Computer Standards & Interfaces 71 (2020) 103443

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

Metamorphic malware identification using engine-specific patterns based on co-opcode graphs

Arzu Gorgulu Kakisim*,[a], Mert Nar[b], Ibrahim Sogukpinar[b]

[a] Istanbul Commerce University, Computer Engineering, Kucukyali, Istanbul, Turkey
[b] Gebze Technical University, Computer Engineering, Gebze, Kocaeli, Turkey

ARTICLE INFO

Keywords:
Metamorphic malware
Malware detection
Opcode graph
Virus generation kits
Static analysis

ABSTRACT

A metamorphic virus is a type of malware that modifies its code using a morphing engine. Morphing engines are used to generate a large number of metamorphic malware variants by performing different obfuscation techniques. Since each metamorphic malware has its own unique structure, signature based anti-virus programs are ineffective to detect these metamorphic variants. Therefore, detection of these kind of viruses becomes an increasingly important task. Recently, many researchers have focused on extracting common patterns of metamorphic variants that can be used as micro-signatures to identify the metamorphic malware executables. With the similar motivation, in this work, we propose a novel metamorphic malware identification method, named HLES-MMI (Higher-level Engine Signature based Metamorphic Malware Identification). The proposed method firstly constructs a unique graph structure, called as *co-opcode graph*, for each metamorphic family, then extracts engine-specific opcode patterns from the graphs. Finally, it generates higher-level signature belonging to each family by representing the extracted opcode-patterns with a binary vector. Experimental results on four datasets produced by different morphing engines demonstrate the effectiveness and efficiency of the proposed method by comparing with several existing malware identification methods.

1. Introduction

Computer viruses are malicious programs that can pose serious threats for our assets and may even cause serious damages for communications and information systems worldwide if necessary precautions are not taken [1–4]. According to the report published by Malwarebytes Labs [5], approximately 750.3 million malware programs were detected in 2018. Moreover, another report [6] remarks that over 350,000 new malicious programs are registered in every day. The most important factor causing this situation is that there exist many toolkits that even non-experts can produce malicious software by applying advanced techniques such as polymorphism and metamorphism.

Metamorphism enables virus to generate its variants by utilizing a morphing engine which incorporates different obfuscation techniques [7]. These obfuscation techniques such as subroutine permutation, dead code insertion, and instruction substitution [8] enable malware to bypass signature-based approaches. For example, when a virus consisting of n subroutines is given, it is possible to produce n number of variants by reordering these subroutines with subroutine permutation. Many variants can be generated by adding dead instructions that do not cause any change in the function of the program or substituting a single instruction or an instruction block with another instruction or an instruction block with the same functionality. Thus, each variant becomes a logically equivalent version of the original virus, but it has a different internal structure from the virus. Herewith, since each new variant has a new signature, traditional malware detection programs that use signature-based browsers become ineffective or unsuccessful to detect this type of malware [9]. Therefore, no malware detection company can claim to achieve 100% detection.

In recent years, it has been observed that control flow modification causes an anomaly in usage of jump opcodes while data flow modification causes an anomaly in usage of pop/push opcodes [10]. Thus, many methods have been proposed to detect malicious programs by observing such anomalies extracted from opcode distributions belonging to the codes of malware [11,12]. However, the most of existing opcode-based methods have several disadvantages. Obfuscations techniques such as dead code insertion alter the opcode distribution of malware in a way that adversely affects the accuracy of opcode frequency-based methods [13]. N-gram-based methods generates large number of features that causes the curse of dimensionality [14]. Sequence-based methods generally use the sequence alignment algorithm whose time complexity is quite high in order to discover the common

* Corresponding author.
E-mail addresses: akakisim@ticaret.edu.tr (A.G. Kakisim), m.nar@gtu.edu.tr (M. Nar), ispinar@gtu.edu.tr (I. Sogukpinar).

https://doi.org/10.1016/j.csi.2020.103443
Received 30 July 2019; Received in revised form 5 April 2020; Accepted 6 April 2020
Available online 09 April 2020

sequence between two different opcode sequences. Traditional opcode graph-based methods lead to difficult due to generally utilizing NP-complete techniques such as graph isomorphism and maximum common sub-graphs. Thus, the need to develop a rapid and scalable detection approach by obtaining distinctive patterns belonging to metamorphic malware executables is still an open and important problem.

In the literature [7,8], some few researchers make efforts for hunting metamorphic engines of malware construction kits. They generally aim to discover the attributes that increase the distance between metamorphic families, where "family" refers to a set of variants by generated from the same toolkit. In a similar manner, some efforts [15] have focused on identifying the metamorphic engine of a given executables by deeply analyzing the textual nature of the code. They assert that if the distinctive patterns belonging to each engine are extracted, we can use these patterns to detect the variants generated by the same toolkit [16].

Considering the mentioned problems, in this work, we focus on developing an alternative metamorphic malware identification approach by extracting meaningful engine-specific patterns from the variants' codes generated by toolkits. The main idea, that motivates us to explore engine-specific patterns, is that each morphing engine performs a certain algorithm to construct the strings of the malware and utilize a set of tools to apply the obfuscation techniques. Accordingly, it may be possible to appear some semantic or lexical discriminators of the toolkit in each sample generated by the toolkit. With this motivation, we have hypothesized that some opcode pairs and a set of the opcodes consisting of these opcode pairs will be frequently observed in the variants produced by the same engine. Moreover, we have hypothesized that there exist different opcode pairs that are often observed together for the variants produced by a different metamorphic engine.

In this work, we propose Higher-Level Engine Signature based Metamorphic Malware Identification (HLES-MMI) method that identifies metamorphic malware families based on computing the similarities among the higher-level engine signatures. Proposed method represents each family with a graph structure, called as *co-opcode graph*, to discover these opcode pairs which are evaluated as discriminators of the engine. In co-opcode graph structure, nodes represent unique opcodes to be observed in the samples while the edges refer to co-occurrence relationships among the opcodes. Proposed method discovers the largest strongly connected opcode sub-graph from co-opcode graph. Afterwards, the method generates *higher-level engine signature* for each family using the set of opcodes in the sub-graph to extract distinctive set of opcode pairs for each family. The proposed method provides considerable fast identification because it applies only a signature matching process. Additionally, the pre-processing step for a given new sample is to only extract unique opcodes in the code of sample. Therefore, our method also provides real-time detection.

The rest of the paper is organized as follows: Section 2 reviews the related works. In Section 3, we present our proposed method in detail. The experimental results are provided in Section 4. Finally, we conclude the paper in Section 5.

2. Related works

Metamorphic malware detection has been an important research topic due to the rapid spread of metamorphic viruses, which have the ability of self-replicating, in digital world [17–20]. Many researchers have been focused on extracting some common attributes belonging the malware programs by applying data mining and learning methods [21–23].

One of the pioneering approaches in this area introduced by Kolter and Maloof [14]. They extracted the most relevant byte n-grams and implemented some classification techniques in order to detect metamorphic variants. Wong and Stamp [7] used Hidden Markov Model (HMM) method to take the advantage of statistical properties of ordered opcode sequences. Their experiments showed that there exists a statistical similarity between variants belonging to the same metamorphic family. However, it is possible to evade HMM detection by using some obfuscation techniques such as dead code insertion and instruction substitution [8,24]. Therefore, Toderici and Stamp [25] combined HMM method with the chi-squared test technique to improve the detection performance. Deshpande et al. [26] proposed an approach based on eigen vector analysis of raw byte data belonging to the metamorphic families. Sahay and Sharma [27] presented opcode frequency-based approach which firstly groups the metamorphic malware programs using k-Means clustering algorithm, and then extracted some relevant features for each group. Raphel and Vinod [11] used heterogeneous opcode-based feature space that includes uni-grams, bi-grams and branch instructions of the samples.

Many methods in the literature have introduced graph-based approaches. Bruschi et al. [28] evaluated malware detection problem as a sub-graph isomorphism problem by extracting the control flow graph (CFG) of the malware. They applied the CFG matching process to detect metamorphic malware programs. However, since the complexity of the graph isomorphism problem is NP-complete, their method has scalability problem. To enhance the detection process of CFG approach, Bonfante et al. [29] obtained reduced graph from CFGs and used them as malware signatures. Eskandari and Hashemi [30] combined CFGs and API calls to obtain more distinctive patterns from executables. To avoid the graph isomorphism problem, they converted each API-CFG structure to a feature vector and used the classification methods for malware detection. Moreover, Eskandari and Raesi [31] claimed that semantic signatures could be obtained from reduced CFGs. They extracted frequent sub-graphs corresponding to common sub-codes of the programs to generate signatures of the executables. Khalilian et al. [12] constructed weighted directed opcode graph based on bi-gram relations and extracted the frequent sub-graphs from these opcode graphs to capture common patterns belonging to each metamorphic malware family. Alam et al. [32] proposed a control flow graph-based approach, namely ACFG (Annotated Control Flow Graph). Their method used an intermediate language, which is used for reducing the number of instructions, to obtain reduced CFGs. The method applied sub-graph isomorphism to detect the malware programs. With their other method [33], they aimed to select an appropriate number of features for statistical analysis of MAIL pattern distributions by sliding a window (SWOD) with different size. Furthermore, they generated signatures belonging to MAIL distributions of the malware samples for detecting malware programs.

Some methods measure the software similarity between the variants of metamorphic malware families, then classify the variants as a metamorphic family or benign according to similarity results [34]. Rad and Masrom [35] proposed a method which compares the similarity among the frequency-based opcode histograms of metamorphic viruses in order to analyze the malware families. Patel [36] used some similarity metrics such as edit distance and similarity-index to measure the similarity between morphed virus and non-virus. Runwal et al. [37] proposed a method based on opcode graph similarity to detect metamorphic malware executables. They constructed weighted directed graphs for each malware sample derived from opcode sequences of the malware. They revealed that the graph structures belonging to same metamorphic malware family are quite similar. Mirzazadeh et al. [13] observed that inserting dead codes to the codes of malware samples causes considerably differentiation in the graph structures of the samples belonging to the same family. Therefore, they applied Linear Discriminant Analysis to extract more meaningful edges from the graph structures. Shanmugam et al. [38] proposed an opcode-based similarity approach that uses simple substitution cipher to compute similarity scores according to the statistics of opcodes. Another new similarity-based method was proposed by Radkani et al. [39]. They introduced entropy-based dissimilarity measure to measure the distance between metamorphic malware and benign. They used K-Nearest Neighbor

procedure ConstructCo-opcodeGraphs($S = \{S^1, \ldots, S^D\}$)
  $V \leftarrow$ Extract unique opcodes in S ;
  $N \leftarrow$ The number of unique opcodes in S;
  for $d \leftarrow 1$ to D **do** ▷ $S^d = s_1^d, \ldots, s_n^d$
    $G^d \leftarrow$ Construct an empty graph for d-th family; ▷ $G^d = (V^d, E^d)$
    for $k \leftarrow 1$ to n **do**
      $Op_k \leftarrow$ Obtain unique opcodes of S_k^d ;
      for $i \leftarrow 1$ to $N - 1$ **do**
        for $j \leftarrow i + 1$ to N **do**
          if opcode $i \in Op_k$ and opcode $j \in Op_k$ **then**
            ▷ Computing the weight for the edge between v_i, $v_j \in G^d$
            $L^d_{v_i,v_j} \leftarrow L^d_{v_i,v_j} + 1$;
          end if
        end for
      end for
    end for
    $L^d_{v_i,v_j} = L^d_{v_i,v_j}/N$; ▷ Normalization
  end for
  return $\{G^1, \ldots, G^D\}$
end procedure

Algorithm 1. Constructing co-opcode graph of each metamorphic family and benign type.

classification method by combining their distance metric to classify metamorphic malware programs.

Some researchers have focused on code authorship attribution in the domain of malware analysis to characterize details of binary codes [16]. Code authorship attribution of the malware can provide to identify the metamorphic engine of the malware variant. Considering a malware construction kit, its morphing engine performs a certain algorithm to construct the opcode sequences of the malware and uses set of tools to apply some obfuscation techniques. In this case, it may be possible to appear some semantic or lexical discriminators of the construction kit in each sample generated by this kit. Chouchane and Lakhotia [40] showed that engine attribution can be useful to identify the metamorphic malware programs by generated the same morphing engine. They proposed a code scoring technique to explore forensic discriminators from the code segments of malware programs. Additionally, Chouchane et al. [15] focused on engine attribution problem by defining three approaches: morphed variant recognition problem, detection of frequency of codes substituting by engines, and generalization of first two problems. Canfora et al. [41] aimed to explore some special usage of opcodes to evaluate static properties of the whole code. The main idea is that virus writers and virus generators intensively use push/pop pairs to modify the body of the code of malware. They analyzed such statistical anomalies by computing some ratios such as ratio between push and pop opcodes, and the ratio between the number of consecutive push opcodes and the total push opcodes.

3. Proposed work

It is assumed that there are some common patterns which increase similarity between the variants in the same family [12,33]. We estimate that the similarity between two variants belonging to same family can be revealed by discovering engine-specific patterns. To analyze engine-specific patterns, we revealed two hypotheses. First, there exists a unique opcode diversity of each metamorphic malware construction kits. Second, there exists a correlation in the co-occurrence of unique opcode pairs observed in the codes of malware samples. We estimate that the opcode diversity and the co-occurrence relations among the opcode pairs will be similar in the codes of the metamorphic samples produced by the same toolkit. To observe the similarity among the variants of same family, we calculate the ratio of maximum number of unique opcodes observed in the assembly code of each variant to the maximum number of unique opcodes that can be observed in different variants with different families. To analyze the co-occurrence of unique opcode pairs in the codes of variants of same family, we use co-occurrence relationship approach [42,43]. There exists a co-occurrence relationship between two unique opcodes which are observed in any samples of same family. We define a relation with weight n between two unique opcodes, if this opcode pair are observed together in n samples.

Taking these two hypothesizes into consideration, to describe each metamorphic malware construction toolkit, we construct a unique directed and weighted graph model, called as *co-opcode graph*. The nodes represent unique opcodes to be observed in the samples. The number of nodes corresponds to the maximum number of unique opcodes that can be observed in different samples of different families. The co-occurrence relationships among opcodes refer to the edges between the nodes of co-opcode graph. The weights of the edges depend on the number of co-occurrence of the opcode pairs. Let S be collection of samples and $S = \{S^1, S^2, ..., S^D\}$ be the set of samples of metamorphic malware families or benign where S^F represents the set of samples of the family F or benign type F. The collection of the family F is defined by $S^F = \{s_1^F, s_2^F, ..., s_n^F\}$ where s_n^F represents the n^{th} sample in S^F. For the family F, the co-opcode graph can be represented as $G^F = \{V^F, E^F\}$ where $V^F = \{v_1, v_2, ..., v_N\}$ is a set of nodes which denotes unique opcodes in all samples and $E^F \subseteq V^F \times V^F$ is a set of edges which indicates the relations among nodes in V^F. The relations of G^F is represented by a link matrix $L^F \in R^{N \times N}$ where $L^F_{v_i,v_j}$ is greater than 0 if nodes v_i and v_j have a co-occurrence relationship, and $L^F_{v_i,v_j}$ is equal to 0 otherwise. The value of $L^F_{v_i,v_j}$ is equal to n if opcodes i and j are observed together in n samples $\in S^F$.

Algorithm 1 demonstrates the construction phase of co-opcode graphs in detail. Firstly, the unique opcodes observed in all existing malware and benign samples are extracted. For each family, an empty co-opcode graph in which each node is assigned to a unique opcode is created. Thereafter, the unique opcodes for each sample of the family are extracted. It is checked whether each unique opcode pair (i, j) is present in each sample of the family. If opcode pair (i, j) is observed in the sample, the edge weight between v_i and v_j is increased by 1. Once the final weights of the edges in the graph are obtained, the weights of the edges are normalized by dividing by the maximum number of unique opcodes which are observed in all different variants.

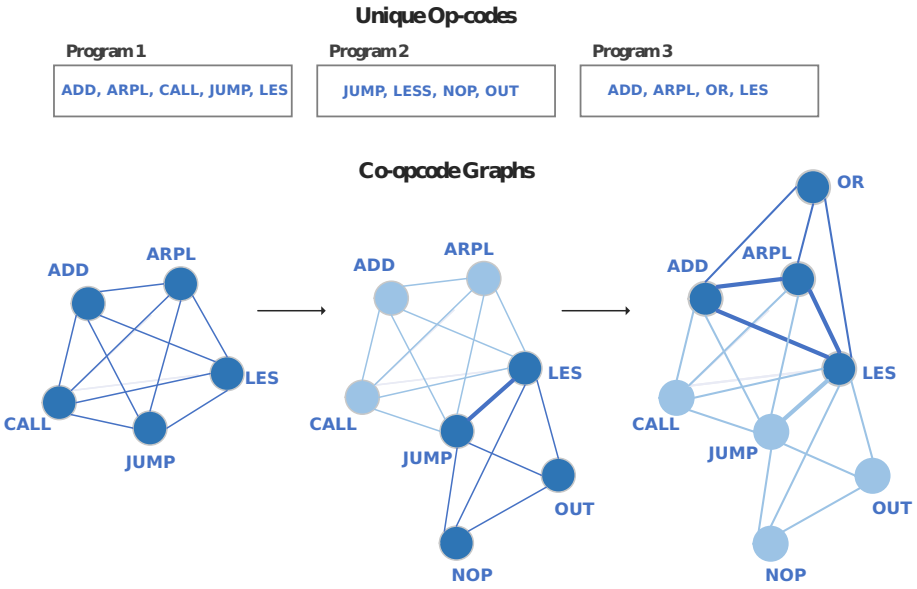

Fig. 1. Illustration of co-opcode graph representation.

Fig. 1 is provided to illustrate the structure of the co-opcode graph. The co-opcode graph is constructed using the examples of unique opcodes which are extracted from codes of three malware samples belonging to the same metamorphic malware family. For these three samples, the number of unique opcodes is equal to 8. Therefore, the weight of the relationship between opcodes once occurred together is 0.125. The weight of edge between JUMP and LES is 0.25 because they occurred twice together. Likewise, the weight of edges between ARPL, ADD and LES is equal to 0.25.

The co-opcode graphs of metamorphic malware families can enable us to determine which metamorphic family is more similar to a new incoming suspicious program. Let G^F be the co-opcode graph corresponding to a family *F*, G^s be the co-opcode graph corresponding to a new incoming suspicious sample, and N be the number of distinct opcodes. Note that G^F and G^s are $N \times N$. We compute the similarity between the sample s and the family F using 2-D correlation coefficient which is defined as below:

$$CGS_{G^s,G^F} = \frac{\sum_N \sum_N (G^s_{NN} - \overline{G^s})(G^F_{NN} - \overline{G^F})}{\sqrt{(\sum_N \sum_N (G^s_{NN} - \overline{G^s})^2)(\sum_N \sum_N (G^F_{NN} - \overline{G^F})^2)}} \quad (1)$$

where $\overline{G^s}$ and $\overline{G^F}$ indicates the mean of the weights of all edges belonging to graph G^s and graph G^F, respectively.

After obtaining the graph structure for each metamorphic family, we can classify a new sample by calculating the similarity between the co-opcode graph of the sample and the co-opcode graph of metamorphic families. Algorithm 2 provides Co-opcode Graph Similarity based Metamorphic Malware Identification method (CGS-MMI). The method constructs co-opcode graph for a new sample, then computes the similarity between the sample and the metamorphic families. The family with the highest similarity value is selected as most likely family for this sample.

In this work, the main aim is to find a distinctive higher-level engine signature for each metamorphic family by extracting meaningful patterns from co-opcode graphs. Also, we create higher-level signature for different benign type such as applications and system files. To generate higher-level engine signatures, we extract the largest strongly connected component from the co-opcode graph [44]. A connected component of a graph $G = (V, E)$ is a sub-graph $G' = (V', E')$ whose nodes are connected to each other by paths. That is, if two nodes v_i and v_j are in the same component V', then there exists a path between them. The largest connected component refers to a maximum set of nodes such that each node is reachable from each of the others. It enables us to determine the set of opcodes that frequently occur together. However, some unique opcode pairs are not frequently observed in the samples of a malware family. Therefore, the relation among the pairs can be weak. We can eliminate the weak relations using a threshold value to obtain a strongly connected sub-graph from the co-opcode graph.

Fig. 2 provides an illustration for the largest strongly connected component of the co-opcode graph. Let the weights of the thick edges are 0.25 and the weights of thin edges are 0.125 as shown in Fig. 2. When the threshold value is equal to 0.2, the largest strongly connected component of the co-opcode graph includes the set of these opcodes: ADD, ARPL, JUMP and LES.

After extracting the set of distinctive opcode patterns belonging to

Require: $S = \{S^1, \ldots, S^D\}, t$

▷ Learning Process

1: $\{G^1, \ldots, G^D\} \leftarrow$ ConstructCo-opcodeGraphs($\{S^1, \ldots, S^D\}$);

▷ Constructing co-opcode graph for the new sample t

2: $\{G^t\} \leftarrow$ ConstructCo-opcodeGraphs(t);

▷ Detection Process

3: **for** $d = 1$ to D **do**

4: Compute CGS_{G^t,G^d} using Eq. (1);

5: **end for**

6: **Output:** A malware family or benign according to highest similarity.

Algorithm 2. Co-opcode graph similarity based metamorphic malware identification: CGS-MMI.

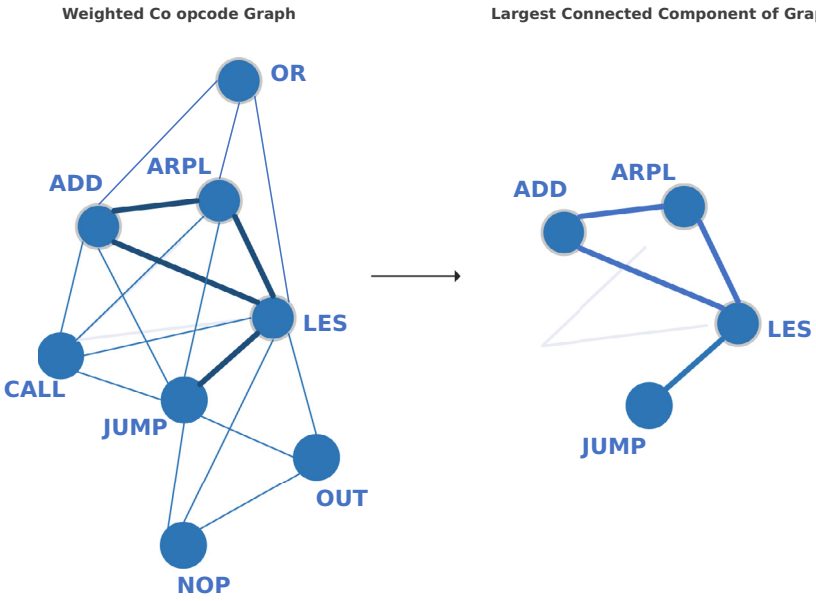

Fig. 2. Illustration of the largest strongly connected component of a co-opcode graph.

procedure ExtractEngineSignature($\{G^1, \ldots, G^D\}$, α)
  for $d \leftarrow 1$ to D **do**
    for $i \leftarrow 1$ to $N-1$ **do**
      for $j \leftarrow i+1$ to N **do**
        if $L^d_{v_i,v_j} \leq \alpha$ **then** ▷ Eliminating weak relations in G^d
          $L^d_{v_i,v_j} \leftarrow 0$;
        end if
      end for
    end for
    $LCC \leftarrow$ Find the largest connected component of G^d;
    $Len \leftarrow$ Length of the largest connected component LCC;
    $Sign^d \leftarrow$ Construct an empty vector with zeros; ▷ Size: 1 by N
    for $k \leftarrow 1$ to Len **do**
      $Sign^d_{LCC_k} \leftarrow 1$;
    end for
  end for
  return $\{Sign^1, Sign^2, \ldots, Sign^D\}$
end procedure

Algorithm 3. Extracting engine-specific higher-level signature for each family.

each family by applying largest strongly connected component approach, we use these patterns to construct a higher-level engine signatures. Algorithm 3 is provided to demonstrate signature extraction phase of HLES-MMI. The method eliminates weak relations from the co-opcode graphs using a threshold value α, then finds the largest connected components of the graphs. Therefore, a set of most relevant opcodes for each metamorphic family are obtained. To generate higher-level engine signatures, the method creates one characteristic binary vector for each metamorphic malware family. Let be $Sign^F \in \{0, 1\}^{1 \times N}$ binary vector of the family F. The signature corresponds to this vector in which the values corresponding to the opcodes existing in the connected component are 1, otherwise 0. The maximum length of this vector is equal to the number of unique opcodes. To use the method as a malware detection method, in a similar way, the signatures for benign types are obtained.

The identification process of HLES-MMI is given in Algorithm 4. First, the method applies a learning process which includes co-opcode graph construction and higher-level engine signature extraction. Second, it constructs a higher-level signature for the new incoming suspicious program. The unique opcodes of the new program are obtained. The values of its vector corresponding to the obtained unique opcodes are assigned to 1. Thereafter, Normalized Mutual Information metric [45] is used to calculate the similarity between signatures. Let *Sign* be the signature belonging to a metamorphic family, and *Sign′* be the signature of a new unknown sample. NMI is defined as follows:

$$NMI(Sign, Sign') = \frac{MI(Sign, Sign')}{\sqrt{(H(Sign), H(Sign'))}} \quad (2)$$

where $H(Sign)$ and $H(Sign')$ are the entropies of *Sign* and *Sign′*. Their mutual information $MI(Sign, Sign')$ indicates the mutual dependence between these samples.

Require: $S = \{S^1, \ldots, S^D\}$, N, α, t
▷ Learning Process
$\{G^1, \ldots, G^D\} \leftarrow$ ConstructCo-opcodeGraphs($\{S^1, \ldots, S^D\}$);
$\{Sign^1, \ldots, Sign^D\} \leftarrow$ ExtractEngineSignature($\{G^1, \ldots, G^D\}, \alpha$);
▷ Constructing higher-level signature for the new suspicious file t
$Op_t \leftarrow$ Extract unique opcodes belonging to file t;
$Sign^t \leftarrow$ Construct an empty vector with zeros; ▷ Size: 1 by N
$Sign^t_{Op_i} \leftarrow 1$;
▷ Detection Process
for $d = 1$ to D **do**
  $Sim_{t,d} \leftarrow$ Compute similarity between $Sign^t$ and $Sign^d$;
end for
Output: A malware family or benign according to highest similarity.

Algorithm 4. Higher-level engine signature based malware identification: HLES-MMI.

4. Experimental results

4.1. Datasets and experimental setup

In order to validate the effectiveness of the proposed method, we have conducted our experiments on four datasets produced by four different metamorphic malware construction kits which are commonly used in the literature [12,32,46,47]. These are PSMPC (Phalcon-Skism Mass-Produced Code Generator), NGVCK (Next Generation Virus Creation Kit), G2 (Second Generation Virus Generator), and MWOR (Metamorphic Worm). PSMPC is a script-driven software which differentiates decryption routines and structures in new variants of a malware. It uses a generator which works as a code-morphing engine [48]. G2 is an enhanced version of the PSMPC. G2 adds some new functions to the structure of PSMPC using more decryptor morphing. NGVCK [49] is a Win32 application which applies random function ordering, junk code insertion, and user-defined encryption methods to create new variants. It uses advanced assembly source-morphing engine. MWOR [24], that carries its own morphing engine, creates worms which are self-replicating malware programs. MWOR injects dead codes into malware' codes per each infection.

For our experiment, we selected 4085 metamorphic malware and 6857 benign programs. The malware executables were generated using the generators obtained from VX Heavens website [49]. We used the configuration files of PSMPC and G2 toolkit to generate PSMPC and G2 variants. We randomly changed the values of some parameters. These are encrypted parameter which is used for generating a random encryption function, resident parameter which specifies the creation of a memory resident or run time virus, and infection parameter which is optional counter limiting the number of infections per run of the virus to a specific number. Similarly, NGVCK variants were generated by selecting different encryption parameters. We used different padding ratios ranging from 0.5 to 4.0 to produce MWOR variants. The benign applications (APPS) were collected from download.com website [50]. The benign programs and operating system files were obtained from WIN7, WIN 10, and CYGWIN. The file sizes range from 1 KB to 18 MB. Detail information of our datasets is given in Table 1.

We applied 10-fold cross validation to randomly select training and test samples. To investigate the stability of the proposed learning process by the different dimensions of the data, we also employed four training groups as 90%, 80%, 60%, 40% and four test groups as 10%, 20%, 40%, 60%, respectively. To measure the classification and detection performance, different performance metrics are used, which are True Positive Rate (TPR), False Positive Rate (FPR), Accuracy (ACC), and Standard Deviation (STD). These metrics are used widely for comparing malware detection methods in literature [31,33,51]. Distorm3 [52,53] is used to generate assembly codes from portable executable files. Experiments were implemented on a 64-bit Ubuntu operating system running on the Intel Core i5-4440K CPU working 3.10 GHz processor and 16 GB memory.

We have implemented Opcode Graph Similarity (OGS) approach [37] and Hidden Markov Model (HMM) [10] method to compare with our method. We extracted the sequence of opcodes for each sample in the dataset for OGS approach. Thereafter, we created a weighted directed graph that each distinct opcode indicates a node of the graph and

Table 1
Dataset description.

Malware		Benign	
Types	# of Samples	Types	# of Samples
G2	974	CYGWIN	2000
PSMPC	410	WIN7	1032
MWOR	701	WIN10	1825
NGVCK	2000	APPS	2000

each distinct edge of the graph is the relation among the consecutive opcodes. In our experiment, the opcode graphs were generated for all samples and K-Nearest Neighbors technique was used with k = 5 [54] by implementing distance formula presented in [37]. Opcode sequences were used as observation sequences to train HMM as mentioned in [10]. Nltk library of Python [55] was utilized to apply HMM model for all metamorphic malware families. We used 10-fold cross-validation for both two methods.

4.2. Identification and detection results

In this section, we demonstrate the effectiveness and efficiency of the proposed methods CGS-MMI and HLES-MMI. To provide a fair comparison, the experiments were carried out on two datasets with different sizes. We created a smaller dataset by selecting 100 samples from each family and benign type since some of the methods used for comparison are ineffective on a large number of samples in terms of running time and memory space. Both the total number of malware samples and the total number of benign samples are equal to 400 for small dataset.

We first tested the CGS-MMI and HLES-MMI methods to identify the families of the new suspicious samples. In the experiments, the maximum number of unique opcodes was obtained as 1055. The number of nodes in the opcode-graphs of G2, PSMPC, MWOR and NGVCK were equal to 105, 103, 167 and 161, respectively. In Table 2, we report the TPR and FPR results for CGS-MMI and HLES-MMI method on 800 samples. As can be seen, the metamorphic malware variants were identified with 100% accuracy for all train and test groups with different sizes. These results show that our two identification methods are effective in distinguishing metamorphic malware construction kits even if very few samples are trained.

Proposed methods are also trained and tested for malware detection to investigate their detection performance. For all experiments, the samples of a certain family were labeled as malware and the benign programs were labeled as benign. We used HMM method for the comparison. For these experiments, the threshold α was set to 0.75, and the training percentage was set to 90%. The detection results are presented in Table 3. It was observed that the proposed methods achieve higher performance compared to HMM method. CGS-MMI detected all metamorphic samples with 100% TPR. Similarly, HLES-MMI identified all metamorphic samples except some NGVCK samples and provided better FPR for malware class when compared to CGS-MMI.

By including all families and benign samples, we provide the metamorphic malware detection results of CGS-MMI and HLES-MMI methods in Table 4. If a sample was identified as a malware family (G2, PSMPC, MWOR, NGVCK) by the method, it was labeled as malware at the end of the identification process of the method. Similarly, if the sample was identified as a benign type (CYGWIN, WIN, APPS), it was labeled as benign. By this way, we can compare the detection performance of all methods. According to TPR results of CGS-MMI method, the method detected metamorphic malware samples with 100% accuracy. Although HMM achieved higher accuracy than CGS-MMI method on same dataset, its FPR value for benign class is higher than zero, which indicates that some malware variants are not detected by HMM method. With HLES-MMI method, very few numbers of NGVCK samples were identified as benign, and very few numbers of benign samples were identified as MWOR and NGVCK. Although OGS-KNN method detected all metamorphic malware samples with 100% TPR, it identified a large number of benign programs as malware. In addition, ACFG and SWOD methods obtained lower TPR values for malware when compared to our method. The performance values of the ACFG and SWOD methods are based on the values presented in [33]. The performances of these methods were tested on the NGVCK, G2 and MWOR variants that are similar to our samples in our dataset. Considering small dataset, proposed methods CGS-MMI and HLES-MMI achieved highest performances with 97.7% and 99.3% accuracy, respectively. As the number of malware samples in the train set was increased, we observed that the TPR value of HLES-MMI increased. According to the results, our methods achieved the higher accuracy for the large dataset when compared to the other methods. The average standard deviations of obtained accuracy from each fold of ten-fold cross-validation indicates that we can obtain similar detection values for each fold with our methods.

In addition, we reported training and testing time of the methods to evaluate the efficiency of our methods. Training time given in Table 4 indicates the time to construct co-opcode graphs and higher-level signatures. Testing time is the time to check if a sample is benign or malware. Training time is the sum of the training times belonging to all training samples from each fold of ten-fold cross-validation. In the same way, testing time is the sum of the testing times belonging to all test samples from each fold. When we compare the testing time of the methods, we observe that CGS-MMI, HLES-MMI and HMM outperform the other methods. Our methods and HMM are more efficient because they apply limited number of graph or signature matching to determine if a new sample is a malware or not. However, OGS method compares the opcode graph of new sample with the graphs of all samples in training set. The testing times of ACFG and SWOD methods were given by calculating the total time for 10 cross validation based on the testing time given in the paper [33]. The testing times of ACFG and OGS-KNN are higher than SWOD because ACFG utilizes graph matching to check the new sample while OGS-KNN calculates the graph similarity between

Table 2
The TPR and FPR results for metamorphic family identification with CGS-MMI and HLES-MMI on 400 malware and 400 benign samples.

Methods	Train-Test Percentage	90%-10%		80%-20%		60%-40%		40%-60%	
		TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
CGS-MMI	G2	1	0	1	0	1	0	1	0
	PSMPC	1	0	1	0	1	0	1	0
	MWOR	1	0	1	0	1	0	1	0
	NGVCK	1	0	1	0	1	0	1	0
HLES-MMI	G2	1	0	1	0	1	0	1	0
	PSMPC	1	0	1	0	1	0	1	0
	MWOR	1	0	1	0	1	0	1	0
	NGVCK	1	0	1	0	1	0	1	0

Table 3
Malware detection results of CGS-MMI and HLES-MMI.

Methods	Family	Class	TPR	FPR	ACC	STD-fold
CGS-MMI	G2	M	1	0	1	0
		B	1	0		
	PSMPC	M	1	0	1	0
		B	1	0		
	MWOR	M	1	0.029	0.974	0.024
		B	0.971	0		
	NGVCK	M	1	0.024	0.981	0.019
		B	0.975	0		
HLES-MMI	G2	M	1	0	1	0
		B	1	0		
	PSMPC	M	1	0	1	0
		B	1	0		
	MWOR	M	1	0.005	0.996	0.012
		B	0.994	0		
	NGVCK	M	0.988	0.002	0.996	0.008
		B	0.997	0.011		
HMM	G2	M	1	0.056	0.991	0.017
		B	0.944	0		
	PSMPC	M	1	0.055	0.989	0.014
		B	0.945	0		
	MWOR	M	0.995	0.025	0.971	0.014
		B	0.975	0.005		
	NGVCK	M	1	0.085	0.972	0.022
		B	0.915	0		

Table 4
Metamorphic malware detection results of different methods.

Methods	Classes	# of samples	TPR	FPR	ACC	STD-fold	Train Time	Test Time
CGS-MMI	M	400	**1**	0.051	0.977	0.016	2.4	68.22
	B	400	0.948	**0**				
HLES-MMI	M	400	0.995	0.009	**0.993**	0.009	5.31	**1.12**
	B	400	**0.991**	0.005				
CGS-MMI	M	4085	**1**	0.062	0.961	0.006	5.33	156.71
	B	6857	0.937	**0**				
HLES-MMI	M	4085	0.996	0.013	**0.989**	0.004	48.42	**19.12**
	B	6857	**0.986**	0.004				
HMM	M	400	0.983	0.039	0.972	0.033	11.02	41.05
	B	400	0.961	0.017				
OGS-KNN	M	400	**1**	0.595	0.723	0.113	1760.3	4160.8
	B	400	0.421	**0**				
ACFG*	M	1020	0.970	0.043	0.960	–	–	~ 1890.1
	B	2330	0.957	0.030				
SWOD*	M	1020	0.973	0.113	0.920	–	–	~ 150.92
	B	4285	0.887	0.037				

the new sample and all other observed samples. The size of a sample increases the size of the graphs generated by ACFG also increases, and thus the graph matching process can be time consuming. Although the testing time of CGS-MMI method is higher than HMM method, its training time is quite lower than HMM. The main difference between CGS-MMI and HLES-MMI is the training and testing time. Since HLES-MMI method extracts higher-level signature using the largest connected component approach from the co-opcode graphs, the training time becomes much higher compared to CGS-MMI method. However, the testing time of CGS-MMI method is higher than HLES-MMI method because the method computes the similarity among the co-opcode graphs. As a result, it is observed that our methods are more convenient for real time applications when compared to testing time of the methods.

We evaluate the HLES-MMI method using different α values to investigate how the method is affected by the threshold. In Fig. 3, 4 and 5, we report the TPR, FPR, Accuracy and standard deviation performances of HLES-MMI by varying the value of α from 0.5 to 0.9. The results show that the method achieves highest accuracy, when the value of threshold is equal to 0.75.

HLES-MMI method detects some benign programs as malware. To observe that misclassified benign programs is more similar to what kind of the metamorphic families, we provide Table 5. The values in the Table 5 indicate misclassification ratios for benign types. Misclassification ratios are obtained by computing how many percent of total samples of a benign type are detected as a metamorphic malware family. When all samples obtained with 10-fold cross validation adjustment were included, the total number of benign test samples was 6857. The total numbers of test samples for G2, PSMPC, MWOR and NGVCK were 974, 410, 700, and 2000, respectively. From the results, it is observed that benign programs were never identified as G2 and PSMPC. The misclassified samples of CYGWIN were mostly associated with MWOR, while the misclassified samples of WIN were identified as NGVCK.

To analyze the results in Table 5, we provide Fig. 6 which indicates the opcode diversity of metamorphic malware and benign samples. As can be seen, the opcode diversities of metamorphic malware families are different from each other. However, G2 and PSMPC samples have similar opcode diversity because G2 is an enhanced version of the PSMPC [48]. The opcode diversities of metamorphic variants generally have a lower variety than benign programs. Additionally, variants of each metamorphic family have a similar diversity within itself. Diversities of APPS samples have spread over a wide area, and many of them have higher diversity than the others. Therefore, the number of unique opcodes belonging to metamorphic and benign samples becomes an important pattern to distinguish malware samples from benign samples. However, the opcode diversities of MWOR and NGVCK variants are quite similar to the diversities of CYGWIN and WINDOWS programs. Due to the proximity of opcode diversities, some benign samples can be identified as malware. The reason for this may be simply that NGVCK is WIN32 application, and MWOR is a Linux-based application [10].

Although G2 and PSMPC samples have similar opcode diversity, we can identify samples of these families accurately with our methods since the ratios of the co-occurrences of opcode pairs can be quite different. To demonstrate the differences of edge values of co-opcode graphs, we randomly selected a sub-graph from the co-opcode graphs for each

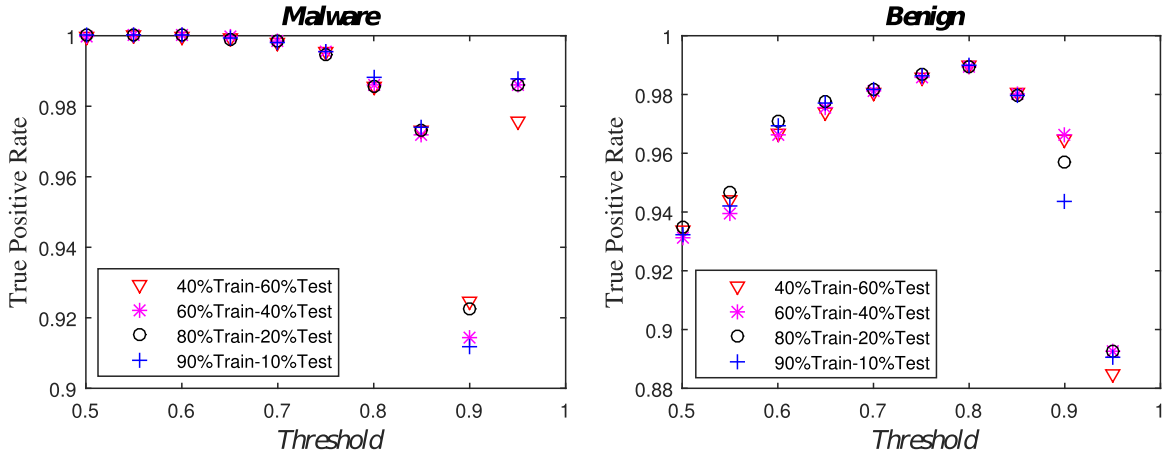

Fig. 3. True positive results of HLES-MMI with different threshold values.

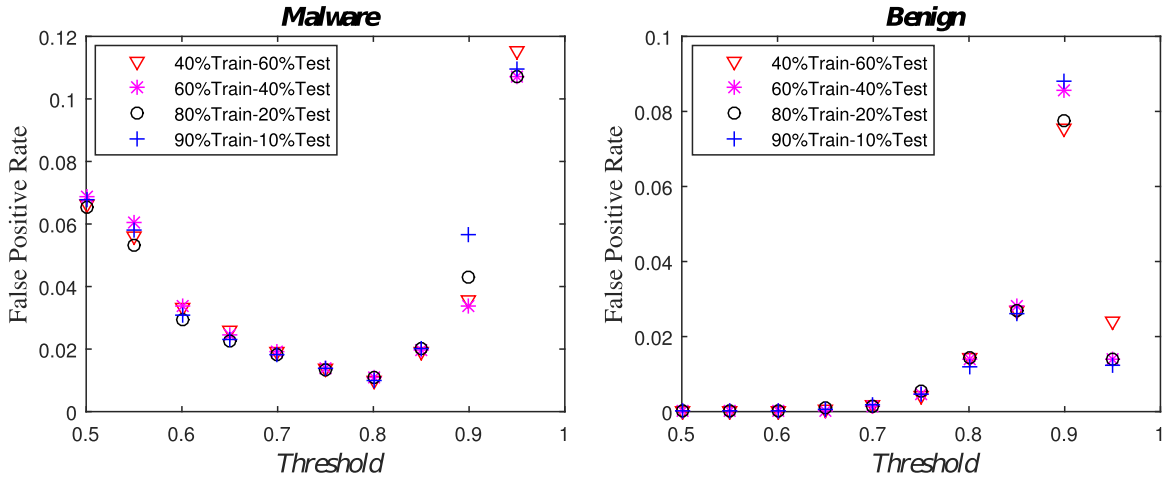

Fig. 4. False positive results of HLES-MMI with different threshold values.

metamorphic family and benign type. The set of opcodes in the sub-graphs consists of ten opcodes (LEAVE, POP, JNO, SAL, DAS, STD, ROR, LODSB, INTO, SHL). Fig. 7 shows the values of adjacency matrix of the sub-graph belonging to each metamorphic family and benign type. The values of adjacency matrix represent the strength of the relationship observed between these opcode pairs. In the Fig. 7, if the edge value between opcode pairs is high, it is shown with lighter color. The edge value is shown with darker color, if it is observed with low value. The values between these ten opcode pairs are quite high for CYGWIN, WIN and APPS, which are useful to separate benign samples from malware. We also observed that the weights of different families between opcode pairs differ. The difference is observed even among two families such as G2 and PSMPC, one of which is an enhanced version of the other. For instance, the edge value among the opcodes DAS and LEAVE is greater than 0.9 for G2 family while this edge value for PSMPC family is lower than 0.1.

The higher-level engine signatures obtained in a training phase of HLES-MMI are given in the Fig. 8. The signature for each family and benign type corresponds to a binary vector with length of unique opcode numbers. In the Fig. 8, zero values are shown with dark color, and the other values are shown with light color. The opcode set of the largest connected component belonging to APPS was observed as larger than the others. Therefore, many opcodes are represented by the value of 1 in the binary vector of APPS. These opcodes are often observed together in the samples of APPS. However, in the samples of G2 and PSMPC families, the number of opcodes frequently observed together is quite low. Fig. 8 shows that the opcode pairs frequently appeared in each family differ from the opcode pairs appeared in other families.

Table 5
Misclassification results of HLES-MMI method.

	G2	PSMPC	MWOR	NGVCK
CYGWIN	0	0	0.0340	0.0005
WIN	0	0	0.0025	0.0049
APPS	0	0	0.0005	0

According to the results, we realize that the method extracts different opcode patterns which represent each family and generates different higher-level signatures.

5. Conclusion and future works

In this work, we have proposed two effective metamorphic malware identification methods to distinguish metamorphic malware samples by extracting engine-specific patterns from morphing engines. One of the proposed methods identifies metamorphic variants according to co-opcode graph similarity, while the other method detects malware variants with using higher-level signature that are extracted from co-opcode graphs. The experiments have conducted on four different metamorphic malware construction kits to demonstrate the effectiveness and efficiency of our methods. The proposed methods achieve higher performance when compared the other baseline methods. Our future work will focus on extending our methods by discovering new different engine patterns.

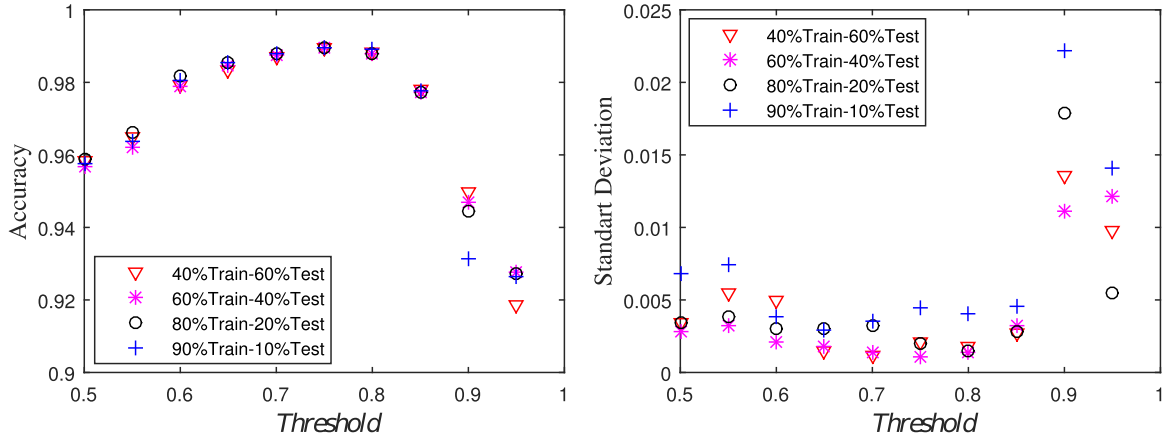

Fig. 5. Accuracy and standard deviation results of HLES-MMI with different threshold values.

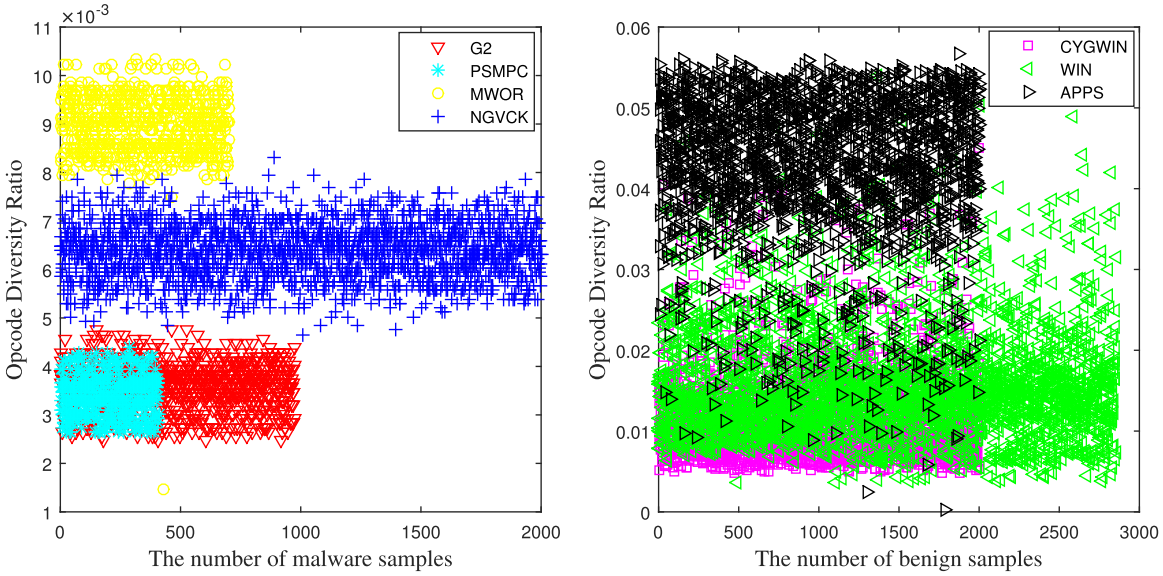

Fig. 6. Opcode diversity of malware and benign samples.

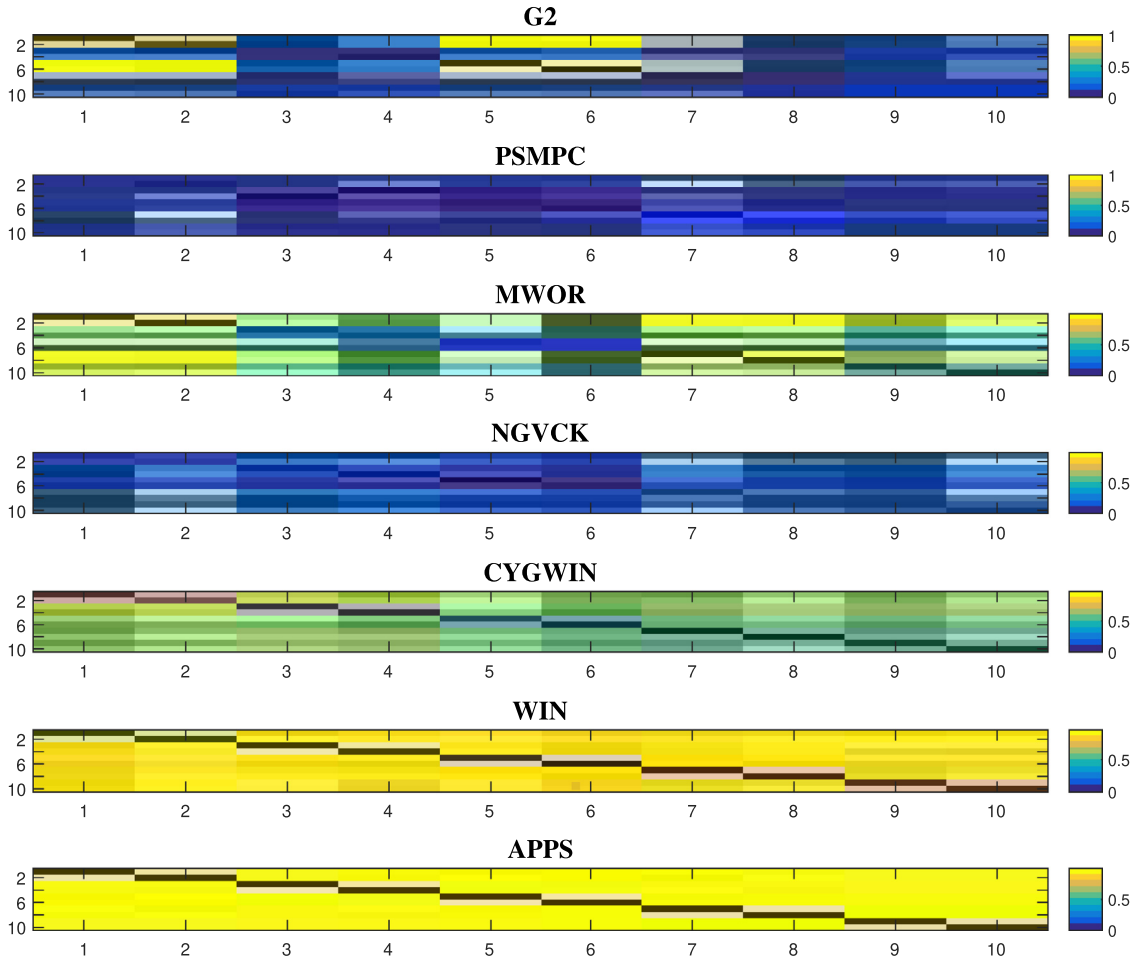

Fig. 7. The edge weights of co-opcode graphs with selected ten opcodes for metamorphic malware families and benign types.

CRediT authorship contribution statement

Arzu Gorgulu Kakisim: Conceptualization, Methodology, Software, Writing - original draft. **Mert Nar:** Investigation, Resources, Software, Writing - review & editing. **Ibrahim Sogukpinar:** Supervision, Project administration, Writing - review & editing.

Declaration of Competing Interest

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational

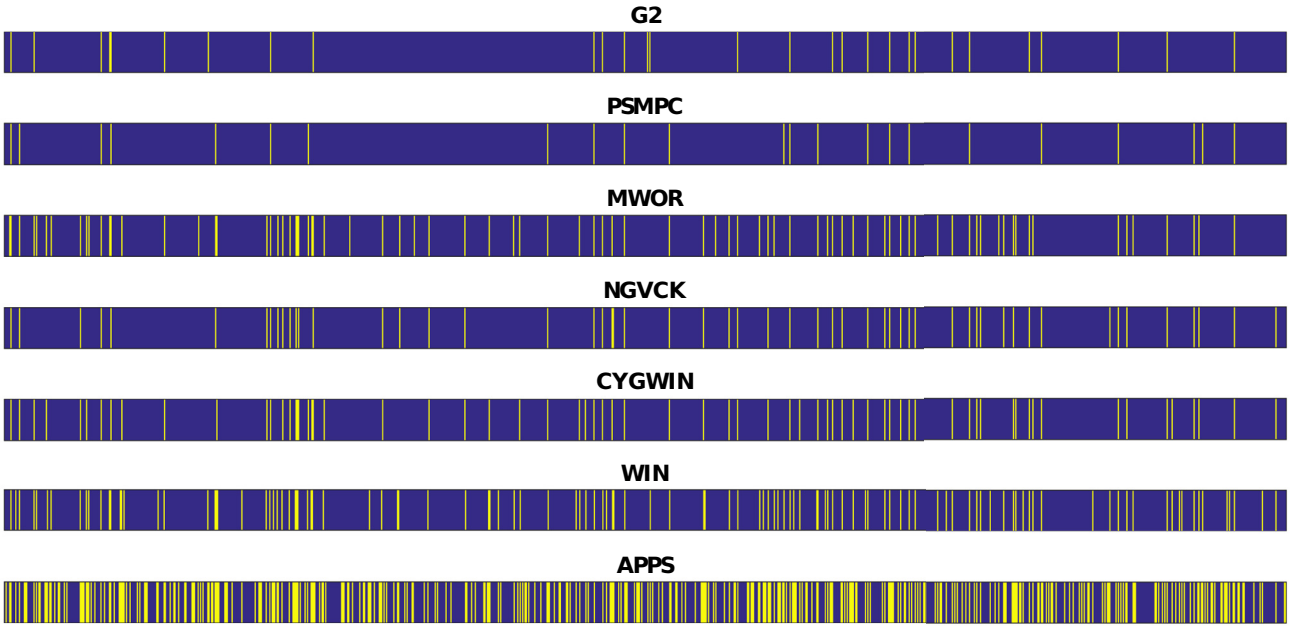

Fig. 8. Higher-level engine signatures for metamorphic families and benign types.

grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgement

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK), Grant No: ARDEB-116E624.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.csi.2020.103443 .

References

[1] K.W. Hamlen, V. Mohan, M.M. Masud, L. Khan, B. Thuraisingham, Exploiting an antivirus interface, Comput. Stand. Interfaces 31 (6) (2009) 1182–1189.
[2] S.-J. Wang, D.-Y. Kao, Internet forensics on the basis of evidence gathering with peep attacks, Comput. Stand. Interfaces 29 (4) (2007) 423–429.
[3] S. Pokharel, K.-K.R. Choo, J. Liu, Mobile cloud security: an adversary model for lightweight browser security, Comput. Stand. Interfaces 49 (2017) 71–78.
[4] S. Talukder, Tools and techniques for malware detection and analysis, arXiv:2002.06819(2020).
[5] MalwareBytes-Labs, 2019 State of Malware, (2019). https://blog.malwarebytes.com/malwarebytes-news/ctnt-report (accessed April 15, 2019)
[6] AV-Test-Institute, 2019 New Malware, (2019). https://www.av-test.org/en/statistics/malware/ (accessed April 15, 2019)
[7] W. Wong, M. Stamp, Hunting for metamorphic engines, J. Comput. Virol. 2 (3) (2006) 211–229.
[8] D. Lin, M. Stamp, Hunting for undetectable metamorphic viruses, J. Comput. Virol. 7 (3) (2011) 201–214.
[9] P. OKane, S. Sezer, K. McLaughlin, Obfuscation: the hidden malware, IEEE Secur. Priv. 9 (5) (2011) 41–47.
[10] M. Stamp, Information Security: Principles and Practice, John Wiley & Sons, 2011.
[11] J. Raphel, P. Vinod, Heterogeneous opcode space for metamorphic malware detection, Arab. J. Sci. Eng. 42 (2) (2017) 537–558.
[12] A. Khalilian, A. Nourazar, M. Vahidi-Asl, H. Haghighi, G3md: Mining frequent opcode sub-graphs for metamorphic malware detection of existing families, Expert Syst. Appl. 112 (2018) 15–33.
[13] R. Mirzazadeh, M.H. Moattar, M.V. Jahan, Metamorphic malware detection using linear discriminant analysis and graph similarity, 5th International Conference on Computer and Knowledge Engineering (ICCKE), IEEE, 2015, pp. 61–66.
[14] J.Z. Kolter, M.A. Maloof, Learning to detect malicious executables in the wild, Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 470–478.
[15] R. Chouchane, N. Stakhanova, A. Walenstein, A. Lakhotia, Detecting machine-morphed malware variants via engine attribution, J. Comput. Virol. Hack.Tech. 9 (3) (2013) 137–157.
[16] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, A. Matyukhina, Code authorship attribution: methods and challenges, ACM Comput. Surv. 52 (1) (2019) 3.
[17] D. Ucci, L. Aniello, R. Baldoni, Survey of machine learning techniques for malware analysis, Comput. Secur. (2018).
[18] I. Haq, S. Chica, J. Caballero, S. Jha, Malware lineage in the wild, Comput. Secur. 78 (2018) 347–363.
[19] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, L. Mao, Maldae: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics, Comput. Secur. 83 (2019) 208–233.
[20] S. Huda, J. Abawajy, B. Al-Rubaie, L. Pan, M.M. Hassan, Automatic extraction and integration of behavioural indicators of malware for protection of cyber–physical networks, Fut. Gener. Comput. Syst. 101 (2019) 1247–1258.
[21] Y. Dai, H. Li, Y. Qian, R. Yang, M. Zheng, Smash: a malware detection method based on multi-feature ensemble learning, IEEE Access 7 (2019) 112588–112597.
[22] C.C. San, M.M.S. Thwin, Proposed effective feature extraction and selection for malicious software classification, Advances in Biometrics, Springer, 2019, pp. 51–71.
[23] M. Nunes, P. Burnap, O. Rana, P. Reinecke, K. Lloyd, Getting to the root of the problem: a detailed comparison of kernel and user level data for dynamic malware analysis, J. Inf. Secur. Appl. 48 (2019) 102365.
[24] S.M. Sridhara, M. Stamp, Metamorphic worm that carries its own morphing engine, J. Comput. Virol. Hack. Tech. 9 (2) (2013) 49–58.
[25] A.H. Toderici, M. Stamp, Chi-squared distance and metamorphic virus detection, J. Comput. Virol. Hack.Tech. 9 (1) (2013) 1–14.
[26] S. Deshpande, Y. Park, M. Stamp, Eigenvalue analysis for metamorphic detection, J. Comput. Virol. Hack. Tech. 10 (1) (2014) 53–65.
[27] S.K. Sahay, A. Sharma, Grouping the executables to detect malwares with high accuracy, Procedia Comput. Sci. 78 (2016) 667–674.
[28] D. Bruschi, L. Martignoni, M. Monga, Detecting self-mutating malware using control-flow graph matching, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2006, pp. 129–143.
[29] G. Bonfante, M. Kaczmarek, J.-Y. Marion, Control flow graphs as malware signatures, International workshop on the Theory of Computer Viruses, (2007).
[30] M. Eskandari, S. Hashemi, A graph mining approach for detecting unknown malwares, J. Visual Lang. Comput. 23 (3) (2012) 154–162.
[31] M. Eskandari, H. Raesi, Frequent sub-graph mining for intelligent malware detection, Secur. Commun. Netw. 7 (11) (2014) 1872–1886.
[32] S. Alam, R.N. Horspool, I. Traore, I. Sogukpinar, A framework for metamorphic malware analysis and real-time detection, Comput. Secur. 48 (2015) 212–233.
[33] S. Alam, I. Sogukpinar, I. Traore, R.N. Horspool, Sliding window and control flow weight for metamorphic malware detection, J. Comput. Virol. Hack. Tech. 11 (2) (2015) 75–88.
[34] A. Walenstein, A. Lakhotia, The software similarity problem in malware analysis, Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.
[35] B.B. Rad, M. Masrom, Metamorphic virus variants classification using opcode

frequency histogram, Proceedings of the 14th WSEAS International Conference on COMPUTERS. July 23-25, 2010. Corfu Island, Greece, (2010). 147–145
[36] M. Patel, Similarity Tests for Metamorphic Virus Detection, (2011). Master's Project 175
[37] N. Runwal, R.M. Low, M. Stamp, Opcode graph similarity and metamorphic detection, J. Comput. Virol. (1–2) (2012) 37–52.
[38] G. Shanmugam, R.M. Low, M. Stamp, Simple substitution distance and metamorphic detection, J. Comput. Virol. Hack. Tech. 9 (3) (2013) 159–170.
[39] E. Radkani, S. Hashemi, A. Keshavarz-Haddad, M.A. Haeri, An entropy-based distance measure for analyzing and detecting metamorphic malware, Appl. Intell. (2018) 1–11.
[40] M.R. Chouchane, A. Lakhotia, Using engine signature to detect metamorphic malware, Proceedings of the 4th ACM workshop on Recurring malcode, ACM, 2006, pp. 73–78.
[41] G. Canfora, F. Mercaldo, C.A. Visaggio, P. Di Notte, Metamorphic malware detection using code metrics, Inf. Secur. J. 23 (3) (2014) 57–67.
[42] H. Small, B.C. Griffith, The structure of scientific literatures i: identifying and graphing specialties, Sci. Stud. 4 (1) (1974) 17–40.
[43] X. Wang, Q. Cheng, W. Lu, Analyzing evolution of research topics with neviewer: a new method based on dynamic co-word networks, Scientometrics 101 (2) (2014) 1253–1271.
[44] J. Leskovec, C. Faloutsos, Sampling from large graphs, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 631–636.
[45] A. Strehl, J. Ghosh, Cluster ensembles—a knowledge reuse framework for combining multiple partitions, J. Mach. Learn. Res. 3 (Dec) (2002) 583–617.
[46] A. Sharma, S.K. Sahay, Evolution and detection of polymorphic and metamorphic malwares: a survey, arXiv:1406.7061(2014).
[47] A.G. Kakisim, M. Nar, N. Carkaci, I. Sogukpinar, Analysis and evaluation of dynamic feature-based malware detection methods, International Conference on Security for Information Technology and Communications, Springer, 2018, pp. 247–258.
[48] P. Szor, The Art of Computer Virus Research and Defense: ART COMP VIRUS RES DEFENSE _p1, Pearson Education, 2005.
[49] V. Heaven, Computer Virus collection, (2014). http://83.133.184.251/virensimulation.org/ (accessed April 15, 2019)
[50] Download.com, Popular Apps, (2019). https://download.cnet.com/ (accessed April 15, 2019)
[51] M. Nar, A.G. Kakisim, N. Çarkaci, M.N. Yavuz, I. Sogukpinar, Analysis and comparison of opcode-based malware detection approaches, 2018 3rd International Conference on Computer Science and Engineering (UBMK), IEEE, 2018, pp. 498–503.
[52] G. Dabah, distorm3, (2019). http://code.google.eom/p/distorm (accessed April 15, 2019)
[53] M. Nar, A.G. Kakisim, M.N. Yavuz, İ. Soğukpinar, Analysis and comparison of disassemblers for opcode based malware analysis, 2019 4th International Conference on Computer Science and Engineering (UBMK), IEEE, 2019, pp. 17–22.
[54] T.M. Cover, P.E. Hart, et al., Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.
[55] NLTK, NLTK 3.4 Documentation, (2001). https://www.nltk.org/_modules/nltk/tag/hmm.html (accessed April 15, 2019)