms in the Area of Cybersecurity towards Requirements of Labour Market.

References

[1] A. Fedák and J. Stulrajter, Fundamentals of Static Malware Analysis: Principles, Methods, and Tools., 2010, In Science and Military, pp. 45-53

[2] E. Masabo, K. Kaawaase, J. Sansa-Otim, J. Ngubiri, and D. Hanyurwimfura., 2018, In A State of the Art Survey on Polymorphic Malware Analysis and Detection Techniques

[3] J. Bai, J. Wang, and G. Zou., A Malware Detection Scheme Based on Mining Format Information., 2014, In TheScientificWorldJournal

[4] A. Kumar, K. S. Kuppusamy, and A. Gnanasekaran., A learning model to detect maliciousness of portable executable using integrated feature set., 2017, In Journal of King Saud University - Computer and Information Sciences

[5] A. Moser, Ch. Kruegel, and E. Kirda., Limits of Static Analysis for Malware Detection., 2007, In Twenty-Third Annual Computer Security Applications Conference., pp. 421-430

[6] I. Firdausi, Ch. Lim, A. Erwin, and A. Nugroho., Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection., 2010, In Advances in Computing, Control, and Telecommunication Technologies, International Conference., pp. 201-203

[7] P. V. Shijo, and A. Salim., Integrated Static and Dynamic Analysis for Malware Detection., 2015, In Procedia Computer Science

[8] N. Lutsiv, T. Maksymyuk, M. Beshley, O. Lavriv, L. Vokorokos, and J. Gazda., Deep Semisupervised Learning-Based Network Anomaly Detection., 2022, In Heterogeneous Information Systems. CMC-Computers, Materials & Continua., pp. 413-431

[9] Online repository of malware samples VirusShare.com [Online] Avaliable: https://virusshare.com

[10] Statcounter., GlobalStats., [Online] Avaliable: https://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-201001-202001

[11] PortableFreeware.com., [Online]. Avaliable: https://www.portablefreeware.com

[12] PortableApps.com., [Online]. Avaliable: https://portableapps.com

[13] N. Ádám, B. Madoš, A. Baláž, and T. Pavlik., Artificial neural network based IDS., 2017, In Proc. of the 15th International Symposium on Applied Machine Intelligence and Informatics., pp. 159-164

[14] U. Bayer, E. Kirda, and C. Kruegel, Improving the efficiency of dynamic malware analysis., 2010, In Proceedings of the 2010 ACM Symposium on Applied Computing., pp. 1871-1878

[15] Z. Dankovičová, D. Sovák, P. Drotár, and L. Vokorokos,. Machine learning approach to dysphonia detection.2018 In Applied Sciences., p. 1927

[16] B. Kang, T. Kim, H. Kwon, Y. Choi, and E. G. In., Malware Classification Method via Binary Content Comparison., 2012, In Proceedings of the 2012 ACM Research in Applied Computation Symposium., pp. 312-316

[17] A. Baláž, N. Ádám, E. Pietriková, and B. Madoš., ModSecurity IDMEF module., 2018, In Proc. of the 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 77-81

[18] D. Uppal, M. Vishakha, and V. Verma., Basic survey on Malware Analysis, Tools and Techniques., 2014, In International Journal of Computer Science & Applications., pp. 103-112

[19] H. Tamada et al., Dynamic software birthmarks based on API calls., 2006 In IEICE Transactions on Information and Systems

[20] M. Sikorski, and A. Honig., PRACTICAL MALWARE ANALYSIS., 2012, In no starch press

[21] T. K. Ho., Random decision forests. In Proceedings of 3rd International Conference on Document Analysis and Recognition., 1995, In Montreal, Quebec, Canada, pp. 278-282

[22] L. Breiman, and A. Cutler., "Random forests." Machine learning., 2014, In Netherlands, pp. 5-32

[23] H. Zhang., Exploring conditions for the optimality of naive Bayes., 2005, In International Journal of Pattern Recognition and Artificial Intelligence., pp. 183-198

[24] S. Suthaharan., Support vector machine., 2016, In Machine learning models andalgorithms for big data classification., Springer, Boston, pp. 207-235

[25] F. Pedregosa, et al., Scikit-learn: Machine Learning, In {P}ython., 2011, In Journal of Machine Learning Research., Vol. 12, pp. 2825-2830

[26] I. H. Witten, and E. Frank., Data mining practical machine learning., 2005, In Amsterdam: Morgan Kaufman, pp. 53-129