A Cross-Platform Malware Variant Classification based on Image Representation

Hamad Naeem[1], Bing Guo[1*], Farhan Ullah[1,2] and Muhammad Rashid Naeem[1]
[1] College of Computer Science, Sichuan University, Chengdu, 610065, P.R. China
[2]Department of Computer Science, COMSATS University Islamabad, Sahiwal Campus, Sahiwal 57000, Pakistan
[E-mail : hamadnaeemh@yahoo.com, guobing@scu.edu.cn, farhankhan.cs@yahoo.com, rashidnaeem717@yahoo.com]
*Corresponding author: Bing Guo

Received November 24, 2017; revised November 14, 2018; accepted January 17, 2019; published July 31, 2019

Abstract

Recent internet development is helping malware researchers to generate malicious code variants through automated tools. Due to this reason, the number of malicious variants is increasing day by day. Consequently, the performance improvement in malware analysis is the critical requirement to stop the rapid expansion of malware. The existing research proved that the similarities among malware variants could be used for detection and family classification. In this paper, a Cross-Platform Malware Variant Classification System (CP-MVCS) proposed that converted malware binary into a grayscale image. Further, malicious features extracted from the grayscale image through Combined SIFT-GIST Malware (CSGM) description. Later, these features used to identify the relevant family of malware variant. CP-MVCS reduced computational time and improved classification accuracy by using CSGM feature description along machine learning classification. The experiment performed on four publically available datasets of Windows OS and Android OS. The experimental results showed that the computation time and malware classification accuracy of CP-MVCS was higher than traditional methods. The evaluation also showed that CP-MVCS was not only differentiated families of malware variants but also identified both malware and benign samples in mix fashion efficiently.

Keywords: CSGM, Internet security, Grayscale image, CP-MVCS, Malware Detection, Machine Learning

The authors thank Dr. Bing Guo for many valuable comments that have improved the quality of this manuscript. This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant No.61332001； The National Natural Science Foundation of China under Grant No. 61772352, 61472050; the Science and Technology Planning Project of Sichuan Province under Grant No. 2019GZDZX0045, 2018GZDZX0031, 2018GZDZX0004, 2017GZDZX0003, 2018JY0182.

http://doi.org/10.3837/tiis.2019.07.023 **ISSN : 1976-7277**

1. Introduction

Rapid growth in malicious code variants has posed a serious threat to internet security. Symantec antivirus exposed more than 401 malicious code variants in a recent technical report (2016) [31]. Besides this, the presence of malware in mobile phones also increased. In the 2016 year, Kaspersky Lab reported that the majority of mobile phones were insecure insense of unpatched vulnerabilities. In the 2017 year, the cyber financial threat report showed that banking malware attacked most of the users in Brazil, Vietnam, India, Russia, Germany, and the US. The report exposed 767072 trojans' attacks in banks [32]. In the 2014 year, the Android technical reported 3.26 million malicious attacks. The number of users that encountered Android malware attacks was 767,072 [1].

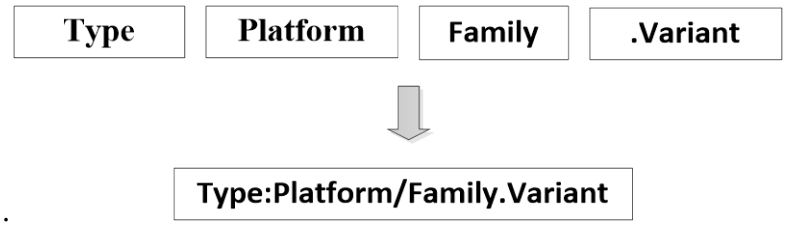


Fig. 1. Naming Scheme used by Microsoft Antivirus [30]

Malware not only divided into various classes but also identified by platforms, families and variants. In this paper, we considered the naming convention used by Microsoft antivirus [30]. According to that naming convention, the first field indicated the platform on which the malware designed to execute such as Windows, MacOS and Android etc. The second field signified the family of malware that was determined by the structural similarity between different malware. The third field was variant which used to distinguish between different versions of the same family. The **Fig. 1** showed the naming scheme used by Microsoft antivirus. Malware variants belonging to the same family must have a similar structure, and it is challenging to classify them into their relevant family.

Most of the scanning methods use specific signatures to detect malware. These scanning methods are based on various techniques such as static and dynamic analysis. Static analysis works by disassembling and executing the code without a virtual environment. On the other hand, dynamic analysis works by running the code in a virtual environment. The dynamic study is time intensive and resource consuming as each malware executable must be executed for a specific time duration [3].

Most of the static analysis techniques used signatures for malware identification. These methods showed low computational cost and high true positive rate. However, these techniques could easily expose by ordinary code obfuscation. Currently, instead of straight malicious code obfuscation, malware authors rely on packing tools. A packing tool is an application that converts one binary executable by using different compression techniques,

that appears in different file size than the original file size. Besides this, the signatures of these variants are also different from the actual one. Consequently, the Signature based techniques need to store new signatures of these variants. It leads to an exponential growth in the signature size and makes signature-based static analysis techniques less effective [33].

The expansion of malware is not only towards Windows OS but also towards also other platforms such as Android, Linux and OS X. According to an Android report of the year 2014, they detected 1.5 million Android malware applications. Likewise, Android malware, an exponential increase in Linux and OS X malware are also analysed. Hence, the focus of conventional malware detection methods such as static and dynamic code analysis is on a specific platform. Moreover, the static code analysis works by disassembling an executable file to analyse its control flow structure, and the dynamic code analysis works by real-time execution of binary in a virtual environment to examine its behavioural characteristics.

Consequently, these methods are consuming too many resources due to the third party decompile tools.

To target these challenges, this paper presented the following contributions:

1) We proposed a method that could detect malware of multiple operating systems such as Windows OS and Android OS without knowing the difference between each operating system.
2) We converted malware detection into an image classification problem by transforming malware binary into a grayscale image. The method was capable of differentiating both packed and unpacked malware binaries.
3) We extracted CSGM features that were more suitable for malware variant detection and classification. The CSGM features consisted of local and global features of the malware image.
4) Experimental outcomes indicated that CP-MVCS not only increased the classification rate up to 98.40% for 25 malware families but also achieved 97.29% classification accuracy for two classes(malware or benign).

The remaining sections of the paper are arranged as follows. Section 2 states related works for malware detection, section 3 introduces a comprehensive methodology of CP-MVCS, section 4 shows experimental results and discussion. Finally, section 5 presents the conclusion and future directions.

2. Related Works

One of the most emerging issues in the scientific community is malware variants detection and family classification, which has drawn the significant attention of security analysts. Currently, several works have been proposed such as graph-based methodologies [4] and [5], sequence-based instruction methods [6] and [7], API based monitoring methods [8] and [9]

and behavioural methods [10] etc. Even though these works are helpful to detect both malware and benign samples, but still, they have some problems such as they are unable to deal with new malware variants. For example, Naqqash Aman et al. [11] proposed a technique that used an enhanced and scalable version of the Cuckoo sandbox to generate behaviour reports. Later, these behaviour reports extracted features for training a machine learning classifier. Although their work could detect both pack and unpack malware executable still they were slow due to the usage of third party tool for example Cuckoo sandbox.

Recently, several works have been proposed to analyse malware through visualization, for instance, Kyoung Soo Han et al. [12] suggested a method that used entropy graphs to detect and classify new malware and their variants. Similarly, Jae Hyun Lim et al. [2] proposed a way that transformed malware binary information into image matrices to classify new malware variants. Even though these works were useful to classify un-packed malware binaries, but still they were unable to deal with packed malware binaries.

Various texture based visualization methods have been introduced to study new malicious code variants such as Nataraj et al. proposed that an image texture analysis be more suitable to classify the families of malware variants as compared to other existing malware analysis techniques. Nataraj converted malware binaries into images and then classified GIST based texture features with the nearest neighbour classifier. Parallel to Nataraj technique, kesav et al. [14] also proposed a low-level texture feature extraction technique for malware analysis. Kesay converted malware binaries into images and then extracted discrete wavelets transform based texture features for classification. Aziz et al. [15] extracted wavelet transform based texture features to identify new malware and their variants, and then supplied to feed forward artificial neural network for classification. Next, Konstantinos Kosmidis [16] described a two-step malware variant detection and classification method. In the first step, binary texture analysis applied through GIST. In the second step, these texture features classified by using machine-learning techniques such as classification and clustering to identify malware. Subsequently, Ban Xiao Fang et al. [17] proposed a malware detection method that extracted local binary features using SURF and then did fast fingerprint matching with LSH schemes.

Although works mentioned above [13], [14], [15], [16] and [17] are helpful to detect and classify new malware and their variants, still they have some limitations. For instance, on the one hand, global texture features lose local information needed for classification. On another hand, they have significant computation overheads to process a vast amount of malware.

3. Methodology

CP-MVCS designed to identify the families of known or unknown malware variants of Windows OS and Android OS. The entire architecture of the proposed method was shown in **Fig. 2**. There are sixth phases in proposed malware detection architecture. First, the user scans the applications on a computer or mobile applications such as antivirus tools etc. Second, if the application is recognised, then the scan results are sent to the user's computer or mobile phone.

Third, if the application is unrecognised, then the system transforms the classes. Dex file (in case of Android OS) or executable file (in case of Windows OS) into a grayscale image, as discussed in section 3.1. Fourth, these grayscale images further store on a database of back-end server. Fifth, these grayscale images feed into GPU computing pool of Matlab for identification with the trained CSGM features, as discussed in section 3.2. Sixth, the results are sent to the user's computer or mobile phone.

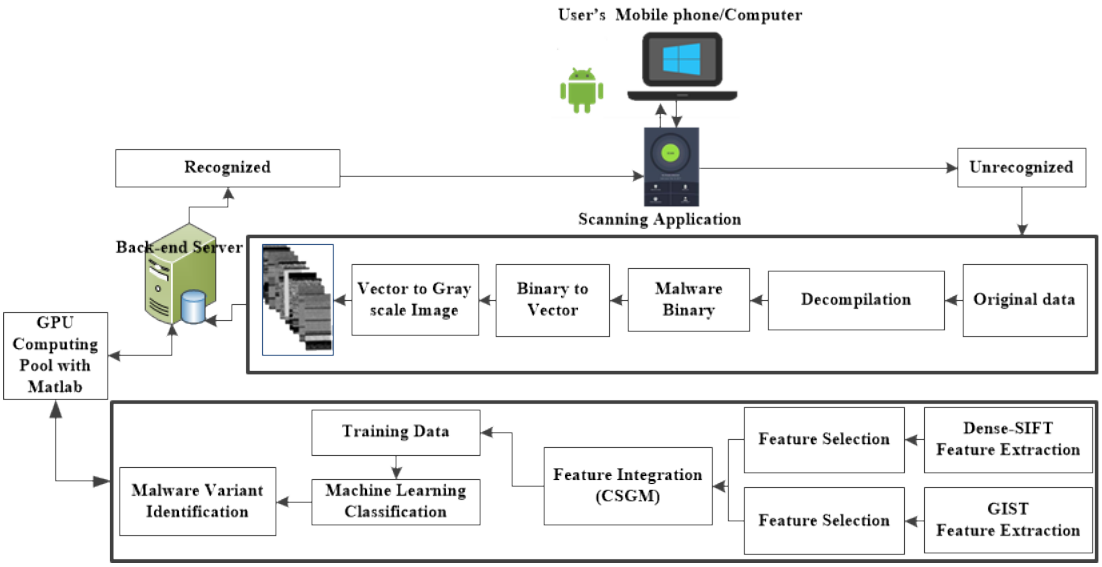


Fig. 2. The entire architecture of CP-MVCS

3.1 Malware Binary Preprocessing

In this paper, there was a need to unzip apk file for Android code visualization. Each apk file contained Dalvik Executable (DEX). We obtained byte code from apk file in three steps. First, we decompressed the apk file and received the class. Dex file. Second, we converted the class. Dex file into Java.Class file using the dex2jar tool. Finally, we used JD-GUI decompiler to extract byte code from Java.class file. The entire process is shown in **Fig. 3**. In the case of Windows code visualization, we took an executable file as an input data directly. Hence, there was no need to decompile code for Windows OS.

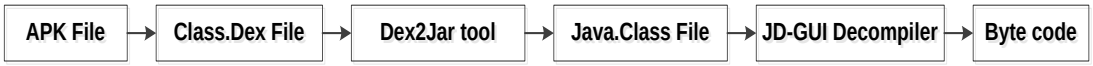


Fig. 3. Android APK decompilation

The concept of CP-MVCS is purely based on the construction of malware information from binary to an image. Our method for malware binary file to grayscale image transformation consisted of three significant steps. Firstly, the malware binary bit string separated into substrings. Each of substring was 8 bits in length and denoted as a pixel. Each eight-bit deliberated as an unsigned integer (0-255). Secondly, the malware binary bit string converted into a 1-D vector of decimal numbers. Thirdly, the 8-bit vector of decimal numbers

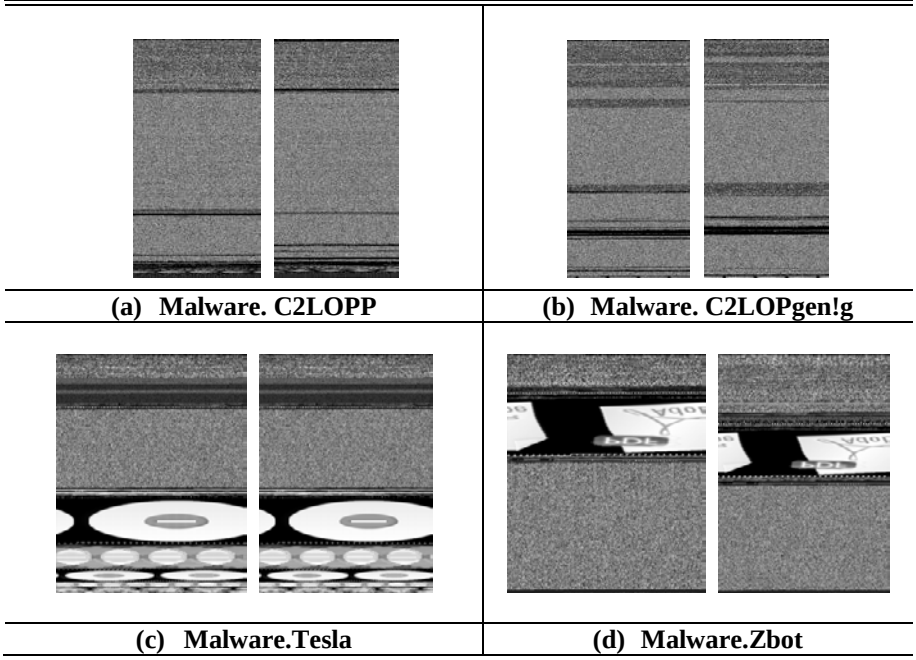
transformed into a two-dimensional matrix of the specified width. Finally, the two-dimension array directly converted a grayscale image. The entire process is shown in **Fig. 2**. Furthermore, the algorithm 1 described above steps briefly.

Algorithm 1: Malware Binary to Grayscale Image Transformation

Input: *Malware Binary File* M_B

```
1:  File_Size ← get_file_size(M_B)
2:  Width     ←  get_width(File_Size)
3:  Row       ← size(M_B)/Width
4:  Column    ← Width/8
5:  Initialize arr[Row][Column]
6:  for i=0 to Row do do
7:      for  j=0 to  Column do do
8:          arr[i][j]=convert 8-bit to unsigned integer (0-255)
9:      end for
10: end for
11: Grayscale Image ← convert 8-bit  vector to grayscale image
```

Output: *Grayscale Image*



(a) Malware. C2LOPP | **(b) Malware. C2LOPgen!g**

(c) Malware.Tesla | **(d) Malware.Zbot**

Fig. 4. Demonstration of benign and malware images

Fig. 4 showed visualization results of proposed methods. From **Fig. 4**, it observed that malware variants belonging to the same family were similar in texture and layout. The malware variants belonging to the related family were different from each other in texture and design. Beside this, we also observed that malware image size is not fixed and the reason behind that it depended on the size of a malicious software program.

3.2 CSGM Features Extraction

CSGM features extraction of CP-MVCS purely based on an image level, as both Dense SIFT and GIST descriptors extracted independent features from malware image. There are three steps to complete CSGM features extraction. The local features of malware image extracted by using Dense SIFT descriptor in the first step, which described the edges and corners of an image. The dimension of Dense SIFT descriptor reduced through Bag of Feature (BOF) [19] paradigm. There are four phases to reduce the dimension of Dense SIFT descriptor through BOF method. In the first phase, key features of malware image extracted from a dense grid with the help of Dense SIFT detector. In the second phase, Dense SIFT descriptor selected to compute 128-dimensional local features' vector from the rectangular area where each key feature expressed by a function as shown in Eq. (1).

$$\mathrm{DSIFT}_{\mathrm{FV}} = \mathrm{f}_{\mathrm{dsift}}(\mathrm{M}_{\mathrm{I}}, \mathrm{R}_{\mathrm{s}}, \mathrm{S}_{\mathrm{size}}, \mathrm{B}_{\mathrm{bounds}}) \qquad (1)$$

Where M_I represented a binary image of dimension $D_h \times D_w$; R_S denoted resizing parameter; S_{Size} denoted the size of Dense SIFT descriptor; B_{bounds} represented the rectangular area around each key point of a binary image. In our case, M_I = (256,256) pixels; R_S= (200,200) pixels; S_{Size} =16 and B_{bounds} was by default complete malware image respectively.

In the third phase; the most illustrative patches of $\mathrm{DSIFT}_{\mathrm{FV}}$ needed to be identified, which clustered into a pre-defined number of clusters using K-means [20] clustering technique, whereas the whole process known as dictionary learning as shown in Eq. (2).

$$\mathrm{Dictionary}_{\mathrm{L}} = \mathrm{f}_{\mathrm{kmeans}}(\mathrm{DSIFT}_{\mathrm{FV}}, \mathrm{D}_{\mathrm{size}}) \qquad (2)$$

Where D_{size} denoted dictionary size, whose value was 256 in our case.

At the end; $DSIFT_{FV}$ assigned to closest visual features of pre-defined $Dictionary_L$, whereas the entire process known as descriptor quantisation. Once the whole $dictionary_L$ learnt, each quantised descriptor visualized in the form of a histogram. The value of histogram occurrences varied between [0-1] scale and then generated BOF based Dense SIFT description by calculating minimum Euclidean distance between $DSIFT_{FV}$ and $\mathrm{Dictionary}_{\mathrm{L}}$, as shown in Eq. (3).

$$\mathrm{DSIFT}_{\mathrm{MFD}} = \mathrm{f}_{\mathrm{histogram}}(\mathrm{DSIFT}_{\mathrm{FV}}, \mathrm{Dictionary}_{\mathrm{L}}) \qquad (3)$$

In our case, the total dimensionality of $DSIFT_{MFD}$ was 256. The complete sequence of reducing the dimension of Dense SIFT descriptor through BOF method is shown in **Fig. 5**.

The global features of malware image computed by using GIST descriptor [21] in the second step, which provided texture and spatial layout of an image. There were three phases to describe malware images through GIST descriptor. In the first and second phase of GIST description, malware image filtered through a filter of various scales and locations and then

separated into blocks. In the last step, the average value of each block computed by GIST descriptor. GIST description for malware image calculated by using Eq. (4).

$$GIST_{MFD} = f_{gist}(M_I, [R_s, N_b, B_{overlap}, F_n, N_{sq}]) \quad (4)$$

Where M_I represented a malware image of dimension $D_h * D_w$; R_S denoted resizing parameter; $N_b = (b_x, b_y)$ represented a standard block size to divide malware image into horizontal and vertical locations;

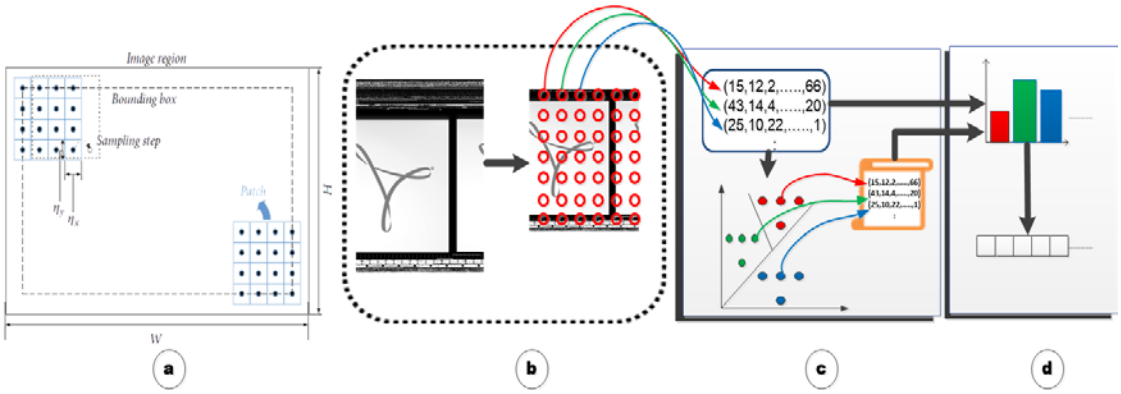


Fig. 5. BOF based on Dense SIFT description for malware image

$B_{overlap}$ denoted overlap parameter to separate binary image in overlap or non-overlap blocks; F_n represented a number of filters used to filter binary image, and N_{sq} denoted the number of statistical quantities, i.e. mean and deviation used while computing the features description. For our case, M_I = (256,256) pixels; R_S = (200,200) pixels; N_b = (4, 4); $B_{overlap}$ =0; F_n =16; N_{sq}=1 and dimensionality of $GIST_{MFD}$ was 256 ($F_n \times N_b \times N_{sq}$ =16×16×1=256) respectively. The sequence of steps to compute GIST description for malware image was shown in **Fig. 6**.

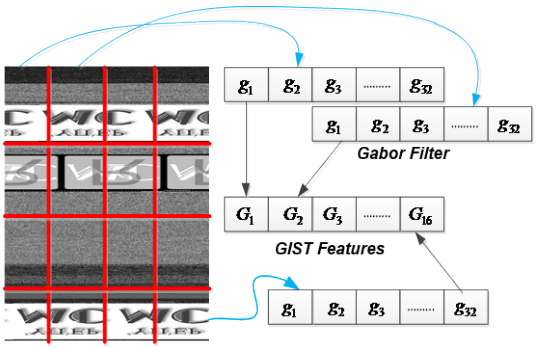


Fig. 6. GIST description for malware image

Finally, the combination of BOF based Dense SIFT descriptor with GIST descriptor obtained by integration, and dimensionality of resulting CSGM features was 512. The CSGM features description expressed by Eq. (5).

$$CSGM_{FD} = wDSIFT_{MFD} + (1\text{-}w)GIST_{MFD} \quad (5)$$

Here, *w* represented weighting parameter whose value depended on feature contribution in malware classification. According to Hamad et al. [22], a small-scale feature point has more considerable similarity to local information but requiring a broader range of global information. Formula computed the Gaussian weighted [22] value for each pixel of the binary image in Eq. (6).The brief explanation on CSGM features description is shown in algorithm 2.

$$w(x,y) = 1 - e^{\frac{-((x-x_{feature})^2-(y-y_{feature})^2)}{2\sigma^2}} \quad (6)$$

Algorithm 2: CSGM Features Extraction

Input: $Dictionary_L$ ← *Dictionary Learning*
$DSIFT_{FV}$ ← *DSIFT Feature Vector*
$DSIFT_{MFD}$ ← *DSIFT Malware Feature Description*
$GIST_{MFD}$ ← *GIST Malware Feature Description*
W ← *Weight Parameter*

1: ***if*** *$Dictionary_L$* ***!= exist then***
2: *Compute $DSIFT_{FV}$ by using equation in (1)*
3: *Construct $Dictionary_L$ for $DSIFT_{FV}$ by using K-means clustering equation (2)*
4: ***elseif*** *$Dictionary_L$* ***= exist AND*** *$DSIFT_{MFD}$* ***! = exist then***
5: *Compute $DSIFT_{FV}$ by using equation (1)*
6: *Compute minimum Euclidean distance between $DSIFT_{FV}$ and $Dictionary_L$*
7: *Construct histogram for $DSIFT_{FV}$ quantization by using equation (3)*
8: *Save 256-dimensional BOF based $DSIFT_{MFD}$ for each image*
9: *Compute and save 256-dimensional $GIST_{MFD}$ for each image by using equation (4)*
10: *Both $DSIFT_{MFD}$ and $GIST_{MFD}$ are concatenated to form $CSGM_{FD}$ using equation(5)*
11: *Calculate by using Gaussian weight formula in equation (6)*
12: *Save 512-dimensional $CSGM_{FD}$ for each image*
13: ***else*** *Initialize both $Dictionary_L$ and $CSGM_{FD}$ for Classification phase*
14: ***end***

Output: *CSGM Feature Description*

4. Experiments

4.1 Experimental Setup and Setting

The tests performed on CPU version Intel i5-4258U @ 240 GHz, the RAM was 4.0 GB, the operating system was Windows 10, and Matlab version R2017a developed the proposed method. The classifications and detections of CP-MVCS performed by Matlab [23] and Weka [24], which are open source data mining tools. Previously, several works [13, 14, 15, 16, 17] used machine-learning classification for malware detection with Naïve Bayes (NB), Nearest Neighbor (KNN) and Support Vector Machine (SVM). To obtain the best suitable classifier for experimentation, these classifiers compared with each other on Malimg dataset, as shown in **Table 1**. Due to the imbalanced dataset, the F-measure considered for comparison of all classifiers. The F-measure of KNN was 1.7% higher than that of NB and 6.4% higher than that of SVM. Using KNN classification algorithm of CP-MVCS, most of the performance indicators attained better results. For final classification of CP-MVCS, KNN was best suitable classifier between SVM and NB.

Table 1. The comparison of the performance of CP-MVCS on different classification algorithms

Classification Method	TPR	FPR	F-measure	Accuracy
SVM	0.902	0.0017	0.907	0.902
NB	0.965	0.001	0.954	0.967
KNN=3	**0.982**	**0.0007**	**0.971**	**0.984**

Zhihua et al. [36] indicated that texture features were lost in a smaller size image. While texture features were more apparent in larger size image, they also mentioned that bigger image required high computational time for training. From **Table 2**, CP-MVCS took more time to train larger image size. Therefore, we reshaped malware image into 256*256 size, as CP-MVCS performed better on that image size.

Table 2. The comparison of the performance of CP-MVCS on different image ratios

Image Ratio	Accuracy (%)	Prediction Time (sec)
156*156	95.57	9.43
256*256	**98.40**	**9.62**
356*356	98.87	16.39

Finally, three kinds of matrices such as True positive ratio, False positive ratio, F-measure and accuracy used for performance evaluation. Here, True positive and False positive represented the number of malware samples false and true classified. Similarly, the True negative and False negative represented the number of benign samples false and true classified.

$$\text{True Positive Ratio} = \frac{\text{True Positive}}{\text{True Positive+False Negative}} \tag{7}$$

$$\text{False Positive Ratio} = \frac{\text{False Positive}}{\text{False Positive+True Negative}} \tag{8}$$

$$\text{F-measure} = \frac{2*\text{True Positive}}{2*\text{True Positive+FalsePositive+False Negative}} \tag{9}$$

$$\text{Accuracy} = \frac{\text{True Positive+True Negative}}{\text{True Positive+True Negative+FalsePositive+False Negative}} \tag{10}$$

4.2 Malware Datasets Collection

Four public datasets: Malimg dataset, Malheur dataset, Virus share dataset and Android dataset used for evaluating CP-MVCS. Malimg dataset consisted of 9339 samples belonging to 25 Windows malware families and contained 617 benign samples. The dataset is the mixture of both packed and unpacked malware samples obtained from vision research lab of University California [25]. Malheur database obtained from a security research group of the University of Erlangen [26], which consisted of 3131 unpacked samples from 24 malware families. For experimental purpose, we selected only 17 malware families who had at least 13 samples. Virus share dataset [27] obtained from Virus share portal, which comprised over unpacked 2630 samples belonging to 14 Windows malware families. To assure that all those samples were malware, we scanned them using Virus Total [34]. We selected only those samples, which were reported as malicious. For labelling these samples, we used the label provided by Microsoft antivirus, as listed in **Table 3**. Beside this, we collected 4000 Android malware samples and 2000 benign Android samples from IKM Laboratory National Cheng Kung University, Taiwan [28].

Table 3. Information on malware samples

Dataset	Malware Families
Malimg [25]	Adialer.C, AgentFY.I, Allaple.A, Allaple.L, Aluerongen!.J, Autorun.K, C2LOPgen!g, C2LOP.P, Dialplatform.B, Dontovo.A, Fakerean, Instantaccess, Lolyda.AA1, Lolyda.AA2, Lolyda.AA3, Lolyda.AT, Malexgen!.J, Obfuscator.AD, Rbot!gen, Skintrim.N, Swizzorgen!E, Swizzorgen! I,VB.AT,Wintrim.BX,Yuner.A
Virus share [27]	Adnel, Ardamax, Bartallex, Conficker, Cutwail, IRCBOT, Locky, Rbot, Sirefef, Tesla, Zbot, Phobi, Rootkit, Agent
Malheur [26]	Adult Browser, Allapl.E, Bancos, Casino, Fly studio, Looper, Porndialer, Rotator, Salty, Spygames, Swizzor, Vapsup, Viking Dll, VikingDz, Virut, Woikoiner, Zhelatin

4.3. Experimental Results

4.3.1 Effect of Different sizes of Training Sets on Classification Rate of CSGM features

Several experiments conducted to compare the performance of CSGM features over traditional features, i.e. GIST features and Dense SIFT features. GIST is the global descriptor whose dimensionality is 512, whereas dimensionality of Dense SIFT descriptor is 512. Apart from these, CSGM features are the combination of both Dense SIFT and GIST features whose

dimensionality is 512. In this experiment, CP-MVCS evaluated with a compressed training set. The experimental samples took from Malheur dataset [26]. The research conducted on three types of features named Dense SIFT features (shape, edge), GIST features (texture) and CSGM features (edge, Shape, and texture). By comparing classification results of different feature extraction techniques in **Fig. 7**, it concluded that the CSGM features were superior to individual GIST features or Dense SIFT features. The comparison of feature extraction techniques measured by overall accuracy. The lowest classification rate on CSGM features was 62.33% when the size of the training set reduced to 10% of the total dataset. When the training set was about 80 % of the entire dataset, the maximum classification rate on CSGM features was 94.13%. Besides this, the accuracy of Dense SIFT features 4.89% was higher than GIST features. Even though GIST features achieved the lowest efficiency, but it increased the efficiency of Dense SIFT features by 12.01%. Hence, it concluded that the addition of GIST features naturally enhanced the effect of the overall classification. The overall result also showed that the classification accuracy of CP-MVCS was not over-reliant on the training set. To reduce the risk of misclassification, we selected 80% training and 20% test data for further experimentation.

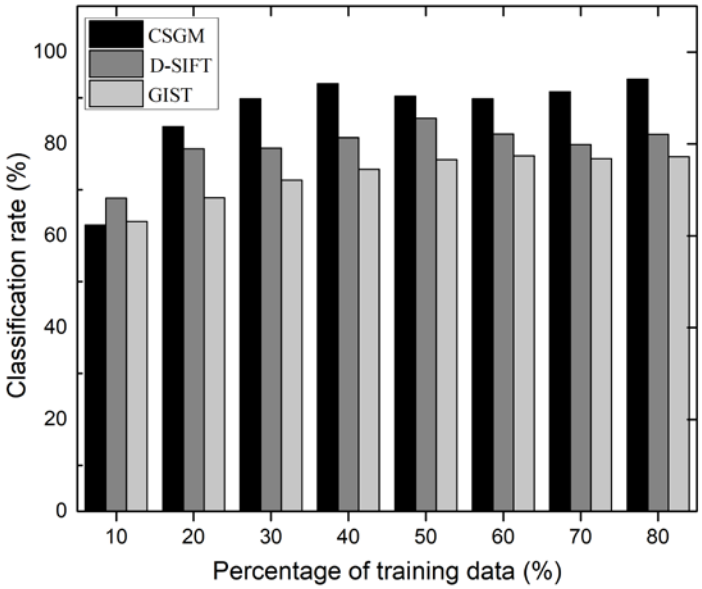


Fig. 7. Classification rate for different sizes of training sets

4.3.2. Effect of CSGM features on Classification Rate and Run Time Cost

Three cases considered for evaluating classification performance and run time cost between CSGM features and traditional features. The classification stability of CSGM features identified by examining the classification performance of each family with overall run time improvement in case I. The detection performance with betterment in running time of CSGM features evaluated by unknown malware detection in case II and case III respectively. The detail experimental results of three cases discussed one by one in this section.

The case I: The critical section of the experimental repository was malware samples, collected from Malimg dataset [25]. Malimg dataset randomly separated into a training set, and the resemblance between families of the training set computed through three different types of

features in this case. The comparison of the overall classification rate among CSGM features and traditional features was shown in **Table 4**. As shown in **Table 4**, CSGM features outclassed the conventional features in the terms overall classification accuracy. The maximum accuracy obtained on CSGM features was 98.40%, whereas it provided the highest TP rate (98.20%) and lowest FP rate (0.07%) among traditional features.

Table 4. Comparison of classification rate among CSGM features and former features in Case I

Features Type	Feature-length	TPR (%)	FPR (%)	Classification Accuracy (%)
Dense SIFT	512	89	0.19	89.29
GIST	512	93.10	0.13	93.44
CSGM	**512**	**98.20**	**0.07**	**98.40**

The classification accuracy results for three different types of features were shown in **Fig. 8**. The experimental outcomes also showed that the CSGM features achieved better classification accuracy than former features among most malware families. The The entire classification of CSGM features between each malware family was shown with various evaluation matrices in **Fig. 9**.

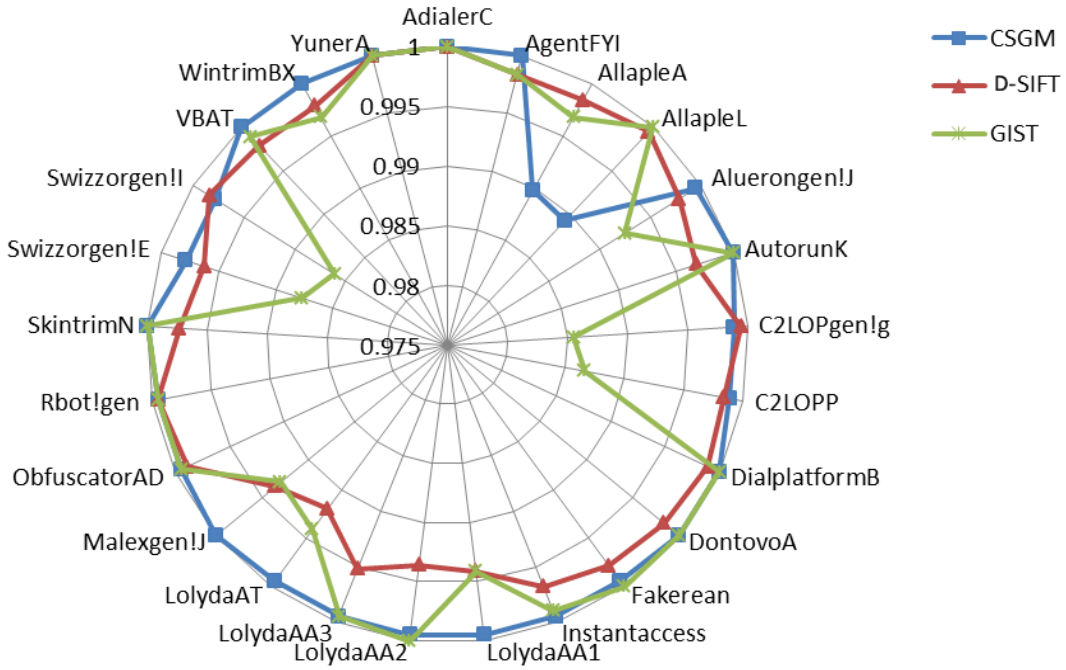


Fig. 8. Comparison of classification accuracy among CSGM features, Dense SIFT features and GIST features**.**

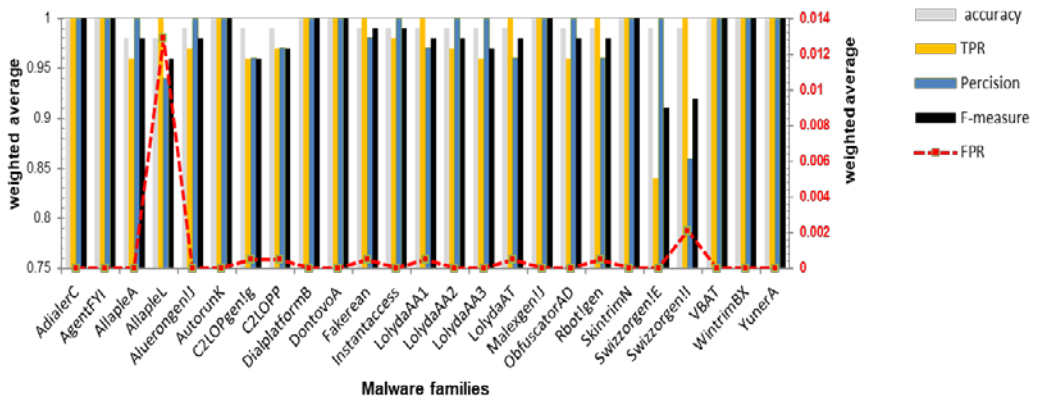


Fig. 9. Classification of CSGM features among each malware family with various performance matrices

The experimental results showed that average classification accuracy of CSGM features ranged from 0.98-1**.** The results showed that the CSGM features classified most of the malware families with better accuracy, TP rate, F-measure and lowest FP rate. The range of TP rate was 0.84-1 and FP rate was 0-0.0129. The classification accuracy of CSGM features among each malware family also presented with confusion matrix in **Fig. 10**.

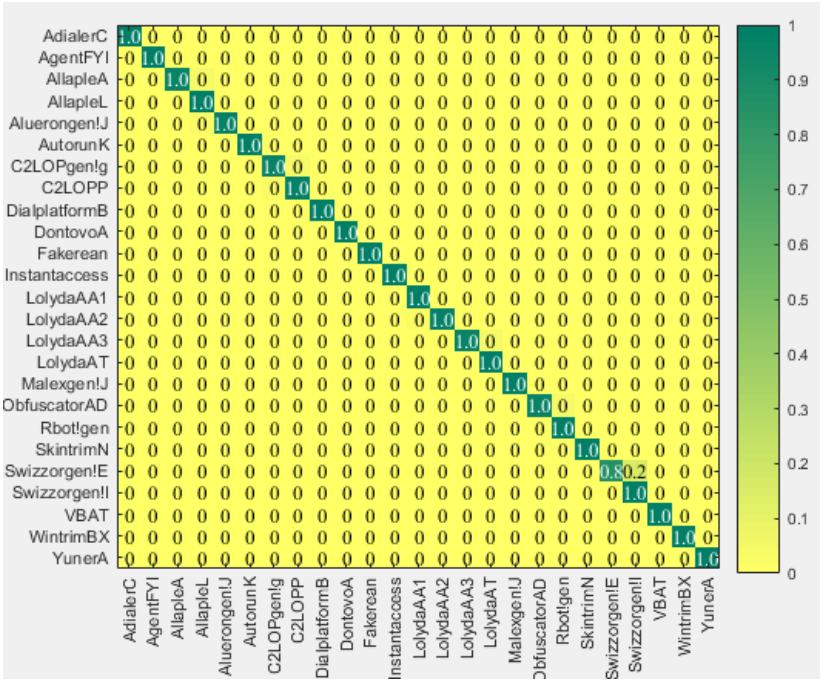


Fig. 10. Confusion matrix among each malware family for CSGM features

The confusion matrix indicated that CSGM features separated most of the malware families efficiently. However, there was confusion among families Allaple .A, Allaple .L, Aluerongen! .J, C2LOP.P, C2LOPgen!.g, Lolyda.AA1, Lolyda.AA2, Lolyda.AT,

Lolyda.AA3, Obfuscator.AD, Rbot! gen, Swizzorgen! .E and Swizzorgen! .I. Most of these were variants of Allaple, C2LOP, Lolyda and Swizzorgen families, which meant that image visualization of variants of these families appears similar in structure.

Case II: The entire families of Malimg dataset [25] combined as a malware set. The dataset consisted of 9339 malware samples and 617 benign samples. The samples include in test data were unknown as they were not used in the training phase. The overall detection accuracy of three diverse types of features was shown in **Table 5**.

Table 5. Comparison of classification rate between CSGM features and former features in Case II

Features Type	Feature-length	TPR (%)	FPR (%)	Classification Accuracy (%)
Dense SIFT	512	88.46	0.20	88.68
GIST	512	90.84	0.19	91.16
CSGM	**512**	**97.20**	**0.09**	**97.29**

CSGM features outperformed conventional features regarding overall detection accuracy. The overall detection accuracy (97.29 %) with highest TP rate (97.20 %) and lowest FP rate (0.09 %) achieved on CSGM features. Hence, in the case of maximum training samples, it concluded that the CSGM features were more suitable to deal with unknown malware detection than the previous features.

Case III: To minimise the effect of unbalance data in Malimg dataset [25], the ratio of data kept same to ensure that the proportion of each family of training and test sets remained the same. After picking 617 malware and 617 benign samples, three different types of features used to evaluate the detection performance of unknown malware. The dimension of features minimized to 256. The results for the detection accuracy of each kind of features were shown in **Table 6**. The accuracy of GIST features 8.9% was higher than that of the Dense SIFT features. Though the accuracy of the Dense SIFT features was lowest, it considerably enhanced the overall efficiency of GIST features by 1%. It showed that the addition of local feature always increased the effect of detection.

Table 6. Comparison of classification rate between CSGM features and former features in Case III

Features Type	Feature-length	TPR (%)	FPR (%)	Classification Accuracy (%)
Dense SIFT	256	89.00	10.57	89.43
GIST	256	97.97	3.07	97.40
CSGM	**256**	**98.00**	**2.04**	**97.96**

Run time cost: The prediction time of different features recorded on Malimg dataset [25]. The dimension of each feature was 512. In **Fig. 11**, the prediction time of GIST features was 0.4 seconds longer than Dense SIFT features. While the prediction time of GIST features was 0.2 seconds longer than CSGM features. Although the extraction time of CSGM features was slightly shorter than GIST features, still it was more accurate to process small-scale and large-scale malware features, as discussed in the above experimentation.

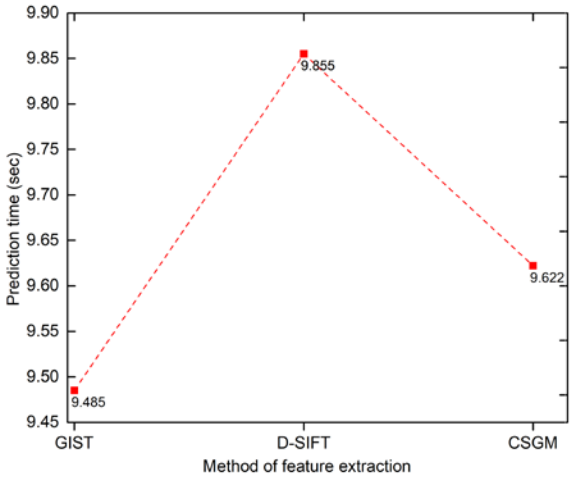
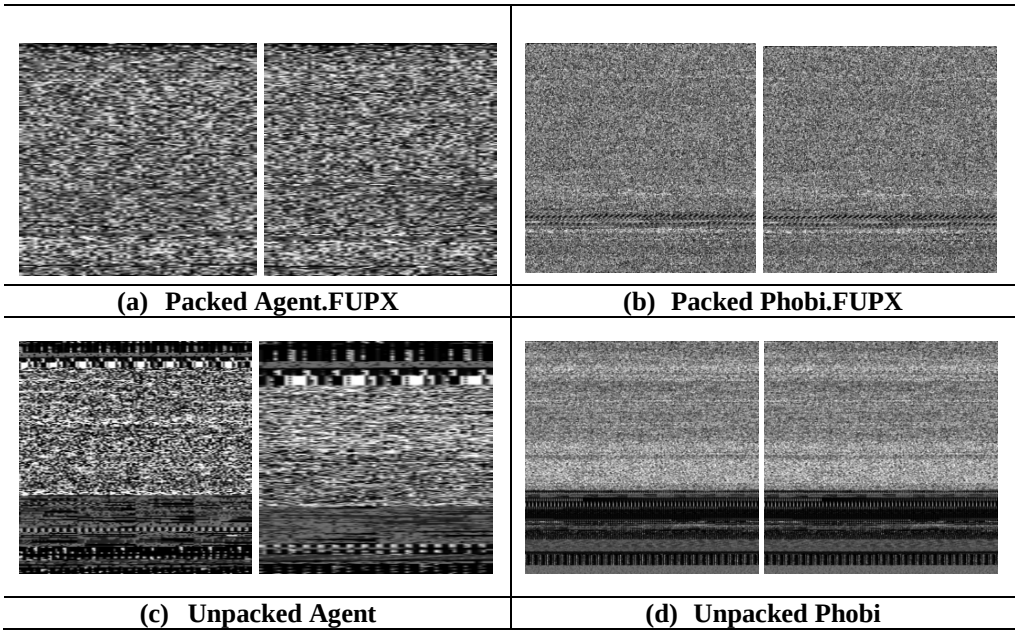


Fig. 11. Prediction time cost for CSGM, GIST and Dense SIFT features

4.3.3. Effect of Packed Malware on CP-MVCS Classification Rate

There was a misunderstanding that if two malware binaries belonging to different families were packed (code obfuscation) using the same packer, then they had similar characteristics. Unfortunately, the conventional static code analysis techniques did not capture the structural features of a malware binary. After packing with the same packer, the images of malware variants belonging to different families were indeed different [33]. In our malware visualization results, the visual similarity among packed variants of a family maintained as shown in **Fig. 12 (a-b)**. We also observed that unpacked variants of both families were completely different from their packed variants in **Fig. 12 (c-d)**.

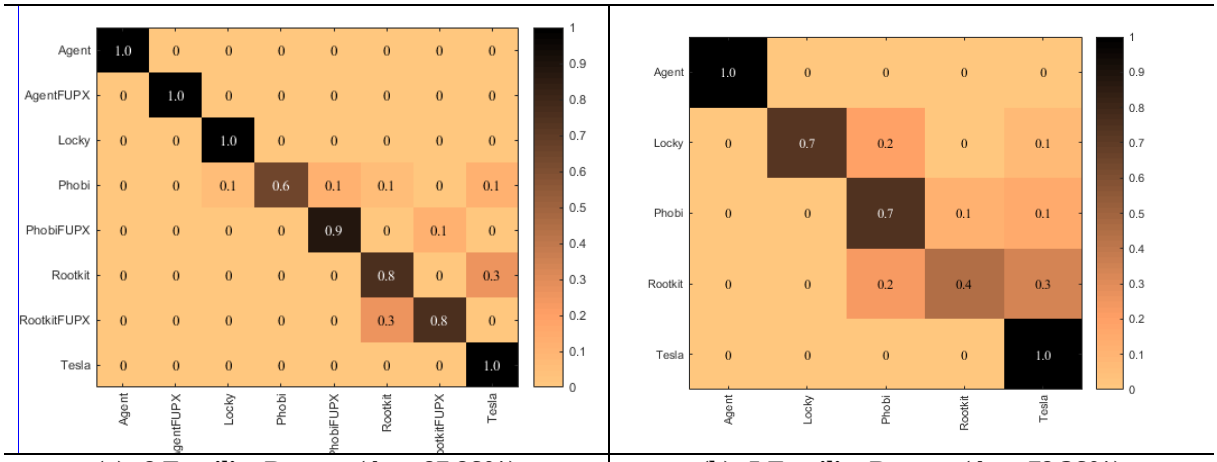


(a) Packed Agent.FUPX | **(b) Packed Phobi.FUPX**

(c) Unpacked Agent | **(d) Unpacked Phobi**

Fig. 12. Unpacked and packed variants of Agent and Phobia families

For our case, we collected 181 unpacked binary executables from Virus share [27] portal. As we performed supervised learning of classification, we labelled these binary executables using Virus total [34] search engine. To evaluate the performance of CP-MVCS over packed malware, we selected 48 unpacked binary executables and packed them using FUPX packing tool [35]. We performed dataset mixing using two different ways. First, we prepared eight families dataset by mixing unpacked and the packed malware families. We treated packed families as new families as shown in **Fig. 13 (a)**. We attained 87.86% classification accuracy while took only 181 samples (133 unpacked samples and 48 packed samples) of 8 malware families. Second, we prepared five families dataset by mixing packed and unpacked malware families. We treated packed and unpacked families as a single family as shown in **Fig. 13 (b)**. We obtained 78.26% classification accuracy while took only 181 samples (133 unpacked samples and 48 packed samples) of 5 malware families. For both experiments in CP-MVCS, the packed variants did not misclassify with their unpacked variants. The results showed that similarity was preserved in the dataset, which contained both unpacked and packed malware variants.



(a) 8 Families Dataset (Acc=87.86%) **(b) 5 Families Dataset (Acc=78.26%)**

Fig. 13. Confusion matrix for both Unpacked and Packed malware families

4.3.4 Effect of Datasets from Different Operating Systems on CP-MVCS Classification Rate

The classification accuracy of CP-MVCS verified on malware samples of Windows OS and Android OS. The experiment performed on five popular malware datasets of Windows OS and Android OS. In each dataset, the number of malware families described in **Table 3**. The performance of CP-MVCS over Windows OS and Android OS was shown in **Table 7**.

Table 7. Effect of datasets from different operating systems on CP-MVCS classification rate

Dataset	Operating System	TPR	FPR	Classification Accuracy
Virusshare [27]	Window OS	0.962	0.005	0.962
Malheur [26]	Window OS	0.975	0.0005	0.975
IKM Laboratory [28]	Android OS	0.975	0.03	0.963
Virusshare + IKM Lab	Window OS + Android OS	0.909	0.09	0.909

To verify the performance of CP-MVCS on combine dataset of Android OS and window OS, we collected 1191 Windows malware samples,1999 Android malware samples and 1000 benign Android samples for the experiment. We observed that performance of CP-MVCS was less accurate as compared to individual datasets of window OS and Android OS, but still, it was acceptable. For combine datasets of Android OS and window OS, CP-MVCS obtained maximum classification accuracy, as shown in **Table 7**. Hence, it proved that the classification accuracy of CP-MVCS was not over-reliant on malware samples of the specific operating system.

4.3.5 Comparison of CP-MVCS with Existing Previous Methods

CP-MVCS compared with texture-based visual analysis and bigram based static analysis to evaluate the overall performance regarding classification accuracy. The texture based visualization approaches proposed in [13], [15] and [16] compared with CP-MVCS were shown in **Table 8**. The comparison conducted within Malimg dataset [25]. CP-MVCS showed highest classification rate (99.21%) by processing 5288 samples from 8 malware families in Malimg dataset. Similar to that, CP-MVCS also showed better classification rate (98.40%) by taking 9339 samples from 25 malware families in Malimg dataset. Hence, it proved that CP-MVCS was more appropriate for processing the massive amount of malware samples as compared to other proposed works in [13], [15] and [16].

Table 8. Comparison of CP-MVCS with proposed malware visualization approaches

Method	Feature Types	No. of Malware Families	No. of Malware Samples	Classification Accuracy (%)
Nataraj et al [13]	GIST	8	1713	99.93
Aziz et al [15]	Gabor Wavelet, DWT, GIST	8	3320	98.88
Our method	**CSGM**	**8**	**5288**	**99.21**
Nataraj et al [13]	GIST	25	9339	97.18
Konstantinos etal [16]	GIST	25	9339	91.60
Our method	**CSGM**	**25**	**9339**	**98.40**

Further, CP-MVCS compared with bigram based static analysis approach proposed in [29]. The complete comparison of both techniques with Malimg dataset [25] was shown in **Table 9**.

Table 9. Comparison of CP-MVCS with traditional n-gram static analysis approach

Method	Feature Types	No. of Families	No. of Malware Samples	Classification Accuracy (%)
Karim et al [29]	Bigram	8	1713	98
Our method	**CSGM**	**8**	**5288**	**99.21**

The overall classification time of CP-MVCS was 3.7 seconds; whereas the total classification time of proposed bigram based static analysis in [29] was 5 seconds. From **Table 9**, it

observed that overall classification accuracy of CP-MVCS was much better as compared to bigram based static analysis approach proposed in [29].

5. Conclusion and Future work

In this paper, a CP-MVCS has been proposed to analyze malware by processing malware binaries as images visually. Our proposed CP-MVCS showed that variants belonging to same malware family appear similar when they are converted to images. To get an ideal performance of CP-MVCS, the visual similarities among malware variants are computed through CSGM features description, and they are classified with various machine-learning methods. Experimental outcomes showed that CP-MVCS detects malware variants and organize families with a small false positive and a high true positive rate. It is not only differentiated families of malware variants but also identifies mix malware and clean samples efficiently. Although proposed CP-MVCS approach for malware variant detection and classification is novel, still it is unable to detect section relocation. Due to this shortcoming, we need to explore more localized feature extraction schemes such as LBP (Local Binary Pattern). Even though in CP-MVCS, the classification time cost is better as compared to traditional methods but still it needs to be more improved. We will use the PCA (Principal Component Analysis) to target this issue. Moreover, in CP-MVCS, we used BOF model for local features description, and local feature description is computed through K-means clustering algorithm that is much slower. Hence, instead of K-means clustering algorithm, in future, we can use other clustering algorithms such as GMM (Gaussian Mixture Model) for local features description in BOF model.

References

[1] Thuy Yilin Ye, Lifa Wu, Zheng Hong and Kangyu Huang, "A Risk Classification Based Approach for Android Malware Detection," *KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS*, vol. 11, no. 2, pp. 959-981, February 2017. Article (CrossRef Link).

[2] KyoungSoo Han, Jae Hyun Lim and Eul Gyu Im, "Malware Analysis Method using Visualization of Binary Files," in *Proc. of 2013 ACM Conf. on Research in Adaptive and Convergent Systems*, pp. 317-321, October 1-4, 2013. Article (CrossRef Link).

[3] Asaf Shabtai, Robert Moskovitch, Yuval Elovici and Chanan Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16-29, February 2009. Article (CrossRef Link).

[4] S. Cesare,Y. Xiang and W. Zhou, "Control Flow-Based Malware Variant Detection," *IEEE Transaction Dependable and Secure Computing*, vol. 11, no. 4, pp. 307-317, July 2014. Article (CrossRef Link).

[5] Shanhu Shang, Ning Zheng and Jian Xu, "Detecting malware variants via function-call graph similarity," in *Proc. of 2010 IEEE Conference on Malicious and Unwanted Software*, pp. 113-120, October 19-20, 2010. Article (CrossRef Link).

[6] T. Abou-Assaleh, N. Cercone, V. Keselj and R. Sweidan, "N-gram based detection of new malicious code," in *Proc. of 2004 IEEE Conference on Privacy, Security and Trust*, pp. 193-196, October 18, 2004. Article (CrossRef Link).

[7] Igor Santos, Jaime Devesa, Félix Brezo, Javier Nieves and Pablo Garcia Bringas, "OPEEM: A Static-Dynamic Approach for Machine-Learning Based Malware Detection," in *Proc. of 2012*

Springer Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions, pp. 271-280, 2012. Article (CrossRef Link).
[8] Vinod P. Nair , Harshit Jain, Yashwant K. Golecha, Manoj Singh Gaur and Vijay Laxmi, "MEDUSA: Metamorphic malware dynamic analysis using signature from API," in *Proc. of 2010 ACM Conf. on Security of Information and Networks*, pp. 263-269, September 7-11, 2010.
[9] Qi-Guang Miao, Yun-Wang and Ying-Cao, "API Capture – A tool for monitoring the behavior of malware," in *Proc. of 2010 IEEE Conference on Advanced Computer Theory and Engineering*, pp.390-394, September 20, 2010. Article (CrossRef Link).
[10] Matt Fredrikson, Somesh Jha and Mihai Christodorescu, "Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors," in *Proc. of 2010 IEEE Conference on Security and Privacy*, pp. 45-60, May 16-19,2010. Article (CrossRef Link).
[11] Naqqash Aman, Naqqash Amanan, Fahim H. Abbasi and Farrukh Shahzad, "A Hybrid Approach for Malware Family Classification," in *Proc. of 2017 Springer Conference on Applications and Techniques in Information Security*, pp. 169-180, 2017. Article (CrossRef Link).
[12] KyoungSoo Han, Jae Hyun Lim, Boojoong Kang and Eul Gyu Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, no. 1, pp. 1-14, April 29, 2014. Article (CrossRef Link).
[13] L. Nataraj,S. Karthikeyan,G. Jacob and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proc. of 2011 ACM Conf. on Visualization for Cyber Security*, pp. 1-4, July 20, 2011. Article (CrossRef Link).
[14] Kesav Kancherla and Srinivas Mukkamala, "Image Visualization based Malware Detection," in *Proc. of 2013 IEEE Conference on Computational Intelligence in Cyber Security*, pp. 40-44, April 16-19,2013. Article (CrossRef Link).
[15] Aziz Makandar and Anita Patrot, "Malware Class Recognition Using Image Processing Techniques," in *Proc. of 2017 IEEE Conference on Data Management, Analytics and Innovation*, pp. 76-80, February 24-26, 2017. Article (CrossRef Link).
[16] Konstantinos Kosmidis, "Machine Learning and Images for Malware Detection and Classification," 2017.
[17] Ban Xiaofang, Chen Li, Hu Weihua and Wu Qu, "Malware Variant Detection Using Similarity Search over Content Fingerprint," in *Proc. of 2014 IEEE Conference on Control and Decision*, pp. 5334-5339, May 31-June 2, 2014. Article (CrossRef Link).
[18] Ultimate Packer for Executables UPX. http://upx.sourceforge.net/
[19] Francesco Ciompi, Colin Jacobs, Ernst Th. Scholten, "Bag-of-frequencies: a descriptor of pulmonary nodules incomputed tomography images," *IEEE Transaction on Medical Imaging*, vol. 34, no. 4, pp. 962-973, November, 2014. Article (CrossRef Link).
[20] Yan Ke, R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors", in *Proc. of 2004 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 506–513, 27 June-2 July, 2004. Article (CrossRef Link).
[21] Aude OlivaAntonio Torralba, "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145-175, May, 2001. Article (CrossRef Link).
[22] Hamad Naeem, Bing Guo, Muhammad Rashid Naeem,Muhammad Aamir,Muhammad Sufyan Javed, "A new approach for image detection based on refined Bag of Words algorithm," *Optik - International Journal for Light and Electron Optics*, vol. 140, pp. 823-832, July, 2017. Article (CrossRef Link).
[23] Weka. http://www.cs.waikato.ac.nz/ml/weka
[24] Matlab. https://www.mathworks.com/products/matlab.html
[25] Malimg Dataset. https://vision.ece.ucsb.edu/
[26] Malheur Dataset. https://www.sec.cs.tu-bs.de/data/malheur/
[27] Virus Share. http://www.virusshare.com/
[28] IKMLaboratory. https://sites.google.com/site/nckuikm/home/

[29] Md. Enamul Karim, Andrew Wallenstein, Arun Lakhotia and Laxmi Parida, "Malware phylogeny generation using permutations of code," *Journal of Computer Virology*, vol. 1, no. 1, pp. 13-23, December 20, 2005. Article (CrossRef Link).
[30] S. Nari, A. Ghorba.ni. "Automated malware classification based on network behavior," in *Proc. of international conference on computing, networking and communications (ICNC)*, pp. 642-647, 2013. Article (CrossRef Link).
[31] Symantec, "Internet security threat report," 2017.
[32] Kaspersky Lab, "Cyber financial threat report," 2017.
[33] T. Ban, R. Isawa, S. Guo, D. Inoue, K. Nakao, "Efficient Malware Packer Identification Using Support Vector Machines with Spectrum Kernel," in *Proc. of Eighth Asia Joint Conference on Information Security*, pp. 69-76, 2013. Article (CrossRef Link).
[34] Virus Total. https://www.virustotal.com/en/statistics/
[35] FUPX Packer. https://portableapps.com/apps/utilities/free_upx_portable
[36] Zhihua C, Fei X, Xingjuan C, Yang C, Gai-ge W, Jinjun C, "Detection of Malicious Code Variants Based on Deep Learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187-3196, 2018. Article (CrossRef Link).

Hamad Naeem He graduated from NFC IET, Multan, Pakistan, and Chongqing University, Chongqing, China in 2012 and 2016, and received B.S. and M.E., respectively. He received excellent master student award from Chongqing University, China in 2016. Currently, he is pursuing Ph.D. degree in Software Engineering at Sichuan University, China. He has published various articles in reputed SCIE and EI journals/conferences. His research interest includes malware detection, image processing, internet security, and machine learning.

Bing Guo He received his BS degree in Computer Science from the Beijing Institute of Technology in China, and MS and PhD degrees in Computer Science from the University of Electronic Science and Technology of China, China, in 1991, 1999, and 2002, respectively. He is currently a Professor in the School of Computer Science at the Sichuan University, China. His current research interests include embedded real-time system and big data management.

Farhan Ullah Farhan Ullah received his MS in computer sscience degree from CECOS University Peshawar, Pakistan, in 2012 and BS in computer science degree from University of Peshawar, Pakistan, in 2008. He is currently pursuing PhD degree in computer science from School of Computer Science, Sichuan University Chengdu, China. He has authored/co-authored 16 publications including 8 SCI/SCIE indexed journals. His research interests include software similarity and data science.

Muhammad Rashid Naeem Muhammad Rashid Naeem was born in Rawalpindi, Punjab, Pakistan in 1990. He received his bachelor's degree in software engineering from International Islamic University Islamabad, Pakistan and master's degree in software engineering from Chongqing University, China in 2012 and 2015 respectively. Currently, he is PhD student at Sichuan University, China. He is author of various articles published in reputed journals and conferences. His current research interests include mutation-testing, software testing through static analysis and machine learning techniques.