Multinomial Malware Classification Via Low-Level Features

By

Sergii Banin, Geir Olav Dyrkolbotn

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2018 USA

Providence, RI (July 15th - 18th)

Digital Investigation 26 (2018) S107–S117

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2018 USA — Proceedings of the Eighteenth Annual DFRWS USA

Multinomial malware classification via low-level features

Sergii Banin [a, *], Geir Olav Dyrkolbotn [a, b]

[a] Department of Information Security and Communication Technology, NTNU, Gjøvik, Norway
[b] Norwegian Defence Cyber Academy (NDCA), Jørstadmoen, Norway

Keywords:
Information security
Malware detection
Malware classification
Multinomial classification
Low-level features
Hardware activity

A B S T R A C T

Because malicious software or (”*malware*”) is so frequently used in a cyber crimes, malware detection and relevant research became a serious issue in the information security landscape. However, in order to have an appropriate defense and post-attack response however, malware must not only be detected, but also categorized according to its functionality. It comes as no surprise that more and more malware is now made with the intent to avoid detection and research mechanisms. Despite sophisticated obfuscation, encryption, and anti-debug techniques, it is impossible to avoid execution on hardware, so hardware (“*low-level*”) activity is a promising source of features. In this paper, we study the applicability of low-level features for multinomial malware classification. This research is a logical continuation of a previously published paper (Banin et al., 2016) where it was proved that memory access patterns can be successfully used for malware detection. In this research we use memory access patterns to distinguish between 10 malware families and 10 malware types. In the results, we show that our method works better for classifying malware into families than into types, and analyze our achievements in detail. With satisfying classification accuracy, we show that thorough feature selection can reduce data dimensionality by a magnitude of 3 without significant loss in classification performance.

1. Introduction

Malware detection is an important part of information security. Recently there were several major cyber attacks that influenced power grids, banking and transportation systems, manufacturing facilities and so on Reuters (2017), The Verge (2017) and all of them used malware for achieving their final goals. Despite the use of antivirus solutions, complicated anti-detection techniques allowed adversaries to avoid defense mechanisms. This fact points out a need for improvements in malware detection.

Malware is used for different purposes: to show ads to users, spread spam, track user activity, steal data, create backdoors and so on. Malware is often not created with a single specific purpose, but rather as a part of more advanced threats. APT or Advanced Persistent Threat is a human being or organization (WAMPTY Enterprise) that operates a campaign of intellectual property theft, the undermining of a company's or country's operations through stealthy, targeted, adaptive and data focused (Cole, 2012) attack techniques. *Something* has to exploit a victim's weaknesses, *something* has to aid in the installation of persistence tools, *something* has to communicate with command and control servers, and *something* has to perform actions in the victim system. Even though specific actions might be launched manually from the command and control server, they may rely on remote access trojans and backdoors (Rudd et al., 2017) present in the victim system. As we can see, malware could be used for different purposes and goals.

Because of the variety of malware functionality, it is important not only to detect malice (*malware detection*), but to differentiate between different *kinds* of malware (*multinomial malware classification* or *malware classification*) in order to provide better understanding of malware capabilities, describe vulnerabilities of systems and operations as well as to use appropriate protection and post-attack actions.

Malware *classification* or *categorization* is a common problem that is analyzed in many research articles (Tabish et al., 2009; Sathyanarayan et al., 2008). There are two widely used malware categorization approaches: *malware types* and *malware families*. However, literature studies show that authors rarely provide proper definitions of these terms. This can lead to the various misunderstandings and non-valid comparisons. E.g. in Tabish et al. (2009), authors mention *viruses, backdoors, trojans* etc. while talking about classifying *malware types* and *families*. Another example

* Corresponding author.
E-mail addresses: sergii.banin@ntnu.no (S. Banin), geir.dyrkolbotn@ntnu.no (G.O. Dyrkolbotn).

https://doi.org/10.1016/j.diin.2018.04.019
1742-2876/

of inconsistent terminology can be found in Sathyanarayan et al. (2008). In this paper, authors claim that their system is capable of detecting the *malware families* (in their case *trojans, backdoors, worms*). Nevertheless, they compare their results to the results from other papers where research was done on the *malware types*. Authors of Saeed et al., (2013) attempted to elaborate on the definition of the term *malware*; however, later on they use term *malware family* when talking about *viruses, trojans, worms* and other *malware types*. It might happen, that the use of inconsistent terminology is more common among academics and not malware analysis practitioners. Therefore, we must emphasize, in this paper that we use the following definitions created after reviewing descriptions of malware categories provided by well-known vendors (e.g. Microsoft, Symantec etc):

Malware type is assigned according to general functionality.
Malware is grouped into a **malware family** according to its particular functionality.

Where general functionality is about *what* malware does (which goals it pursues), and particular functionality is about *how* malware acts (which methods it uses in order to achieve its goals).

As it appears, it is insufficient to know that some malware is affecting operations: knowledge about its category (*family* or *type*) can aid in restoring a system's state as well as in developing new security mechanisms to prevent similar problems in the future. This necessitates standard definitions of different malware kinds and methods that allow the effective categorization of detected malware.

To avoid detection, malware creators develop additional evasive methods to thwart detection by antimalware software. They utilize various obfuscation techniques such as metamorphism, polymorphism, encryption, dead code insertions, and instruction substitution (Schiffman, 2010). Such methods allow altering the appearance of a file and its static characteristics. The basic example is changing hash sums (such that SHA-1 or md5) used as file signatures by means of changing different strings in the file. Moreover, dead code insertions will change opcode sequences in the executable, making detection more difficult.

There are two main ways to perform malware analysis which are widely used and described in the literature (Distler and Hornat, 2007; Kendall): *static* and *dynamic*. *Static analysis* is performed without execution of a malicious file. The main purpose of this approach is to collect different static properties: bytes, opcodes and API n-grams frequencies, properties of Portable Executable header, strings (e.g. commandline commands, URLs etc) and others (Schiffman, 2010; Uppal et al., 2014). *Dynamic analysis* is done by executing malware in a controlled environment (a virtual machine or emulator) and recording actions it has done in the system. These include patterns of a registry, network and disk usage, monitoring of API-calls, tracing of executed instructions, investigation of memory layout and so on (Egele et al., 2012). Specialized sandboxes like Cuckoo (Cuckoo Sandbox, 2015) or other Virtual Machines can be used. They might be assisted by a debugger or other tracing software. Some authors assume (Egele et al., 2012; Prakash et al., 2015) disk and network activities are essential for malware detection, but few authors explored the capabilities of memory properties analysis (Kawakoya et al., 2013; Khasawneh et al., 2015).

Though malware creators use a variety of sophisticated evasive techniques (Rudd et al., 2017), it is impossible to avoid execution on the system's hardware. Earlier low-level (or hardware) activity has proven to be efficient in malware detection (Banin et al., 2016). In this paper, we use a similar technique for multinomial malware classification. Achieved results and findings will be used in future work, where combinations of high- and low-level activity will be used for malware categorization according to the specific context.

In this paper, we use sequences of memory access operations generated by a set of malicious executables as a source of features for machine learning algorithms. We apply dynamic analysis inside the virtualized environment as it is a safe (we don't let real malware samples spread outside of our environment) and time-efficient solution (experiments on physical machines would take significantly longer). We find the best features for distinguishing between ten predefined malware families and ten types. However, our models should be simple enough so that we can build a connection between low-level and high-level activity in the future work. Therefore, we may choose less accurate but simpler models to make analysis easier. Our initial hypothesis predicts that since malware types and families have a valuable difference in high-level behavior, we might be able to find distinctive low-level behavior patterns among malware categories. In the future work, we will test our models on the dataset of newer malware in order to check their capabilities against previously unknown (as for the models) malware. Our second hypothesis is that since malware families are assigned according to their particular functionality (e.g. exploiting of a certain vulnerability), they might generate more explicit activity that allows distinguishing better between families than between types.

The remainder of the paper is arranged in the following order: Section 2 contains State of the Art, Section 3 describes our methodology, Section 4 describes our results, Section 5 presents analysis of the results achieved, Section 6 presents a series of short remarks, conclusions, and a projection of future work.

2. State of the art

As was written above, in order to perform appropriate counteractions (to prevent) or postactions (to recover), we need additional information about malware category. With knowledge about malware types, we can apply appropriate defense mechanisms: e.g. in order to protect against Ransomware, we should keep an up-to-date backup of the data, while defense against self-replicating (Viruses) malware could be implemented with a thorough managing of a network traffic and removable media. In addition to knowledge about malware type, knowledge about malware family can help to set up appropriate defense mechanisms. Moreover, information about malware family can serve well in incident response actions: proper definition of malware family points to the potentially affected system components.

Many authors have performed research on malware classification. Different techniques and features are used to classify unknown malware into known malware categories or to detect outliers and perform a thorough analysis of such anomalies. For example, the authors of Kong and Yan (2013) combined different types of malware attributes (opcodes, API calls, flags, registers etc) in order to classify malware into 11 families. They used discriminant distance metric learning and pairwise graph matching in ensembled classifier to create an efficient framework that is capable of detecting previously unknown samples. Authors of Tian et al (2008) used a length of functions for classifying Trojans into 7 different families. They created pretty fast (*O(n)* training and classification time) and relatively accurate (around 80% average accuracy) method for malware classification. They also warn, that their approach might not be as successful on other malware types such as Viruses, where malicious code is difficult to extract. The same authors in their newer paper (Tian et al., 2010) used API calls and their parameters as features for malware detection, and classification of 10 malware families. They managed to achieve up to 97% accuracy in malware detection, and up to 95% accuracy in malware classification.

Nevertheless, malware analysis always challenges. Authors of Branco et al (2012) did a thorough review of anti-debug, disassembly and VM techniques on the dataset of more than 4 million malware samples. As was shown, around 34% of malware is packed, while most of the packers listed in the paper contain some kind of anti-debug or anti-reverse engineering techniques. Moreover, among samples considered nonpacked more than 68% contain obfuscation, 43% contain anti-debugging and 12% contain anti-disassembly techniques. This gives a clear view of a need of advanced dynamic analysis. However, more than 81% contain anti-VM techniques. The presence of anti-VM techniques might cause some problems for dynamic analysis. The authors didn't mention how they created their dataset and how the distribution of anti-techniques might be different from a real world. For example, Symantec published a paper where they claim that around 18% of malware stop execution when detected while being launched on a virtual machine (Wueest, 2014). Also, they say that a significant amount of organizations were planning to use server virtualization by the end of 2015. This means that malware may run on virtual machines or even created specifically to act on VMs and use their vulnerabilities (Wueest, 2014). Thus dynamic malware analysis, which is often performed in virtualised environments (Gandotra et al., 2014), is a relevant and promising research topic.

Dynamic malware analysis could be done on the different levels regarding to how “far” the features are from the hardware. For example, API calls or network analysis can be considered as high-level features and were proved to be reliable features for malware analysis (Gandotra et al., 2014). On the other hand, memory activity (Banin et al., 2016) (Kawakoya et al., 2013), opcodes (Khasawneh et al., 2015; Kirat et al., 2014), file system activity (Kirat et al., 2014) and other hardware-based features (Ozsoy et al., 2016) can be used for malware detection and considered as low-level features.

When studying memory access traces, we use Intel Pin (IntelPin, 2017), a binary instrumentation tool that allows us to capture detailed information about every single access to memory. Malware analysis usingf Intel Pin was described earlier in Kawakoya et al (2013) and Banin et al (2016). Authors of Kawakoya et al (2013) tested model in a virtual environment and in a real environment with installed Windows XP or Xen Linux. They recorded the following features: API calls (both system or user) if any file or folder was modified, calls which created symbolic or hard links, calls and arguments passed to function *exec()*, and instructions that executed memory operations such as *read* and *write*. While in our paper we target separate memory access operations generated by separate opcodes, authors of Kawakoya et al (2013) used *basic blocks* of a program. The basic block is a sequence of instructions executed between conditional branching instructions. Together with other properties of memory access operations in the basic blocks, authors studied memory range, the presence of certain operations and the size of transferred data. Using records of the execution trace, it was possible to create regular expressions and security policies, to use them for malware detection. Finally, 100% detection rate was achieved for both Windows and Linux on the original and obfuscated malware samples. Authors also state that their approach allows one accurately detect malware, and achieved 93.68% code and path coverage of input-dependent executables. Ensemble learning method for malware detection with a use of a number of properties extracted with Intel Pin (IntelPin, 2017) was proposed in the Khasawneh et al (2015). Authors used the frequency of opcode occurrence, presence of a particular opcode, difference between the frequency of opcode in malware and benign executables, distance and presence of memory references, and total number of load and store memory operations as well as branches. For each sample in their dataset, they recorded a generalized feature vector for every ten thousands of executed instructions, reaching up to 95.9% of classification accuracy.

Paper Banin et al (2016) is worth special attention, since the authors used memory access traces for malware detection. Their initial goal was to show that low-level features (memory access patterns in their case) are applicable for malware detection tasks. They used a virtualized environment and Intel Pin (IntelPin, 2017) to record memory access operations produced by malicious and benign executables. Using Machine Learning, they achieved more than 98% of accuracy for malicious against benign classification. Even though they used a different feature selection method, this work created a baseline for our research. However, the authors of Banin et al (2016) didn't take into account malware categories present in their dataset which points to one of the main goals of our research: testing whether memory access operations applicable for multinomial malware classification.

Additionally, behavior analysis has its disadvantages: it might be vulnerable to anti-emulation, when malware is created with capabilities not to reveal it's functionality in emulated and virtual environments. Even though it might not be the biggest problem, behavior based detection methods have another disadvantage: malware cannot be detected being executed (Rudd et al., 2017). The speed of detection depends on the features used for detection. E.g. if we use n-grams (or 1-g) of API calls (or any other high-level event), our detection system will not make a decision before a certain API call is executed. However, single API call invokes the execution of many (rough analysis allows us to say hundreds) of opcodes with different parameters. So it is hypothetically possible to detect a needed high-level event before it is completed, since opcodes provide better data granularity. This is yet another reason to study low-level features in malware context. However, we need significantly more storage to store information about executed opcodes and their parameters than for API calls: from what was said above, it is easy to see that amount of data (to store and to analyze) can be larger in several magnitudes larger. In it's turn, memory access sequence takes less space and can be stored as a sequence of binary elements (*R* for *read* and *W* for *write* operation). It therefore simplifies process of pattern search and matching. Memory accesses potentially provide granularity better than opcodes: it is therefore possible to detect execution of opcodes sequence before it is finished. Moreover, since not all opcodes generate memory activity (Banin et al., 2016) this method should create smaller performance overhead while giving detection system more time to make a decision.

Taking into account the presence of anti-debug techniques mentioned above, as well the contiguous growth of virtualization solutions' market share (PRNewswire, 2016) our research can aid for out-of-VM security solutions. Since many virtualized solutions might contain sensitive information, vendors won't always have access to the systems, while malware capable of escaping virtual environment can undermine not only host system (Wueest, 2014), but other guest systems as well. Methods that allow monitoring the state of guest system from outside a virtual machine can improve the security of a virtualized environment without breaking ethical and privacy policies. In their paper (Hua and Zhang, 2015, Gu et al., 2012), the authors designed and implemented a VMM-based hidden process detection system. Their system is placed outside of a protected virtual machine and interacts with a virtual machine manager. During the virtual machine introspection they inspect low-level state of protected virtual machine and track presence of hidden processes or lack of critical processes. In Gu et al. (2012), authors created a system, that can detect OS type with a use of fingerprints extracted from virtual machine memory without false positives. Authors of

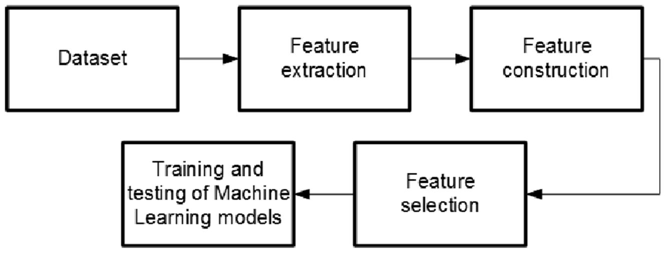

Fig. 1. Simplified experimental flow.

Ozsoy et al (2016) proposed Malware-Aware Processors, where they suggest hardware-based online monitoring of malware. As a features for malware detection they use frequency of memory read and write operations, memory address histogram, frequency and existence of opcodes and instruction categories. Moreover, hardware manufacturers tend to invest in hardware-based security solutions (Ozsoy et al., 2016). Because of everything written above, the results of our research may contribute to different aspects of digitized society, from improving the security of operations to helping security measures agree with ethical and privacy considerations.

3. Methodology

In this section we describe our experimental flow and explain details about dataset, feature selection, chosen machine learning methods, and hardware. We also outline several phases of analysis that we perform on the achieved results.

In our study we followed the scheme provided in Fig. 1. We first created two datasets: one for malware families, and another for malware types. We then extracted features by recording memory access traces from each sample. Afterwards, we constructed n-grams of a size 96 for each sample. Lastly, we performed feature selection and trained Machine Learning Models. A detailed scheme of our experimental flow is shown in Fig. 2 and described below in this Section.

3.1. Dataset

The initial dataset was created under the initiative of the Testimon (T. R. Group, 2017) research group and consisted of 400 k malware samples. All malware samples were PE32 executables. This dataset was previously used for research purposes and described in more details in Shalaginov et al (2016). The malware that we used in our research was selected under the following criteria: the file should not be a DLL (only EXE files), it should not contain AntiDebug or AntiVM features, it contains GUI and files were sorted ascending according to a size of a file. Information about file type, AntiDebug, AntiVM and GUI were gained through the use of peframe (Amato, 2016). As our research is aimed on proof of concept, dealing with DLLs and AntiDebug features was not a case, so we eliminated potential problems by filtering such things out (though we argue that study of AntiDebug influence will be an important part of future work). As we described in previous sections we can skip dealing with AntiVM, however we should remember this for assessing the results. We selected malicious files where *peframe* detected a presence of GUI for a simple reason: malware samples without GUI can fall into idle mode soon after starting, making it hard to collect enough data and increasing the time of dynamic analysis (Banin et al., 2016). The presence of GUI should not significantly influence the results because it is present in every single sample. Because if something influences every sample we might assume, that results will be equally biased. We also decided to select small files because our goal was to prove a presence of features that can help to distinguish between 10 malware categories. If we used big files with long execution times it would be more likely to find a unique feature for each malware sample, which is good for classification accuracy, but won't contribute to understanding our findings and won't prove that our hypothesis works.

The main goal of this research is to check how memory access patterns can aid in malware classification. In order to do this, we created two datasets: the first contain 10 malware types and the second contain 10 malware families. We decided to choose the following malware types: *backdoor, pws, rogue, trojan, trojandownloader, trojandropper, trojanspy, virtool, virus, worm*. Additionally we chose the following malware families: *agent, hupigon, obfuscator, onlinegames, renos, small, vb, vbinject, vundo, zlob*. The reason for such choice was that these types and families were prevalent in our malware dataset. We tried to create a balanced dataset, so each malware category contained around 100 samples. However, not all of the files launched, so they were rejected before analysis. Our datasets contained 952 files for malware types and

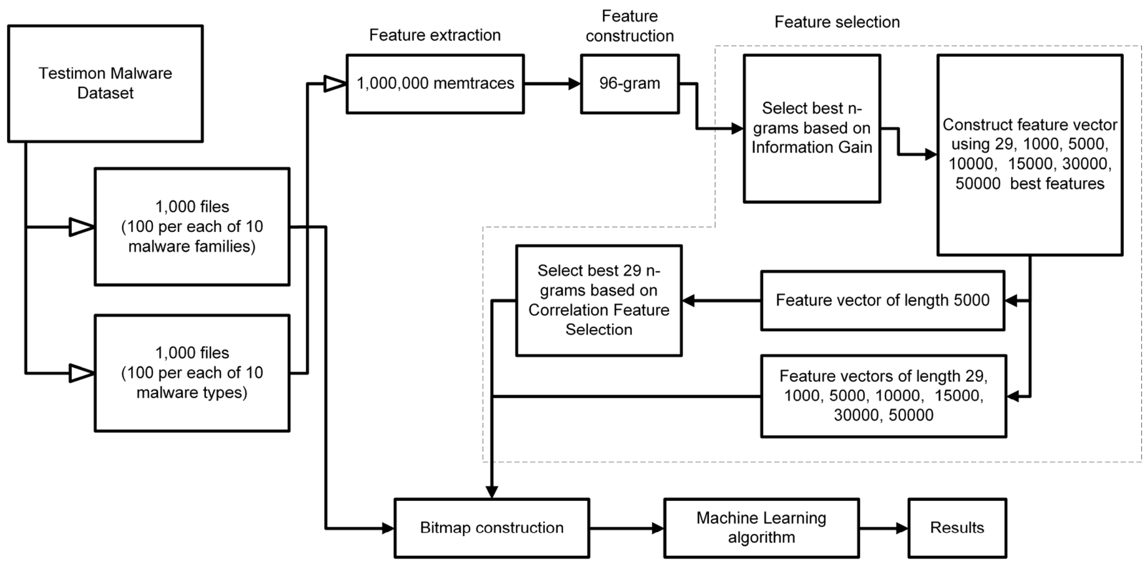
Testimon Malware Dataset
1,000 files (100 per each of 10 malware families)
1,000 files (100 per each of 10 malware types)
Feature extraction
1,000,000 memtraces
Feature construction
96-gram
Feature selection
Select best n-grams based on Information Gain
Construct feature vector using 29, 1000, 5000, 10000, 15000, 30000, 50000 best features
Feature vector of length 5000
Select best 29 n-grams based on Correlation Feature Selection
Feature vectors of length 29, 1000, 5000, 10000, 15000, 30000, 50000
Bitmap construction
Machine Learning algorithm
Results

Fig. 2. Detailed experimental flow.

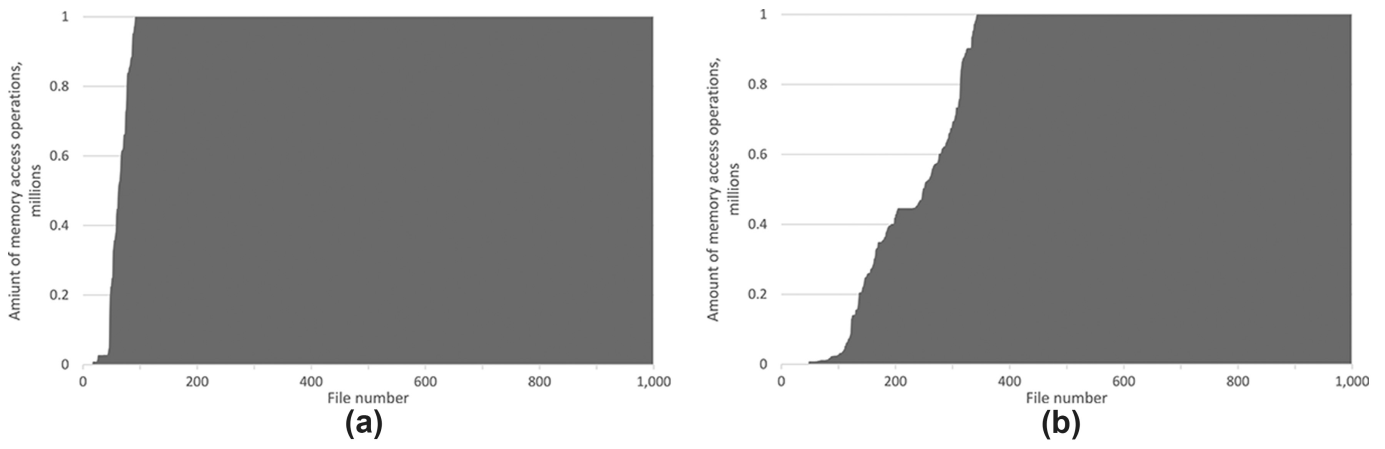

Fig. 3. Memory access operation numbers for families and types.

983 files for malware families. We can therefore assume that our datasets are approximately balanced, and we don't need to analyze the influence of sample distribution on the final results.

3.2. Feature construction and selection

The first task is to record a sequence of the first 1,000,000 (one million) memory access operations performed by an executable. We record only the type of operation: *W* for *Write*, and *R* for *Read*. This length of a sequence was chosen based on results from previous research (Banin et al., 2016) where it provided the best accuracy for malware against benign classification. We also found, that not all the executables can produce a greater or equal amount of memory access operations. On the Fig. 3, the charts show the distribution of memory access operations gained from types and families datasets. We analyze all samples regardless the amount of memory access operations they produced. We do not truncate or fill missing operations with zeros: instead we work with the available amount of data. We explain our choice in the following paragraphs.

As was stated in Section 2 in some scenarios it might important to detect malicious process as fast as possible. Also in Section 1 we stated that our models should be simple enough to perform high level analysis of the findings in future work. So we need to find features that do not rely on how long the process is executed. The sequence of memory access operations is later on divided into overlapping n-grams of a length 96. An n-gram is a sub-sequence of length *n* of original sequence of length *L*. For example if an original sequence of length $L = 6$ *[WRWWRW]* is divided into n-grams of length $n = 4$ (4-g) then our n-grams set will look the following way: {WRWW,RWWR,WWRW}. Each n-gram starts from the second element of the previous one: they overlap on the n-1 elements as it is shown on the Fig. 4.

This n-gram size was also chosen due to findings published in previous research (Banin et al., 2016). We might notice, that out of 2^{96} possible n-grams we have to select the most relevant, thus significantly reduce feature space. N-grams are later stored for feature selection. Some malware researchers use file-wise frequency of features as feature values: file-wise frequency is a ration between number of observations of a certain feature in the file and overall number of all observation of all features. In our case n-grams are stored without file-wise frequency for two reasons. First, we can not guarantee the amount of memory access operations (how long the file will run before stop) produced by a random file. Second, if we are able to find unique memory access patterns that comply with our classification goal, we can continue with more in-depth analysis of results, provide better high-level description of low-level findings.

As numbers of features are too big to just simply feed them to the machine learning models additional feature selection methods are therefore required. We obtained more than 15 M of features for malware families dataset, and more than 6 M of features for malware types dataset. The numbers are big but not surprising: sequence of memory access operations is basically a binary sequence with two possible elements *R* or *W*, so each sequence on 1 M operations can potentially contain up to 1M-96 + 1 different 96-g. However, during preliminary experiments we found that such amounts of data are too big to use in general-purpose machine learning libraries. Also models built on high-dimensional data provide results that are harder to interpret by human analysis. We used a feature selection method based on Information Gain. Information gain is an attribute quality measure based on class entropy and class conditional entropy given the value of attribute (Kononenko and Kukar, 2007). We ranked all features according to their Information Gain and selected 50,000 with highest rank. We chose this number for several reasons. First, is computational complexity while training ML models, and second because we know from previous research (Banin et al., 2016), that we need around 400 features for class to get good classification accuracy. This reason is however more empirical since in this paper we study multinomial classification. After feature selection we use several conventional Machine Learning models in order to check our hypothesis, quality of feature selection and get some additional findings. To study the classification performance dependency on number of features we also selected best 5, 10, 15 and 30 thousands of features. We also performed correlation-based feature selection (CfSubsetEval (Hall, 1998) from Weka (W. University, 2016)) on the best 10,000 features. This method selects features based on their correlation between class and other features. In simple words: the best feature is the one that correlates with classes and does not correlate with other features (does not bring redundant information). This gave us the 29 best features, and we will show in Section

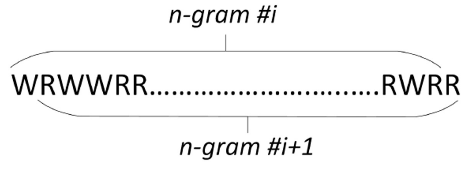

Fig. 4. Example of overlapping n-grams.

Table 1
Classification performance for families and types datasets.

Number of features	Accuracy for families						Accuracy for types					
	kNN	RF	J48	SVM	NB	ANN	kNN	RF	J48	SVM	NB	ANN
29	0.282	0.274	0.265	0.246	0.232	0.271	0.201	0.204	0.2	0.201	0.198	0.206
5000	0.806	0.802	0.800	0.651	0.646	N/A	0.642	0.637	0.623	0.468	0.430	N/A
10,000	0.802	0.807	0.793	0.607	0.599	N/A	0.663	0.678	0.648	0.461	0.412	N/A
15,000	0.800	0.802	0.795	0.607	0.600	N/A	0.665	0.661	0.645	0.455	0.415	N/A
30,000	0.814	0.818	0.814	0.591	0.606	N/A	0.673	**0.688**	0.666	0.419	0.412	N/A
50,000	0.833	**0.845**	0.827	0.648	0.572	N/A	0.668	0.675	0.665	0.375	0.386	N/A
CfSbased 29 features	**0.784**	0.781	0.769	0.740	0.724	0.783	**0.668**	**0.668**	0.626	0.584	0.498	0.617

Bold emphasizes which ML algorithm achieved best results for families and types datasets. Also it is important to show, what was achieved by ML algorithms with 29 features.

4 that they perform almost as good as thousands of features. However, it was impossible to get a larger number of features from correlation-based feature selection due to computational issues. On the Table 1 and on the Fig. 5 we also present results for best 29 features selected by Information gain, as so we can compare feature selection performance on similar feature numbers. We omit results for feature numbers between 29 and 5000 to simplify presented material as they don't add any significant information to the reader.

3.3. Machine learning algorithms

As a machine learning (ML) methods we chose the following: k-Nearest Neighbors (kNN), RandomForest (RF), Decision Trees (J48), Support Vector Machines (SVM), Naive Bayes (NB) and Artificial Neural Network (ANN). The following parameters (default for Weka package) were used for ML algorithms: kNN used $k = 1$; RF had 100 random trees; J48 used pruning confidence of 0.25 and minimum split number of 2; SVM used radial basis as function of kernel; NB used 100 instances as preferred batch size; ANN used 500 epochs, learning rate 0.3 and a number of hidden neurons equal to half of the sum of a number of classes and a number of attributes. The results for ANN are shown only for the smallest amount of features since machine learning software Weka (W. University, 2016) was not able to finish training of such big neural networks. This fact can be explained by means of time complexity for training. According to S. for Machine Learning (2017) computational complexity of ANN is $O(nMPNe)$ where n is a number of input variables (size of a feature set), M number of hidden neurons, P number of output values (10 in our case, since we have 10 classes), N number of samples, e number of learning epochs. Artificial Neural Networks built by Weka by default has 1 hidden layer, when the number of hidden neurons is taken as $M = (P + n)/2$ and e equals 500. For the dataset with 29 features, 10 classes, around 1000 samples and 5-fold cross validation it took $5 \times 9 seconds \approx 45 seconds$ to train models. The time complexity in this case is $O(3 \cdot 10^8 operations)$.For example, for 5000 features the time complexity would be around $O(6 \cdot 10^{13})$ what will take roughly 10^5 times more time to complete a task which is not suitable for our purposes since $45s \cdot 10^5 \approx 52 days$ of training time.

We held our experiments on Virtual Dedicated Server (VDS) with Intel Core CPU running at 3.60 GHz, 4 cores, SSD RAID storage and 48 GB of virtual memory. As a main operating system Ubuntu 14.04 64bit was used. Additionally, MySQL 5.5, PHP 5.5.9 and VirtualBox 5.0.16 were used. Windows 7 32-bit was installed on the VirtualBox virtual machine as a guest OS. It is widely spread (Netmarketshare, 2016) and malware written for 32-bit OS's will run on 64-bit OS as well as well. We also met some virtualization problems and were not able to run VM with 64-bit OS installed.

3.4. Analysis

During the analysis stage we try to explain achieved results in terms of numbers and words. We perform two types of analysis: statistical and context using sub-categories of our two datasets (different than original 10 classes). In statistical analysis we look into per-category classification accuracy and use statistical measures to explain differences in performance of machine learning models for different malware categories. During context analysis we are seeking an understanding of classification performance with a use of malware functionality description. As a results of analysis we not only understand how distribution of subcategories influence on per-category classification accuracy, but also show how human understandable explanation of malware functionality can contribute to an explanation of malware classification performance.

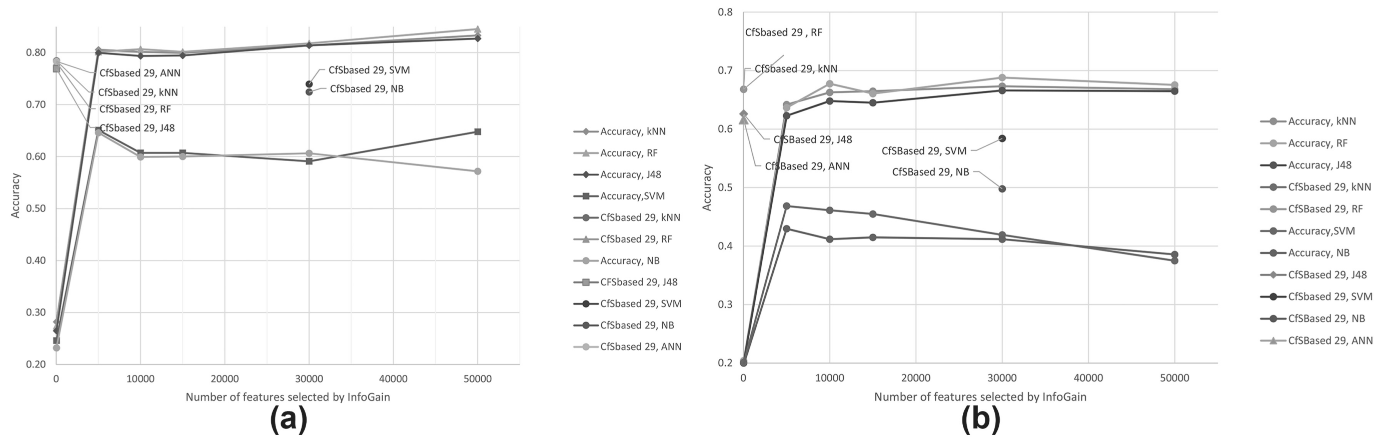

Fig. 5. Classification performance for families (a) and types (b) datasets.

With these findings we contribute to our future work where we are going to correspond low- and high-level activity and make results achieved with low-level features more understandable.

We also compare our results with results from a paper (Shalaginov et al., 2016) where authors used similar malware categories but did static analysis and used different ML algorithms.

4. Results

In this section we present results and key finding of our experiments. In order to test the quality of our ML models we used 5-fold cross validation. As a classification quality measure, we use accuracy: it allows us to compare results in this paper with results from previous study published in Banin et al (2016). It also shows how many instances have been correctly identified in our multinomial classification problem. In the Table 1 we present results for classification accuracy of different ML algorithms as a function of number of features. Each cell contains the accuracy that a certain ML method achieved with a given number of features. The last row shows accuracy that given ML method achieved with 29 features selected based on correlation (Hall, 1998). We separated it from other feature sets as here we used different feature selection method.

The results are also presented in Fig. 5. As we can see SVM and NB, in general, showed lower accuracy than other methods. This is interesting, since SVM performed pretty well in previous studies (Banin et al., 2016). Additionally, in general, neither SVM nor NB improve their performance with an increased number of features. We can also see other ML methods (kNN, RF and J48) slightly improve their performance as number of features increase. However, using 50,000 features instead of 5000 to gain a few extra percents of accuracy is not necessarily an efficient method, when our goal is to better understand how low-level features can be used for multinomial malware classification. Therefore we might put emphasis on a little bit less accurate but more understandable model. Because of this we decided to compare ML methods performance when only 29 features used. On the charts, these results are shown with separate points aligned to the most representative results on the horizontal axis. With 29 features (selected by corellation-based feature selection) given, ML methods such as kNN, RF and J48 show either small drop in performance or even some increase when compared to 5000 features. Other ML methods such as SVM and NB show significant grows of classification accuracy when given fewer features. One of the possible reasons could be that SVM is not originally designed for multinomial classification, it means that in order to deal with more than 2 classes it has to build several one-versus-all or one-versus-one classifiers. SVM is also known to have problems in so-called HDLSS datasets. High Dimension Low Sample Size dataset is a dataset, where the number of features is much bigger than the number of samples (it is true for most of our datasets, where number of samples is no bigger than 1000, while feature number starts from 5000). This fact was pointed out in different studies such as Ahn (2006) and Marron et al (2007); Rennie et al (2003). Naive Bayes classifier on its turn assumes that features are independent. But when we used Information Gain for feature selection, we can have a lot of potentially correlated features, thus Naive Bayes showed low classification performance in comparison to dataset where features where selected with respect to their mutual independence. Such behavior of Naive Bayes was studies in Rennie et al (2003). Also it is worth mentioning, that all the ML methods showed poor performance when 29 features selected by Information Gain were used.

As we are able to see, classification performance are better when our algorithms are used for distinguishing between malware families and worse for malware types. We can explain this fact by referring to Section 1, where we provided definitions for term malware family and malware type. From the definitions it is easy to understand, that since malware sample is put into malware type according to general functionality it might be harder to distinguish between such categories, since a goal that malware achieves can be achieved by different methods. On the other hand, malware families are about particular functionality, which means that methods used by samples within family should be more similar. This interpretation can be strengthened by the following observations: from malware families dataset we were able to extract more than 15 millions of uniques features, while types dataset gave us “only” 6 millions of such. It is worth mentioning that it took 1.66 h (5987 s) to run through more than 15 millions of features extracted from malware samples divided in families, and select 50,000 based on Information Gain. And it took 1.72 h (6192 s) to run through more than 6 millions of features extracted from malware samples divided in types, and select 50,000 based on Information Gain. In the next section we will provide more analysis of the achieved results and describe some valuable findings.

5. Analysis

In this section we analyze our findings. First, we analyze our results by means of statistics, e.g. we use our posterior knowledge of achieved accuracy and additional subcategories in order to explain why some malware categories are easier to classify than other. After we perform context analysis and try to show how human understandable description of malware categories can assist us in analyzing classification performance. Later we compare our results with results presented in Shalaginov et al (2016) since authors used similar malware categories for their research.

5.1. Statistical analysis

For the analysis we will focus on the classification results from kNN algorithm. As it can be seen from Table 1 kNN provided best classification accuracy for both families and types when given 29 features selected with correlation-based (Hall, 1998) features selection. Also we chose this feature set for deeper analysis since following a rule of a thumb “less is more” we think that smaller feature set is much easier to analyze and it complies with our goals from Section 1. The question about a trade-off between model complexity and accuracy is not properly studied in the literature. However, the authors of Canali et al (2012) state that best models usually rely on a few features.

For our analysis we performed the following steps.

1. We recorded per-sample classification results and created a table where information about classification of each sample in our dataset is stored.
2. To this table we added a column where information about additional subcategories of each sample is stored. For samples in the malware types dataset, we added information about malware families, and vice versa.
3. As table from Step 1 allows us to calculate per-family (or per-type) classification accuracy we examined the influence of additional subcategories on the efficiency of our machine learning model to detect a certain malware type or family.

In the following paragraphs we describe our analysis workflow in a more detailed manner.

As we obtained our results based on average from a 5-fold cross validation, we decided to run 5-fold cross validation 5 times. Each cross-validation was done with a different random seed value in Weka (W. University, 2016). This allowed us to make each sample to be in the test set (in other words - not used in model generating) more than once. We combined achieved results and took final information about classification result for a certain sample by the majority of results from all 5 runs. In order to analyze the influence of additional subcategory on classification performance, we calculated entropy and coefficient of unalikeability of subcategories for each malware family or type. The (informational) entropy (Shannon, 1948) is calculated as $H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$ where X is a variable (subcategory in this case), x_i is an i^{th} value of a variable, and $p(x_i)$ is a probability of a variable X to obtain value x_i. In simple words, entropy is often used to show randomness of a certain variable. It is also used in static malware analysis for detection of packers (Lyda and Hamrock, 2007). So in our case higher entropy will be a sign that certain category (type or family) are more diverse in terms of subcategories. In the matter of interest, we also used a coefficient of unalikeability (Perry and Kader, 2005). It is an index of qualitative variation that measures variance of a nominal attribute (like our subcategory). It is calculated as $u = \frac{\sum_{i \neq j} c(x_i, x_j)}{n^2 - n}$ where $c(x_i, x_j) = \left\{ \begin{matrix} 1, & x_i \neq x_j \\ 0, & x_i = x_j \end{matrix} \right\}$. It is a very simple coefficient, however it efficiently reflects variance of a nominal variable: if all the data are equal (variable obtain a certain value for all positions) than unalikeability is 1, and 0 if all positions are different. As we will show later unalikeability has a strong negative correlation with entropy. After looking at results from Section 4 and taking into account our initial hypotheses our first guess was that the more subcategories are found within a specific class the more difficult it is to generalize over that class. However pure number of subcategories will not reflect their real distribution, and that was an important reason to introduce some more advanced measures described above. On the Table 2 we present analysis of subcategory distribution on the classification accuracy.

As we can see, entropy in terms of subcategories in general is higher for malware types than for malware families. This can be easily explained by the fact that malware types samples are represented by higher number of subcategories. In order to illustrate findings from Table 2 we will use charts on Fig. 6. As we have written above, we can see a strong negative correlation between entropy and accuracy. However accuracy does not strongly dependent on neither unalikeability nor entropy. As we can see for both families and types datasets we can find classes with relatively high and low accuracy regardless the fact they share similar amount of subcategories and similar value entropy.

On the Table 3 we show Pearson's correlation between corresponding columns in Table 2 respectively. As we can see from these tables, accuracy is strongly affected by the number of subcategories or entropy in types dataset. This is yet another proof that families are assigned due to particular functionality, thus more alike within one family, and more diverse within several families. Table 3 also shows that entropy and coefficient of unalikeability has very strong (close to -1) correlation. However it is also worth to analyze several specific cases in order to understand the nature of different classification accuracy for different malware categories.

5.2. Context analysis

In this subsection we analyze how human understandable description of additional subcategories influence classification performance. Within each (families and types) dataset we compare categories with high and low per-category accuracy by means of their two most frequent subcategories.

First, we will take a look at families dataset. For example malware family *zlob* has high classification accuracy (0.99) and relatively low entropy (0.29). It belongs to two types (subcategories) such as *trojandownloader* and *trojan* with classwise frequencies of 0.95 and 0.05 respectively. However, another malware family *vbinject* has even lower entropy (0.08) but much lower accuracy (0.59). It also belongs to two types such as *virtool* and *trojan* with classwise frequencies of 0.99 and 0.01 respectively.

Second, let's take a look at types dataset. Malware type *virus* has relatively high entropy (5.42) and relatively high accuracy (0.81). Samples of this type belong to 55 different families (subcategories). And two most frequent are *small* and *radix* with classwise frequencies of 0.12 and 0.05 respectively. On its turn, malware type *rogue* has the highest accuracy of 0.86 with way lower entropy of 2.08. Samples of this type belongs to only 9 families, two most frequent of which are *fakexpa* and *internetantivirus* with classwise frequencies of 0.43 and 0.35 respectively. On the other hand, malware type *worm* has entropy (5.69) slightly higher than virus, but almost twice lower accuracy (0.43). Samples of worm type belongs to 63 families, two most frequent of which are *roram* and *kelvir* with classwise frequencies of 0.08 and 0.05 respectively.

In the first case we might admit, that trojandownloader and trojan families might have relatively similar behaviour, because first downloads and installs another malicious software, while others are trojans by itself. Yet they share a similar feature: they might look legitimate, and trick user to download and/or run them. On the other hand virtools are aimed on modification of other malicious software in order to hide them from antivirus software. At a first glance it might look like trojandownloaders and trojans are more similar than virtools and trojans. However in both cases trojans make up only small amount of all samples. This is a very important finding and we will return to it later.

Table 2
Accuracy (acc.), unalikeability (unalike.), entropy and number of subcategories (subN) for malware families (a) and types (b). Onlinega. stands for onlinegames, trojandr - for trojandropper, trojando. - for trojandownloader.

class	acc.	unalike.	entropy	subN
agent	0.56	0.23	2.43	8
vbinject	0.59	0.98	0.08	2
obfuscator	0.64	0.98	0.08	2
hupigon	0.69	0.88	0.34	2
vb	0.75	0.36	1.83	8
small	0.84	0.73	0.92	7
vundo	0.88	0.94	0.22	3
renos	0.91	1.00	0.00	1
onlinega.	0.99	1.00	0.00	1
zlob	0.99	0.90	0.29	2

(a) Families

class	acc.	unalike.	entropy	subN
worm	0.43	0.02	5.69	63
pws	0.54	0.06	4.50	40
trojan	0.54	0.12	4.14	37
trojandr.	0.62	0.22	3.35	26
backdoor	0.67	0.11	4.18	40
trojanspy	0.71	0.27	2.92	22
trojando.	0.74	0.27	2.75	20
virtool	0.77	0.24	2.53	15
virus	0.81	0.02	5.42	55
rogue	0.86	0.31	2.08	9

(b) Types

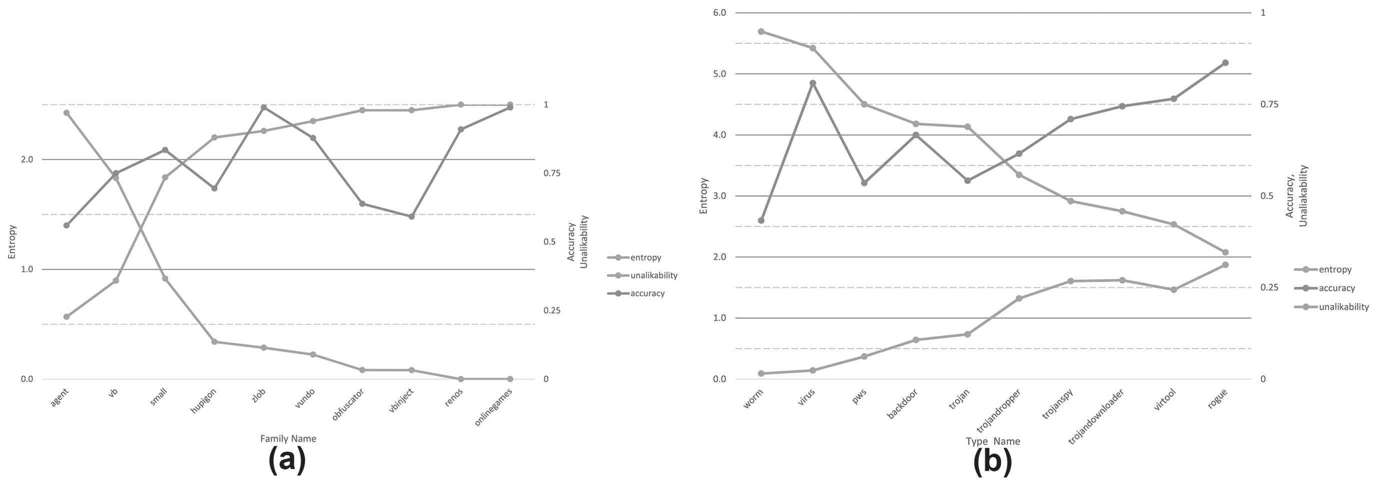

Fig. 6. Per-family (a) and per-type (b) entropy (left vertical axis), unalikeability and accuracy (right vertical axis).

Table 3
Correlation between accuracy (acc.), unalikeability (unalike.), entropy and number of subcategories (subN) for columns of Table 2a and b.

	acc.	unalike.	entropy	subN
acc.	1.0	0.43	-0.44	-0.37
unalike.	0.43	1.0	-1.00	-0.92
entropy	-0.44	-1.00	1.0	0.93
subN	-0.37	-0.92	0.93	1.0

(a) Families

	acc.	unalike.	entropy	subN
acc.	1.0	0.59	-0.60	-0.61
unalike.	0.59	1.0	-0.98	-0.96
entropy	-0.60	-0.98	1.0	0.99
subN	-0.61	-0.96	0.99	1.0

(b) Types

In the second case we should also study what our subcategories are. *Small* malware family are multipurpose malware, that is often used for downloading and executing additional files. They used in the initial infection of visitors to websites. They also tend to drop and use kernel mode driver for its purposes. *Radix* on its turn is a mass-mailing malware that propagates by send a copy of itself via e-mail with a use of its own SMTP engine. *FakeXPA* and *internetantivirus* are programs than pretend to scan systems for malware and display fake warning about malicious programs found on victim system. After that they ask you to pay for removing fake threats. There is no surprise that they have similar functionality and is a part of malware type *rogue*. *Roram* spreads via IRC channels. *Kelvir* also spreads via chat programs, but instead of IRC it uses MSN or Windows Messenger. To sum up this paragraph: *small* and *radix* are pretty different by functionality, while *fakexpa* and *internetantivirus* are way more similar. Yet malware type to which they belong are easy to generalize over. At the same time *roram* and *kelvir* are different only by the name of the chat program they use for proliferating. However, we might assume that our methodology is not capable of generalizing over such functionality.

5.3. Classification performance comparison

It is worth comparing our work with a paper by Shalaginov et al. (2016) where the authors used malware dataset with a similar malware categories. It the Table 4 per-category True Positive and False Positive rates from Shalaginov et al (2016) and our work are present. They did not include accuracy measure in their work, so we compare our results using True and False Positive rates. Authors of that work used a Neuro-Fuzzy approach for malware classification, while we will use results achieved by kNN because, as it was said earlier, it brought us best results in case of 29 features. As we can see, in most cases TP rate from our work is higher, and FP rate is somewhat lower than in Shalaginov et al (2016). However, for such

Table 4
Comparison of our results to the results from Shalaginov et al (2016).

Shalaginov et al	Family	vb	hupigon	vundo	obfuscator	agent	renos	small	onlinegames	vbinject	zlob
	TP rate	0.3595	0.8080	0.5405	0.1222	0.1633	0.3276	0.5229	0.6084	0.2076	0.4295
	FP rate	0.0226	0.2033	0.0233	0.1185	0.0341	0.0101	0.1397	0.0303	0.0261	0.0262
	Type	trojan	pws	trojando.	worm	virtool	backdoor	virus	rogue	trojandr.	trojanspy
	TP rate	0.6084	0.1954	0.1385	0.1608	0.2112	0.3392	0.0857	0.0000	0.0744	0.0769
	FP rate	0.5220	0.0432	0.0517	0.0097	0.0614	0.1528	0.0098	0.0000	0.0193	0.0152
Our work	Family	vb	hupigon	vundo	obfuscator	agent	renos	small	onlinegames	vbinject	zlob
	TP rate	0.75	0.695	0.879	0.639	0.56	0.91	0.835	0.99	0.586	0.99
	FP rate	0.028	0.036	0.02	0.061	0.015	0.009	0.046	0	0.02	0.003
	Type	trojan	pws	trojando.	worm	virtool	backdoor	virus	rogue	trojandr.	trojanspy
	TP rate	0.542	0.535	0.745	0.433	0.765	0.667	0.808	0.864	0.615	0.71
	FP rate	0.046	0.046	0.026	0.049	0.033	0.057	0.038	0.02	0.026	0.029

categories as *hupigon* or *trojan* our results are worse. Authors of paper in interest used different features: they used static features from PE header. And Table 4 is yet more proof that static analysis may be outperformed by dynamic analysis. Specifically we can explain our relative success by the fact, that malware categories (both families and types) in use are assigned based on the *functionality* (dynamic characteristics), so our dynamic approach may be more suitable for such tasks. It is also worth mentioning that they had bigger dataset, so our results might be influenced as well if we increase number of samples. Authors of Shalaginov et al (2016) used up to 20 features to complete their goals, what makes their work useful for malware analysts. This fact ensures us that using smaller (even though a bit less accurate) models gives better contribution for the scientific community. Finally, they show overall classification accuracy around 39.6% while ours is around 78.4%.

As we shown in this Section, subcategories might be a key to explain classification performance of our malware classification approach, but only of the many. As we shown in previous paragraphs, subcategories does not directly influence classification accuracy neither by means of variety, nor by their amount. However analysis of subcategories pointed us to a very important finding: our approach is better in detecting and generalizing over a certain types of behavior, and worse for others. It means that in order to improve classification accuracy, in the future work we have to study how usage of context as ground truth will influence classification accuracy. We will elaborate on this more in the next section.

6. Conclusion and future work

In this paper we showed that patterns of memory access operations can be used for malware classification. We tested our method over the datasets with malware types and families. At a first glance, an achieved accuracy of 0.688 and 0.845 is not that high, however it is important to remember that in our case we have 10 classes and random guess on the balanced dataset will not exceed 0.1. So our results are way better than theoretical random guess generator. It is also important to notice that we went down from millions of potential features to 50,000, and from them extracted 29 best features that allowed us to have compact yet relatively accurate models.

We have also shown that our approach performs better in some conditions, while worse in others. As we stated before, ground truth and context might help to improve classification accuracy. As a ground truth we might use high-level activity, to do so we should study what high-level activity (e.g. which API calls) are represented by certain memory access patterns. This study might also bring additional meaning to memory access sequences, because now a 96-g similar to *WWWRWRW* ... *WWRWW* does not say anything to a human analyst. While context analysis might involve capturing opcodes as well as the content of the memory. It can also be useful to use variative n-gram length, however, it might be extremely time and memory consuming. We also plan to evaluate models built with our approach against previously unknown, or assumed to be new, malware samples Nevertheless, our research topic is shown as promising, capable of bringing valuable findings and worth of further studies.

References

Ahn, J., 2006. High Dimension, Low Sample Size Data Analysis.

Amato, G., 2016. Peframe. https://github.com/guelfoweb/peframe. (Accessed 27 October 2016).

Banin, S., Shalaginov, A., Franke, K., 2016. Memory Access Patterns for Malware Detection, Norsk Informasjonssikkerhetskonferanse (NISK), pp. 96–107.

Branco, R.R., Barbosa, G.N., Neto, P.D., 2012. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. Black Hat.

Canali, D., Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., Kirda, E., 2012. A quantitative study of accuracy in system call-based malware detection. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis - ISSTA 2012, p. 122.

Cole, E., 2012. Advanced Persistent Threat: Understanding the Danger and How to Protect Your Organization. Newnes.

Cuckoo Sandbox, 2015. Cuckoo Sandbox: Automated Malware Analysis. https://www.cuckoosandbox.org/. (Accessed 15 April 2016).

Distler, D., Hornat, C., 2007. Malware analysis: an introduction. SANS Inst. InfoSec Read. Room 18–19.

Egele, M., Scholte, T., Kirda, E., Kruegel, C., 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. (CSUR) 44, 6.

Gandotra, E., Bansal, D., Sofat, S., 2014. Malware analysis and classification: a survey,. J. Inf. Secur. 5, 56.

Gu, Y., Fu, Y., Prakash, A., Lin, Z., Yin, H., 2012. Os-sommelier: memory-only operating system fingerprinting in the cloud. In: Proceedings of the Third ACM Symposium on Cloud Computing, ACM, p. 5.

Hall, M.A., 1998. Correlation-based Feature Subset Selection for Machine Learning [Ph.D. thesis]. University of Waikato, Hamilton, New Zealand.

Hua, Q., Zhang, Y., 2015. Detecting malware and rootkit via memory forensics. In: Computer Science and Mechanical Automation (CSMA), 2015 International Conference on. IEEE, pp. 92–96.

IntelPin, 2017. A Dynamic Binary Instrumentation Tool.

Kawakoya, Y., Iwamura, M., Shioji, E., Hariu, T., 2013. Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 123–143.

C. M. K. Kendall, Practical malware analysis, in: Black Hat Conference USA.

Khasawneh, K.N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., Ponomarev, D., 2015. Ensemble learning for low-level hardware-supported malware detection. In: Research in Attacks, Intrusions, and Defenses. Springer, pp. 3–25.

Kirat, D., Vigna, G., Kruegel, C., 2014. Barecloud: bare-metal analysis-based evasive malware detection. In: USENIX Security Symposium, pp. 287–301.

Kong, D., Yan, G., 2013. Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1357–1365.

Kononenko, I., Kukar, M., 2007. Machine Learning and Data Mining: Introduction to Principles and Algorithms. Horwood Publishing.

Lyda, R., Hamrock, J., 2007. Using entropy analysis to find encrypted and packed malware. IEEE Secur. Priv. 5.

Marron, J.S., Todd, M.J., Ahn, J., 2007. Distance-weighted discrimination. J. Am. Stat. Assoc. 102, 1267–1271.

Netmarketshare, 2016. Desktop Operating System Market Share. https://www.netmarketshare.com/operating-system-market-share.aspx?Ωqprid=10\&qpcustomd=0\&qpcustomb=2016. (Accessed 22 November 2017).

Ozsoy, M., Khasawneh, K.N., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., Ponomarev, D., 2016. Hardware-based malware detection using low-level architectural features. IEEE Trans. Comput. 65, 3332–3344.

Perry, M., Kader, G., 2005. Variation as unalikeability. Teach. Stat. 27, 58–60.

Prakash, A., Venkataramani, E., Yin, H., Lin, Z., 2015. On the trustworthiness of memory analysis #x2014;an empirical study from the perspective of binary execution. IEEE Trans. Dependable Secure Comput. 12, 557–570.

PRNewswire, 2016. Virtual Desktop Infrastructure Market to See 27.35% Cagr Driven by Byod to 2020. http://www.prnewswire.com/news-releases/virtual-desktop-infrastructure-market-to-see-2735-cagr-driven-Ωby-byod-to-2020-566513421.html. (Accessed 29 September 2017).

Rennie, J.D., Shih, L., Teevan, J., Karger, D.R., 2003. Tackling the poor assumptions of naive bayes text classifiers. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp. 616–623.

Reuters, 2017. Ukraine's Power Outage Was a Cyber Attack: Ukrenergo. https://www.reuters.com/article/Ωus-ukraine-cyber-attack-energy/ukraines-power-outage-was-a-cyber-Ωattack-ukrenergo-idUSKBN1521BA.

Rudd, E., Rozsa, A., Gunther, M., Boult, T., 2017. A survey of stealth malware: attacks, mitigation measures, and steps toward autonomous open world solutions. IEEE Communications Surveys & Tutorials.

S. For Machine Learning, Time Complexity: Graph & Machine Learning Algorithms, 2017. https://github.com/guelfoweb/peframe. (Accessed 23 November 2017).

Saeed, I.A., Selamat, A., Abuagoub, A.M., 2013. A survey on malware and malware detection systems. Int. J. Comput. Appl. 67.

Sathyanarayan, V.S., Kohli, P., Bruhadeshwar, B., 2008. Signature generation and detection of malware families. In: Australasian Conference on Information Security and Privacy. Springer, pp. 336–349.

Schiffman, M., 2010. A Brief History of Malware Obfuscation: Part 2 of 2.

Shalaginov, A., Grini, L.S., Franke, K., 2016. Understanding neuro-fuzzy on a class of multinomial malware detection problems. In: Neural Networks (IJCNN), 2016 International Joint Conference on. IEEE, pp. 684–691.

Shannon, C.E., 1948. A mathematical theory of communication, part i, part ii. Bell Syst. Tech. J. 27, 623–656.

T. R. Group, 2017. Testimon Research Group. https://testimon.ccis.no/.

Tabish, S.M., Shafiq, M.Z., Farooq, M., 2009. Malware detection using statistical analysis of byte-level file content. In: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics. ACM, pp. 23–31.

The Verge, 2017. The Petya Ransomware Is Starting to Look like a Cyberattack in Disguise. https://www.theverge.com/2017/6/28/Ω15888632/petya-goldeneye-ransomware-cyberattack-ukraine-russia.

Tian, R., Batten, L.M., Versteeg, S., 2008. Function length as a tool for malware classification. In: Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on. IEEE, pp. 69–76.

Tian, R., Islam, R., Batten, L., Versteeg, S., 2010. Differentiating malware from cleanware using behavioural analysis. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. IEEE, pp. 23–30.

Uppal, D., Sinha, R., Mehra, V., Jain, V., 2014. Malware detection and classification based on extraction of api sequences. In: Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, pp. 2337–2342.

W. University, 2016. Weka: Data Mining Software in Java. http://www.cs.waikato.ac.nz. (Accessed 30 October 2017).

G. Hoglund, What APT Means To Your Enterprise.

Wueest, C., 2014. Threats to virtual environments, symantec research. Mountain view. Symantec.