UDC 004.056

Malware Detection System Based on Static and Dynamic Analysis Using Machine Learning

Alan Nafiiev[1], Andrii Rodionov[1]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Institute of Physics and Technology, Build. 1, 37, Beresteiskyi avenue, Kyiv, 03056, Ukraine*

Abstract

Cyber wars and cyber attacks are an urgent problem in the global digital environment. Based on existing popular detection methods, malware authors are creating ever more advanced and sophisticated malware. Therefore, this study aims to create a malware analysis system that uses both dynamic and static analysis. Our system is based on a machine learning method - support vector machine. The set of data used was collected from various Internet sources. It consists of 257 executable files in .exe format, 178 of which are malicious and 79 are benign. We use 5 different types of data representation: binary information, trace instructions, control flow graph, information obtained from the dynamic operation of the file, and file metadata. Then, using multiple kernel learning, we combine all data views and create one summative machine learning model.

Keywords: malware detection, malware dynamic analysis, feature selection, multiple kernel learning

Introduction

In the history of mankind, wars are an integral part of the nature of Homo sapiens. And as the current situation in the world shows, despite the rapid technological breakthrough of the 21st century, war is still inherent in modern man. However, with the growth of the global digitalization of the world, the type of war is also changing. Today we can observe that almost the most important role is played by cyber attacks. Since even one malicious file can harm the vital infrastructure. Therefore, this study is aimed at combating malware, namely the difficult associated with its detection. Malware analysis can be performed by static or dynamic methods. In this work, we use both and combine them.

The aim of this work is to create a binary classification system for executable files. A system that equally well detects both malware with a simple structure and a complex, isomorphic malware that can change its structure during activity. Obviously, to solve such a task, one data view will not be enough. To successfully analyze the true nature of a malicious file, we should extract as much information from it as possible. Therefore, we will use 5 different types of data representation:

1. Binary
2. Trace
3. Control flow graph
4. Dynamic
5. File Info

Binary - uses the byte information contained in the binary executable file. Trace - disassembled code is used. These two static methods were described in detail in our previous work [1]. CFG - a method based on control flow graph, the graphlet kernel is used [2]. Dynamic - a method based on dynamic analysis, where we get information about a file while it is active in a virtual environment [3]. File Info - method based on file metadata. Each method is described in more detail in the section Feature Selection.

Now, with the five methods of file representation, the task is to apply this data for classification. For each method, we constructed a machine learning model based on the support vector machine algorithm. This algorithm was chosen because it showed good accuracy results in our work, where we compared different machine learning methods [4]. And the main selection criterion was the fact that SVM uses the properties of kernels, which allows us to apply multiple kernel learning [5, 6]. By means of which several data views can be combined into one model. And this model takes into account the features of each of the methods during training.

This approach is described in more detail in the section “Kernel and training”.

1. Dataset

The dataset used consists of 257 .exe files (178 malware and 79 benign). There are 7 types of malware files: InstallCore, CryptoRansom, TheZoo, Zeus, Zbot, Zeroaccess, Winwebsec. The distribution infographic can be seen in **Figure 1**. All malware files were taken from various sites: "virusshare.com", "malicia-project.com", "thezoo.morirt.com". The benign files were taken from the installed application folders of legal software from different categories. Files were also taken from the site "exefiles.com". The input dataset was separated as a test and training set in the ratio of 30/70, respectively. Both sets included 7 malware types.

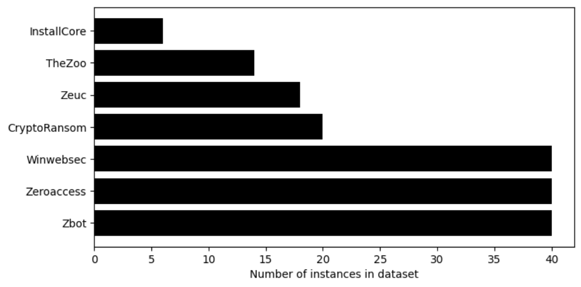


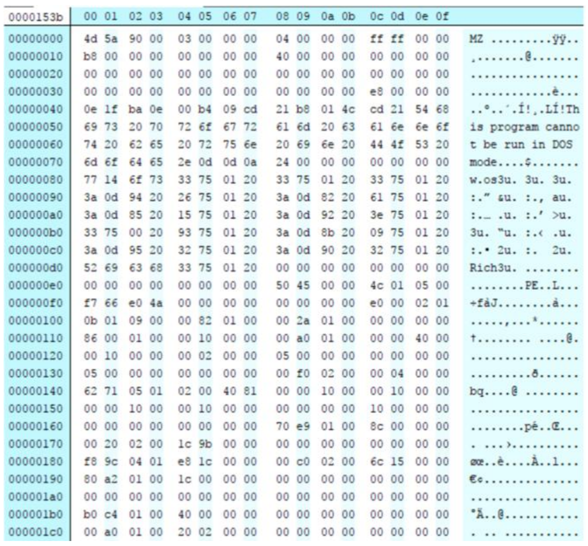
Figure 1: Infographics of file types distribution

2. Feature Selection

This section describes 5 different approaches to representing an .exe file. We have focused on collecting and processing data to form features on which the support vector machine will learn.

2.1. Binary

To generate the input data for the machine learning model, we use the bit information of the executable file represented using the PE format. For each item in the dataset, we generate an array containing the byte sequence of the .exe file.



```
0000153b  00 01  02 03  04 05  06 07  08 09  0a 0b  0c 0d  0e 0f
00000000  4d 5a  90 00  03 00  00 00  04 00  00 00  ff ff  00 00   MZ .........ÿÿ..
00000010  b8 00  00 00  00 00  00 00  40 00  00 00  00 00  00 00   ¸.......@.......
00000020  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00   ................
00000030  00 00  00 00  00 00  00 00  00 00  00 00  e8 00  00 00   ............è...
00000040  0e 1f  ba 0e  00 b4  09 cd  21 b8  01 4c  cd 21  54 68   ..º..´.Í!¸.LÍ!Th
00000050  69 73  20 70  72 6f  67 72  61 6d  20 63  61 6e  6e 6f   is program canno
00000060  74 20  62 65  20 72  75 6e  20 69  6e 20  44 4f  53 20   t be run in DOS
00000070  6d 6f  64 65  2e 0d  0d 0a  24 00  00 00  00 00  00 00   mode....$.......
00000080  77 14  6f 73  33 75  01 20  33 75  01 20  33 75  01 20   w.os3u. 3u. 3u.
00000090  3a 0d  94 20  26 75  01 20  3a 0d  82 20  61 75  01 20   :.” &u. :., au.
000000a0  3a 0d  85 20  15 75  01 20  3a 0d  92 20  3e 75  01 20   :.… .u. :.’ >u.
000000b0  33 75  00 20  93 75  01 20  3a 0d  8b 20  09 75  01 20   3u. “u. :.‹ .u.
000000c0  3a 0d  95 20  32 75  01 20  3a 0d  90 20  32 75  01 20   :.• 2u. :.  2u.
000000d0  52 69  63 68  33 75  01 20  00 00  00 00  00 00  00 00   Rich3u. ........
000000e0  00 00  00 00  00 00  00 00  50 45  00 00  4c 01  05 00   ........PE..L...
000000f0  f7 66  e0 4a  00 00  00 00  00 00  00 00  e0 00  02 01   ÷fàJ........à...
00000100  0b 01  09 00  00 82  01 00  00 2a  01 00  00 00  00 00   .....‚...*......
00000110  86 00  01 00  00 10  00 00  00 a0  01 00  00 00  40 00   †........ ....@.
00000120  00 10  00 00  00 02  00 00  05 00  00 00  00 00  00 00   ................
00000130  05 00  00 00  00 00  00 00  00 f0  02 00  00 04  00 00   .........ð......
00000140  62 71  05 01  02 00  40 81  00 00  10 00  00 10  00 00   bq....@ ........
00000150  00 00  10 00  00 10  00 00  00 00  00 00  10 00  00 00   ................
00000160  00 00  00 00  00 00  00 00  70 e9  01 00  8c 00  00 00   ........pé..Œ...
00000170  00 20  02 00  1c 9b  00 00  00 00  00 00  00 00  00 00   . ...›..........
00000180  f8 9c  04 01  e8 1c  00 00  00 c0  02 00  6c 15  00 00   øœ..è....À..l...
00000190  80 a2  01 00  1c 00  00 00  00 00  00 00  00 00  00 00   €¢..............
000001a0  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00   ................
000001b0  b0 c4  01 00  40 00  00 00  00 00  00 00  00 00  00 00   °Ä..@...........
000001c0  00 a0  01 00  20 02  00 00  00 00  00 00  00 00  00 00   . .. ...........
```

Figure 2: File representation in PE format

Then, based on the byte sequences obtained, construct a graph as follows: all 256 possible bytes (00, 01, ... ff) are the vertices of the graph. For each pair of bytes in the graph it is counted how many times the first byte was immediately followed by the second byte. So, we find the probability that some byte X will be followed by byte Y.

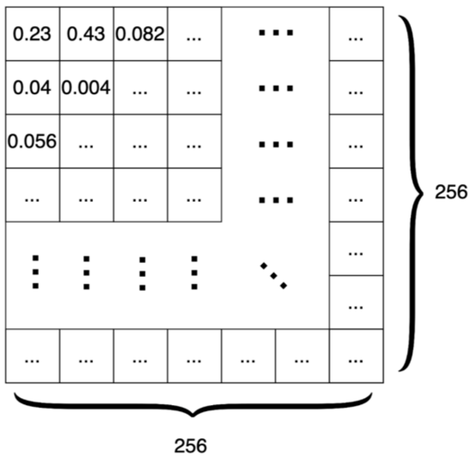


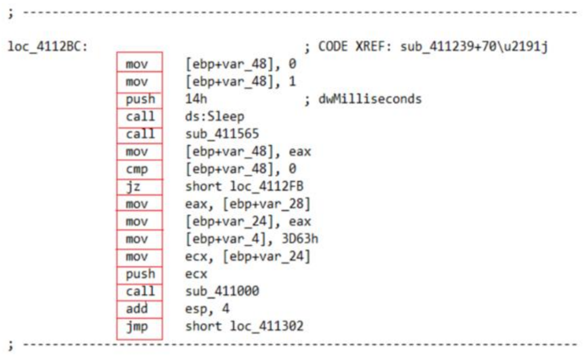
Figure 3: Adjacency matrix

We group all the adjacency matrices into one final matrix, which is sent to the support vector machine algorithm. The algorithm uses a Gaussian kernel, where for two graphs their adjacency matrices are taken and their kernel is calculated using the following formula:

$$K_G(x, x') = \sigma^2 e^{-\frac{1}{2\lambda^2}\sum_{i,j}\left(x_{ij}-x'_{ij}\right)^2} \qquad (1)$$

2.2. Trace

The disassembled code generated by IDA Pro is used for the trace type of data representation. A fragment of such code can be seen in **Figure 4**. The Markov chain is constructed similarly to how it was constructed for binary files. Only instead of byte values for vertices of the graph we use disassembled instructions (mov, push, call, jz, etc.)



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

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 4: A fragment of the disassembled code

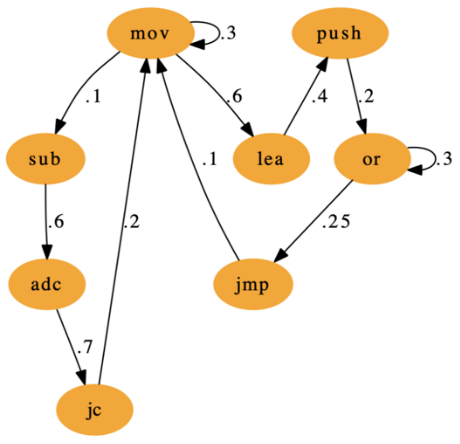


Figure 5: Graph

Based on the obtained graphs, we construct an adjacency matrix for each file. The size of the adjacency matrix determines the set of unique instructions, which was formed as follows: the 187 most frequent instructions in the files were taken. Just as in the binary file representation, we use the Gaussian kernel (1) in the support vector machine algorithm.

2.3. Control Flow Graph

A control flow graph is a graph representation that models all of the paths of execution that a program might take during its lifetime. In the graph, the vertices are the basic blocks, sequential code without branches or jump targets, of the program, and the edges represent the jumps in control flow of the program. One of the advantages of this representation is that it is very difficult for a polymorphic virus to create a semantically similar version of itself while modifying its control flow graph enough to avoid detection. A fragment of the CFG can be seen in **Figure 6**.

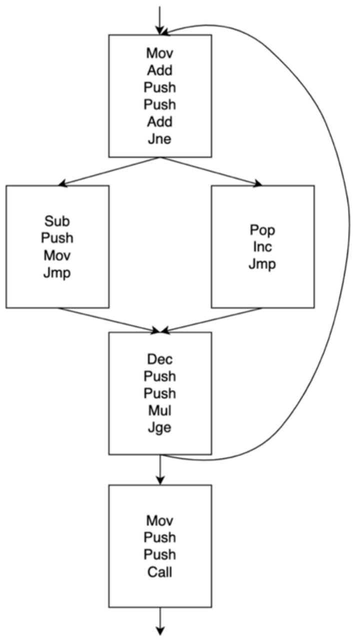


Figure 6: A fragment of the Control Flow Graph

For further graph processing we chose the graphlet kernel because of its computational efficiency [7]. The essence of the graphlet kernel is that it does not take into account what instructions are inside vertices, but just compares their structure. A k-graphlet is defined as a subgraph of graph G with the number of subgraph nodes equal to k. As k was chosen 4 as such which gives the highest accuracy and AUC.

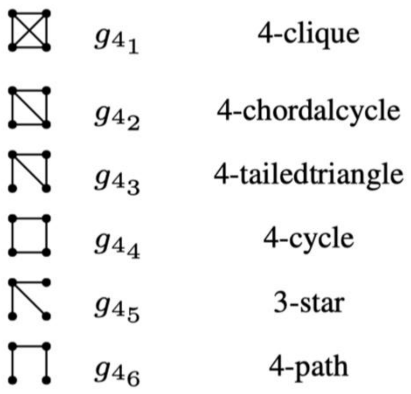


Figure 7: k-graplets, where k = 4

If $\overrightarrow{f_G}$ is a feature vector, where each feature is the number of times a unique graphlet of size k occurs in G, the normalized probability vector is:

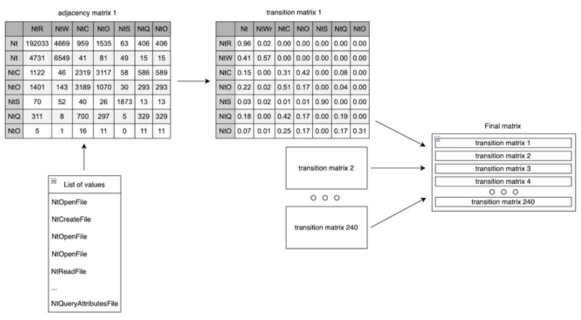
$$\overrightarrow{D_G} = \frac{\overrightarrow{f_G}}{\#\ of\ all\ graphlets\ of\ size\ k\ in\ G}$$

and the graphlet kernel is defined as:

$$K_g(G, G') = \overrightarrow{D_G^T}\overrightarrow{D_{G'}}$$

2.4. Dynamic method

For dynamic file representation, we used the Drakvuf Sandbox malware analysis system, which allows us to monitor malware at the user and OS kernel level without the need to install an agent in the guest OS [8]. The system is built on the Xen virtualization platform, uses the LibVMI API and the DRAKVUF engine. With this system we collect data on the operation of metamorphic malware and monitor: process execution, file operations, system calls and kernel function traces. The principle behind the formation of features is the same as in the binary representation of the data. You can see a diagram of this process in **Figure 8**. We also use the Gaussian kernel (1) for this data representation. This dynamic method was described in more detail in our previous work [3].



	NtR	NtW	NtC	NtO	NtS	NtQ	NtO
Nt	192033	4669	959	1535	63	406	406
Nt	4731	6549	41	81	49	15	15
NtC	1122	46	2319	3117	58	586	589
NtO	1401	143	3189	1070	30	293	293
NtS	70	52	40	26	1873	13	13
NtQ	311	8	700	297	5	329	329
NtO	5	1	16	11	0	11	11

	Nt	NtWr	NtC	NtO	NtS	NtQ	NtO
NtR	0.96	0.02	0.00	0.00	0.00	0.00	0.00
NtW	0.41	0.57	0.00	0.00	0.00	0.00	0.00
NtC	0.15	0.00	0.31	0.42	0.00	0.08	0.00
NtO	0.22	0.02	0.51	0.17	0.00	0.04	0.00
NtS	0.03	0.02	0.01	0.01	0.90	0.00	0.00
NtQ	0.18	0.00	0.42	0.17	0.00	0.19	0.00
NtO	0.07	0.01	0.25	0.17	0.00	0.17	0.31

Figure 8: Final matrix formation scheme

2.5. Miscellaneous File Information

For this data view, we collected seven pieces of information about the different data views described earlier. The following data is taken about the file:

1. Entropy
2. Packed
3. Binary Size
4. Number Edges
5. Number Vertices
6. Number Static Instructions
7. Number Dynamic Instructions

We use file entropy, similar to previous work [9]. To find whether a file was packed or not, we used the PEID signature method [10]. Binary Size is used as the file size in megabytes. We also use the number of vertices and edges in the control flow graph. Finally, we use the average number of instructions in the disassembled file and the average number of instructions/system calls in the dynamic analysis. For the file information feature vector, we use a standard squared exponential kernel:

$$K_{SE}(x, x') = \sigma^2 e^{-\frac{1}{2\lambda^2}\sum_i (x_i - x'_i)^2}$$

3. Kernel and training

In our previous work, we experimented with simply averaging the features of different methods to obtain the final machine learning model [3]. However, this approach did not yield significant improvements in accuracy. Therefore, we use multiple kernel learning, since this approach combines the features of different methods directly during model training, that is, it allows us to combine different types of data representation at a deeper level.

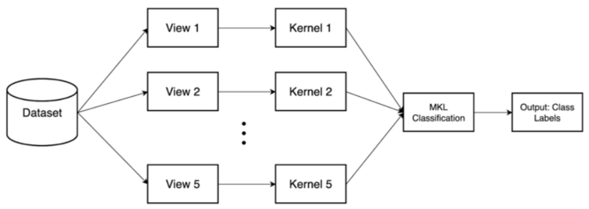


Figure 9: Architecture diagram for classification system

The goal of classical kernel-based learning with support vector machines is to learn the weight vector, α , describing each data instance's contribution to the hyperplane that separates the points of the two classes with a maximal margin and can be found with the following optimization problem:

$$\min_{\alpha} \underbrace{\left(\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i \right)}_{S_k(\alpha)} \quad (2)$$

subject to the constraints:

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C$$

where y_i is the class label of instance x_i.
With multiple kernel learning, each individual kernel's contribution, β, must also be found such that:

$$K_{comb}(x_i, x_j) = \sum_{k=1}^{M} \beta_k K_k(x_i, x_j)$$

K_{comb} is a combination of M kernels with $\beta_k \ge 0$, where each kernel K_k uses a unique set of features, which is a different kind of data representation. The general outline of the algorithm is to first combine the kernels with $\beta_k = \frac{1}{M}$, find α, and then iteratively continue optimizing for β and α until convergence. To solve for β, assuming a fixed set of support vectors (α), the following semi-infinite linear program has been proposed [6]:

$$\max \theta \quad (3)$$

$$w.r.t. \;\; \theta \in \mathbb{R}, \beta \in \mathbb{R}^K$$

subject to constraints:

$$\sum_{k=1}^{M} \beta_k S_k(\alpha) \ge \theta \quad (4)$$

$$\sum_{k} \beta_k = 1$$

$$\theta \ge 0$$

for all $\alpha \in \mathbb{R}^N$ with $0 \le \alpha \le C$ and $\sum_i y_i \alpha_i = 0$, and where $S_k(\alpha)$ is defined in eq. (2). M is the number of kernels to be combined. This is a semi-infinite linear program because all of the constraints in eq. (4) are linear, and there are infinitely many of them, one for each $\alpha \in \mathbb{R}^N$ satisfying $0 \le \alpha \le C$ and $\sum_i y_i \alpha_i = 0$ cannot go to ∞ because of the constraint $\sum_{k=1}^{M} \beta_k S_k(\alpha) \ge \theta$. Finding the maximal theta corresponds to finding a saddle-point of the following min-max optimization problem:

$$\max_{\beta} \min_{\alpha} \sum_{k=1}^{M} \beta_k S_k(\alpha) \ge \theta$$

If α is the optimal solution, then $\sum_{k=1}^{M} \beta_k S_k(\alpha) = \theta$ would be minimal satisfying eq. (4) for all α. To find solutions for α and β, an iterative algorithm was proposed that first uses a standard support vector machine algorithm to find α (eq. (2)), then fixes α to the solution, and solves eq. (3) to find β. Although this algorithm is known to converge, there are no known convergence rates [11]. Therefore, the following stopping criterion was proposed [6]:

$$\epsilon^{t+1} \ge \epsilon^t := \left| 1 - \frac{\sum_{k=1}^{M} \beta_k^t S_k(\alpha^t)}{\theta^t} \right|$$

Results

Table 1 shows the accuracy results of all models. All kernels - the model that combines all 5 methods of data representation: Dynamic, Binary, Trace, CFG, and File info. Static - the model that combines static analysis methods:

Table 1
Performance results of the machine learning models

	F-score		Precision		Recall		roc_auc	pr_auc
	0	1	0	1	0	1		
All kernels	0.9200	0.9622	0.8846	0.9807	0.9583	0.9444	0.9876	0.9945
Static	0.8571	0.9345	0.8400	0.9433	0.8750	0.9259	0.9606	0.9838
Dynamic	0.8163	0.9158	0.8000	0.9245	0.8333	0.9074	0.9328	0.9710
Binary	0.7755	0.8971	0.7600	0.9056	0.7916	0.8888	0.9205	0.9678
Trace	0.7843	0.8952	0.7407	0.9215	0.8333	0.8703	0.9120	0.9619
CFG	0.7200	0.8679	0.6923	0.8846	0.7500	0.8518	0.8842	0.9516
File info	0.6153	0.8076	0.5714	0.8400	0.6666	0.7777	0.8109	0.9107

Binary, Trace, CFG and File info. We can see the AUC curves in **Figure 10** and **Figure 11**. From **Figure 12** to **Figure 18** we can see the confusion matrix of each model. It is not difficult to notice that the "All kernels" model shows the best result (roc auc - 0.9876) for all metrics. The Static model also shows good accuracy (roc auc - 0.9606). It is worth noting that the dynamic model was better than any static method, but worse than all static methods taken together - the Static model. As expected, the model based on file metadata had the worst accuracy (roc auc - 0.8109). The CFG model showed more acceptable accuracy (roc auc - 0.8842), but was slightly worse than the Binary and Trace models, which showed approximately equal results for all metrics. However, the Binary is still slightly better, (roc auc - 0.9205) versus (roc auc - 0.9120).

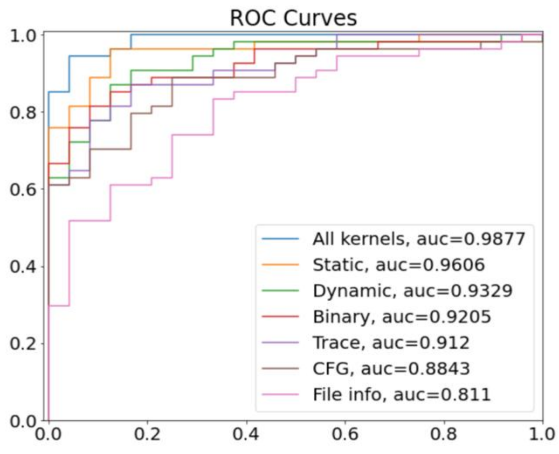


Figure 10: Roc curves

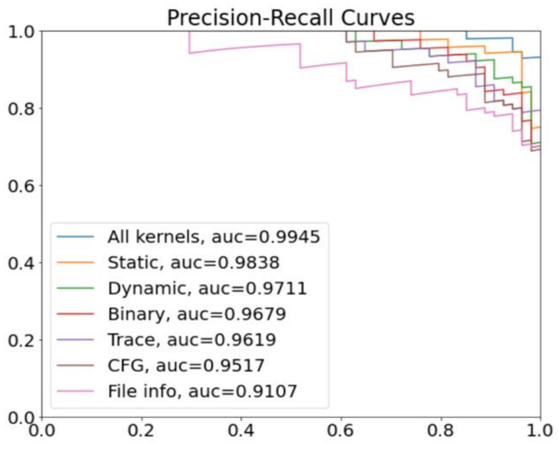


Figure 11: Precision-Recall curves

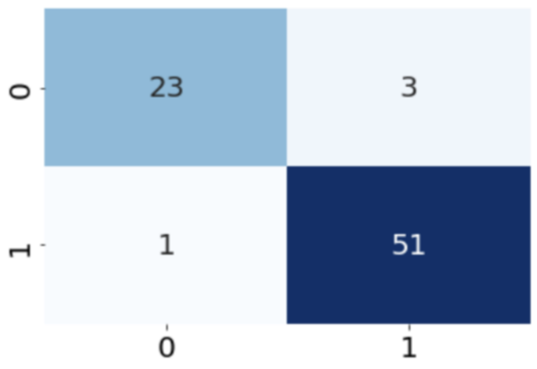


Figure 12: Confusion matrix of "All Kernels" model

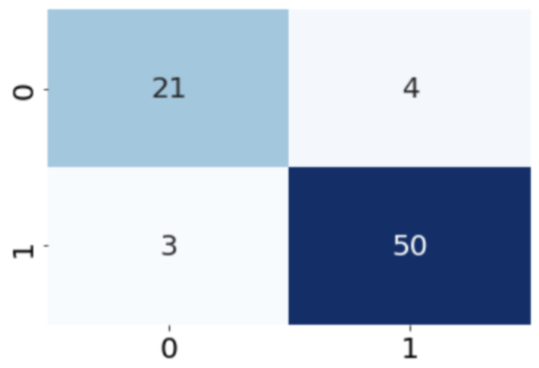


Figure 13: Confusion matrix of Static model

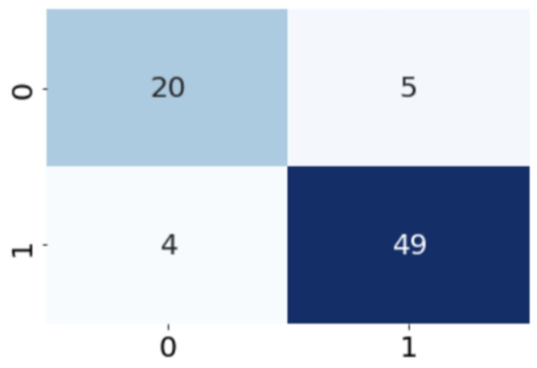


Figure 14: Confusion matrix of Dynamic model

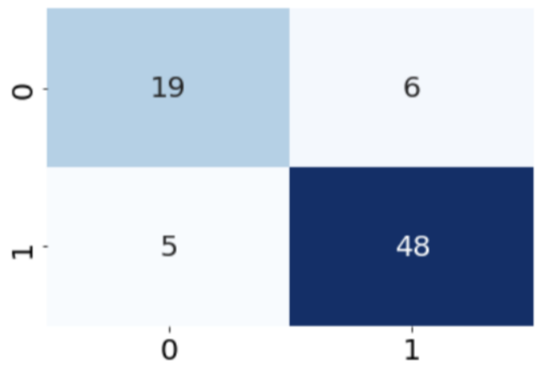


Figure 15: Confusion matrix of Binary model

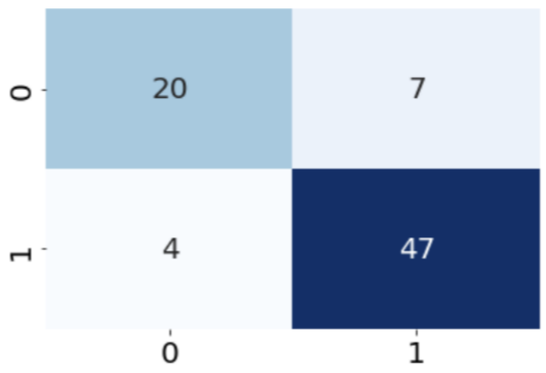


Figure 16: Confusion matrix of Trace model

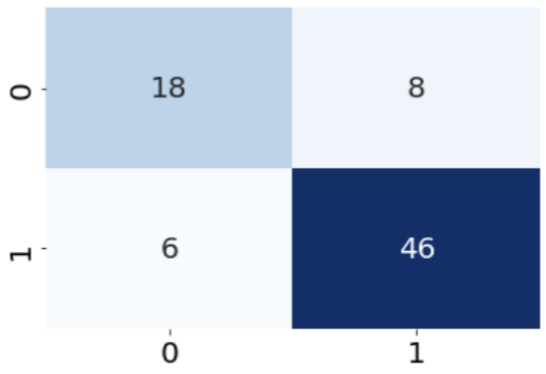


Figure 17: Confusion matrix of CFG model

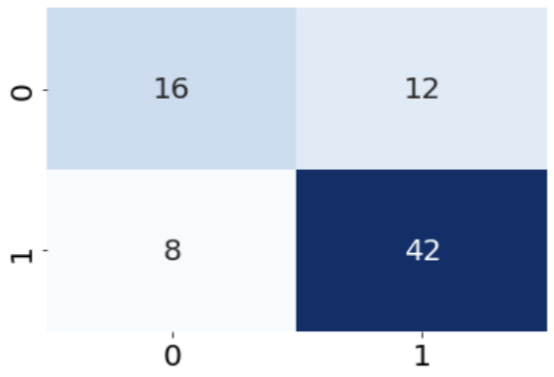


Figure 18: Confusion matrix of File info model

Conclusions

In this study, we described 5 methods of file representation. 4 methods of static analysis and 1 method of dynamic analysis. Using multiple kernel learning, the final model was created, which combines all the data representation methods we described. This final model showed the highest accuracy on all metrics. Also, the dynamic model was better than any static method, but worse than all static methods combined. Based on this, it can be argued that it is useful to use all available information about executable files, rather than just a single data representation, in order to classify malicious files qualitatively. It is worth noting that in order to use such a file classification system in a real case, a large dataset should be collected that includes as many types of malicious files as possible.

References

[1] Alan Nafiiev, Hlib Kholodulkin, Andrii Rodionov and Dmytro Lande, “Comparative analysis of machine learning models with different types of data representations for detecting malicious files”, International Scientific Conference "Information Technology and Implementation", December 2022.

[2] Nino Shervashidze, S.V.N. Vishwanathan, Tobias H. Petri, Kurt Mehlhorn, Karsten M. Borgwardt, “Efficient graphlet kernels for large graph comparison”, Journal of Machine Learning Research, January 2009.

[3] Alan Nafiiev, Hlib Kholodulkin, Andrii Rodionov, “Malware dynamic analysis system based on virtual machine introspection and machine learning methods”, Information Technologies and Security 2022.

[4] Alan Nafiiev, Hlib Kholodulkin, Andrii Rodionov, “Comparative analysis of machine learning methods for detecting malicious files”, Theoretical and Applied Cybersecurity, Algorithms and methods of cyber attacks prevention and counteraction (2022).

[5] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan, Multiple Kernel Learning, Conic Duality, and the SMO Algorithm, Proceedings of the Twenty-First International Conference on Machine Learning, 2004.

[6] Soren Sonnenburg, Gunnar Raetsch, and Christin Schaefer, A General and Efficient Multiple Kernel Learning Algorithm, Nineteenth Annual Conference on Neural Information Processing Systems (2005).

[7] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna, “Polymorphic Worm Detection Using Structural Information of Executables”, Recent Advances in Intrusion Detection, 2006

[8] Tamas K Lengyel, Steve Maresca, Bryan D Payne, George D Webster, Sebastian Vogl, and Aggelos Kiayias. Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In The 30th Annual Computer Security Applications Conference, pages 386–395, 2014

[9] Robert Lyda and James Hamrock, Using Entropy Analysis to Find Encrypted and Packed Malware, IEEE Security & Privacy 5 (2007)

[10] Portable Executable iDentifier, Accessed 6 October 2011. http://peid.info/.

[11] Rainer Hettich and Kenneth Kortanek, Semi-Infinite Programming: Theory, Methods, and Applications, SIAM Review 35 (1993), 380–429.