Date of publication DEC-31, 2024, date of current version AUG-16, 2024.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.23.4.3765

Detection of Windows Portable Executable Malware using NLP Techniques and Proxy-server

MAKSYM MISHCHENKO, MARIIA DOROSH

Department of Information Technology and Software Engineering, Chernihiv Polytechnic National University, Chernihiv, Ukraine
(e-mail: max.mishchenko771@gmail.com, mariyaya5536@gmail.com)

Corresponding author: Maksym Mishchenko (e-mail: max.mishchenko771@gmail.com)

ABSTRACT This paper aims to investigate the effectiveness of virus detection in Windows Portable Executable file using NLP, machine learning and a computer network proxy. Selected classification performance metrics are the accuracy and F1-score of the virus type classification in a specific file and the average time spent on analyzing the file. To classify viruses, a static analysis of the Optional Header Directories section in PE file is conducted. The list of imported libraries is vectorized using the word2vec model and submitted for classification by the Random Forest Classifier, Support Vector Machine and Multilayer Perceptron models. As a result, the best training mean accuracy of 94% and F1 score of 0.94 for the Random Forest Classifier model is achieved. To determine the effectiveness of virus file detection, a local area network (LAN) of three computers and a proxy server is configured. The conducted experiments on the detection of malicious files with the use of a proxy shows request time of 2.3 seconds for Support Vector Machine, 2.28 seconds for Multilayer Perceptron and 2.6 seconds for Random Forest Classifier. For reducing delay, ssdeep based cache is introduced, which reduces delay to 2.1 seconds for Random Forest Classifier and 2.15 seconds delay for Multilayer Perceptron. The proxy classification F1 score obtained on the evaluation proxy data confirmed and outperformed the F1 score obtained on the training dataset. This gives grounds for asserting the feasibility of using a proxy server and NLP techniques to detect Windows Portable Executable malware.

KEYWORDS cybersecurity; NLP; word2vec; proxy-server; machine learning; Windows Portable Executable; malware; ssdeep; LAN.

I. INTRODUCTION

IN today's world, most businesses use computer networks for their operations. The main advantages of corporate networks are the collection of corporate information in one place, ensuring the functioning of web servers, mail servers, and DNS servers, convenient search and communication between employees and, as a result, faster performance of corporate tasks by employees. The main threat to a corporate computer network is the leakage of confidential data, virus infection or hacking by third parties. For example, in the report on the security results of Cisco [1], it was indicated that 51.5% of the respondents' companies faced cyber-attacks that led to the leakage of network data; 51.1% also encountered a system or network outage; 46.7% were also victims of ransomware viruses. From the report, it can be seen that a certain percentage of companies faced several cyber attacks at the same time, which led to such consequences as interruption of IT communications (62.6%), disruption of supply chains (43%) and disruption of internal operations (41.4%). According to a report from the VirusTotal website [30], as of 2023, the most popular sources of attacks include email attachments, files such as excel, word, pdf, iso, exe, etc. Stable during 2020 - 2023, Windows Portable Executable file format, which can be EXE and DLL types, is common malware source.

A firewall or a proxy server is usually used to protect corporate computer networks from malware and other types of attacks. Firewall allows to prevent cyber threats thanks to the filtering of unwanted traffic based on set rules, analysis of anomalies, in-depth packet inspection, and more. Modern firewalls - Next Generation Firewall, are also integrated with the intrusion prevention system (IPS). Despite the advantages, the main disadvantage of firewalls is the high cost of purchase and maintenance.

An alternative to a firewall can be a proxy server, the main function of which is to redirect and analyze traffic from the client or organization to the server. Unlike a firewall, a proxy server does not have built-in features to protect against cyber threats, so the implementation of protection methods is left to a cyber security specialist. However, a proxy server can have advantages such as the ability to integrate with various custom traffic checks, client address masking, and traffic caching.

Thus, the motivation for conducting this study is to validate the possibility of using a proxy server to protect the computer network from malicious software - malware of the Windows Portable Executable type, and to determine the effectiveness of this method.

II. RELATED WORKS

Scanning network traffic, in particular files, entering corporate computer networks, is a necessary factor in ensuring the security of internal data and preventing the failure of network components. Proxy servers or firewalls with a given set of rules or signatures are usually used to filter files. A relatively new method of detecting and neutralizing malicious files is to detect the threat using machine learning methods before it enters the corporate network. Such approaches are effective in combating zero-day attacks, but are resource intensive.

In our work, we analyzed the conducted studies in which the proxy server is used to analyze network traffic and detect threats in it. For example, Mamoru M. [2] developed a method for detecting malicious traffic based on lexical analysis of campus network proxy server logs and MTA and D3M datasets [3], which are about 1Gb pcap files with different types of threats, such as Blackhole Exploit Kit, Elenore, and Mpack. As a result, the best F-measure of 0.96 was obtained and it was concluded that the developed method can detect new malicious traffic in proxy server logs.

Another example of using a proxy server is the work of Alexander Moshchuk, Tanya Bragin et al. [4], that presented a methodology for detecting malicious Web content before it reaches the user's Web browser. Detection of malicious web content was carried out by interception of web content by a proxy server and execution in a dedicated virtual machine. If the download resulted in unauthorized additional actions, the web content was considered threatening. As a result, the authors managed to develop and optimize a proxy prototype for detecting malicious web content, which on average adds 600 ms of delay per request, which is insignificant, considering the average request time.

The main difficulty in implementing network traffic analysis is the fact that the vast majority of traffic is encrypted using SSL or TLS protocols, which makes it impossible to read its content without access to the encryption keys. To analyze and modify encrypted network traffic, cyber analysts can use a MITM (Men In The Middle) approach, in which the cyber analyst creates a new SSL session for the client, and the network clients are issued SSL certificates to be used by them for https connections.

At the same time, a proxy can be used by attackers, including for MITM attacks and competitive attacks. For example, in their work [5], Carlos Novo and Ricardo Morla investigated and modeled adversarial attacks in which attackers create additional delay in Command and Control (C2) traffic by adding or delaying TCP packets on a proxy server to avoid detection a C2 attack by IDS systems. In our work, the MITM approach is used to scan files at the entrance to the computer network, in order to detect and block malicious software.

As a method of detecting malicious software files at the entrance to the network, static analysis of malware is conducted. Static analysis consists of analyzing a file without running it and monitoring the execution results. A set of imported libraries in the file is analyzed. Thus, the task of analysis is reduced to solving the problem of supervised text classification. To solve this problem, the existing techniques of natural language processing (NLP) are analyzed.

In our previous work, we used the TF-IDF statistic in combination with the n-gram method to vectorize disassembly opcodes in ELF files [7]. The vectorized opcodes were fed as input to different models for classification, resulting in a best performance of 84% accuracy and F1-score 0.84 for the stochastic gradient descent support vector model.

Among the existing architectures of neural networks for the classification of text data, the Encoder-Decoder architecture and Transformers can be distinguished. The Encoder-Decoder architecture was created on the basis of the Recurrent Neural Network architecture and is used to perform machine translation [8], sentiment analysis of text [9] and image to textual description [10]. The transformers architecture was proposed in 2017 by Ashish Vaswani, Noam Shazeer et al. and showed better results compared to encoder-decoder models: less time for training and more parallelization in solving the problem of machine translation of text [11]. The architecture of transformers has also gained popularity in research aimed at analyzing and classifying malicious software, especially static file analysis. For example, Khan S. and Nauman M in their study [24] created a model based on the transformers architecture, which was able to achieve an accuracy of 90% to 97% in the classification of various types of malicious Windows PE files. In their work authors used transformers to process PE file opcodes, in our work on other hand only import table section from Optional Headers of PE file is being used.

The Bidirectional Encoder Representations from Transformers (BERT) model was built on the basis of the transformers architecture. BERT receives a sequence of digital representations of tokens or words and generates a corresponding sequence of semantically encoded vectors representing the text. The model uses the Masked language model (MLM), which consists in predicting randomly masked words based on the context of the document. BERT can be used not only for text vectorization, but also for solving NLP tasks in general, provided that suitable final neural layers are added to the model. A special feature of BERT is that it can accept texts with a length of no more than 512 characters [12].

As an improvement of the model, RoBERTa [13] and ELECTRA [14] were developed. In the RoBERTa model, a modified MLM was presented, consisting in a dynamic change of the masking pattern applied to the input text. In the ELECTRA model, MLM was modified so that the words were not hidden, but replaced by synthetically generated alternatives. Thus, the task of the model is not to predict omitted words, but substituted words.

A significant breakthrough was the construction of the Generative Pre-trained Transformer (GPT) model [25], whose authors proposed an approach to training on a large volume of unlabeled data, in which fine-tuning of the model is performed separately for each task. This made it possible to obtain an effective transfer of knowledge without significant restructuring of the model.

In his study [15] Jan Sawicki et al. made a comparative review of many modern NLP techniques. They mentioned that statistical approaches, such as bag of words, require a carefully cleaned text to be entered. More modern word embedding techniques, on the other hand, can work with raw original text. One of the word embedding techniques is the word2vec technique.

The word2vec technique was proposed by Tomas Mikolov et al. in [16] and [17] as the latest architecture for word vectorization as an alternative to the already existing Neural Net Language Models and n-gram models. Within word2vec, 2 architectures were proposed: continuous bag of words model (CBOW) and continuous skip-gram model. The results of the comparison of the created models showed an advantage in accuracy over the already existing Neural Network Language Models (NNLM) and Recurrent Neural Network Language Models (RNNLM).

The essence of the CBOW method is to use the context surrounding the word to predict the word in the middle of the context. The skip-gram model, on the other hand, predicts surrounding words based on an existing word. Given a sequence of words, the goal of the Skip-gram model is to maximize the mean log-likelihood of surrounding words. So the cost function goal will be minimizing a negative log-likelihood, which is the same, as maximizing the positive log-likelihood:

$$J_0 = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0} \log p(w_{t+j} \mid w_t), \qquad (1)$$

where J_0 – the cost function for skip-gram model, T – the corpus size, C - the size of the training context (the larger the context, the more instances for training, leading to increased accuracy), w_t – the center word, $\log p(w_{t+j}|w_t)$ – log likelihood of occurring surrounding word w_{t+j} given center word w_t.

As a result of analyzing existing methods of classification malware WPE files, it can be concluded, that proposed method for detection of Windows Portable Executable malware using NLP techniques differs from existing ones by using:

- "Import Table" data directories from "Optional headers" section of WPE file instead of opcodes;
- word2vec models for vectorization of "Import Table" of WPE file, instead of transformer models or TF-IDF methods;
- proxy-server for malware detection.

As presented in Section 4 of this paper, our method shows improvement in F1 score and time of WPE malware classification over some of the existing methods.

III. MATERIAL AND METHODS

A. PE FILE STATIC ANALYSIS APPROACH

In this work, attention is focused on the classification of PE files - the most common format of executable files for systems on the Windows operating system. PE files consist of several structures, the main ones are MS-DOS stub, PE signature, COFF file header, Optional header and Section header [6]. A typical structure of a PE file is shown in Figure 1. Optional Header Directories, a structure that stores imported and exported libraries and functions used to execute a PE file, were chosen for the static analysis of the malware. Import Table will be used for this work.

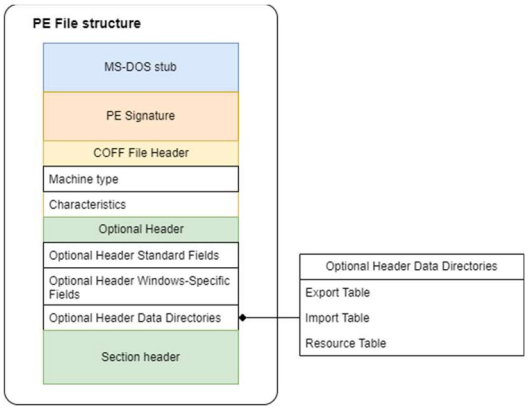

Figure 1. Illustrative PE file structure

Considering the fact that this section may be missing in some files, the system will mark such files as potentially dangerous during operation. Such files will not be involved in training process. In the training, test and validation dataset, all files have Import Table section. In the input set of files, a sequence of functions imported by PE files is taken for analysis. For each function, a word of the form is generated:

$$w = library_i \cap function_{ij}, \qquad (2)$$

where $library_i$ - library, that is imported in Import table of PE file, $function_{ij}$ - function, that belongs to $library_i$ and is used in PE file. In Figure 2, an example of parsed and processed Import Table section for PE file is presented that is marked as Trojan.

KERNEL32.DLL_GetProcAddress KERNEL32.DLL_GetModuleHandleA
KERNEL32.DLL_VirtualQuery ADVAPI32.DLL_RegEnumKeyExA
ADVAPI32.DLL_RegCreateKeyExA ADVAPI32.DLL_RegOpenKeyExA
ADVAPI32.DLL_RegSetValueExA ADVAPI32.DLL_RegDeleteValueA
ADVAPI32.DLL_RegCloseKey ADVAPI32.DLL_RegFlushKey
ADVAPI32.DLL_RegQueryValueExA ADVAPI32.DLL_RegEnumValueA
ADVAPI32.DLL_RegDeleteKeyA ADVAPI32.DLL_RegQueryInfoKeyA
ADVAPI32.DLL_RegOpenKeyExA ADVAPI32.DLL_RegCloseKey
ADVAPI32.DLL_RegQueryValueExA COMCTL32.DLL_ImageList_GetIconSize
COMCTL32.DLL_ImageList_Draw COMCTL32.DLL_ImageList_Remove
COMCTL32.DLL_ImageList_SetIconSize COMCTL32.DLL_ImageList_Add
COMCTL32.DLL_InitCommonControls COMCTL32.DLL_ImageList_DragLeave

Figure 2. Truncated text with parsed Optional Header Directories of Trojan PE file

Thus, the input dataset is a set of text documents - a sequence of imported libraries for each file. Each of these documents should be classified by the type of virus the file carries, or not a virus if the file is safe. The NLP techniques, described in the second section of this paper, are analyzed to vectorize the input set of documents. The TF-IDF and n-gram statistics had already been used in our previous research, so it was decided to use another NLP model to compare the results and find a better solution. As a result, it was decided to use the word2vec technique based on the skip-gram model for vectorization of the parsed PE files. The word2vec model uses shallow neural networks, which speeds up model training and requires fewer resources to perform real-time file classification. In our case, when resources for detecting malicious files are limited, this is one of the key factors for choosing a model.

B. CLASSIFICATION MODEL SELECTION

To perform file classification, vectorized text is passed as an input to machine learning models. Support Vector Machine, Random Forest and Multilayer Perceptron models are compared to obtain the best result.

The Support Vector Machine method consists in constructing a hyperplane to separate points of different classes in a multidimensional space. The method can be divided into 2 approaches: linear separation and nonlinear separation. With linear separation, the input data does not undergo any transformations – the classes can be immediately separated by constructing a hyperplane. A non-linear approach that uses kernel functions to construct a class partition consists in transforming the input data into a space with a higher dimension, which makes it easier to distinguish similar data that belong to different classes. In our work, we use the most popular Radial Basis Function (RBF) kernel function for constructing hyperplanes:

$$k(x,z) = e^{-\gamma\|x-z\|^2}, \quad (3)$$

where $\gamma > 0$ - a parameter that controls the influence of each instance from the sample on the decision boundary, x, z - sample instances [19]. The greater the Euclidean distance between sample instances, the closer the function value is to zero. This means that such instances most likely belong to different classes.

The Random Forest algorithm is a machine learning method that consists in creating a set of decision trees during training. In the case of classification, the element class is determined by the most frequent result of training the decision tree. Random Forest is often used to classify malicious files using the static textual information of the file. For example, Trung Kien Tran and Hiroshi Sato in their work [18], used a vectorized command API set as input data for a Random Forest model, obtaining the best accuracy of 98% for a 2-class dataset and 95% for a 4-class dataset. In our work, a model with the following hyperparameters is used:

- number_of_trees = 100 – amount of trees in the forest;
- criterion = 'Gini' – Gini Impurity criterion, which is function that measures amount of impurity in the split. The closer value to 0, means less impurity and more quality of the split:

$$Gini = 1 - \sum_{i=1}^{n} p(i)^2, \quad (4)$$

where *p(i)* probability of samples in node, belonging to class *i*; n – amount of classes.

The Multilayer Perceptron algorithm is a feed forward neural network with various possible activation functions. In our work, the ReLU activation function is used:

$$f(x) = max(0,x), \quad (5)$$

where x- input signal. In our work, a multilayer perceptron model with first hidden layer of 100 neurons, second hidden layer of 50 neurons, and third hidden layer with 25 neurons is built.

As an optimization method for Multilayer Perceptron, the Adam algorithm [20] is used, which consists in calculating the learning rate for each weight of the neural network and adjusting it during training.

To evaluate and compare the classification results of the models, the Receiver Operating Characteristics curve (ROC), that is, a graph showing the efficiency of the classification model at all classification thresholds is used. The curve shows the ratio of True Positive to False Positive - the closer this value is to one, the better the model performs.

C. DATASET CREATION

Much attention was paid to the selection and processing of the dataset for training and validation of the PE file classification model. Datasets such as Mal-API-2019 [21], BODMAS [22] and VirusShare [26] were analyzed. Among the analyzed datasets, the VirusShare dataset was selected, which represents static information about malicious files obtained from the VirusShare website. Static information about Windows PE files was generated using the manalyze program [27].

The dataset includes 3 types of malicious EXE and DLL files and a separate type of benign files:

- **Trojans** - viruses that are disguised as bening software. Different types of ransomware and spyware. According to Microsoft worldwide top threat statistics, on May 2024, 2 out of 5 top threats were types of Trojan. [23].
- **Adware** – viruses that are installed in obscured manner and replace search engine results, add advertisement banners to the desktop of OS, etc. Also known as PUA – potentially unwanted application.
- **Worms** - viruses that infect a system and spread to another computer through a network. Worms can encrypt or remove files, sending spam, or can start DDoS attacks, etc.
- **Benign** - not a virus program. It is represented by 4328 examples, that are fetched from "Windows" directory and "Program Files" directory on clean Windows 7 and Windows 10 setups.

The amount of types of malicious and safe software that are used to train machine learning models is shown in Table 1. The percentage distribution of types is shown in Figure 3. Data was splitted - for training we used 80% of dataset, for validation – 20% of dataset.

Table 1. Amount of each virus type in input dataset.

Virus type	Amount in the dataset
Trojan	4628
Adware	4031
Worm	3875
Benign (not a virus)	4328

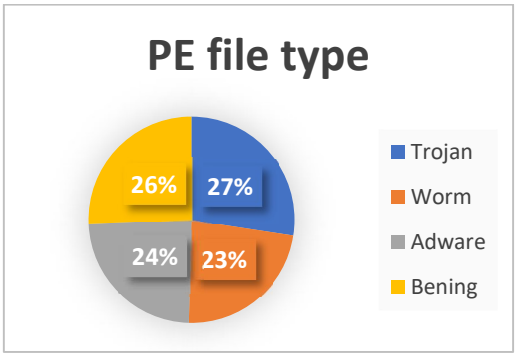

Figure 3. Pie chart with percentage of each type of PE files presented in dataset

D. PROXY SETUP

As an environment for deployment and retraining machine learning models, a server application was created in Python, which was deployed on the same server with a proxy server. The application is launched before the start of the proxy server, trains the model on the training data, and serializes it in the pkl format for further use in the proxy server. This has an advantage over using classic ML-OPS environments, where additional latency is added to model training and inference process due to the need to make API requests or make other network connections.

An opaque proxy server for malware detection was written in Python using the mitmproxy library [28]. The proxy server is deployed on a separate machine with Linux OS, intercepts traffic entering the network, extracts and sends PE files for classification to the trained model. Parsing of the import table PE file is performed on the side of the proxy server using the pefile library [29].

Receiving the prediction result from the model, the proxy server writes the malware type in the headers of the response to the intercepted request and blocks the file in case the file turns out to be malicious. Requests and responses that go from the network and to the network through the proxy server and do not contain PE files are not checked.

All devices in the network are connected to the proxy server. To do this, the IP address of the machine, running the proxy server, and its port are specified in the Internet options of each of the devices on the network as proxy server settings. A trusted SSL certificate signed by the mitmproxy library is installed on each device. This makes it possible to decrypt and analyze the content of protected network traffic on the proxy server. The experimental environment designed for the study is depicted in Figure 4.

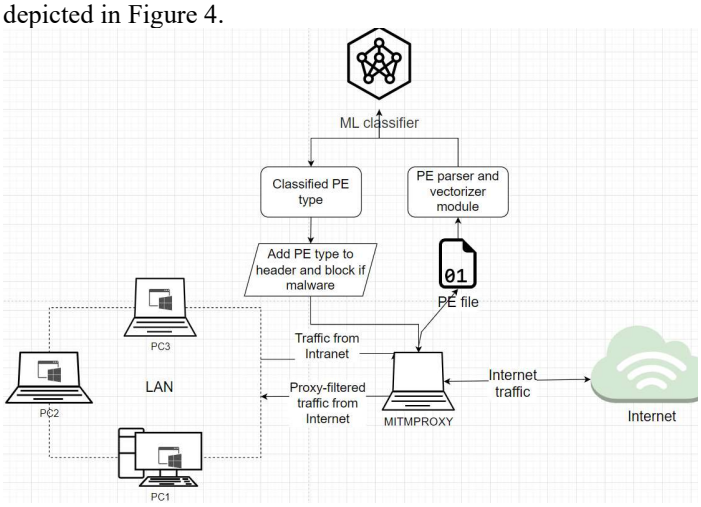

Figure 4 – Diagram of created experimental environment for detection and classification of Windows PE malware in proxy-server

IV. RESULTS

A. CLASSIFICATION MODEL EVALUATION

We used following metrics for evaluation of classification performance of PE files:

1. Precision (formula 6) – measures the part of the real true positives among all positives classified.
2. Recall (formula 7) – measures the part of true positives among false negatives and true positives.
3. F1-score (formula 8) – harmonic mean of precision and recall.
4. Accuracy (formula 9) – mean accuracy for all predictions made by a model.

$$\text{Precision(class=a)} = \frac{TP(class=a)}{TP(class=a)+FP(class=a)}, \quad (6)$$

where TP – true positive, FN – false negative.

$$\text{Recall(class=a)} = \frac{TP(class=a)}{TP(class=a)+FN(class=a)}, \quad (7)$$

where TP – true positive, FN – false negative.

$$F1(class=a) = \frac{2\times\text{Precision(class=a)}\times\text{Recall(class=a)}}{\text{Precision(class=a)}+\text{Recall(class=a)}}, \quad (8)$$

where Precision (class=a) – precision for class a, Recall(class=a) – recall for class a.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (9)$$

where TP – true positives, TN – true negatives, FP – false positives, FN – false negatives.

To evaluate overall performance across all classes, macro average and weighted average values for mentioned metrics are calculated. Macro average – the average value of the metric value for each class is calculated for precision, recall and F1-score. Weighted average - the average value of the metric value for each class multiplied by proportion of true instances for the class is computed for precision, recall and F1-score.

A Receiver Operating Curve (ROC) is constructed to evaluate the learning quality of the models. The curve builds the dependence of two parameters: True positive rate (TPR) and False positive rate (FPR).

$$TPR = \frac{TP}{TP+FN}, \quad (10)$$

where TP - true positive, FN - false negative.

$$FPR = \frac{FP}{FP+TN}, \quad (11)$$

where FP - false positive, TN - true negative.

To adapt the curve for the case of non-binary classification, we use the OVR technique (One-vs-Rest), in which a curve is constructed for each class, taking into account that a correctly classified specific class is positive, and all other classes are negative. Also, additional curves are plotted: micro ROC curve, for which TPR and FPR are calculated using the sum of all TP, FP, TN, FN for all classes; macro ROC curve, for which TPR and FPR are calculated separately for each of the classes and divided by the number of classes.

To avoid overfitting, the sample is divided into training and validation in the proportion of 80% to 20%. For each estimator, cross-validation is performed on different hyperparameter values and the best performing hyperparameters are selected.

Cross-validation tuning of regularization parameter C on 5 train-test splits is conducted for Support Vector Machine, which shows the best mean cv test score of 0.926 for *C=600*.

Results of tuning are presented in Table 2. As a result, the following hyperparameters are chosen for the Support Vector Machine model: *C=600*, *kernel='rbf'* - radial basis function.

Table 2. Cross validation tuning of regularization parameter C for Support Vector Machine model

Value of parameter *C*	Mean cv test score (num_splits=5)	Mean fit time (seconds)	Mean score time (seconds)
0.1	0.838	77.66	4.7
0.5	0.873	31.71	2.12
1	0.876	26.83	2.07
4	0.903	22.14	1.56
10	0.912	16.07	1.21
50	0.92	15.72	1.09
100	0.922	17.28	1.14
500	0.925	24.56	1.18
600	**0.926**	**30.39**	**1.29**
1000	0.925	28.07	1.13
2000	0.924	31.22	1.17

The results of the SVM model evaluation are shown in Table 3. From the table, we can see that the model classified adware the most accurately – 0.97 precision, 0.9 F1-score. The trojan was classified the least accurately – 0.87 precision, 0.88 F1-score. Macro precision was 0.93, weighted precision was also 0.93, macro score and weighted F1-score were both 0.93. ROC curves, plotted for Support Vector Machine, which are displayed in Figure 5, shows high area under curve (AUC) – from 0.97 for Benign ROC and up to 1.00 for Adware ROC. It means that SVM model is capable of correctly classifying each class.

Table 3. Resulted metrics of classification malware and benign PE files using Support Vector Machine classifier.

	Precision	Recall	F1-score
adware	0.97	0.97	0.97
benign	0.96	0.98	0.97
trojan	0.87	0.89	0.88
worm	0.92	0.87	0.90
Macro average	**0.93**	**0.93**	**0.93**
Weighted average	**0.93**	**0.93**	**0.93**
Mean accuracy, %	**93**		
Average WPE file classification time, s	**0.03**		

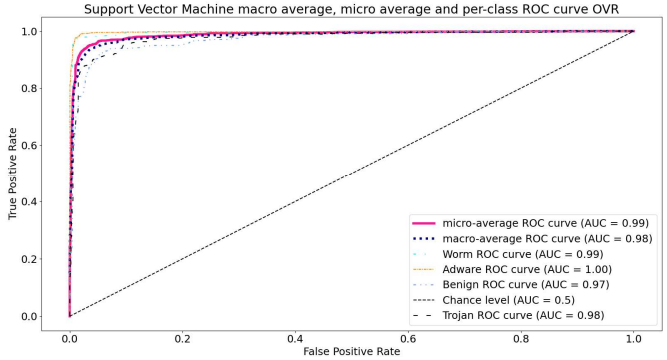

Figure 5. ROC curves for Support vector machine classification of PE files

Next, a classifier based on the Random Forest algorithm was created and validated. The results of model training are presented in Table 4.

Table 4. Resulted metrics of classification malware and benign PE files using Random Forest classifier.

	Precision	Recall	F1-score
adware	0.99	0.97	0.98
benign	0.96	0.98	0.97
trojan	0.88	0.90	0.89
worm	0.92	0.89	0.90
Macro average	**0.94**	**0.94**	**0.94**
Weighted average	**0.94**	**0.94**	**0.94**
Mean accuracy, %	**94**		
Average WPE file classification time, s	**0.04**		

The Random Forest algorithm showed average accuracy - 94%, macro F1-score - 0.94, weighted F1-score - 0.94. Adware threats were classified best – 0.99 precision, 0.97 recall. ROC curves for the Random Forest Classifier algorithm are shown in Figure 6. After analyzing the ROC curves for the Random Forest algorithm, we can conclude that the ability of the model to correctly classify PE files is confirmed by the AUC value for micro ROC – 0.99, macro ROC – 0.99 and for the ROC of each individual class.

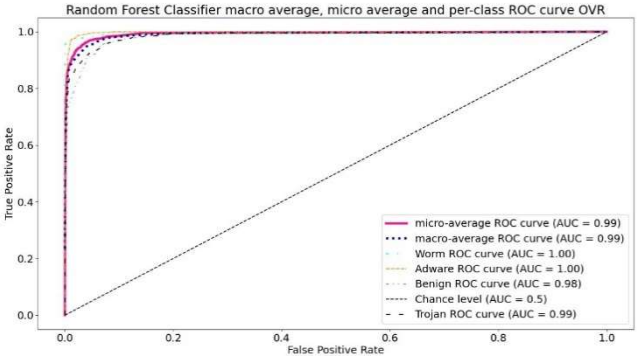

Figure 6. ROC curves for Random Forest classification of PE files

The third analyzed model is the Multi Layer Perceptron, which uses the ReLU activation function and the adam optimizer of the best model weights. The results of model training are presented in Table 5.

Table 5. Resulted metrics of classification malware and benign PE files using Multilayer Perceptron classifier.

	Precision	Recall	F1-score
adware	0.99	0.96	0.97
benign	0.96	0.98	0.97
trojan	0.83	0.86	0.86
worm	0.91	0.85	0.88
Macro average	**0.92**	**0.92**	**0.92**
Weighted average	**0.92**	**0.92**	**0.92**
Mean accuracy, %	**92**		
Average WPE file classification time, s	**0.1**		

As we can see from Table 5, the Multilayer Perceptron model did the best job of classifying adware files - 99% accuracy, 96% recall, F1 score 0.97, the average accuracy of the model is 92%. Macro and weighted average of F1-score for the model are both 0.92. ROC curves for the Multilayer Perceptron model are shown in Figure 7.

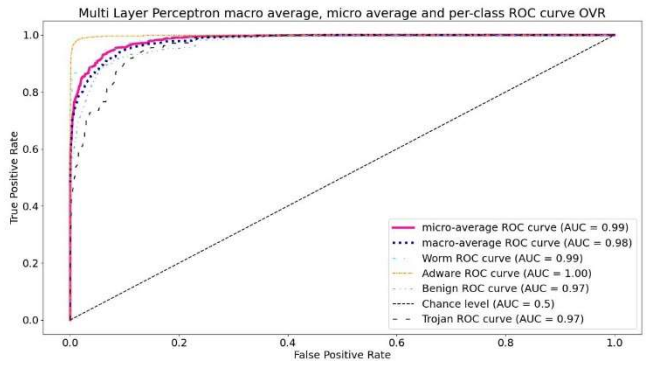

Figure 7. ROC curves for Multilayer Perceptron classification of PE files

Each of the models showed lower F1-score for Trojan-type viruses than for other types of viruses. This may be due to the fact that the Trojan family includes many different sub-types of threats – such as ransomware, spyware, etc. Accordingly, malicious PE Trojan files may have more differences than malicious files of other types, making it more difficult for the models to draw generalization conclusions.

The comparison of the proposed models with existing models for classification WPE files, suggested in works [31-33], are displayed in Table 6.

We compared our results with the works that aim to classify WPE malware using different approaches and data. In work of Lad Sumit et. al. [31], DNN model on the EMBER 2018 dataset with 5 types of features was used: general file information (virtual size, number of imported and exported functions, etc.), header information, imported functions, exported functions, section information. Authors achieved 94.09% of accuracy and 88.66 F1-score. Ye et. al. [32] used Chi-square to classify malicious WPE files by their API calls sequences and achieved 67.5% accuracy and 0.09 seconds of inference time. Koçak, Aynur et al. [33] used IBk algorithms to detect WPE malware by analyzing WPE activity network packets and achieved accuracy of 90.47%, F1-score of 90.4 and average inference time of 0.05 seconds per sample. It can be concluded that our word2vec and Random Forest WPE classification method shows improvements over existing analyzed methods in accuracy, F1-score and inference time.

Table 6. Comparison of created WPE classification model performance with existing models

Method	Input	Accuracy	F1-score	Inference time, seconds
Lad, Sumit & Adamuthe, Amol [31] (DNN)	EMBER 2018 dataset	94.09	88.66	-
Ye et al. [32] (Chi-square)	API calls	67.5	-	0.09
Koçak, Aynur et al. [33] (IBk algorithm)	WPE activity network packets	90.47	90.4	0.05
Our model (word2vec and Random forest classifier)	WPE import table	**94**	**94**	**0.04**

B. PROXY PERFORMANCE EVALUATION

To evaluate the effectiveness of detection of malicious files using a proxy server, an environment was created simulating the operation of a small network and the proxy server located at the entrance to this network. For this purpose, 3 machines running on Windows 10 operating system within one local area network (LAN) were started. In the same LAN we also started one machine running on Linux Ubuntu, on which the proxy server was running. Support Vector Machine, Random Forest Classifier, and Multilayer Perceptron models were trained and deployed on the proxy machine for malware classification.

The dataset for evaluating the effectiveness of the developed environment is 1646 files, 3 batches of 550, 550 and 546 files, respectively, the average file size is about 1 MB, each file is marked with the type of virus or the absence of a virus. Virus types are distributed not evenly: for the first batch Trojan – 27, Worm – 111, Adware – 100, Benign – 312; for the second batch: Trojan – 42, Worm – 119, Adware – 101, Benign 288; for the third batch: Trojan – 31, Worm – 130, Adware – 99, Benign – 286. Benign files are represented with the largest amount of samples in each batch, which is close to real life. All three batches are located in the S3 bucket. Each of the machines in the experiment made a request to its batch. Internet speed was 100 Mbit/s. On each machine, the address and port of the proxy server were configured and SSL certificates were installed, allowing the proxy server to decrypt HTTPS requests. Scripts were simultaneously launched on the machines that iteratively download the file from the S3 bucket, the request was redirected through the proxy server, the proxy server made inference with the trained model to detect the virus type and returned the predicted virus type in the response headers. All performance metrics were calculated as an arithmetic average for all requests from all machines. To determine the delay time added by the virus classification module, requests were made without redirection to the proxy server and inference with the ML model - we received an arithmetic average time per request of 1.4 seconds.

For improving request time with proxy server we decided to adapt cache mechanism, using ssdeep fuzzy hashing algorithm - context-piecewise triggered hashing. This algorithm allows us to compare and measure similarity in percent of two files. In our case, unvectorized, sanitized strings with imported libraries of each input file from test batch are compared before making classification. If we find file in cache with the similarity more or equal than defined threshold, we will output predicted type from cache and set cache flag in response to 1. Otherwise we make classification with model, save ssdeep hash with the predicted type to cache and output predicted type, setting cache flag to 0. We have conducted experiments with 85%, 90% and 95% similarity thresholds to investigate the impact on resulted F1 score and request delay. Results of conducted experiments are shown in Table 7. A plot with the experiment results is presented in Figure 8.

Table 7. Results of virus classification in proxy server

	Random Forest Classifier	Support Vector Machine	Multilayer Perceptron
Average accuracy, %	**98**	**97**	**98**
F1-score	**0.98**	**0.97**	**0.98**
Average request time with proxy, s	**2.6**	**2.3**	**2.28**
Request delay, s	**1.2**	**0.9**	**0.88**
Average request time with proxy and ssdeep cache (85% similarity), s	**2.4**	**2.3**	**2.11**
F1-score (ssdeep cache, 85% similarity)	**0.98**	**0.97**	**0.98**
Cache hits (ssdeep cache, 85% similarity), %		**39%**	
Request delay (ssdeep cache, 85% similarity), s	**1.2**	**0.9**	**0.71**
Average request time with proxy and ssdeep cache (90% similarity), s	**2.3**	**2.3**	**2.21**
F1-score with ssdeep cache (90% similarity)	**0.98**	**0.97**	**0.98**
Cache hits (ssdeep cache, 90% similarity), %		**37%**	
Request delay (ssdeep cache, 90% similarity), s	**0.9**	**0.9**	**0.81**
Average request time with proxy and ssdeep cache (95% similarity), s	**2.1**	**2.3**	**2.15**
F1-score with ssdeep cache (95% similarity)	**0.98**	**0.97**	**0.98**
Cache hits (ssdeep cache, 95% similarity), %		**35%**	
Request delay (ssdeep cache, 95% similarity), s	**0.7**	**0.9**	**0.75**
Average file size, Mb		**1**	
Average request time without proxy, s		**1.4**	

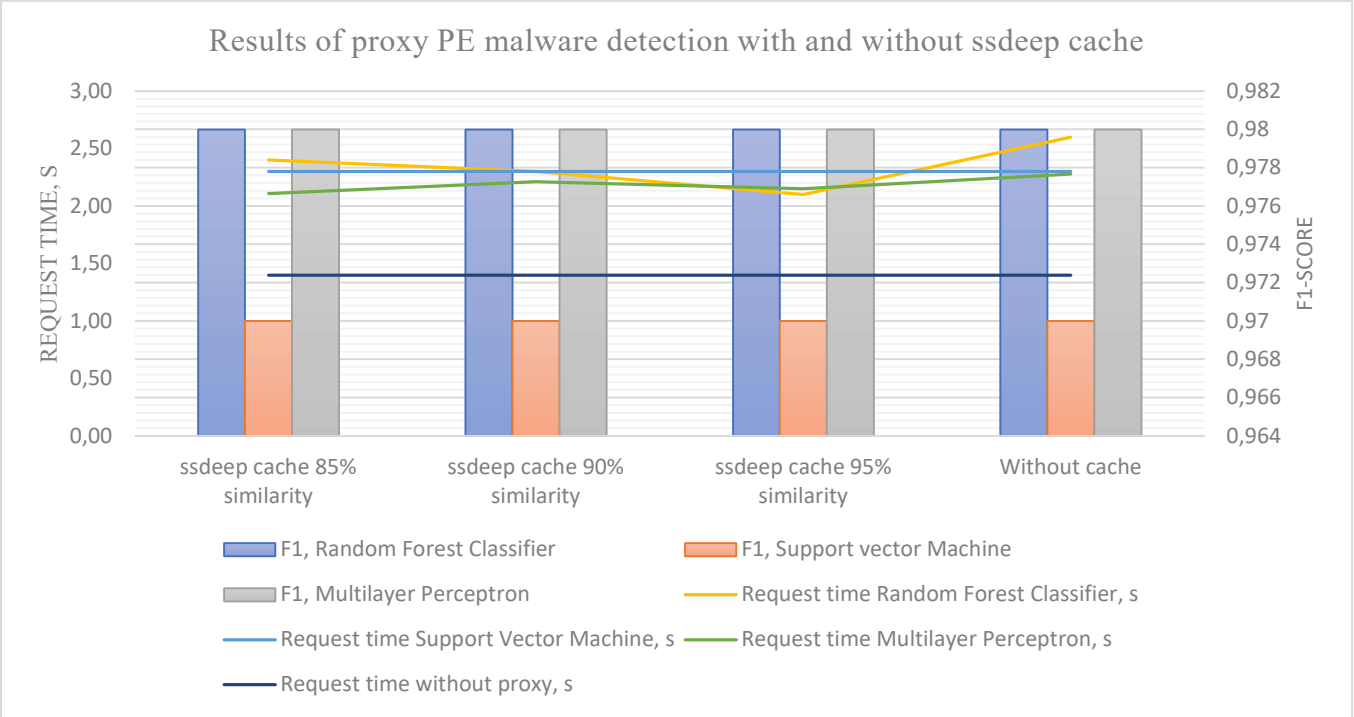

Figure 8. Results of proxy PE malware detection with and without ssdeep cache

As we can see, the best accuracy and F1-score were obtained for Random Forest Classifier and Multilayer Perceptron. Average time of request for Random Forest Classifier is 2.6 seconds, for Multilayer Perceptron – 2.28 seconds, which is not much bigger than average time of request without proxy-server – 1.4 seconds. To summarize, average time delay for classification with Random Forest is 2.6 seconds – 1.2 seconds slower compared to original time of request; for Support Vector Machine is 2.3 seconds – 0.9 seconds slower compared to original time of request; for Multilayer Perceptron is 2.28 seconds – 0.88 seconds slower compared to original time of request.

Introducing of ssdeep cache mechanism insignificantly improved request time delay. With the threshold of 85% ssdeep

similarity, we achieved 2.4 seconds request time for Random Forest Classifier compared to 2.6 seconds request time without caching mechanism. Caching mechanism does not affect F1-scores of predictions.

V. CONCLUSIONS

This study shows that a proxy server and machine learning models can be used to detect and neutralize malicious PE files before they enter a small local area network. The approach with vectorization of the import table section in Windows PE files, using the word2vec NLP model, and subsequent classification of files in the dataset, showed a minimum classification accuracy of 92% and F1-score of 0.92 for the support vector model, 92% accuracy and 0.92 F1-score for the multilayer perceptron, 94% accuracy and 0.94 F1-score for the Random Forest. This proves the feasibility of using the chosen approach to detect malicious PE files. Measurements of the effectiveness of the classification of malicious PE files in the proxy server showed a time delay for proxy requests (the difference between the request time without a proxy and the request time with a proxy): 1.2 seconds for the Random Forest Classifier model, 0.9 seconds for the Support Vector Machine model, and 0.88 seconds for the Multilayer Perceptron model. We decided to use ssdeep-based similarity cache to improve obtained request time. Results show that the best improvement was gained for 95% similarity threshold – 2.1 seconds compared to 2.6 seconds request time for Random Forest Classifier and 2.15 seconds compared to 2.28 seconds delay for Multilayer Perceptron. The classification F1 score obtained on the proxy evaluation data confirmed and outperformed the F1 score obtained on the training dataset. This proves that the developed module, consisting of a proxy server and machine learning models, shows sufficiently high accuracy in the classification of malicious PE files and does not significantly slow down the operation of a local area network.

To investigate the use of a proxy server with networks of a larger scale, future experiments are planned with the use of a larger number of machines. To improve accuracy results, it is planned to add real-time training of trained models with new files, using open virus databases.

References

[1] S. Shankar, "Security Outcomes Report Volume 3", Cisco, 2022, [Online]. Available at: https://www.cisco.com/c/dam/en/us/products/collateral/security/security-outcomes-vol-3-report.pdf?utm_medium=email&utm_source=prospect&utm_campaign=UMB-FY23-Q2-Content-Ebook-Security-Outcomes-Report-V3&utm_term=confirmation&utm_content=UMB-FY23-Q2-Content-Ebook-Security-Outcomes-Report-V3.

[2] M. Mamoru, "Adjusting lexical features of actual proxy logs for intrusion detection," *Journal of Information Security and Applications*, vol. 50, 2020. DOI: https://doi.org/10.1016/j.jisa.2019.102408.

[3] M. Hatada, M. Akiyama, T. Matsuki, T. Kasama, "Empowering anti-malware research in Japan by sharing the MWS datasets," *Journal of Information Processing*, vol. 23, pp. 579–588, 2015. DOI: https://doi.org/10.2197/ipsjjip.23.579.

[4] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, H. Levy, "Spyproxy: Execution-based detection of malicious web content," Department of Computer Science & Engineering, University of Washington, 2007.

[5] C. Novo, R. Morla, "Flow-based detection and proxy-based evasion of encrypted malware C2 traffic," *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, New York, United States, November 13, 2020, pp. 83-91. DOI: https://doi.org/10.48550/arXiv.2009.01122.

[6] "PE Format," Microsoft Learn, 2024, [Online]. Available at: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[7] M. V. Mishchenko, M. S. Dorosh, "Semantic analysis and classification of malware for UNIX-like operating systems with the use of machine learning methods," *Applied Aspects of Information Technology*, vol. 5, no. 4, 371, 2022. DOI: https://doi.org/10.15276/aait.05.2022.25.

[8] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, Doha, Qatar, October 2014, pp. 103-111. DOI: https://doi.org/10.48550/arXiv.1409.1259.

[9] Q. Qi, L. Lin, R. Zhang and C. Xue, "MEDT: Using multimodal encoding-decoding network as in transformer for multimodal sentiment analysis," *IEEE Access*, vol. 10, pp. 28750-28759, 2022. DOI: https://doi.org/10.1109/ACCESS.2022.3157712.

[10] X. Xiao, L. Wang, K. Ding, S. Xiang and C. Pan, "Deep hierarchical encoder–decoder network for image captioning," *IEEE Transactions on Multimedia*, vol. 21, no. 11, pp. 2942-2956, 2019. DOI: https://doi.org/10.1109/TMM.2019.2915033.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, "Attention is all you need," *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017. DOI: https://doi.org/10.48550/arXiv.1706.03762.

[12] D. Tsirmpas, I. Gkionis, G. Th. Papadopoulos, I. Mademlis, "Neural natural language processing for long texts: A survey on classification and summarization," *Engineering Applications of Artificial Intelligence*, vol. 133, 2024. https://doi.org/10.1016/j.engappai.2024.108231.

[13] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019. https://doi.org/10.48550/arXiv.1907.11692.

[14] K. Clark, M. Luong, Q. V. Le, C. D. Manning, "ELECTRA: pre-training text encoders as discriminators rather than generators," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. https://doi.org/10.48550/arXiv.2003.10555.

[15] J. Sawicki, M.Ganzha, M. Paprzycki, "The state of the art of natural language processing – A Systematic automated review of NLP literature using NLP techniques," *Data Intelligence*, vol. 5, issue 3, pp. 707–749, 2023. https://doi.org/10.1162/dint_a_00213.

[16] T. Mikolov, K. Chen, G. Corrado, J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR,* January 2013. https://doi.org/10.48550/arXiv.1301.3781.

[17] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, vol. 26, 2013. https://doi.org/10.48550/arXiv.1310.4546.

[18] T. Kien, S. Hiroshi, "NLP-based approaches for malware classification from API sequences," *Proceedings of the 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES),* 2017, pp. 101-105. https://doi.org/10.1109/IESYS.2017.8233569.

[19] C. Hui, N. Takashi, N. Yoshiki, "Approximate RBF Kernel SVM and its applications in pedestrian classification," 2008, pp. 1-9. https://doi.org/10.1007/978-1-4020-8450-8_1.

[20] D. P. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization," *Proceedings of the International Conference on Learning Representations*, December 2014. DOI: https://doi.org/10.48550/arXiv.1412.6980.

[21] C. F. Ozgur, A. Javed, S. Kevser, K. Z. Hussain, "Data augmentation based malware detection using convolutional neural networks," *Peer J Computer Science*, vol. 7, 2021. DOI: https://doi.org/10.48550/arXiv.2010.01862.

[22] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, "BODMAS: An open dataset for learning based temporal analysis of PE malware," *Proceedings of the 4th Deep Learning and Security Workshop*, San Francisco, CA, USA, 2021, pp. 78-84. DOI: https://doi.org/10.1109/SPW53761.2021.00020.

[23] Global Threat Activity, Microsoft, 10 May 2024, [Online]. Available at: https://www.microsoft.com/en-us/wdsi/threats.

[24] S. Khan, M. Nauman, "Interpretable detection of malicious behavior in windows portable executables using multi-head 2D transformers," *Big Data Mining and Analytics*, vol. 7, pp. 485–499, 2024. DOI: https://doi.org/10.26599/BDMA.2023.9020025.

[25] A. Radford, and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available at: https://api.semanticscholar.org/CorpusID:49313245.

[26] VirusShare, GitHub, Sep 2, 2020, [Online]. Available at: https://github.com/seifreed/VirusShare.

[27] Manalyze, GitHub, Jan 3, 2024, [Online]. Available at: https://github.com/JusticeRage/Manalyze.

[28] A. Cortesi, M. Hils, T. Kriechbaumer, "{mitmproxy}: A free and open source interactive {HTTPS} proxy," 10 May 2024, [Online]. Available at: https://mitmproxy.org/.

[29] E. Carrera Ventura, "pefile (Version 2023.2.7)," GitHub. [Online]. Available at: https://github.com/erocarrera/pefile.

[30] V. Diaz, "VirusTotal malware trends report: Emerging formats and delivery techniques," July 26, 2023, [Online]. Available at: https://blog.virustotal.com/2023/07/virustotal-malware-trends-report.html.

[31] S. Lad, & A. Adamuthe, "Improved deep learning model for static PE files malware detection and classification," *International Journal of Computer Network and Information Security*, vol. 14, pp. 14-26, 2022. https://doi.org/10.5815/ijcnis.2022.02.02.

[32] Y. Ye, T. Li, Q. Jiang, Y. Wang, "CIMDS: Adapting postprocessing techniques of associative classification for malware detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, pp. 298–307, 2010. https://doi.org/10.1109/TSMCC.2009.2037978.

[33] A. Koçak, E. Söğüt, M. Alkan, O. Ayhan Erdem, "Detection of different windows PE malware using machine learning methods," *Journal of Polytechnic*, vol. 26, issue 3, pp. 1185-1197, 2023. https://doi.org/10.2339/politeknik.1207704.

MAKSYM V. MISHCHENKO, *Postgraduate, PhD student in Information Technology and Software Engineering Department. Chernihiv Polytechnic National University, 95, Shevchenko Street. Chernihiv, 14035, Ukraine. Research fields: Cybersecurity; machine learning; NLP; operating systems; software engineering.*

MARIIA S. DOROSH, *Dr. Sc. (Eng), a Professor of Information Technology and Software Engineering Department, Chernihiv Polytechnic National University, 95, Shevchenko Str. Chernihiv, 14035, Ukraine. Research fields: Modeling and design of intelligent systems; knowledge management; convergence of project management systems; human factor in information security systems of organizations and projects*