Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-024-00513-5

ORIGINAL PAPER

Classification and online clustering of zero-day malware

Olha Jurečková[1] · Martin Jureček[1] · Mark Stamp[2] · Fabio Di Troia[2] · Róbert Lórencz[1]

Received: 31 July 2023 / Accepted: 30 December 2023

Abstract
A large amount of new malware is constantly being generated, which must not only be distinguished from benign samples, but also classified into malware families. For this purpose, investigating how existing malware families are developed and examining emerging families need to be explored. This paper focuses on the online processing of incoming malicious samples to assign them to existing families or, in the case of samples from new families, to cluster them. We experimented with seven prevalent malware families from the EMBER dataset, four in the training set and three additional new families in the test set. The features were extracted by static analysis of portable executable files for the Windows operating system. Based on the classification score of the multilayer perceptron, we determined which samples would be classified and which would be clustered into new malware families. We classified 97.21% of streaming data with a balanced accuracy of 95.33%. Then, we clustered the remaining data using a self-organizing map, achieving a purity from 47.61% for four clusters to 77.68% for ten clusters. These results indicate that our approach has the potential to be applied to the classification and clustering of zero-day malware into malware families.

Keywords Malware classification · Online clustering · Static analysis · Zero-day malware

1 Introduction

Malware is one of the most significant security threats today, which includes several different categories of malicious code, such as viruses, trojans, bots, worms, backdoors, spyware, and ransomware. The number of new malicious software is growing exponentially. Therefore, malware detection is an important issue in cyber security, which is a key area to combat these threats. Every day, approximately 560,000 new malware samples are detected, according to the AV-Test Institute [1]. Due to a large amount of new malware, detailed manual analysis of each one is impractical. Therefore, automatic categorization of malware into groups corresponding to malware families is necessary.

Antivirus companies frequently keep a knowledge base of the behavior of malware families. Samples of the same group share a lot of code and exhibit similar behaviors, making them variants. Such samples are similar to each other in terms of similarity metrics that can also be learned to improve classification accuracy [2].

Malware detection techniques are generally divided into two categories: signature-based and anomaly-detection techniques [3]. Signature-based detection uses a set of predefined signatures, typically sequences of bytes in the malware code, to determine whether or not a scanned software program is malicious. The signature-based method compares the program's content with known signatures, and if a match is found, the program is reported as malicious. The signature-based approach's main limitation is its inability to detect newly developed (zero-day) malware, which are emerging threats previously unknown to the malware detection system, as well as evolving threats like metamorphic and polymorphic malware [4].

✉ Olha Jurečková
jurecolh@fit.cvut.cz

Martin Jureček
martin.jurecek@fit.cvut.cz

Mark Stamp
mark.stamp@sjsu.edu

Fabio Di Troia
fabio.ditroia@sjsu.edu

Róbert Lórencz
robert.lorencz@fit.cvut.cz

[1] Faculty of Information Technology, Czech Technical University in Prague, Prague, Czechia

[2] Department of Computer Science, San Jose State University, San Jose, CA, USA

Published online: 12 February 2024

Machine learning technologies are becoming more popular and are also being introduced into malware analysis and malware detection. Today, malware can be identified using one of three methods: static analysis, dynamic analysis, or hybrid analysis. Static analysis is a method of examining malware without running it. This is typically accomplished by analyzing the code of a binary file to comprehend its functionality and identify any malicious activity. Dynamic analysis involves executing the malware sample in a safe setting, like a sandbox, and watching its behavior in real-time. It is necessary to continuously monitor the malware's file system, registry, and network activity to detect any malicious behavior, such as data exfiltration or unauthorized connections to remote servers. Dynamic and static analysis components are combined in hybrid methods [5].

Malware classification is the process of categorizing malware samples into previously studied and known families. On the other hand, malware clustering divides unlabeled data into different clusters so that similar data fall into the same cluster and dissimilar data fall into different clusters. Clustering algorithms have been used to detect zero-day malware, i.e., previously unknown malware [6]. The groups formed through classification or clustering methods are then distributed to malware analysts, which usually focus only on a few malware families. This grouping can save malware analysts a significant amount of time since they may manually analyze malware samples similar to those already analyzed.

Malicious and benign samples are represented using vectors of features extracted using static or dynamic analysis [5]. While static analysis is faster than dynamic analysis since it does not require running samples, dynamic analysis extracts more relevant features, such as system calls or network data, than those extracted from static analysis. Our work is based on the EMBER dataset [7], which contains features extracted from static analysis.

Zero-day malware is challenging to detect using traditional signature-based detection techniques since no signature for such malware was created and appended in the database of known signatures [8]. The detection of zero-day malware is also difficult for a detection system based on machine learning, which is more robust and can better adapt to new threats however is more prone to have a high false positive rate than the signature-based detection method.

We propose a new malware family classification and clustering system designed for the online processing of zero-day malware. Sample by sample is processed in real-time and assigned to existing or newly emerging malware families. According to the classification score of the multilayer perceptron, malware samples are classified into known malware families or determined to be clustered using self-organizing maps into new malware families. Using our approach, we classified 97.21% of streaming data with a balanced accuracy of 95.33%. Then, we clustered the remaining data using a self-organizing map, achieving a purity from 47.61% to 77.68% depending on the number of clusters. The innovative content of our work is as follows:

- we proposed the architecture of the classification and clustering system, which enables the handling of even zero-day malware,
- we experimentally proved that our system is efficient enough to classify or cluster significantly more than 560,000 samples, the average number of malware detected daily.

The rest of the paper is organized as follows: Sect. 2 reviews related works on malware family classification. In Sect. 3, we present three state-of-the-art online clustering algorithms used in the experimental part. Our proposed malware classification model is presented in Sect. 4. Section 5 provides an experimental setup, while the experiments description and the results are presented in Sect. 6. Conclusion and future work are given in Sect. 7.

2 Related work

The background of malware family classification and clustering that has been researched in the past is presented in this section.

The authors of [9] present a non-signature-based virus detection method based on Self-Organizing Maps (SOMs) that can detect files with viruses without knowing virus signatures. Their approach used structural information about the data contained in the executable file. The researchers also developed the program VirusDetector, which can determine whether or not a file is virus-infected. They used the SOM in an unusual way in that it was "trained" with n fractions of the same sample rather than n different samples of data, and it can reflect the presence of data in an executable that is somehow different from the rest.

In [10], the authors proposed a method for automatic analysis of malware behavior using clustering and classification. The authors monitored the malware binaries in a Sandbox environment and generated a sequential report of observed behavior for each binary. Rieck et al. used the CWSandbox monitoring tool for extracting API call names and parameters. The API call names and parameters were encoded into a multi-level representation called the Malware Instruction Set. The sequential messages were then put into a high-dimensional vector space where behavioral similarity could be assessed geometrically, allowing intuitive yet powerful clustering and classification methods to be designed. The embedded messages were then subjected to machine learning techniques for clustering, which enables identifying novel classes of malware with similar behavior and classification of behavior, which allows the assignment of malware to known classes of behavior. Their incremental method for behavior-based analysis is capable of processing the behavior of thousands of malware binaries daily.

The authors of [11] developed a categorization system for automatically grouping phishing sites or malware samples into families that share specific common characteristics. Their system combined the individual clustering solutions produced by different algorithms using a cluster ensemble. Zhuang et al. used the k-medoids clustering method and the hierarchical clustering algorithm as feature selection algorithms to extract different attributes of phishing emails.

In [6], authors describe a framework for malware detection that combines the accuracy of supervised classification methods for detecting known classes with the adaptability of unsupervised learning for detecting new malware from existing ones using a class-based profiling approach. The authors used a two-level classifier to solve the problem of the unbalanced distribution of classes due to a disproportionate number of benign and malicious network flows. Initially, a macro-level binary classifier isolates malicious streams from non-malicious ones. The multiclass classification technique was then also used to categorize malicious flows into one of the already existing malware classes or as a new malware class. The authors developed a class-based probabilistic profiling method to detect malware classes other than those in the training set. Comar et al. presented a tree-based feature transformation to handle the data imperfection issues in network flow data to create more informative non-linear features to detect different malware classes precisely.

The authors of [12] presented a method for the automatic classification of malware families using feed-forward Artificial Neural Networks. They resized and converted the malware binaries to grayscale images. Texture features are extracted using a Gabor wavelet with eight orientations and four scales. The authors used the Mahenhur Dataset, which contains 3,131 malware samples from 24 unique families. A total of 320 features were selected to train the malware using the neural network tool. The authors reported a classification accuracy of 96.35%.

The authors of [13] created a zero-day malware detection system that used relevant features obtained from static and dynamic malware analysis. The dataset used contains 3,130 portable executables (PE) files, including 1,720 malicious and 1,410 benign files. Malicious samples were collected from an online repository of Virus-Share, and the benign files were collected manually from System directories of successive versions of the Windows Operating system. The authors used an information gain method and ranker algorithm to select seven features from the feature set, which were then used to build a classification model using machine learning algorithms from the WEKA library. The authors used seven classifiers, IB1, Naive Bayes, J48, Random Forest, Bagging, Decision Table, and Multi-Layer Perceptron, for distinguishing malicious files from benign ones.

In [14], Radwan presented a method for classifying a portable executable file as benign or malicious using machine learning. The proposed method for extracting the integrated feature set, which used a static analysis method, was created by combining a few selected raw features from the PE files and a set of derived features. The author used a dataset of 5,184 samples, 2,683 of which were malware and 2,501 benign. The dataset was divided into two categories: raw sample dataset (53 features) and integrated dataset (74 features), which included derived and expanded features. Seven different machine learning classification models were used: k-nearest neighbors, Gradient boosted trees, Decision Tree, Random forest, File large margin, Logistic regression, and Naive Bayes. The classification algorithms are evaluated using the train test split method (70/30) and 10-fold cross-validation for splitting raw and integrated datasets.

In [15], the authors proposed a static malware detection technique using the classification method. Zhang et al. used a dataset released by EMBER, where most PE file samples are labeled malicious or benign. Then, using the detection results of Virus Total and K7 Antivirus Gateway (K7GW), the authors relabeled the malware data into several classes, each representing a type of malware. The malware classifiers are constructed using two linear and two ensemble decision tree models. The authors used linear models such as Support vector classifier and logistic regression, and the ensemble decision tree models are random forest and an efficient gradient boosting decision tree named Light gradient boosting machine. The ensemble decision tree models outperformed the other linear models, especially random forest.

The authors [16] proposed a new method for incremental automatic malware family identification and malware classification called MalFamAware, which is based on an online clustering algorithm. This method efficiently updates the clusters as new samples are added without having to rescan the entire dataset. BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) was used by the authors as an online clustering algorithm and was compared with CURE (Clustering using Representatives), DBSCAN, k-means, and other clustering algorithms. Depending on the situation, MalFamAware classifies new incoming malware into the corresponding existing family or creates a class for a new family.

In [17], the authors used self-organizing maps to generate clusters that capture similarities between malware behaviors. In their work, Pirscoveanu et al. used features chosen based on API calls. These features represent successful and unsuccessful calls (i.e., calls that have succeeded, resp. failed in changing the state of the system on the infected machine) and the return codes from failed calls. Then they apply principal component analysis (PCA) to reduce the set of features.

Using the elbow method and gap statistics, the authors then determined the number of clusters. Each sample was then projected onto a two-dimensional map using self-organizing maps, where the number of clusters equaled the number of map nodes. The authors used the dataset to create a behavioral profile of the malicious types, which was passed to a self-organizing map to compare the proposed clustering result with labels obtained from Antivirus companies via VirusTotal.[1]

In [18], the authors classified malware using continuous system activity data (such as CPU use, RAM/SWAP use, and network I/O). They also used SOFM (Self Organizing Feature Maps) to process machine activity data to capture fuzzy boundaries between machine activity and classes (malicious or benign). First, the authors used SOFM as a stand-alone malware classification method that uses machine activity data as input. In their paper Burnap et al. state that they trained two maps because it was difficult to separate clean files from malicious ones on one map due to the competitive nature of the SOFM. They used benign samples to train the "Good" map, and malicious samples were used to train the "Bad" map. The authors also mention that they created a voting system that gathers accurate classifications during counter-testing for each sequence provided in the maps. Testing with unseen data was accomplished by comparing the Best Matching Unit (BMU) output activity from each map for a given input vector. The authors then used the BMU output from the SOFM as a feature and combined the SOFM with an ensemble classifier built on a Logistic regression model. Finally, the authors' method demonstrated increased classification accuracy compared to classification algorithms such as Random forest, Support vector machines, and Multilayer perceptron.

3 Theoretical background

Cluster analysis or clustering is an unsupervised machine learning method of identifying and grouping a set of abstract objects into classes of similar objects (called clusters). Intuitively, data from the same cluster should be more similar to each other than data from different clusters. Sequential clustering algorithms are considered simple and fast and are among those that produce a single clustering as a result. In the following algorithms described in this section, all input data to be clustered are presented to the algorithms only once. Specifically, the algorithms are Online k-means (OKM), Basic Sequential Algorithmic Scheme (BSAS), and Self-Organizing Map (SOM). All these algorithms belong to the class of online clustering algorithms, and we applied them to cluster the samples into malware families.

3.1 Online k-means (OKM) algorithm

First, we introduce the online k-means (OKM) algorithm, also known as sequential k-means or MacQueen's k-means [19]. The sequential k-means algorithm sequentially clusters a new example and updates the centroid for that particular cluster. One disadvantage of the online k-means algorithm is that the number of clusters, k, must be determined in advance. OKM algorithm can be initialized in different ways, for example, by selecting the first k data points or randomly selecting k data points from the entire data set. The pseudocode for the online k-means algorithm is given in Algorithm 1 below [20].

Algorithm 1 Sequential k-means algorithm (OKM)

```
Input: a number of clusters $k$ to be created, a set of data points $X$
Output: a set of $k$ clusters
initialize cluster centroids $\mu_1, \ldots, \mu_k$ randomly
set the counts $n_1, \ldots, n_k$ to zero
repeat
    select a random point $x$ from $X$ and find the
    nearest center $\mu_i$ to this point
    if $\mu_i$ is closest to $x$ then
        increment $n_i$
        replace $\mu_i$ by $\mu_i + \frac{1}{n_i}(x - \mu_i)$
    end if
until interrupted
```

3.2 Basic sequential algorithmic scheme (BSAS)

The Basis Sequential Algorithmic Scheme (BSAS) [21] is a well-known clustering method in which all feature vectors are presented to the algorithm only once, and the number of clusters is not known a priori. The clusters are gradually generated as the algorithm evolves. The basic idea of BSAS is to assign each newly considered feature vector x to an existing cluster or create a new cluster for that vector depending on the distance to already created clusters.

The distance $d(x, C)$ between a feature vector x and a cluster C may be defined in several ways. We will consider $d(x, C)$ as the distance between x and the centroid of C. The BSAS has the following parameters: the dissimilarity threshold Θ, i.e., the threshold used for creating new clusters, and a number q, i.e., the maximum number of clusters allowed. When the distance between a new vector and any other clusters is beyond a dissimilarity threshold, and if the number of the maximum clusters allowed has not been reached, a new cluster containing the new presented vector is created. The value of the threshold Θ directly affects the number of clusters formed by BSAS. If the user chooses the too small value of Θ, then unnecessary clusters will be created, while if the user chooses the too large value of Θ, less than an appropri-

[1] https://www.virustotal.com

ate number of clusters will be formed. The pseudocode for the BSAS algorithm is given below in Algorithm 2.

```
Algorithm 2 Basic Sequential Algorithmic Scheme (BSAS)
Input: the dissimilarity threshold $\Theta$, the maximum allowed number of
    clusters $q$, and a set of data points $X$
Output: a set of clusters
initialize $m = 1$
select a random point $x_1$ from $X$
define the first cluster $C_m = \{x_1\}$
for each $x$ in $X \backslash \{x_1\}$ do
    find $C_k : d(x, C_k) = min_{1 \leq i \leq m} d(x, C_i)$
    if $d(x, C_k) > \Theta$ and $m < q$ then
        $m = m + 1$
        $C_m = \{x\}$
    else
        $C_k = C_k \cup \{x\}$
        update the centroid of $C_k$
    end if
end for
```

3.3 Self-organizing map (SOM)

A self-organizing map (SOM) was proposed by Finnish researcher Teuvo Kohonen in 1982 and is, therefore, sometimes called a Kohonen map [22]. The SOM is an unsupervised machine learning technique that transforms a complex high-dimensional input space into a simpler low-dimensional (typically two-dimensional grid) discrete output space while simultaneously preserving similarity relations between the presented data. Self-organizing maps apply competitive learning rules where output neurons compete with each other to be active neurons, resulting in only one of them being activated at any one time. An output neuron that wins the competition is called a winning neuron.

Before running the algorithm, several parameters need to be set, including the size and shape of the map, as well as the distance at which neurons are compared for similarity. After selecting the parameters, a map with a predetermined size is created. Individual neurons in the network can be combined into layers.

SOM typically consists of two layers of neurons without any hidden layers [23]. The input layer represents input vector data. A weight is a connection that connects an input neuron to an output neuron, and each output neuron has a weight vector associated with it. The formation of self-organizing maps begins by initializing the synaptic weights of the network. The weights are updated during the learning process. The winner is the neuron whose weight vector is most similar to the input vector.

The winning neuron of the competition or the best-matching neuron c at iteration t (i.e., for the input data x_t) is determined using the followingequation

$$c(t) = \arg\min \{\|x(t) - w_i(t)\|\}, \text{ for } i = 1, 2, \ldots, n$$

where $w_i(t)$ is the weight of i-th output neuron at time t, and n is the number of output neurons. After the winning neuron c has been selected, the weight vectors of the winner and its neighboring units in the output space are updated. The weight update function is defined as follows:

$$w_i(t+1) = w_i(t) + \alpha(t) h_{ci}(t) \left[x(t) - w_i(t)\right],$$

where $\alpha(t)$ is the learning rate parameter, and $h_{ci}(t)$ is the neighborhood kernel function around the winner c at time t. The learning rate is the speed with which the weights change. The connection between the input space and the output space is created by the neighborhood function, which also determines the rate of change of the neighborhood around the winner neuron. This function affects the training result of the SOM procedure. A Gaussian function is a common choice for a neighborhood function h_{ci} that determines how a neuron is involved in the training process:

$$h_{ci}(t) = \exp\left(-\frac{d_{ci}^2}{2\sigma^2(t)}\right) \alpha(t).$$

where d_{ci} denotes the distance between the winning neuron c and the excited neuron i, $\sigma^2(t)$ is a factor used to control the width of the neighborhood kernel at time t. The learning rate $\alpha(t)$ is a decreasing function toward zero.

SOM can be used in a variety of ways, including clustering tasks. The authors of [24] assumed that each SOM unit is the center of a cluster, and as a result, the k-unit SOM performed a k-means-like task. The authors also added that when the radius of the neighborhood function in the SOM is zero, the SOM and k-means algorithms strictly correspond to one another.

The basic SOM algorithm can be summarized by the following pseudocode:

```
Algorithm 3 Self-organizing map (SOM)
Input: dimension and size of the output space, distance function, neigh-
    borhood function, learning rate, and a set of data points $X$.
Output: a set of clusters
initialize the weights of each neuron
$t = 1$
select randomly an input vector from the set of training data $X$
for each input vector do
    calculate the distances measure between
    the input vector and all the weights
    vectors.
    find the best matching neuron $c(t)$ at
    iteration $t$.
    update the weight vectors of the neurons.
    $t = t + 1$ and update neighborhood size and
    learning rate.
end for
```

Table 1 The list of symbols and their descriptions

Symbol	Description
S	the streaming data $\{s_t, s_{t+1}, s_{t+2}, \ldots\}$ containing unlabeled malicious samples captured from time t
T	the dataset with labeled malicious samples captured before the time t
k	the number of classes, i.e., malware families, of the samples from T
p_i	for the *single-classifier method*, it is the probability estimation of the classifier that a given test sample belongs to the i-th class
p'_i	for the the *multi-classifier method*, it is the probability estimation of the i-th classifier that a given test sample belongs to the i-th class
τ	for the *single-classifier method*, it is the threshold for determining whether a sample will be classified or clustered
τ'	for the *multi-classifier method*, it is the threshold for determining whether a sample will be classified or clustered

4 The proposed approach

This section contains a description of the proposed system for the classification and clustering of malware families. The definition of the problem that our system attempts to solve is as follows.

Let $S = \{s_t, s_{t+1}, s_{t+2}, \ldots\}$ be a streaming data containing unlabeled malicious samples captured from time t. Let us also have a dataset T with labeled malicious samples captured before the time t where labels are divided into k different classes corresponding to k known malware families. The goal is to process s_i, $i \geq t$, as follows:

1. if s_i is from the known malware family, then assign it to this family,
2. otherwise:
   (a) if s_i is similar to some already clustered (unlabeled) samples $s_j \in S$, where $t \leq j \leq i$, then assign s_i to the corresponding cluster
   (b) otherwise, create a new cluster and assign s_i to it.

We use the notation summarized in Table 1 to describe your approach. Our approach attempts to solve this problem in two phases:

- *First phase*: deciding which stream data samples to classify and which to cluster,
- *Second phase*: classification and clustering of samples based on the decision from the *first phase*.

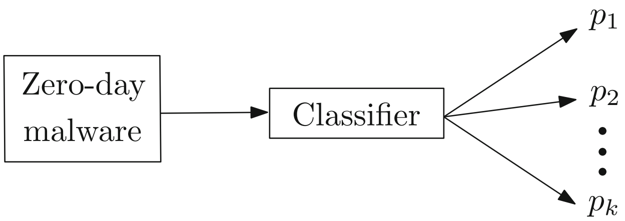


Fig. 1 Classification probabilities prediction $(p_1, \ldots, p_k)$ from a multiclass classifier

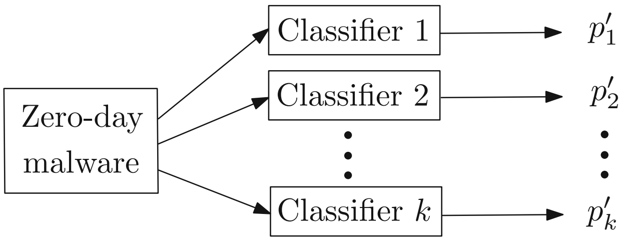


Fig. 2 Classification probabilities prediction $(p'_1, \ldots, p'_k)$ from the k binary classifiers

In the *first phase*, the streaming data S is first preprocessed using the standard score and the PCA algorithm. Then, the classification probabilities for the classes (i.e., known malware families) are predicted using already trained one or more classifiers. We considered two different methods for computing the classification probabilities prediction. In the first method, the classification probabilities prediction for a given classifier is defined as a vector $(p_1, \ldots, p_k)$ of calibrated probabilities, where p_i is the probability estimation of the classifier that a given test sample belongs to the i-th class. The classification probabilities prediction from the second method is defined as a vector $(p'_1, \ldots, p'_k)$ of probabilities, where p'_i is the probability estimation of the i-th classifier that a given test sample belongs to the i-th class. The concrete calculation of classification probability predictions depends on the given classifier and will be discussed in Sect. 6.2.

Thus, the first method relies on one multiclass classifier, as shown in Fig. 1, where this classifier was trained using the labeled data from the dataset T with k classes.

On the other hand, the second method relies on k binary classifiers, as illustrated in Fig. 2. In this case, the i-th classifier corresponds to the i-th class, i.e., the dataset T is divided into two classes: samples from the i-th class, and the second class consists of samples that do not belong to the i-th class. This division is applied for each of the k classes separately. Then, k binary classifiers were trained on such data, and the i-th classifier provided p'_i, which is the probability prediction that a test sample belongs to the i-th class. In the rest of the paper, the first method will be referred to as the *single-classifier method* and the second as the *multi-classifier method*.

The reason why we considered both methods is that the performance of these methods varies depending on the data structure. The *multi-classifier method*, where we trained separate classifiers for each class, can be suitable if the classes have different characteristics. However, it may also lead to redundancy in the learned features. In addition, training k binary classifiers slows down the training process compared to training one multiclass classifier.

Streaming data samples s_i are divided into two chunks according to the classification probabilities prediction. In both methods, maximal probability $\max_{1 \leq j \leq k} p_j$, resp. maximal probability $\max_{1 \leq j \leq k} p'_j$ is compared to some threshold parameter τ, resp. τ'. A test sample for which this maximal probability is greater or equal to the threshold is called *high-confidence sample*. On the other hand, *low-confidence samples* are samples where the maximal probability from the classification probabilities prediction vector is lower than a given threshold.

In the *second phase*, *high-confidence samples* are classified into the known malware families and *low-confidence samples* proceed into the online clustering algorithm. The same feature set extracted using PCA in the *first phase* was used for classification and clustering. The threshold τ, resp. τ' is a parameter of our approach, and it determines the amount of stream data that will be classified or clustered. The proposed architecture is depicted in Fig. 3.

Testing various clustering algorithms to find the best clustering is essential since online clustering methods may exhibit varying performance traits based on the dataset. The main difference between our approach and existing works regarding malware family classification is that our method processes the streaming data in real-time, while some other works rely on batch processing. Both streaming data processing and batch processing have their advantages and disadvantages. While streaming data processing can provide a faster decision to samples as they occur, on the other hand, processing in large batches may be more efficient since it can be parallelized.

5 Experimental setup

This section presents the dataset used in the experimental part, and the metrics for evaluating the classification and clustering results are explained. The implementation of our proposed model and methods for evaluating classification and clustering results are based on scikit-learn[2] and PyClustering[3] libraries. All experiments in this work were executed on a single computer platform having two processors (Intel Xeon Gold 6136, 3.0GHz, 12 cores each), with 64 GB of RAM running the Ubuntu server 18.04 LTS operating system.

Table 2 The size of training labeled data set D, size of streaming unlabeled data set S, and the overall dataset size, i.e., $|D| + |S|$

Malware Family	$\|D\|$	$\|S\|$	Size
Xtrat	16,689	19,280	35,969
Zbot	10,782	13,293	24,075
Ramnit	10,275	10,320	20,595
Sality	9,522	9,050	18,572
Ursnif	0	5,733	5,733
Emotet	0	4,904	4,904
Sivis	0	2,803	2,803

5.1 Dataset

We worked with the EMBER dataset [7] that contains features from portable executable files extracted using static analysis, which aims at searching for information about the file structure without running a program. The features were extracted using the LIEF open source package [25] and includes metadata from portable executable file format [26], strings, byte and entropy histograms. The feature set consists of 2,381 features that are described in [7].

The EMBER dataset contains 400,000 labeled malware samples divided into a training set (300,000 samples) and a test set (100,000 samples) according to the following date. Samples that appeared until October 2018 are included in the training set, while samples appeared between November and December 2018 are included in the test set. The training set contains samples from more than 3,000 malware families. However, we focus primarily on the four most prevalent malware families: Xtrat, Zbot, Ramnit, and Sality. The training dataset T used in our model consists of samples from the EMBER training set with labels corresponding to these four malware families. The streaming data S used in our model consists of samples from the EMBER test data set with labels corresponding to these four malware families and three additional malware families: Emotet, Ursnif, and Sivis. We considered three new families to get closer to the real situation when new malware families are constantly being created. One of our goals is to verify whether our proposed model can identify new families using online clustering.

Table 2 summarizes the number of samples used in the experimental part, arranged in descending order of sample count for each of the seven prevalent malware families from the EMBER dataset.

The following is a brief description of the malware families. More information about malware families and technical details can be found in [27].

[2] https://scikit-learn.org

[3] https://pyclustering.github.io

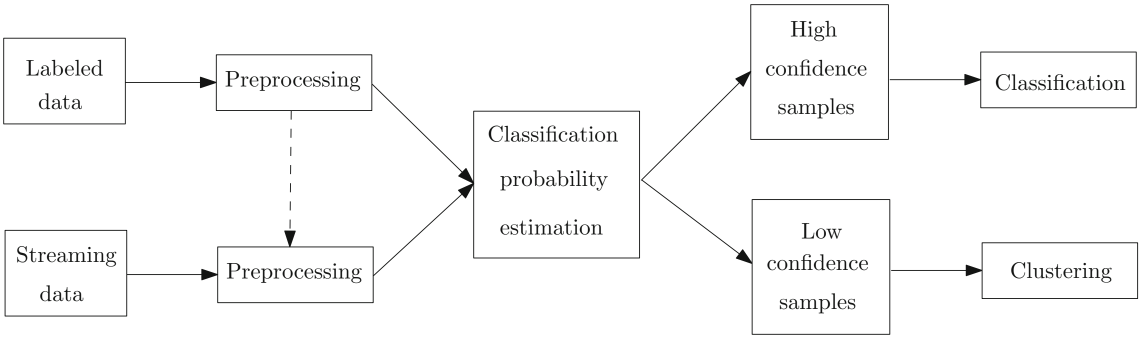


Fig. 3 The architecture of our proposed model for processing zero-day malware to malware families

The Xtrat malware family is able to steal sensitive data from infected devices, including login passwords, keystrokes, and information from online forms. Zbot, also known as Zeus, is a Trojan horse frequently used to steal financial data, including credit card numbers and login information for online banking. The Ramnit is a worm that has the ability to steal login passwords, financial information, and other sensitive data. It is also capable of downloading additional malware onto compromised devices. Sality is malware that has the ability to replicate itself and propagate over networks. It can infect executable files and change the code within to avoid detection.

Emotet is a modular malware that mainly targets affected computers to steal sensitive data. It is usually spread through phishing emails and can use social engineering tactics to deceive users into downloading and installing the malware. Ursnif is a banking Trojan that can steal private data such as usernames, passwords, and credit card numbers. Typical infection vectors are phishing emails or drive-by downloads. Sivis is a backdoor Trojan that belongs among more recent malware families. Sivis often spreads via phishing emails or by taking advantage of vulnerabilities in outdated software. Once Sivis is activated, attackers may utilize the victim's computer to carry out orders, steal data, or launch more attacks.

5.2 Evaluation metrics

Our dataset contains samples from seven classes that have different sizes. We used balanced accuracy (BAC) to evaluate the imbalanced testing set for the multiclass classification problem. The balanced accuracy score is defined as the average of true positive rates (recalls) across all k classes:

$$BAC = \frac{1}{k}\sum_{i=1}^{k} TPR_i,$$

where TPR_i is the true positive rate for class C_i. The balanced accuracy helps identify whether the classifier performs well in all classes or is biased towards a particular class.

In the clustering part, we evaluated the quality of clusters using two standard measures: purity and silhouette coefficient (SC). Let the purity of cluster C_j be defined as $\text{Purity}(C_j) = \max_i p_{ij}$, where p_{ij} is the probability that a randomly selected sample from cluster C_j belongs to class i. The overall purity is the weighted sum of individual purities and is given as follows:

$$\text{Purity} = \frac{1}{n}\sum_{j=1}^{k} |C_j|\text{Purity}(C_j).$$

where n is the size of a dataset.

While purity uses labels when evaluating the quality of clusters, the silhouette coefficient does not depend on labels. It can therefore be used in the validation phase to determine the number of clusters. The average silhouette coefficient [28] for each cluster is defined as follows.

Consider n samples $x_1, \ldots, x_n$ that have been divided into the k clusters $C_1, \ldots, C_k$. Average distance between $x_i \in C_j$ to all other samples in cluster C_j is given by

$$a(x_i) = \frac{1}{|C_j| - 1}\sum_{\substack{y \in C_j \\ y \neq x_i}} d(x_i, y).$$

Let $b_k(x_i)$ be the average distance from the sample $x_i \in C_j$ to all samples in the cluster C_k not containing x_i :

$$b_k(x_i) = \frac{1}{|C_k|}\sum_{y \in C_k} d(x_i, y).$$

Let $b(x_i)$ be the minimum of $b_k(x_i)$ for all clusters C_k, where $k \neq j$. The silhouette coefficient of x_i is given by combining $a(x_i)$ and $b(x_i)$ as follows:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}.$$

The silhouette coefficient $s(x_i)$ ranges from -1 to 1, with higher scores indicating better performance. Finally, the average silhouette coefficient for a given dataset is defined as the average value of $s(x_i)$ over all samples in the dataset.

The choice of metric for evaluating the quality of clusters depends on the information we have about the samples. Some antivirus companies may receive hundreds of thousands of new samples daily, but it is not known, immediately after their appearance, whether they are malicious. However, these samples are analyzed (manually or through automated processes based on machine learning), and the corresponding labels are created. For this reason, we also assume in our work that we also have the labels available for evaluating clusters, i.e., respective malware families.

6 Experimental results

This section contains a description of individual experiments. For both methods, i.e., for the *single-classifier method* with one multiclass classifier and the *multi-classifier method* with four binary classifiers, we considered the following three classifiers: Multilayer perceptron (MLP), Random forest (RF), and k-nearest neighbors (KNN). First, we performed feature extraction and hyper-parameters tuning of these three classifiers. Then, the relationship between BAC and the percentage of classified samples (i.e., number of *high-confidence samples* divided by $|S|$ times 100%) is presented for both methods for calculating the classification probabilities prediction vector. Finally, for the *single-classifier method* only, we present the relationship between the number of clusters and the quality of the clusters given in terms of purity and average silhouette coefficient.

6.1 Preprocessing

The standard score and PCA algorithm were applied to the data set T containing the labeled samples. The standard score, or z-score, converts a value x to a standard score z via $z = (x - \bar{x})/s$, where $\bar{x}$ is the mean and s is the standard deviation. The PCA [29] is an unsupervised learning algorithm used for dimensionality reduction. We used the PCA to extract new, uncorrelated features that are linear combinations of the original features given by the EMBER dataset described in Sect. 5.1. The same preprocessing methods, i.e., the standard score for data normalization and PCA for feature extraction, were also applied to unlabeled streaming data S.

In this experiment, we considered the options for the optimal number of features from the interval $\{20, 30, 40, \ldots, 200\}$. Table 3 shows the optimal number of features and the balanced accuracy achieved on the training data D for the multiclass classifier and four binary classifiers.

6.2 Classifiers selection

In the *single-classifier method* and the *multi-classifier method*, we considered the following three classifiers: MLP, RF, and KNN. We tuned the hyper-parameters of the MLP, RF, and KNN classifiers using the grid search that exhaustively considered all parameter combinations. The following searching grid parameters were explored for MLP:

- hidden layer sizes: (100,0), (200,0), (400,0), (100,50), (200,100), (400,100), (400,200)
- activation function: relu, tanh, logistic
- solver for weight optimization: lbfgs, adam
- alpha: 0.0001, 0.001, 0.01

The parameter alpha controls the strength of regularization applied to the neural network's weights. The names of the activation functions and the solvers are taken from `neural_network.MLPclassifier` class from the scikit-learn library, which was used in the experiments. For random forest, we explored the number of trees in the forest, the maximal depth of trees, and the criterion that measure the quality of a split:

- number of estimators: 100, 500, 1000
- maximal depth: 7, 8, 9, 10
- criterion: gini, entropy

The names of the criteria are taken from `ensemble.RandomForestClassifier` class from the scikit-learn library, which was used in the experiments. Finally, for the KNN, we considered the following numbers of nearest neighbors, k: 1,3,5,7,9,11. The selected values of the hyper-parameters for the MLP, RF, and KNN models are given Table 4.

According to the experimental results described in Table 3, the MLP achieved the highest classification accuracy for the multiclass classifier and for all binary classifiers. Figure 4 demonstrates that the MLP classifier outperformed the RF and the KNN classifiers in terms of ROC curves and the AUC (Area Under Curve) measure.

In the following experiments, we will use MLP to determine which stream data samples to classify and which to cluster. For a test sample, the output of the MLP with the softmax activation is a probability distribution over the possible classes. The predicted class for a test sample is then the highest probable class.

6.3 Data stream splitting

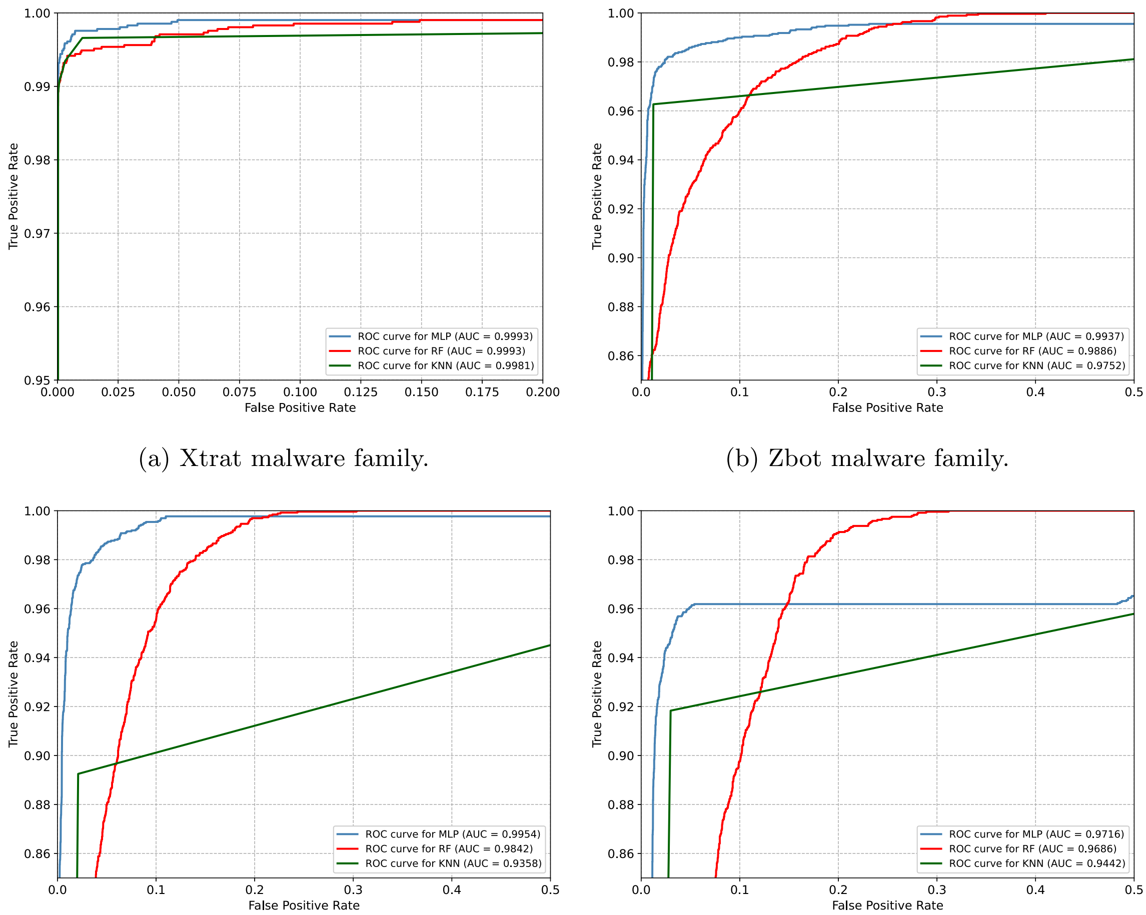
At the end of the *first phase* of our model, streaming data is divided into the *high-confidence samples* and the

Table 3 An optimal number of features extracted using PCA and the balanced accuracy for the multiclass classifier (class_all) and four binary classifiers (class_family) trained for the corresponding malware families

Classifiers	MLP		RF		KNN	
Classes	# Features	BAC	# Features	BAC	# Features	BAC
class_all	170	96.8%	180	93.20%	190	94.65%
class_Xtrat	180	99.63%	130	99.52%	160	99.61%
class_Zbot	160	97.46%	150	92.40%	180	97.46%
class_Ramnit	160	96.46%	190	92.19%	140	93.77%
class_Sality	110	95.47%	160	90.41%	190	94.44%

Table 4 Hyperparameter tuning for the multiclass MLP (class_all) and four binary MLPs (class_family) trained for the corresponding malware families

Classifiers	MLP				RF			KNN
Parameters	Hidden_layer_sizes	Activation	Solver	Alpha	Criterion	Max_depth	n_estimators	*k*
class_all	(400, 200)	relu	adam	0.001	Entropy	10	500	1
class_Xtrat	(400, 200)	relu	adam	0.001	Entropy	10	500	5
class_Zbot	(200, 0)	relu	adam	0.001	Entropy	10	100	1
class_Ramnit	(400, 200)	relu	adam	0.0001	Entropy	10	1000	1
class_Sality	(400, 200)	relu	lbfgs	0.0001	Gini	10	1000	1



(a) Xtrat malware family.

(b) Zbot malware family.

(c) Ramnit malware family.

(d) Sality malware family.

Fig. 4 ROC curves and AUC values of four binary classifiers used in the *multi-classifier method*. The ROC curves are zoomed in the upper left corner

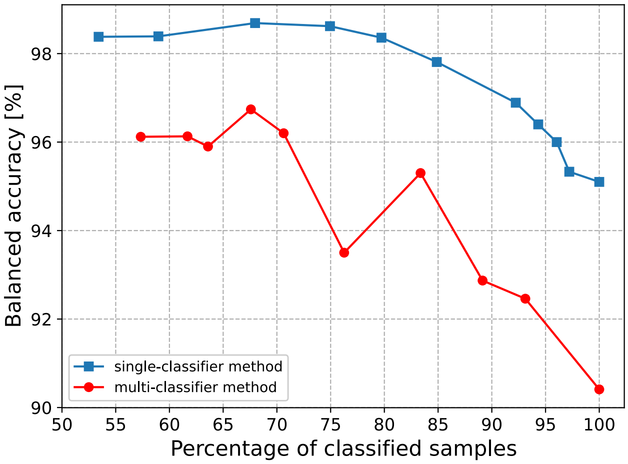


Fig. 5 Relation between the percentage of classified samples and the balanced accuracy

low-confidence samples according to the classification probabilities prediction vector. Figure 5 shows the relation between the balanced accuracy and the percentage of classified samples for various thresholds τ, resp. τ'. Specifically, we experimented with the following values of the thresholds: 0.1, 0.2, …, 0.9, 0.99, 0.999, …, 0.99999999.

The *single-classifier method* achieved the highest BAC, 98.60%, for the threshold $\tau = 0.99999$, classifying 67.97% of the samples. While the *multi-classifier method* achieved the highest BAC, 96.74%, for the threshold $\tau' = 0.9999$, classifying 67.58% of the samples.

The results show that the *single-classifier method*, where one multiclass classifier was used to determine the data to be clustered, outperforms the *multi-classifier method* based on four binary classifiers. For this reason, in the following section, we will present the clustering results only using the *single-classifier method.*

A threshold τ is the parameter of our model and can be used to influence the BAC. However, we do not know the optimal number of clusters in advance for the *low-confidence samples*. One way to determine the number of clusters is based on the silhouette coefficient, where labels are not required for its computation. Specifically, we may cluster incoming *low-confidence samples* simultaneously for several numbers of clusters. Based on these silhouette coefficient time series, we may predict future silhouette coefficient values for different numbers of clusters. Then we can select the number of clusters for which the highest silhouette coefficient is expected.

Since the optimal value of the parameter τ is not known in advance, therefore, in the following experiments, we considered only two extreme cases:

- $\tau = 0.6$, when almost all streaming data is classified (specifically, it was approximately 98%),
- $\tau = 0.9999999$, when approximately half of the streaming data was classified (specifically, it was approximately 55%).

6.4 Clustering

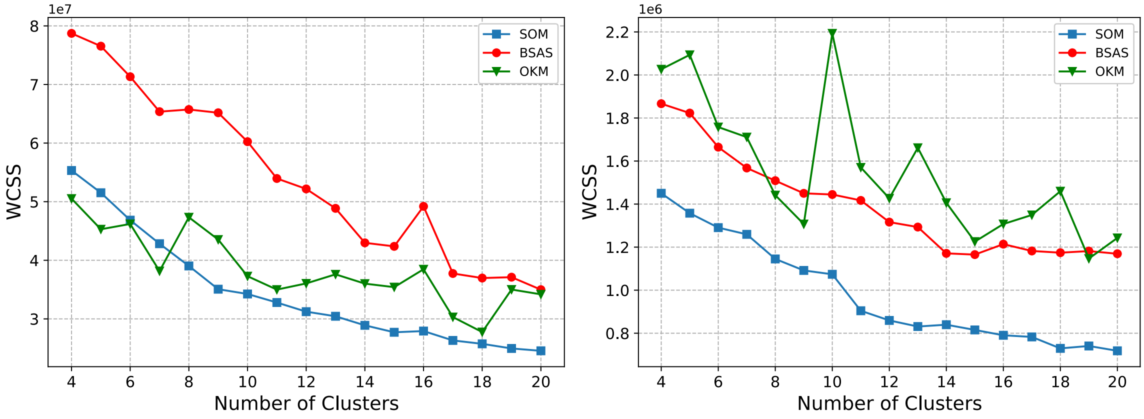
For various numbers of clusters, we conducted experiments where three online clustering algorithms were applied to the *low-confidence samples*. We used the elbow method to determine the optimal number of clusters. Figure 6 for different values of the parameter τ show the relationship between the number of clusters and Within-Cluster Sum of Square (WCSS), which is the sum of the squared distance between each point of the cluster and its centroid. Since the plots do not exhibit clear elbow points, we present clustering results for clusters between four to ten. The number of clusters determined the number of output neurons in SOM and the maximum number of clusters for BSAS. At BSAS, we experimented with different values of the dissimilarity threshold Θ. The highest average silhouette coefficients and purities of clusters were achieved for the default value of $\Theta = 1$.

The relation between the number of clusters and the purity of clusters, respectively, the silhouette coefficient, is depicted in Fig. 7. This relation corresponds to the parameter $\tau = 0.6$ for which the *single-classifier method* achieved the BAC, 95.33%, classifying 97.21% of the samples from S. The results show that SOM online clustering algorithm outperformed the other two algorithms except in one case where OKM achieved higher purity for the number of clusters equal to five.

While Fig. 7 for the parameter $\tau = 0.6$ represents the case where 97.21% of streaming data S were classified, on the other hand, Fig. 8 for the parameter $\tau = 0.9999999$ represents the case when only 55.44% of the samples were classified, achieving a BAC of 99.14%.

The results from Fig. 8 show that SOM online clustering algorithm outperformed the other two algorithms in terms of silhouette coefficient in all cases. For all numbers of clusters, SOM and OKM algorithms achieved significantly higher purities than BSAS algorithm. Note that all the online clustering algorithms achieved higher purities of clusters for $\tau = 0.6$ for almost all numbers of clusters compared to the purities achieved for the parameter $\tau = 0.9999999$.

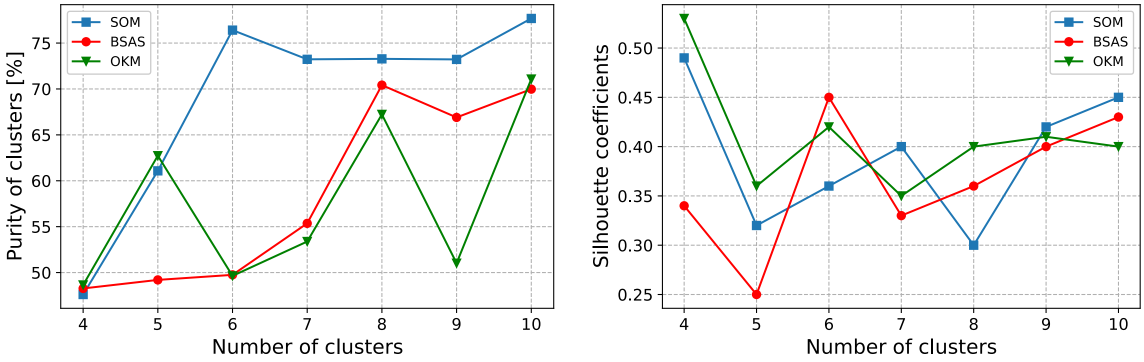
To summarize the results, we classified 97.21% of streaming data with a balanced accuracy of 95.33% and clustered the remaining data using SOM online clustering algorithm, achieving a purity from 47.61% for four clusters to 77.68% for ten clusters. These results indicate that our approach has the potential to be applied to the classification and clustering of zero-day malware into malware families.



(a) $\tau = 0.9999999$

(b) $\tau = 0.6$

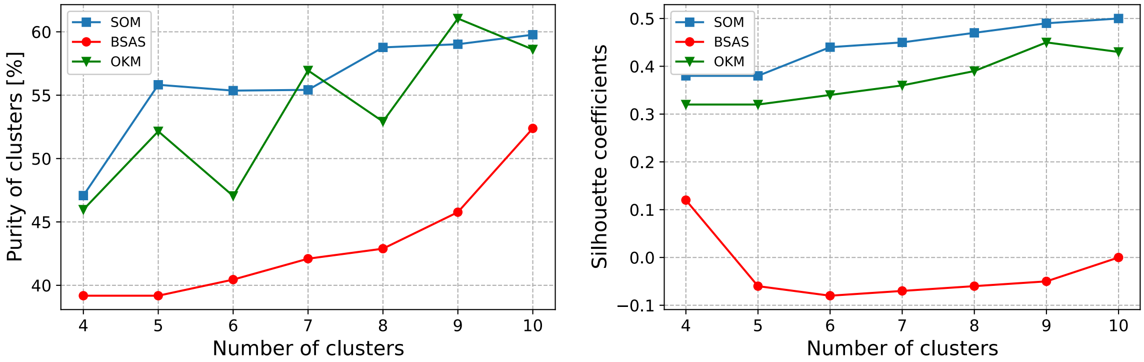
Fig. 6 The relation between the number of clusters and the WCSS for the parameter $\tau = 0.9999999$ (a), respectively, the parameter $\tau = 0.6$ (b)



(a) Purities of clusters.

(b) Average silhouette coefficients.

Fig. 7 The relation between the number of clusters and the purity of clusters (a), respectively, the average silhouette coefficient (b). For the parameter $\tau = 0.6$, 2.79% of the samples from S were clustered



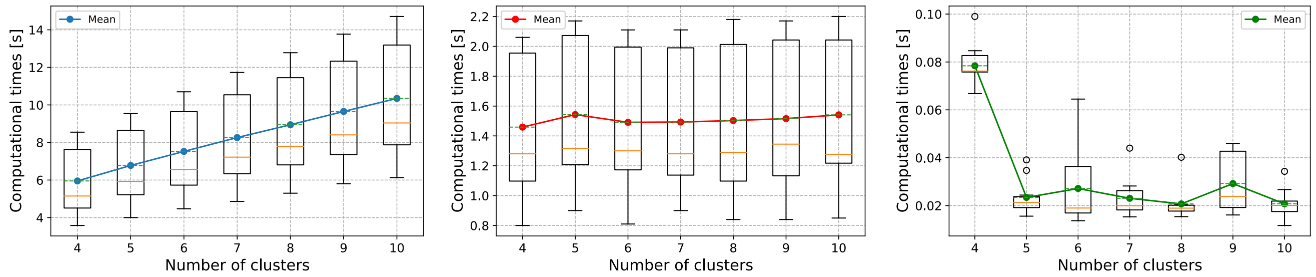
(a) Purities of clusters.

(b) Average silhouette coefficients.

Fig. 8 The relation between the number of clusters and the purity of clusters (a), respectively, the average silhouette coefficient (b). For the parameter $\tau = 0.9999999$, 44.56% of the samples from S were clustered

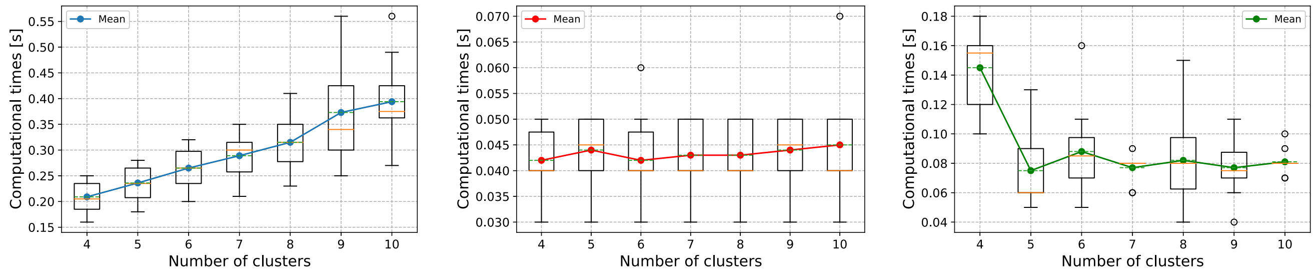
6.5 Computational times

This section focuses on the computational times of classification and clustering of malware families. We run our proposed approach ten times, and the results of the classification part are reported in the form of mean and standard deviation, while the results of the clustering part are shown as boxplot graphs. The dataset D of size 47,268 samples was used for training the MLP classifier, and the computational times for the classification and clustering parts were obtained for the processing of streaming data S of size 65,383 samples. The training time of the MLP took 81.80 s on average, with a standard devia-



(a) SOM (b) BSAS (c) OKM

Fig. 9 The computational times of the clustering algorithms for the parameter $\tau = 0.9999999$



(a) SOM (b) BSAS (c) OKM

Fig. 10 The computational times of the clustering algorithms for the parameter $\tau = 0.6$

tion of 24.48 s. The computation times of the classification and clustering parts depend on the parameter τ, which is used in dividing the streaming data into those to be classified and those to be clustered. For the parameter, $\tau = 0.9999999$, the MLP classification took 0.33 s on average, with a standard deviation of 0.02 s, while for the parameter $\tau = 0.6$, the MLP classification took 0.38 s on average, with a standard deviation of 0.01 s. The Figs. 9 and 10 show the computational times of individual clustering algorithms for the parameter $\tau = 0.9999999$ and $\tau = 0.6$, respectively. The differences in the computational times of individual clustering algorithms for different values of the parameter τ are because the parameter τ affects the size of the data to be clustered. The parameter $\tau = 0.9999999$ was chosen so that roughly half of the used streaming data (more precisely, 55% on average for the considered ten experiments) was clustered, while for the parameter $\tau = 0.6$ only approximately 2% of the streaming data were clustered. Based on the given computational times, we can estimate that the implementation of our proposed approach can process more than 3,000 samples per second, which is sufficient to process 560,000 samples, which according to the AV-Test Institute [1] are detected on average per day.

7 Conclusions

Our approach can play a useful role for malware researchers in classifying and clustering malware into families and studying how the families evolve over time. The proposed model was designed in an online form to provide decisions immediately as samples occurred. In our work, the training data were strictly separated from the test data based on the date of appearance of malware samples. In addition, the test data contained new malware families not presented in the training set, corresponding to the emergence of new malware families. Following these conditions that align with the real world, we classified zero-day malware with a balanced accuracy of 95.33% and clustered with a purity of up to 77.68%. Experimental results indicate that the proposed model can accurately classify and cluster malware into families.

A paper's direct extension is to process streaming data containing malicious and benign samples. This is a more challenging problem since the *low-confidence samples* also consist of benign files that can break the structure of the clusters. Future work may also focus on the prediction of the optimal threshold τ, based on which it is determined which zero-day malware should be classified and which should be clustered. The optimal threshold is the value at which we obtain the highest overall accuracy of the classification and clustering of stream data. This task is challenging since the optimal threshold is related to the number of new malware families, which may be hard to predict.

Acknowledgements This work was supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 ”Research Center for Informatics” and by the Grant Agency of the CTU in Prague, grant No. SGS23/211/OHK3/3T/18 funded by the MEYS of the Czech Republic.

Funding Open access publishing supported by the National Technical Library in Prague.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. AV-TEST: AV-TEST malware statistics. https://www.av-test.org/en/. [Accessed on April 27, 2023] (2023)
2. Jureček, M., Jurečková, O., Lórencz, R.: Improving classification of malware families using learning a distance metric. In: ICISSP, pp. 643–652 (2021)
3. Idika, N., Mathur, A.P.: A survey of malware detection techniques. Purdue Univ. **48**(2), 32–46 (2007)
4. Lakhotia, A., Kapoor, A., Kumar, E.: Are metamorphic viruses really invincible. Virus Bull. **12**, 57 (2004)
5. Damodaran, A., Troia, F.D., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. J. Comput. Virol. Hacking Tech. **13**, 1–12 (2017)
6. Comar, P.M., Liu, L., Saha, S., Tan, P.-N., Nucci, A.: Combining supervised and unsupervised learning for zero-day malware detection. In: 2013 Proceedings IEEE INFOCOM, pp. 2022–2030. IEEE (2013)
7. Anderson, H.S., Roth, P.: Ember: an open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
8. Radhakrishnan, K., Menon, R.R., Nath, H.V.: A survey of zero-day malware attacks and its detection methodology. In: TENCON 2019-2019 IEEE Region 10 Conference (TENCON), pp. 533–539. IEEE (2019)
9. Yoo, I.S., Ultes-Nitsche, U.: Non-signature based virus detection: towards establishing a unknown virus detection technique using SOM. J. Comput. Virol. **2**, 163–186 (2006)
10. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. J. Comput. Secur. **19**(4), 639–668 (2011)
11. Zhuang, W., Ye, Y., Chen, Y., Li, T.: Ensemble clustering for internet security applications. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **42**(6), 1784–1796 (2012)
12. Makandar, A., Patrot, A.: Malware analysis and classification using artificial neural network. In: 2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15), pp. 1–6. IEEE (2015)
13. Gandotra, E., Bansal, D., Sofat, S.: Zero-day malware detection. In: 2016 Sixth International Symposium on Embedded Computing and System Design (ISED), pp. 171–175. IEEE (2016)
14. Radwan, A.M.: Machine learning techniques to detect maliciousness of portable executable files. In: 2019 International Conference on Promising Electronic Technologies (ICPET), pp. 86–90. IEEE (2019)
15. Zhang, S.-H., Kuo, C.-C., Yang, C.-S.: Static pe malware type classification using machine learning techniques. In: 2019 International Conference on Intelligent Computing and Its Emerging Applications (ICEA), pp. 81–86. IEEE (2019)
16. Pitolli, G., Laurenza, G., Aniello, L., Querzoni, L., Baldoni, R.: MalFamAware: automatic family identification and malware classification through online clustering. Int. J. Inf. Secur. **20**, 371–386 (2021)
17. Pirscoveanu, R.-S., Stevanovic, M., Pedersen, J.M.: Clustering analysis of malware behavior using self organizing map. In: 2016 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), pp. 1–6. IEEE (2016)
18. Burnap, P., French, R., Turner, F., Jones, K.: Malware classification using self organising feature maps and machine activity data. Comput. Secur. **73**, 399–410 (2018)
19. Abernathy, A., Celebi, M.E.: The incremental online k-means clustering algorithm and its application to color quantization. Expert Syst. Appl. **207**, 117927 (2022)
20. Duda, R., Hart, P.: k-means clustering. https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/C/sk_means.htm. [Accessed on April 27, 2023] (2008)
21. Koutroumbas, K., Theodoridis, S.: Pattern Recognition. Academic Press, Cambridge (2008)
22. Kohonen, T.: The self-organizing map. Proc. IEEE **78**(9), 1464–1480 (1990)
23. Asan, U., Ercan, S.: An Introduction to Self-organizing Maps. Springer, Cham (2012)
24. Bação, F., Lobo, V., Painho, M.: Self-organizing maps as substitutes for k-means clustering. In: Computational Science–ICCS 2005: 5th International Conference, Atlanta, GA, USA, May 22–25, 2005, Proceedings, Part III 5, pp. 476–483. Springer (2005)
25. Thomas, R.: LIEF–library to instrument executable formats. Available: https://lief-project.github.io/. [Accessed on April 27, 2023] (2017)
26. Microsoft: PE Format—Win32 apps. Available: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format. [Accessed on April 27, 2023] (2023)
27. Micro, T.: Threat encyclopedia. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/. [Accessed on April 27, 2023] (2023)
28. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**, 53–65 (1987)
29. Webb, A.R., Copsey, K.D., Cawley, G.: Statistical Pattern Recognition. Wiley Online Library, Hoboken (2011)