UDC [004.493.004.94]:004.8
DOI: 10.24025/2306-4412.3.2023.286374

MALWARE DETECTION MODEL BASED ON MACHINE LEARNING

Alan Nafiiev
PhD Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",
Institute of Physics and Technology, Kyiv, Ukraine
https://orcid.org/0009-0004-8604-377X
Dmytro Lande
Doctor of Technical Sciences, Professor
Institute for Information Recording of National Academy
of Sciences of Ukraine, Kyiv, Ukraine
https://orcid.org/0000-0003-3945-1178

Abstract. Every year, malware authors create more and more sophisticated and clever malware that can harm our computers. Traditional methods, which are based on searching for program signatures are no longer effective in solving the problem of malware detection. It is being replaced by automated file analysis, which is a more promising approach to detecting suspicious files. Machine learning methods are increasingly used to detect such malware programs. However, such solutions may require a lot of computing resources to perform their operations. Therefore, the task of creating an optimal machine learning model in terms of learning speed and malware detection accuracy arises. In addition, usually one method of data representation is not sufficient to detect malicious features of files. Therefore, this paper will describe two different methods: one method is based on the binary information of the file, the other one is based on disassembled code of executable files. The purpose of this work is to improve the efficiency of malware detection by optimising feature extraction methods and applying machine learning. The main tasks of the study include: extracting features from exe files, creating several machine learning models and comparing them to determine the most effective one. The dataset used in this study has been collected from various online sources and consists of 12824 executable files in .exe format, of which 11844 files are malicious and 980 are benign. This paper presents recommended methods of feature extraction and input data generation for machine learning models based on the support vector machine algorithm. These methods allow to find the best way to process the features describing a malicious file. Six machine learning models, each of which performed well in terms of F-score, precision, and recall metrics, were created. The model that was created based on the binary type of data representation showed the highest results for all metrics.

Keywords: intrusion detection, PE format, feature extraction, disassembled instructions, support vector machine.

Introduction

Every year there are more and more methods for detecting malicious software that abandon traditional signature approaches. Instead, a bet is made on heuristic analysis, behavioral methods or their combinations (Abri *et al.*, 2019). These methods have gained particular attention due to their ability to detect zero-day malware. This is software whose signature is not available to antivirus systems (Handa *et al.*, 2019; Alazab *et al.*, 2011). The use of machine learning models is the key point of these methods, but it is not without certain challenges. The main ones are the selection and formation of an optimal set of features

(features of the file) that allow to best represent the input data set. In addition, it is important to effectively adapt these features for machine learning models to improve the overall accuracy of the models and optimize the learning time. Our work is focused precisely on these aspects. In particular, in the world scientific literature there are already works that offer various solutions to these issues (Chaudhary, 2021; Kutlay *et al*., 2020; Al-Khshali *et al*., 2020). In our previous study, we have analyzed a set of .exe files with the help of different machine learning algorithms using the binary code of the file as key features (Nafiiev *et al*., 2022). We have considered the executable file in PE format, and also selected the most optimal parameters for classification algorithms. The algorithm based on support vectors turns out to be particularly effective. Therefore, in this work, we will build several machine learning models based on this algorithm. The main purpose of this research is to improve the process of detecting malware with the help of modernized feature extraction methods and the use of machine learning models. Within the framework of this study, we have identified the following tasks:
1. Formation of the input data set for the support vector machine.
2. Research and adaptation of methods for extracting features from files.
3. Construction and evaluation of several machine learning models based on extracted features.
4. Comparison of models in order to choose the most effective one for detecting malware.

1 Materials and methods

This section describes the main stages of our research. Section 1.1 covers the basics of PE format. In Subsection 1.2, we consider the formation of the input data set. Subsection 1.3 describes data representation and feature extraction methods, and Subsection 1.4 introduces the basics of the support vector machine algorithm.

1.1 Description of PE format

There is no doubt that Portable Executable (PE) file format plays a key role in the execution of software in the Windows operating system. It is universal file format for executable files, dynamic link libraries (DLLs) and drivers (Microsoft, n.d.). Structural organization of PE format, which is depicted in Fig. 1, is one of its main features. PE file consists of different sections containing different types of information. For example, PE header contains metadata about the file, such as the date of its creation, file size, version, and so on (Sikorski *et al*., 2012). Import address table indicates external functions that the program uses, and export address table indicates the functions that the program exposes for use by other programs. These tables can provide important information about the behavior of a program because they indicate the interaction of the program with other programs or with the operating system (Sikorski *et al*., 2012; Bilar, 2007). Resource sections can contain different types of data, such as icons, fonts, images, text lines, etc. These resources can be analyzed to obtain additional features for static analysis. Using n-grams of bytes or n-grams of instructions, certain patterns in binary code contained in PE files can be detected. These patterns may indicate typical behavior of a benign program or behavior specific to malware, such as exploitation of known vulnerabilities, attempts to bypass antivirus programs, or the presence of malicious content (Abdessadki *et al*., 2019). Ultimately, the analysis of PE format can provide a significant amount of information that can be used to classify a file as malicious or safe one.

DOI: 10.24025/2306-4412.3.2023.286374

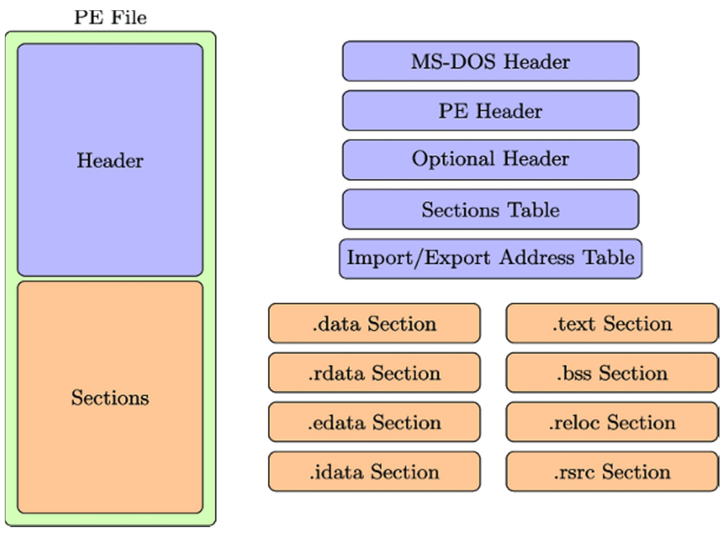


Figure 1. Structure of PE format

1.2 Formation of the input data set

The creation of the input data set is one of the most important steps in building a machine learning model. It is also necessary that such a set is well balanced to obtain the most representative results. The input data was represented by a dataset containing 11.844 malicious .exe files and 980 benign ones. Malicious files were divided into 5 different families: CryptoRansom (5856), Zbot (1973), Zeus (1413), InstallCore (731), Winwebsec (657), Mediyes (463), Zeroaccess (407), Locker (300). An infographic of the distribution of files by type can be seen in Fig. 2.

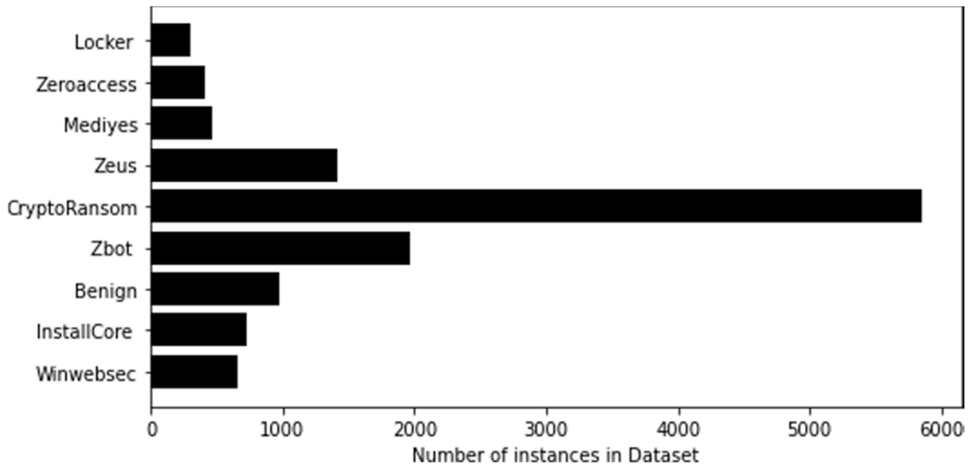


Figure 2. Infographic of distribution of files by type

It is worth noting that in the used data set there are significantly fewer safe files than malicious ones. In real system, this may be a problem, but in our case, such a distribution is acceptable, since our main task in this work is to investigate methods of feature formation. All malicious files were taken from "malicia-project.com" and "virusshare.com" websites. Benign files were taken from the folders of installed legitimate software applications of various categories. They can be downloaded at "download.cnet.com/windows".

1.3 Identification of features

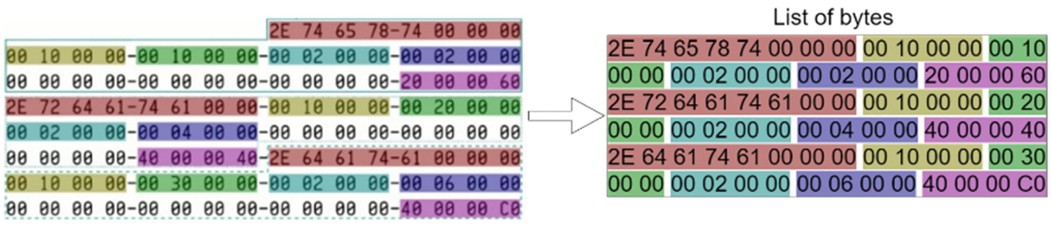
As already mentioned in this Section, PE executable files contain a lot of different information that can be used to detect malware. We can consider each file as a sequence of

© A. E. Nafiiev, D. V. Lande, 2023
DOI: 10.24025/2306-4412.3.2023.286374

bytes, a set of trace instructions, or function calls. Different types of file representation require different approaches to information extraction, selection, and preprocessing. This Section is aimed at solving these tasks.

1.3.1 Binary data representation

To generate input data for a machine learning model, you need to extract useful bit information from the executable file. To do this, each file is processed in turn using the pefile library from the Python programming language. The functions of this module allow to get the bytes of certain fields from different sections of PE-file in the form of a line. For each file from the data set, a list containing the sequence of bytes of this file is formed. A visualization of this process can be seen in Fig. 3.



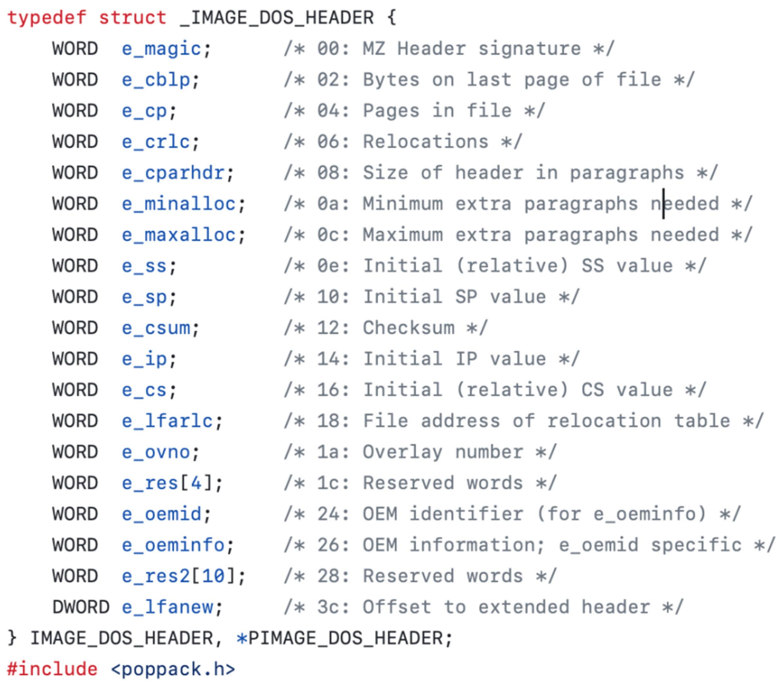
```
                                    2E 74 65 78-74 00 00 00
00 10 00 00-00 10 00 00-00 02 00 00-00 02 00 00
00 00 00 00-00 00 00 00-00 00 00 00-20 00 00 60
2E 72 64 61-74 61 00 00-00 10 00 00-00 20 00 00
00 02 00 00-00 04 00 00-00 00 00 00-00 00 00 00
00 00 00 00-40 00 00 40-2E 64 61 74-61 00 00 00
00 10 00 00-00 30 00 00-00 02 00 00-00 06 00 00
00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 C0
```

List of bytes

```
2E 74 65 78 74 00 00 00 00 10 00 00 00 10
00 00 00 02 00 00 00 02 00 00 20 00 00 60
2E 72 64 61 74 61 00 00 00 10 00 00 00 20
00 00 00 02 00 00 00 04 00 00 40 00 00 40
2E 64 61 74 61 00 00 00 00 10 00 00 00 30
00 00 00 02 00 00 00 06 00 00 40 00 00 C0
```

Figure 3. The process of forming a byte vector, which is filled with different fields of PE file structure

This study examines two models of binary data representation. The first model uses only the byte sequence from the header (purple cells in Fig. 1): DOS_HEADER, FILE_HEADER, OPTIONAL_HEADER, SECTION_HEADER, Export, and Import. The second model uses the entire header and resource section (.rsrc) as it is available in the file module (orange cells in Fig 1). Fig. 4 shows an example of the fields used for the DOS Header (The fields, 2016).



```c
typedef struct _IMAGE_DOS_HEADER {
    WORD  e_magic;      /* 00: MZ Header signature */
    WORD  e_cblp;       /* 02: Bytes on last page of file */
    WORD  e_cp;         /* 04: Pages in file */
    WORD  e_crlc;       /* 06: Relocations */
    WORD  e_cparhdr;    /* 08: Size of header in paragraphs */
    WORD  e_minalloc;   /* 0a: Minimum extra paragraphs needed */
    WORD  e_maxalloc;   /* 0c: Maximum extra paragraphs needed */
    WORD  e_ss;         /* 0e: Initial (relative) SS value */
    WORD  e_sp;         /* 10: Initial SP value */
    WORD  e_csum;       /* 12: Checksum */
    WORD  e_ip;         /* 14: Initial IP value */
    WORD  e_cs;         /* 16: Initial (relative) CS value */
    WORD  e_lfarlc;     /* 18: File address of relocation table */
    WORD  e_ovno;       /* 1a: Overlay number */
    WORD  e_res[4];     /* 1c: Reserved words */
    WORD  e_oemid;      /* 24: OEM identifier (for e_oeminfo) */
    WORD  e_oeminfo;    /* 26: OEM information; e_oemid specific */
    WORD  e_res2[10];   /* 28: Reserved words */
    DWORD e_lfanew;     /* 3c: Offset to extended header */
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
#include <poppack.h>
```

Figure 4. Example of DOS_HEADER fields from PE file structure

Data representation using n-grams was one of the first feature generation methods used to detect malware (Raff *et al.*, 2018). In this work, we will use a Markov chain, which can be represented as a graph, in which the vertices are the states of the process (all possible 256 bytes), and the edges are the transitions between the states. As a result, we get a square matrix of $P = \|p_{ij}\|$ transitions with a dimension of 256*256, where each p_{ij} cell contains the probability of transition from one byte to another. The transition matrix is subject to the following conditions:

$$p_{ij} \geq 0, \tag{1}$$

$$\forall i \sum_j p_{ij} = 1. \tag{2}$$

To build such a matrix of transitions, first, based on the collected sequence of bytes for each file, it is necessary to build an adjacency matrix of the form (00, 01, ..., ff)*(00, 01, ..., ff). In this adjacency matrix, the number of times one byte was immediately followed by another byte is counted in each cell. And only then, after the formation of adjacent matrices, transition matrices are built for all files.

At the final stage, a final matrix with dimensions 12824 (the number of all files)*256^2 is formed, in which each line corresponds to one file and includes its transition matrix converted into vector form. Each such vector contains 65536 values. A diagram of this process can be seen in Fig. 5.

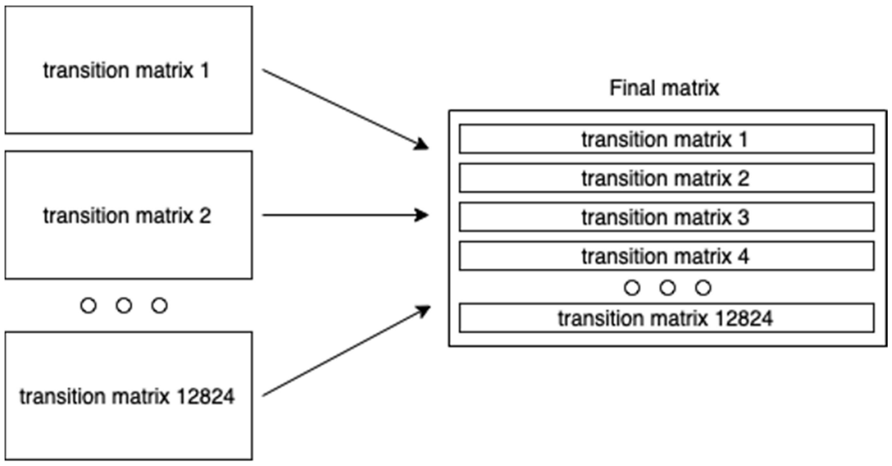


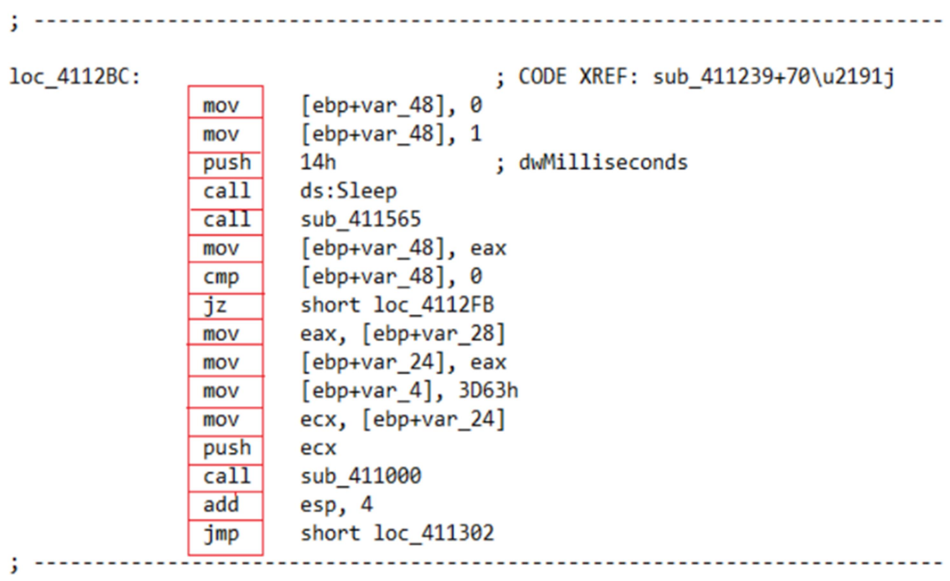
Figure 5. Diagram of formation of the final matrix

Before passing the resulting matrix to the machine learning algorithm, you must first split this matrix into two parts. One part will be used for learning the model and the other part for testing. Since our dataset does not contain a large number of different types of malicious files, it was decided to use 60% of the final matrix for learning and 40% for testing.

1.3.2 Instruction tracing

Executable instruction tracing is also used to detect malware. We generate disassembled code using IDA Pro, which is opened for each file via the command line using a python script. The program saves disassembled instructions of each element in .asm files. This is a file format in which text information is stored. An example of an .asm file is shown in Fig. 6.

DOI: 10.24025/2306-4412.3.2023.286374



```
; ---------------------------------------------------------------------------

loc_4112BC:                             ; CODE XREF: sub_411239+70\u2191j
                mov     [ebp+var_48], 0
                mov     [ebp+var_48], 1
                push    14h             ; dwMilliseconds
                call    ds:Sleep
                call    sub_411565
                mov     [ebp+var_48], eax
                cmp     [ebp+var_48], 0
                jz      short loc_4112FB
                mov     eax, [ebp+var_28]
                mov     [ebp+var_24], eax
                mov     [ebp+var_4], 3D63h
                mov     ecx, [ebp+var_24]
                push    ecx
                call    sub_411000
                add     esp, 4
                jmp     short loc_411302
; ---------------------------------------------------------------------------
```

Figure 6. A fragment of the .asm file
Note: Used instructions are marked in red

After that, instructions (push, call, cmp, mov, jz, etc.), which are assembled into a two-dimensional matrix, are sent to each file in turn. Each line of this matrix corresponds to a sequence of instructions of a specific file. Now the task is to form adjacency matrices for all files. In binary data representation, we used the number of all existing bytes as the dimension of the adjacency matrix. However, in the case of instruction tracing, we cannot use all existing instructions, as their number is counted in the thousands. And this will have a very negative effect on the learning speed of the model. Therefore, it is necessary to limit the set of instructions based on which the adjacency matrix is built. To solve this problem, it was decided to choose the instructions that appear in all files the most times in total. Four sets of instructions containing 74, 126, 187 and 272 elements were formed. Based on each such set, a machine learning model will ultimately be built.

Finally, an adjacency matrix is built for each file in the same way as in binary data representation. The four sets of instructions determine the dimensionality of two-dimensional square adjacency matrices, in which for each pair of instructions in the matrix, the number of times the first instruction followed the second is counted. Now, transition matrices are constructed using adjacency matrices. Transition matrices form the final matrix, as in Fig. 5. As in binary data representation, the final matrix is divided into two parts - learning and testing ones, and the share of the learning sample is 60% of the entire data set.

1.4 Machine learning algorithm

For file classification, we use the support vector machine (SVM) algorithm. This algorithm searches for a hyperplane in the feature space that separates the points of two classes with the maximum distance (Lifshits, 2006; Burges, 1998). The hyperplane found by the SVM is a linear combination of data instances, x_i, with weights, α_i. It is important to note that only points close to the hyperplane will have nonzero values of α. These points are called support vectors. Therefore, the goal of SVM learning is to find the weight vector α that describes the contribution of each data instance to the hyperplane. Using quadratic programming, you can effectively solve the following optimization problem:

$$\max_{\alpha}\left(\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\,\alpha_j y_i y_j \langle x_i, x_j\rangle\right). \quad (3)$$

Under the conditions:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad (4)$$

$$0 \le \alpha_i \le C, \quad (5)$$

where y_i is the label of the class of the x_i instance, $\langle\cdot,\cdot\rangle$ is the scalar product, and C is the regularization parameter that allows "soft fields". That is, some instances may fall between fields, which helps prevent data overlearning and provides better accuracy. The weight vector of the hyperplane is calculated as follows:

$$\omega = \sum_{i} \alpha_i y_i x_i. \quad (6)$$

In the current state of affairs, only linear hyperplanes are possible in the *d*-dimensional space determined by x feature vectors. Using the kernel trick, data instances can be projected into a higher-dimensional space and find a linear hyperplane in it that will be equivalent to a nonlinear hyperplane in the original *d*-dimensional space. A new optimization problem arises:

$$\max_{\alpha}\left(\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\,\alpha_j y_i y_j K(x_i, x_j)\right). \quad (7)$$

The replacement of the scalar product with $K(x_i, x_j)$ kernel function is the only difference from formula (3). In our case, a Gaussian kernel is used, where transition matrices for two instances are taken and their kernel function (kernel) is calculated according to the following formula:

$$K(x, x') = \delta^2 e^{-\frac{1}{2\lambda^2}\sum_{i,j}\left(x_{ij}-x'_{ij}\right)^2}. \quad (8)$$

Now, taking into account α found in equation (7), our solution is given by:

$$f(x) = sgn\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x_j)\right), \quad (9)$$

where class +1 (benign file) is returned if sum ≥ 0, and class -1 (malicious file) is returned if sum < 0.

2 Results and discussion

In this study, six machine learning models, four of which are trace representation type and two are binary type, were created. The indicators of the main metrics can be seen in Table 1. In this table, Binary_r is a model that includes the file resource section. Binary_wr model does not include it. Models called Trace_74 and similar ones are based on the trace type of data representation, where 74 means the number of instructions used as features. Time means the learning time of the model, measured in seconds. ROC and Precision-Recall curves can also be observed in Fig. 7. The results show that all our models have fairly close indicators. However, it is still possible to select the best model, relying on the F-score metric, which is the harmonic mean between precision and recall. Models based on binary data representation have proven to be quite effective, especially when a resource section is used. A slight advantage in all metrics was shown by the Binary_r model. However, the learning time of this model also increased slightly as more data was used. These high metrics are explained by the fact that binary code often displays file properties that are specific for malware, such as specific coding pattern, specific libraries, or command sequences. For security systems that rely on speed of response and high accuracy, models learned on binary representation, especially with the resource section, should be used. Among the models based on disassembled instructions, the model with 74 instructions (Trace_74) turned out to be the most efficient one. In addition, the learning time of this model is significantly less than that of the variants with 126, 187 and 272 features, which is a significant plus. This proves that it is not always necessary to use a large amount of learning data for high classification quality. More precisely, the correct selection of "important" instructions can give an advantage over a large volume. When developing systems for detecting malicious files based on trace instructions, we recommend focus on the quality, not the quantity, of data. This not only improves the accuracy, but also reduces the computational cost and learning time of the model. It is also worth noting that binary representation models have shown a comparable accuracy result to the best model learned on trace instructions.

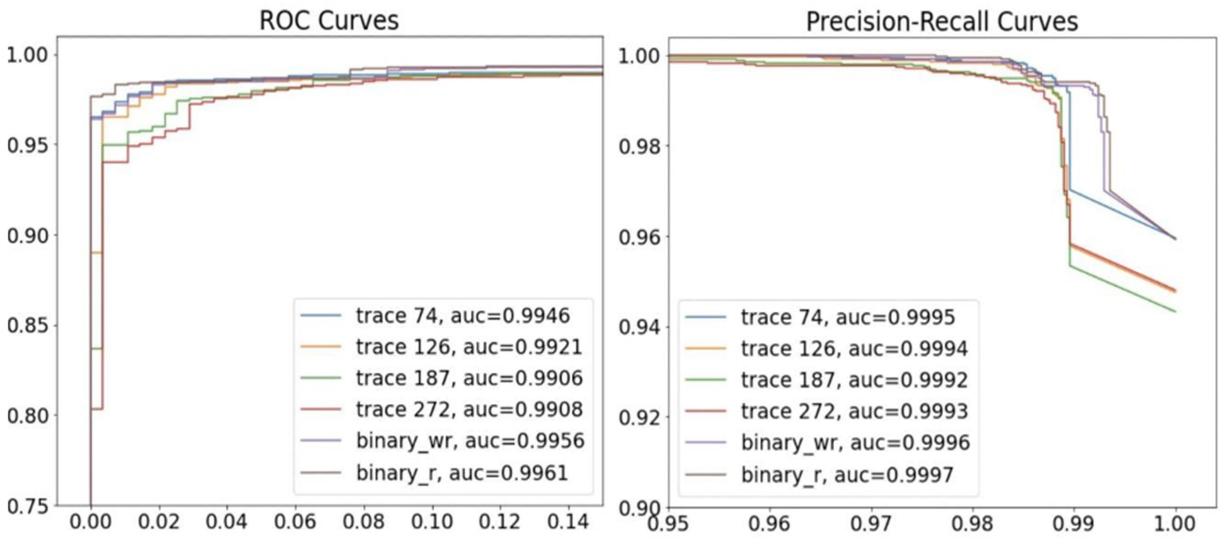


Figure 7. ROC and Precision-Recall curves

The results of our models accuracy may seem too high (Table 1), but some limitations should be taken into account. First, the data set used is not well balanced. Benign files are an order of magnitude fewer than viral ones, so the machine learning model will be better able to recognize malicious files. This is acceptable in our research, but a real file recognition system should use a balanced and carefully filtered data set. Second, there are not many different types of malicious files in the collected data set. Viruses of the same type are very similar to each other, which makes it easy for a machine learning algorithm to recognize such files. Future studies should use as many varieties of malicious files as possible to obtain a more representative result of the detection accuracy.

Table 1. Results of evaluation of machine learning models

	recall		precision		F-score		roc_auc	pr_auc	Time
	0	1	0	1	0	1			
Binary_wr	0.9130	0.9909	0.8873	0.9932	0.9000	0.9921	0.9956	0.9996	795
Binary_r	0.9239	0.9915	0.8947	0.9940	**0.9090**	**0.9928**	**0.9961**	0.9997	816
Trace_74	0.9130	0.9893	0.8689	0.9932	**0.8904**	**0.9912**	**0.9946**	0.9995	99
Trace_126	0.9094	0.9873	0.8479	0.9929	0.8776	0.9901	0.9921	0.9994	255
Trace_187	0.8695	0.9887	0.8571	0.9898	0.8633	0.9893	0.9906	0.9992	490
Trace_272	0.8369	0.9882	0.8461	0.9873	0.8415	0.9877	0.9908	0.9993	747

Conclusions

In this work, a method for detecting malicious programs based on machine learning was presented. We analyzed machine learning models that were built based on two different feature extraction approaches. The first approach used bitwise file information obtained from the PE structure using the Python pefile library. Two models were generated, one of which used the resource section of the file for learning. The second approach used executable instruction tracing. The disassembled code was generated using the IDA Pro program. 4 sets of features, each of which contained a different number of instructions (74, 126, 187 and 272) and corresponded to one machine learning model, were formed. All six models showed a good result of the accuracy of detecting malicious files. Our study demonstrates the importance of choosing a feature extraction approach when building machine learning models. Models based on the binary representation showed the best accuracy, especially the one that included the resource section. On the other hand, there was an opportunity to optimize the malware detection process through instruction tracing, especially when a limited number of instructions was used. It was found that the model learned on the least number of instructions (Trace_74) showed the highest values for all metrics. In addition, this model requires the least time for its learning, which is a significant advantage for using such a model in real file classification systems. Thus, it can be argued that the principle of using all possible data for classification does not always give the best accuracy result. Sometimes it is useful to select a small part of the entire set of features on which the learned model will show the highest accuracy result.

Acknowledgements

None.

Conflict of Interest

None.

References

Abdessadki, I., & Lazaar, S. (2019). A new classification based model for malicious pe files detection. *International Journal of Computer Network and Information Security*, 11(6), 1-9.

Abri, F., Siami-Namini, S., Khanghah, M.A., Soltani, F.M. et al. (2019). The performance of machine and deep learning classifiers in detecting zero-day vulnerabilities. arXiv:1911.09586.

Alazab, M., Venkatraman, S., Watters, P., & Alazab M. (2011). Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Australasian Data Mining Conference* (pp. 171-182).

Al-Khshali, H.H., Ilyas, M., & Ucan, O.N. (2020). Effect of pe file header features on accuracy. In *IEEE Symposium Series on Computational Intelligence.*

Bilar, D. (2007). Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics.*

Burges, C.J.C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2, 121-167.

Chaudhary, P. (2021). Pe file-based malware detection using machine learning. In *Proceedings of International Conference on Artificial Intelligence and Applications* (pp. 113-123).

Handa, A., Sharma, A., & Shukla, S.K. (2019). Machine learning in cybersecurity: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

Kutlay, A., & Karađuzović-Hadžiabdić, K. (2020). Static based classification of malicious software using machine learning methods. *Lecture Notes in Networks and Systems book series*, 83.

Lifshits, Yu. (2006). *Algorithms for internet: Support vector machines*.

Microsoft. "Portable Executable". (n.d.). Retrieved from https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

Nafiiev, A., Kholodulkin, H., & Rodionov, A. (2022). Comparative analysis of machine learning methods for detecting malicious files. *Theoretical and Applied Cybersecurity*, 3(1).

Raff, E., Zak, R. et al. (2018). An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniq*ues.

Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands on Guide to Dissecting Malicious Software.*

The fields used for the DOS Header. (2016). Retrieved from https://github.com/wine-mirror/wine/blob/master/include/winnt.h.

DOI: 10.24025/2306-4412.3.2023.286374

ISSN 2306-4412 (Print), ISSN 2306-4455 (CD-ROM), ISSN 2708-6070 (Online)

МОДЕЛЬ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

А. Е. Нафієв
Аспірант
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", фізико-технічний інститут, м. Київ, Україна
https://orcid.org/0009-0004-8604-377X
Д. В. Ланде
Доктор технічних наук, професор
Інститут проблем реєстрації інформації
Національної академії наук України, м. Київ, Україна
https://orcid.org/0000-0003-3945-1178

Анотація. З кожним роком автори шкідливого програмного забезпечення створюють все більш досконалі та хитромудрі шкідливі програми, які можуть завдати шкоди нашим комп’ютерам. Традиційні методи, які ґрунтуються на пошуку сигнатур програм, перестають бути ефективними для вирішення проблеми детекції шкідливого програмного забезпечення. На зміну приходить автоматизація аналізу файлів, яка є більш перспективним підходом для виявлення підозрілих файлів. Для виявлення таких програм все частіше використовують методи машинного навчання. Однак для виконання своїх операцій такі рішення можуть потребувати багато обчислювальних ресурсів. Тому виникає задача створення оптимальної моделі машинного навчання з погляду швидкості навчання і точності детекції шкідливого програмного забезпечення. Крім того, зазвичай одного методу представлення даних недостатньо для якісного виявлення шкідливих ознак файлів. Тому в цій роботі буде описано два різні методи: один підхід ґрунтується на бінарній інформації файлу, другий полягає у використанні трасувальних інструкцій. Мета цієї роботи – підвищення ефективності виявлення шкідливого програмного забезпечення шляхом оптимізації методів вилучення ознак та застосування машинного навчання. Основні задачі дослідження включають: вилучення ознак з exe. файлів, створення кількох моделей машинного навчання та їх порівняння для визначення найефективнішої моделі. Використаний у цьому дослідженні набір даних був зібраний з різних інтернет-джерел та складається з 12824 виконуваних файлів у форматі .exe, з яких 11844 файлів є шкідливими, а 980 – доброякісними. У статті представлено рекомендовані методи вилучення ознак та генерації вхідних даних для моделей машинного навчання на основі алгоритму машини опорних векторів. Ці методи дозволяють знайти найкращий шлях для обробки ознак, що описують шкідливий файл. Було створено шість моделей машинного навчання, кожна з яких показала високі показники метрик F-score, precision та recall. Модель, яка була створена на основі бінарного типу представлення даних, показала найвищі результати по всіх метриках.

Ключові слова: виявлення вторгнень, PE формат, вилучення ознак, дизасембльовані інструкції, машина опорних векторів.

Дата надходження: 26.08.2023
Прийнято: 12.09.2023

DOI: 10.24025/2306-4412.3.2023.286374