Journal of King Saud University – Computer and Information Sciences xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A learning model to detect maliciousness of portable executable using integrated feature set

Ajit Kumar [a], K.S. Kuppusamy [a,*], G. Aghila [b]

[a] Department of Computer Science, Pondicherry University, Pondicherry 605014, India
[b] Department of Computer Science and Engineering, National Institute of Technology Puducherry, Karaikal 609605, India

ARTICLE INFO

Article history:
Received 14 July 2016
Revised 5 January 2017
Accepted 6 January 2017
Available online xxxx

Keywords:
Malware
Portable executable
Machine learning
Integrated features

ABSTRACT

Malware is one of the top most obstructions for expansion and growth of digital acceptance among the users. Both enterprises and common users are struggling to get protected from the malware in the cyberspace, which emphasizes the importance of developing efficient methods of malware detection. In this work, we propose a machine learning based solution to classify a sample as benign or malware with high accuracy and low computation overhead. An integrated feature set has been amalgamated as a combination of portable executable header fields raw value and derived values. Various machine-learning algorithms such as Decision Tree, Random Forest, kNN, Logistic Regression, Linear Discriminant Analysis and Naive Bayes were adopted in the classification of malware. Using existing raw feature set and the proposed integrated feature set we compared performance of each classifier. The empirical evidence indicates 98.4% classification accuracy in the 10-fold cross validation for the proposed integrated feature set. In the experiments conducted on the novel test data set the accuracy was observed as 89.23% for the integrated feature set which is 15% improvement on accuracy achieved with raw-feature set alone. Classification accuracy with only top N features ($N = 5, 10, 15, 20, 25$) are also experimented and it was observed that with only top 15 features **98%** and **97%** accuracy can be achieved on integrated and raw feature respectively.

© 2017 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Malicious program or malware is an intentionally written program to indulge in various malicious activities, ranging from user's information stealing to cyber-espionage. The behavioral dynamism exposed by the malware is dependent on various factors such as nature of the attack, sophisticated technology and the rapid increase in exploitable vulnerabilities. Malware attacks also increased along with the rapid growth in the use of digital devices and internet. The exponential increase in the creation of new malware in the last five years made the malware detection as a challenging research issue.[1]

Malware detection is the technique for identifying malware in the end devices or networks. Signature based detection and Non-signature based detection are the two major classes of malware detection techniques used today (Idika and Mathur, 2007).

Signature based detection techniques scan for the specific sequence of bytes (i.e. signature) to detect malware, whereas non-signature based techniques detect malware by observing behaviors and patterns against predefined profile. Signature based detection has high detection accuracy but is limited to the signature database, hence requires frequent updates. The major drawback is that it provides an attack window time (time between a vulnerability discovery to signature update or patching) equal to the sum of detection time, signature generation time and updating time. Non-signature based detection can raise false positive or false negative result but it totally eliminates the attack window time. Non-signature based detection techniques can also detect unknown, zero-day and modern malware whereas such malware gets totally undetected in signature-based detection techniques.

This paper proposes a non-signature based approach to detect malware on the basis of an integrated feature set prepared by processing Portable executable (PE) file's header fields values.

Various approaches exist in the non-signature based methods such as heuristics, machine learning and hybrid techniques (Treadwell and Zhou, 2009; Firdausi et al., 2010; Qin et al., 2010; Schultz et al., 2001). The machine learning based malware classification utilizes the structural and behavioral features of malware and benign programs to build a classification model to identify a given sample program as malware or benign. Byte-n-gram,

* Corresponding author.
E-mail address: kskuppu@gmail.com (K.S. Kuppusamy).

[1] Source: http://www.av-test.org/en/statistics/malware/ (Last Accessed:Oct 2016).

http://dx.doi.org/10.1016/j.jksuci.2017.01.003
1319-1578/© 2017 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

opcode-n-gram, string present in binary and PE header fields values are main features used for training and testing malware classifiers (Schultz et al., 2001; Abou-Assaleh et al., 2004; Moskovitch et al., 2008; Santos et al., 2010; Bilar, 2007; Sami et al., 2010; Shankarapani et al., 2011; Shabtai et al., 2009). Various fields of PE header are used to discriminate between malware and benign file. For example, in our pilot study we observed that the values for *NumberOfSections*, *SizeOfInitializedData*, *NumberOfSymbol*, etc. have larger deviation among malware and benign files.

In this proposed work, an integrated feature-based model is made amalgamating the raw values of various fields of PE header along with a set of derived features. The values for derived features are computed by using header fields, binary file data and the guideline proposed in Pietrek (1994). Derived feature set comprises entropy, year, packer information, the number of suspicious sections, the number of non-suspicious sections and 5 other boolean features, which are explained further in Section 3.2.2. Aforementioned integrated features are extracted from the pre-processed malware and benign samples.

Six machine-learning classifiers Logistic Regression, Linear Discriminant Analysis, Decision Tree, Random Forest, K-Neighbors Classifier and Gaussian Naive Bayes are trained on raw and integrated feature set. All the six trained models are tested on various classification metrics using train-test split method (70–30 ratio), 10-fold cross validation and a novel test dataset having 122 samples from different classes of malware and 30 benign samples that are not used in training. To verify the improvement in the classification accuracy and other metrics, the output from the raw and integrated feature set are compared. The proposed method is also compared with the existing model (Bai et al., 2014) which employs only the raw feature set.

The proposed learning model for malware detection has four major objectives as listed below:

- To build a feature set consisting of sixteen components with values derived from PE header fields by comparing with the guidelines provided for respective fields.
- To amalgamate an integrated feature set with derived and raw features that would enhance the accuracy of maliciousness detection of PE file.
- To develop a classification model with six machine learning based classifiers using the integrated feature set.
- To empirically validate the proposed classification model with six phase experimental setup which is aimed towards confirming the performance improvement using established classification metrics.

The remainder of this paper is organized as follows. In Section 2, background information and related works on malware classification with respect to PE files using headers information are discussed. Methodology and details of different features are discussed in Section 3. Experimental setup, results, and analysis are given in Section 4. Conclusions derived from this work are presented in Section 5.

2. Related works

Malware detection is the process of differentiating malware from benign programs. Both the malware sophistication and anti-malware techniques have grown together in an arms race of suppressing each other. Malware detection techniques (Signature-based and Non-signature based) by its type of analysis are categorized into static and dynamic. Static analysis does not execute the sample whereas dynamic analysis executes the sample in a controlled environment (Egele et al., 2012). Non-signature based techniques use both kinds of analysis.

The proposed work is based on static analysis and so this related work section explains some of the notable previous works on non-signature based malware detection based on static analysis of PE files. The proposed work has considered the PE files for maliciousness detection. The reason for selecting PE for the experiment is two folds. Primarily, it is one of the most used file formats due to the wide use of Windows operating system and secondly, around 47.80% of files submitted to *Virustotal*[2] are PE files. Fig. 1 shows the percentage of different file types submitted to *Virustotal*. The data taken from *Virustotal* are grouped to make the visualization more clear and meaningful.

Schultz et al. (2001) introduced the usage of static features of malware and benign programs with data mining methods for malware classification. They experimented with different static features of executable file such as used Dynamic Link Library (DLL), DLL function calls, Application Programming Interface (API) calls, and strings present in binary files. Different data mining classifiers (RIPPER, Naive Bayes and Multi-Naive Bayes) were trained with a standardized dataset of 3265 malware and 1001 benign programs. They claimed 97.6% accuracy for the Multi-naive classifier, double than the signature-based detection for unknown malware.

Shabtai et al. (2009) presented taxonomy for classifying detection methods of malware by machine learning methods based on static features extracted from the executable. They explained various static features, among which PE features are grouped under six categories as data from PE header and physical structure of PE, Optional PE header information, Import section, Export Section, Resource directory, and Version information.

Vinod et al. (2011) used three different types of features- mnemonic n-gram, principal instruction opcodes, and PE header entries. They used scatter criterion for feature selection on two datasets created with pairing obfuscated and packed malware with benign samples. The best performance of TPR 0.968% and FPR 0.138 was reported with Instance Based learner (IBk) on packed malware dataset.

Ye et al. (2008) used API call sequence of PE file (extracted using Import Address Table (IAT) field) with Objective Oriented Association (OOA) mining. With aforementioned method, they achieved an accuracy of 93.7% using Max-Relevance feature selection methods.

Shafiq et al. (2009) extracted 189 features from PE file headers, used three different feature selection methods (Redundant Feature Removal (RFR), Principal Component Analysis (PCA) and Haar Wavelet Transform (HWT)) to reduce dimension of feature set, and trained five data mining classifiers (IBk, J48, NB, RIPPER & SMO). This approach gave an accuracy of 99% and 0.5% of False Positive (FP). They experimented with large data samples that have more than ten thousand samples and the resultant classifier on average took 0.244 seconds to scan a new file.

Wang et al. (2009) proposed a SVM based detection method for detecting unseen PE malware. Using static analysis, PE header entries were extracted and the SVM classifier was trained using selected features. Classification model proposed by Wang et al. (2009) detects viruses and worms with considerable accuracy but the detection accuracy is lower for trojans and backdoors.

Altaher et al. (2012) used API calls of DLL as major feature and Information Gain (IG) to reduce the total number of features to 11. They adopted *evolving clustering* methods to build the rule-based classifier and compared their results (99% Accuracy and 1% FPR) with Naive Bayes, Decision Tree, and Neural Network.

[2] https://virustotal.com/en/statistics/, (Last Accessed: 4 Nov 2016)

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

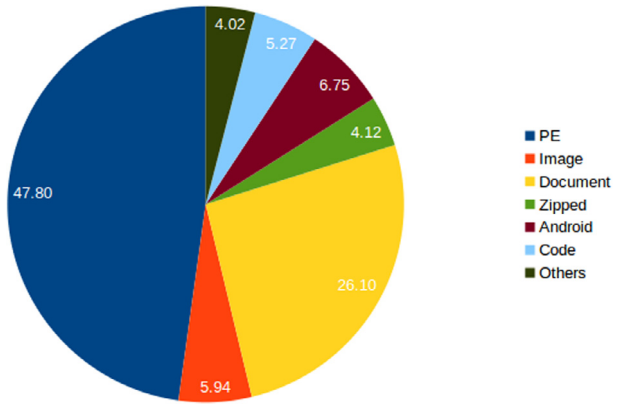

Fig. 1. Percentage of file type submitted to Virustotal (29 Oct- 4 Nov, 2016).

Liao (2012) selected only five fields of PE headers as the feature and implemented a hand-crafted rule-based algorithm for classification. Liaos algorithm achieved 99% accuracy and 0.2% FP for *unknown* malware.

Yan et al. (2013) used all fields of PE headers except *characteristic* and *image resource NameID* fields as numerical feature. Each bit of *characteristic*, DLL and *system call* information are represented as boolean features. Two separate feature-sets (numeric and boolean) were used to train Naive Bayes, kNN, SVM and Decision Tree classifiers. Four different feature selection methods i.e. Relief, Chi-Square, F-statistic and L1-regularized SVC have experimented with Decision tree and boolean feature-set.

Bai et al. (2014) built an initial feature set of 197 features, extracted from different PE header fields. They used filter and wrapper features selection method to reduce the features to 19 and 20 respectively. The selected features were used to train tree based classifiers such as Decision Tree (DT), Random forest (RF), Bagged and Boosted decision tree. The best result of these approaches was mentioned as 99.1% accuracy, 0.998 AUC, and 1.4% FP.

Baldangombo et al. (2013) used the three feature set *DLL calls*, *API calls* and PE *header fields* values as both separately and combined. Information Gain and PCA were used for feature selection and achieved a detection rate of 99.6%.

Belaoued and Mazouzi (2015) presented a real-time PE malware detection system, which is based on the analysis of the information stored in the PE-optional header fields. *Chi-square* and *Phi coefficient* were used for feature selection and with selected features *Rotation forest classifier* was trained and tested.

Ahmadi et al. (2016) proposed a novel paradigm to group malware variants into different families. In this work, feature extraction and selection method are given more importance than classification algorithms. Features are extracted from content and the structure of the sample, so the proposed method will even work on packed and obfuscated sample. They suggested that features can be grouped based on different characteristics of malware behavior, and their fusion is performed according to a per-class weighting paradigm. Aforementioned feature engineering method was experimented with Microsoft Malware Challenge dataset and achieved a very high accuracy rate of 99.77%.

Table 1 presents details of previous works, which used *only PE header fields* or information stored in Data Directories(DD) (DLL and API calls) or a combination of both features to classify malware and benign programs.

The available literature on malware detection using PE header information as features indicates the following observations:

1) Most of the works have only considered the raw value of PE header field as features and used them with other complementing features like DLL and API calls extracted from data directories, or features built using sections header. The proposed work uses a set of features such as entropy, packer info, count of suspicious sections, etc. which are calculated on the basis of header field value and specification provided in the guideline document (Pietrek, 1994).
2) The malware samples used in previous works are either unavailable due to various reasons or the datasets are private to the company. If the samples are available, the preprocessing task such as duplicate removal, file type identification, and labeling that consumes much time could have been avoided. The proposed work, therefore, has considered this issue seriously and provided a standard labeled dataset and code for doing further research on the topic.
3) Previous works have used PE raw features as the complementing feature and used them with other features which in turn improves performance and handle the risk of malware which are carefully created with a benign header but malicious payload in the body. In this proposed work, we have only conducted experiments with feature set built from header values. The purpose is to understand the discriminative limits of PE header value based feature set and study the improvement in performance of the proposed integrated feature set. Since the use of PE *header values* is only effective as the complementary feature set in anti-malware applications, the proposed work does not suggest an independent use of it.

Considering aforementioned observations, we created an integrated feature set by combining a set of derived features with raw features selected from the three main headers namely *DOS header*, *File header* and *Optional header* from the PE samples. When validated with the test dataset created with separate malware and benign samples, the integration of derived features with selected raw features shows improvement in the classification accuracy by 10%.

3. Methodology and feature set

The proposed work is carried out in four main steps: (1) raw samples collection, (2) pre-processing (includes labeling), (3) feature extraction and 4) training & testing. Fig. 2 illustrates all four steps of the work. In the proposed work, raw malware and benign samples are collected from various sources (explained in sub-Section 4.1) and are stored separately for further processing. Raw samples must be processed to make use of machine learning training. The tasks in pre-processing step are file type identification, duplicate removal, and labeling.

Labeling is an important task and it can be performed either locally by installing one or more signature-based anti-virus engine or using cloud-based service where multiple anti-virus engine based scan could be performed in parallel. We labeled each sample in both malware and benign group by cloud based service *Virustotal*.

Labeled samples are then passed to feature extraction step where PE header field's values are extracted to create raw and integrated feature set (explained in further Section 3.1 and 3.2). With raw and integrated feature set, different machine learning algorithms are trained and tested in last step (explained in Section 4 i.e. training and testing).

Feature generation is an important task of the proposed work, Algorithm 1 for that lays down the important steps performed to generate the raw and integrated feature set. All labeled malware and benign samples along with crafted header

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

Table 1
Feature set used in Previous works.

Works	Only Header	Only Data Directory(DD)	Header and DD	Algorithms	FPR	Accuracy
Ye et al. (2008)	No	Yes	No	OOA Rule based	NA	93.00%
Shafiq et al. (2009)	No	No	Yes	IBk,J48,NB,Ripper & SMO	0.50%	99.00%
Wang et al. (2009)	Yes	No	No	SVM	NA	98.86%
Walenstein et al. (2010)	No	No	Yes	NB,DT,SVM,RF,IB5	.014	98.90
Vinod et al. (2011)	No	No	Yes	SMO, IBk,J48,Adaboost1J48, RF	0.13%	96.80%
Altaher et al. (2012)	No	Yes	No	NB, DT, NN and Evolving Clustering method	1.00%	99.00%
Liao (2012)	Yes	No	No	If-else-rules	0.20%	99.00%
Baldangombo et al. (2013)	No	No	Yes	J48, SVM, NB	NA	99.60%
Bai et al. (2014)	No	No	Yes	DT, NB, BoostedTree and BaggedDT	1.40%	99.01%
Ahmadi et al. (2016)	No	No	Yes	XGBoost	NA	99.77%

Objective-Oriented Association (OOA); Instance Based Learner (IBk); Decision Tree (DT) (J48 is a DT implementation present in WEKA); Naive Bayes (NB); Sequential Minimal Optimization (SMO); Support Vector Machine (SVM); Random Forest (RF); Neural Network (NN).

rules are input to the algorithm. Two loops, one for benign and other for malware samples run to bring sample one by one for processing. By calling *FetchFieldsValue()* procedure, all values are extracted from different fields of PE sample file. Raw feature set (i.e. RawF) is updated with all extracted values and appending class label. For integrated feature set (i.e IntF), *CheckRules()* procedure is called by passing extracted values (i.e. *RawValue*) and *rules* and it returned derived values which is used to update integrated feature set along with *FileValue* and class label. This *CheckRules()* procedure takes all raw values as input but rules are checked against only selected fields (explained in Section 3.2.2) and other field values are returned without any change. *ExtractFileFeatures()* procedure takes a sample as input and return other derived values such as file size, file entropy related to file properties (explained in Section 3.2.2).

Algorithm 1. Feature Set Generation

procedure GENERATEFEATURESET Ψ, Ω, Φ
  $\alpha \leftarrow count(\Psi)$ ▷Ψ: malware set
  $\beta \leftarrow count(\Omega)$ ▷Ω: benign set
  $RawF[\alpha+\beta][] \leftarrow 0$
  $IntF[\alpha+\beta][] \leftarrow 0$
for $\kappa \in \Psi$ **do** ▷Extract features from malware
    $Class \leftarrow 0$
    $RawValue \leftarrow FetchFieldsValue(\kappa)$
    $FileValue \leftarrow ExtractFileFeatures(\kappa)$
for $Value \in RawValue$ **do**
    $DerivedValue \leftarrow CheckRules(Value, \Phi)$
    ▷Φ: Rules set
    $RawF \leftarrow UpdateRawF(RawValue \cup Class)$
    $IntF \leftarrow UpdateIntF(FileValue \cup DerivedValue \cup Class)$
for $\kappa \in \Omega$ **do** ▷Extract features from benign
    $Class \leftarrow 1$
    $RawValue \leftarrow FetchFieldsValue(\kappa)$
    $FileValue \leftarrow ExtractFileFeatures(\kappa)$
for $Value \in RawValue$ **do**
    $DerivedValue \leftarrow CheckRules(Value, \Phi)$
    $RawF \leftarrow UpdateRawF(RawValue \cup Class)$
    $IntF \leftarrow UpdateIntF(FileValue \cup DerivedValue \cup Class)$
return$(RawF, IntF)$

3.1. Raw feature set

Raw feature set is created by extracting values from all fields of three main headers (*DOS header*, *File Header* and *Optional header, including standard and Windows-specific fields*) present in every PE file. In total, raw feature set has 55 features in which 19 features are from *DOS header*, 7 from *File Header* and rest 29 are taken from *Optional header*. During training *e_res* and *e_res2* fields are removed from raw feature set as they are reserved (according to the document) and have no values for the samples (including malware and benign samples). So, the final raw feature set has 53 features used for training and testing. PE file processing and field's value extraction are performed on Linux machine with the help of *pefile* python module. Details about in Section 4.1 and 4.2. Eq. 1 shows the features vector for raw feature set where *RawF* represents raw feature set and *DH*, *FH* and *OH* represent the header's fields of *DOS header*, *File header* and *Optional header* respectively.

$$RawF = \begin{bmatrix} DH = \{DH_1, \cdots DH_{19}\} \cup \\ FH = \{FH_1, \cdots FH_7\} \cup \\ OH = \{OH_1, \cdots OH_{29}\} \cup \end{bmatrix} \quad (1)$$

3.2. Integrated feature set

The integrated feature set is created by combining a few selected raw features (explained in Section 3.2.1) and a set of derived features (explained in Section 3.2.2). The proposed method is designed to utilize the combinatorial benefits of both the raw and derived features as integrated features. Integrated feature set has a total of 68 features in which 28 are same as in raw feature set, 26 boolean features are created by expanding individual flags of *Characteristics* and *DLLCharacteristics* from the *File header* and *Optional header Windows specific* and 14 are derived features (explained in further Section 3.2.2). The rationale behind the choice of using binary features for representing *DLLCharacteristic* and *Characteristics* is listed below:

- Using all bits of *DLLCharacteristic* and *Characteristics* as separate binary features will provide more information about PE file than a numeric value.
- The processing of boolean feature will be easier.
- We can see that only 3 binary features from *DLLCharacteristic* and *Characteristics* are in top 10 features and hence other fields can be neglected.

Table 2 summarized the count for raw features, expanded raw features and derived features present in integrated feature set.

Characteristics is one of the important fields under **File Header** and it comprises flags that indicate the attributes of the file. Different flag values of Characteristics fields store different file characteristic such as IMAGE_FILE_DEBUG_STRIPPED flag that tells whether separate debug file is used or not. Most of the previous works have used the integer value of this 16 bits field whereas in the proposed work all 15 flags (as one bit is reserved for future

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

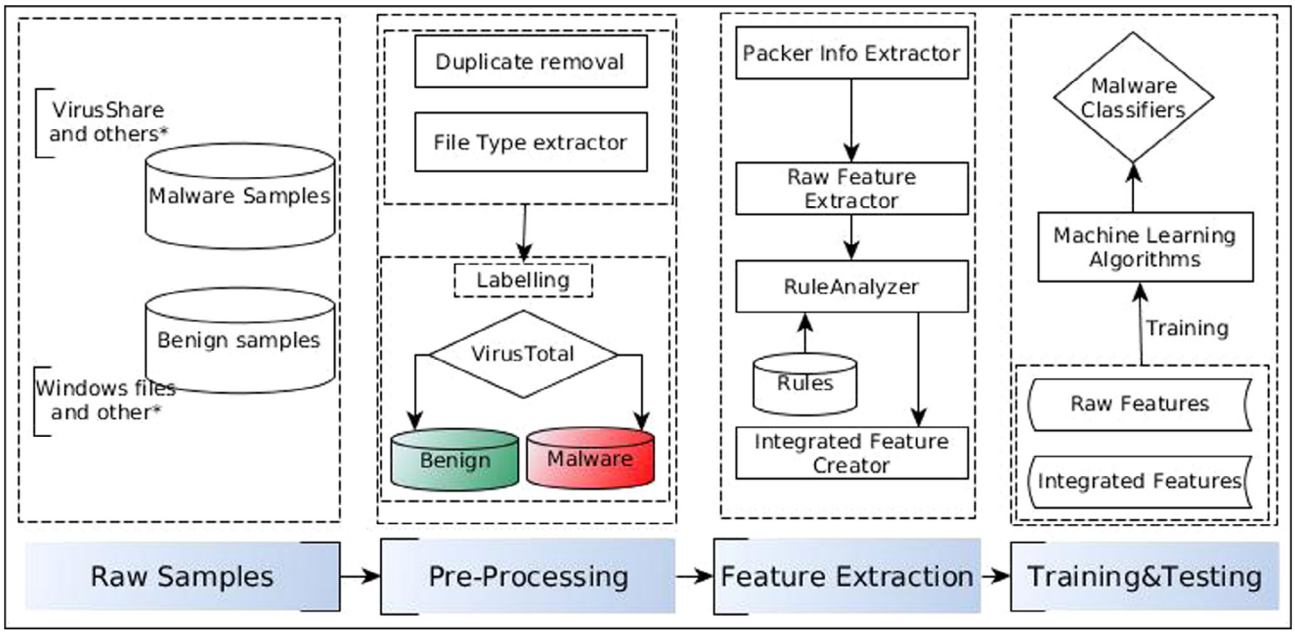

Fig. 2. Process diagram of proposed system.

Table 2
Integrated feature set.

Raw	Expanded	Derived	Total
28	26	14	68

use) used as boolean features for characteristic field and encoded as *FH_char0* to *FH_char14*.

DLLCharacteristics is an important field under *Optional header Windows-specific fields* and contains information about DLL behaviors which are used by linker and loader. Various information is encoded as flags which are represented by on/off bit in 16 bits. For example, value *0x0040* (hexadecimal 0x0040 or decimal 64) in this field indicates that DLL can be relocated at load time. Earlier works have used decimal value of this field as feature whereas, in the proposed work, all in-use bits are used as the boolean feature. So, *DLLCharacteristics* field is coded as *OH_DLLchar0* to *OH_DLLchar10* in proposed integrated feature set.

Eq. 2 shows the feature vector for integrated feature set where *IntF* represents Integrated feature set and *Raw* has selected raw features from *DOS header*(DH_1 *to* DH_6), *File header* (FH_1) and *Optional header* (OH_1 *to* OH_{21}) respectively. *ExpandedRaw* represents boolean features created by expanding flags present in *Characteristics* (C_1 *to* C_{11}) and *DLLCharacteristics* (DC_1 *to* DC_{15}). *Derived* represents the proposed derived features.

$$IntF = \begin{bmatrix} Raw = \{\{DH_1 \cdots DH_6\} \cup \{FH_1\} \cup \{OH_1 \cdots OH_{21}\}\} \cup \\ ExpandedRaw = \{\{C_1 \cdots C_{11}\} \cup \{DC_1 \cdots DC_{15}\}\} \cup \\ Derived = \{D_1 \cdots D_{14}\} \end{bmatrix} \quad (2)$$

The proposed integrated feature set is created with the assumption that integrating raw and derived features will improve the detection accuracy. We conducted rigorous experiments to validate our assumption and find out that experimental results justify the assumption. The details of used raw and derived features for creating proposed integrated feature set are explained further in Sections 3.2.1 and 3.2.2. Experimental setup, result, and analysis are explained in Section 4.

3.2.1. Raw features

Raw features are those features for which values are directly extracted from PE header field for use. In the proposed work, a few of header's field value is used directly, and the selection of these header's fields are made on the basis of comparing their statistical properties (mean and standard deviation) between malware and benign samples. As *DOS header*, NT and versions after *Windows 3.1* do not use DOS (Disk Operating System) header of PE files, most of the fields are not useful. Only e_lfanew is important which has offset of the first byte of new PE header but the proposed work has used a total of 6 fields from DOS header out of total 19 fields, all six fields name are listed in A. *File header* is provided after DOS header, which has abstract information about the whole file. *NumberOfSections* and *Characteristics* are most important than other fields. To know about *total sections* and *type of files* the linker uses these two fields' values. In the proposed work, only *NumberOfSections* is used as raw features while *Characteristics* field's value is used after converting into boolean according to the above explained method. The *Optional header* has two sets of fields: standard fields and Windows specific fields. Eight standard fields are present in **PE32** and nine are in **PE32+**, these fields are similar to COFF header of Linux system. Windows specific fields are totally 21 in number and have information that is required by the linker and loader in window OS. Except for Magic, all other standard fields are used as raw features in the proposed work, 13 windows specific fields are used as raw features while *DLLCharacteristics* field's value used after converted into boolean as explained above. For a complete list of raw features used in the proposed work please refer Appendix A.

3.2.2. Derived features

Derived features are features that are derived from the raw value of PE header by validating with a set of rules. The result of this process is taken as the feature value. Choice for set of derived features are made upon the guidelines given for PE headers' field.[3] Those fields which need to have a value from a predefined set of values as per the guidelines are chosen as derived features. Along with

[3] https://msdn.microsoft.com/en-us/library/windows/desktop/ms680198(v=vs.85).aspx

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

this, the fields whose values depend on another field are also added to the derived feature set. During preprocessing, extracted values for these fields are checked against the rules. The binary output of this comparison is taken as the final value. As the values for these fields are governed by aforementioned guidelines, tampering of their values signals potential threat. There is a high possibility that this tampering of values is caused by malware.

Derived features are not values directly extracted from the header's field; instead, these are values that are amalgamated as a result of comparing raw values against logical and documented rules. They can either be Boolean or Integer. In the proposed work, the structural and logical specifications of PE file are adopted from a widely used document explaining structure of PE file (Pietrek, 1994) and online Microsoft's MSDN document.[4]

For example, if we take *TimeDateStamp* field, the raw value will be simply an integer indicating the number of seconds since 1969 or if an encoding is used by extracting package then it will be in date format. We can see in Fig. 3 that *ValidFile.exe* has a valid date whereas *InvalidFile.exe* does not have a valid date, now using the *TimeDateStamp* raw value we will not have powerful feature so the proposed work does not use these values directly as feature. In the proposed work, *TimeDateStamp* field's value is converted to date and is compared with a range of valid date (From December 31st, 1969, at 4:00 P.M. to the date of the experiment) and the resulted Boolean output is taken as features. Table 3 summarizes all the considered derived values with its counterpart as raw values. It shall be observed that except for *entropy* all other derived features have binary values. The reason for values of *entropy* is explained in the following section.

Entropy can be defined as *measure of efficiency of information storage* (Shannon, 1948). It is directly related to the packing of the file, and packed file will have high entropy resulting in high efficiency of information storage (Yonts, 2012). The *pefile* module has one method named *get_entropy()* that calculates entropy of a given section data. The result of this method is in the form of a quantity of bits per byte and so the maximum entropy will be 8.0. With the inherent properties of measuring the degree of compression, entropy is a very reliable feature to check packer and malicious behavior. Some modern malware or malicious packers try to reduce entropy by inserting *zero bytes* in data. It avoids detection, as many AV software only react to high entropy files. In the proposed work, the entropy of different section of PE and whole file are also calculated and treated as features. As malware have many different names for different sections, only standard section names are considered as the feature and if section's name is present then their respective entropy are added to the feature set, else '−1' is assigned as entropy value. In total, three features (E_text, E_data, E_file) are used based on entropy values. (Markel, 2015; Markel and Bilzor, 2014) have used section's entropy with threshold values 7 and 1 to set value for two boolean features *HighEntropy* and *LowEntropy* respectively. If any section of the corresponding PE file has entropy greater than 7 then *HighEntropy* is set to 1. When the entropy is less than 1, *LowEntropy* is set to 1.

The difference between (Markel, 2015; Markel and Bilzor, 2014) and our study is in the way of using section entropy. In earlier study entropy value is not attached with particular section and file entropy is not used while in the proposed work, we have used three features based on entropy. Out of three, two entropy values are attached to two most frequent sections (.text,.data) of PE file and one uses file entropy value.

File Creation Year or Compilation time is very useful in identification of the malware from the benign program. Malware usually has very suspicious year of creation such as years earlier than 1980 or year beyond the current year, whereas the benign programs have very genuine year of creation. 1980 is considered as a starting year because the first DOS operating system was launched[5] in this year. File creation year can be calculated using *TimeDateStamp* field value of File header for a given PE file. In the proposed work, the calculated file creation year is checked within the valid range of year i.e. 1980–2015, if the year is within the range, then *1* is assigned as feature value else *0* is assigned indicating that the sample has a suspicious year of creation.

Suspicious Section Name is also a very good indicator of malicious file. A standard compiler gives well-defined name to each section like *.data*, *.text*, *.rdata*, etc. It has been observed that, the PE files built using certain non-standard tools tend to have *random* section names. These types of sections are termed as *suspicious*. Section names of each sample is extracted using sections field value of PE header and these sections name are compared with a set of standard section name (Pietrek, 1994), count of match and non-matching section are assigned to two different feature variables *TotalSuspiciousSections* and *TotalNonSuspiciousSections*. Perdisci et al. (2008) have also used number of suspicious sections name as features for detecting packed and non-packed file.

Packer Info would be a potential indicator of malware and benign program because, in our analysis, it is found that nearly 18% of malware sample are packed whereas it is only 13% for benign samples. Both benign and malware can be packed but for different purposes. Benign programs are being packed to protect copyright and license key breaking attempt and also to stop reverse engineering of programs. Attackers pack their programs to bypass the signature-based detection because packed file changes the byte structure of program and it becomes easy to surpass the signature. In the proposed experiment, *packer* is a Boolean feature and will have *0* if file is not packed and *1* if packed with any packer. Packer information of each sample is collected using *PEiD*[6] signature database with *yara*[7] (a Python module for signature matching).

```
ValidFile.exe -> TimeDateStamp : 0x3B7DFE0E [998112782 Seconds]
GMT : Sat, 18 Aug 2001 05:33:02 GMT (file has a valid compilation time)

InvalidFile.exe -> TimeDateStamp : 0x851C3163 [2233217379 Seconds]
GMT : Sun, 07 Oct 2040 10:09:39 GMT (file has a invalid compilation time)
```

Fig. 3. Example of Valid and Invalid TimeDateStamp value.

Table 3
Raw and Derived features.

Features	Raw Value	Derived Values	
		Type	Values
Entropy	Binary value	Integer	[−1, 0–8]
Compilation Time	Integer	Boolean	[0,1]
Section Name	String	Integer	–
Packer Info	NA	Boolean	[0, 1]
FileSize	Integer	Integer	–
FileInfo	String	Boolean	[0, 1]
ImageBase	Integer	Boolean	[0, 1]
SectionAlignment	Integer	Boolean	[0, 1]
FileAlignment	Integer	Boolean	[0, 1]
SizeOfImage	Integer	Boolean	[0, 1]
SizeOfImage	Integer	Boolean	[0, 1]

[4] https://msdn.microsoft.com/en-us/library/windows/desktop/ms680198(v=vs.85).aspx.

[5] http://windows.microsoft.com/en-IN/windows/history#T1=era0 [Accessed May 27, 2016].

[6] http://www.aldeid.com/wiki/PEiD [Accessed May 27, 2016].

[7] http://plusvic.github.io/yara/ [Accessed May 27, 2016]

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

File size is also used as a feature. We had initial assumption that malware file and benign programs file size will have a large difference. Malware writers try to keep file size minimum which helps them to distribute it over web hiding within another program. Malware samples result smaller in size due to avoidance of Graphical User Interface (GUI). Benign programs are free from such size requirement and have genuine size as needed. All libraries and external resources are used as needed to run the program efficiently and effectively. Malware writers truncate the unused function and other external resources from their malicious program, leaving selective API functions and DLL within programs. In previous works, file size has not been considered as a feature but in the proposed work, we have taken this as a feature of type integer that will have a file size in bytes.

FileInfo has set of strings, which contains the metadata about the PE file such as *FileVersion*, *ProductVersion*, *ProductName*, *CompanyName*, etc. Benign programs incorporate a rich metadata while malware writers avoid putting metadata. In the proposed work, metadata information of all samples are tried to extract but it is found that many of malware samples do not have *FileInfo* field or other sub-section of this field. Set of all extracted strings about metadata are large so it is not feasible to consider these strings as features instead of that *FileInfo* feature is considered as Boolean and *1* is assigned if a sample has FileInfo metadata else *0*. A more comprehensive feature set of metadata information can be considered but the proposed work is limited to the presence and absence of the *FileInfo* metadata apart from its value.

ImageBase is a field in the optional header, which has the preferred address of the first byte of the image when loaded into memory (Pietrek, 1994). The condition applied to this field is that it *must be a multiple of 64 k (641024)*. Default value for DLL is *0x10000000* (268435456 in decimal), Window CE exe is *0x00010000* (65536 in decimal) and *0x00400000* (4194304 in decimal) for other Windows OS like Windows XP. To make this field more useful, a pilot study is carried on the samples and it is found that 94.55% malware samples have specified default value while only 77.84% of benign samples have specified a default value and all the value from both classes follow the condition of multiple of 64 K. Value of ImageBase field is very specific to type of file (DLL, exe etc.) So, malware authors do not tamper this field often which is clear from the aforementioned percentage. With the aforementioned result, it is decided to consider this field as Boolean and values were checked against for default values and multiple of 64.

SectionAlignment is another field in the optional header and it is the alignment (in bytes) of sections that determine when they are loaded into memory. According to the Microsoft specification (Pietrek, 1994), it must be greater than or equal to *FileAlignment* and the default is the page size of the architecture. In the proposed work, this field is treated as Boolean and is compared against the specifications. If extracted value follows the specification then this field will have a value *1* else will be assigned as value **0**.

The value of **FileAlignment** field of optional header is the factor (in bytes) that is used to align the raw data of sections in the image file. Microsoft specification (Pietrek, 1994) states that this value should be power of 2 (between 512 and 65536). Default value is 512 and if the *SectionAlignment* is less than the architecture's page size, then this value must match with *SectionAlignment* value. The proposed work follows the given specification and considered this as Boolean feature, with value *1* if the extracted value match the specification else *0*.

SizeOfImage gives the size (in bytes) of the image, including all headers, as the image is loaded in memory (Pietrek, 1994). It must be a multiple of *SectionAlignment*. This field is important to hide the embedded code. To verify this assumption, the specified condition is checked for malware and benign samples. It is found that almost all benign samples follow the specification but approx 4% (3.87%) malware samples do not follow the specification. To benefit the classification process, since it got a significant proportion, this field is also considered as Boolean and assigned value *1* if it matches the specification otherwise **0**.

SizeOfHeaders has value that is equal to the combined size of *MS-DOS stub*, *PE header* and *Section headers*, and is rounded to a multiple of *FileAlignment*. This specification is also validated for malware and benign samples. Among benign samples, 16% did not follow the specification while 78% of malware samples did not follow the specification. In the proposed work, this field is considered as Boolean and assigned *1* if extracted value follow the specification else *0* is assigned.

The rationale for converting numeric values into boolean is to incorporate semantics in feature representation. For example, the feature *CompilationTime* holds *TimeDateStamp* which belongs to 32-bit Integer type. In the proposed work, instead of representing a numeric value, the semantic of earliest file creation date is used as a delimiter to convert the value into boolean representation. Though this looks like losing some content in the original field, it provides an advantage of proper identification with respect to legitimate and tampered value. Moreover, binary representation would lead to simpler processing and build a robust classifier. The core idea of this approach is to harness the value of the field with respect to the context of malware identification. Similar logic is adopted for all other derived features. The empirical validation of this approach is provided with the help of the analysis of experimental results illustrated in Section 4.3 (Result Analysis).

David et al. (2016) have also presented and discussed the structured analysis of many of aforementioned fields and stated that considering these as features can enhance the malware detection rate. Among aforementioned derived features, six features i.e. CompilationTime, FileInfo, SectionAlignment, FileAlignment, SizeOfImage, and SizeOfHeader can be modified by an attacker without affecting of the execution of the program. This inference is derived on the basis of comparative analysis between benign and malware samples present in the dataset.

4. Experimental setup, results and analysis

4.1. Dataset preparation

To evaluate the proposed work and create raw and integrated feature set, malware and benign samples are collected. Malware samples are collected from *virusshare*[8] and benign samples are taken from freshly installed Windows XP and Windows 7 program files. Some of the benign samples are also collected from online free software archive.[9] Apart from this dataset, a separate test dataset (explained in detail in Section 4.3.3) is also created for validation purpose which has 129 malware samples of different type and 30 benign samples. Collected malware and benign samples can have duplicate sample (same sample with a different name) and that will affect the classification of trained model. To identify and remove such duplicate samples using filename alone is not correct as it can lead to errors, so we used hashing technique (MD5) to create file hash. Using hashes, retain the unique sample in both malware and benign group.

In the proposed work, only PE files (exe, DLL etc.) are considered and so it is necessary to select PE files alone. There are many methods available for file type detection, among them we have used Linux's *file* utility to get the file type of each sample and with the Python script we processed output retaining the PE files.

[8] http://virusshare.com/ [Accessed May 27, 2016]
[9] http://download.cnet.com/ [Accessed May 27, 2016].

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

To have an accurate and robust feature set for training and testing, it is necessary to verify the class label of each sample. Among the local installed Anti-virus engines and online multiple engines scanning, we selected online malware scanning service, *Virusotal* for labeling. *Virusotal* offers API-based service which helped to automate the scanning process, we used this service to verify the class label of each sample present in the dataset. *Virusotal* provides scan result of multiple anti-virus engines running in parallel. The selection of AV engines are made based on AV-test[10] ranking. As the leading AV engines (Top n) are given priority instead of treating all engines equal, this approach enhances the probability of more accurate labeling. The decision on the class label is made using Eq. 3.

$$CL(M|B,S) = \begin{cases} M, & If(E_1(S) \vee E_2(S) \vee \ldots E_n(S)), \\ B, & elseIf(!(E_1(S)) \wedge !(E_2(S)) \wedge \ldots !(E_n(S))) \\ & \{\upsilon(E_1) < \upsilon(E_2) < \ldots \upsilon(E_n)\} \end{cases} \quad (3)$$

From Eq. 3 it is clear that a malware label is assigned to the sample S, if any of the scanning engine among E_1 to E_n returns positive whereas a benign label is only assigned to the sample S if all of the scanning engines return negative. In Eq. 3, the AV-test rank of engine E_i is represented as $\upsilon(E_i)$.

After performing the aforementioned pre-processing steps, the final dataset had 2488 benign and 2722 malware samples. The size of the dataset is in alignment with the common standards adopted in machine learning based malware detection research studies (Schultz et al., 2001; Belaoued and Mazouzi, 2015). However, a larger dataset size shall strengthen the classifier and the performance metrics shall improve than the values reported currently in Section 4.3 (Result Analysis). All these samples are passed on to a feature extractor where the different header's field values are extracted from the each data sample using *pefile*[11] python module. It is very efficient and well accepted Python module for PE file processing.

The raw feature set is created by just appending class label with each set of values extracted from all samples.

For creating integrated feature set, we have selected raw values and set of derived features. Raw values are retained from the previous step and for derived features, a python script is written which has methods to check selected field's value with predefined rules (based on MSDN document) and return the result as feature's value. As explained in Algorithm 1, all extracted field values are passed and only selected fields are checked for derived features. Appending selected raw with returned derived values and class label we created the integrated feature set. Some sample headers fields' values are not readable which is the result of obfuscation and hence such samples are neglected by the extractor.

4.2. Experimental system

To validate the proposed idea, an experimental environment is created with Ubuntu Operating system running on Intel(R) Core (TM) 2 Duo CPU E7400@2.80 GHz processor and 4 GB of primary memory and 320 GB of secondary memory. Ubuntu system helps to stop infection of the experimental system as samples are targeted for Windows based OS. Due to static analysis, computation cost is very low and hence all experiments are carried on the normal end-host system. *Scikit-learn* (Pedregosa et al., 2011), a Python-based machine learning tool is used to run all the experiments. It is used to build and test the performance of different classifiers with raw, integrated and test dataset. Train-test split, 10 fold cross-validation and supply of test dataset testing methods of *Scikit-learn* are used to perform various training and testing based experiments. It helps in comparing the performance of various models on many metrics such as accuracy, recall, precision and f1-measure.

Scikit-learn has the implementation of many of well recognized machine learning algorithms such as Decision Tree (DT), Logistic Regression (LR), Random forest, etc. We selected one algorithm from each category which are grouped on the basis of underlying theoretical approaches such as tree-based, Bayesian or probability-based, instance-based, dimensionality reduction-based and ensemble-based algorithms. Six algorithms Logistic Regression (LR), Linear Discriminant Analysis (LDA), Random Forest (RF), k-Nearest Neighbours (kNN), Decision Tree (DT) and Gaussian Naive Bayes (NB) are selected finally and used throughout all experiments. All the aforementioned algorithms are trained and tested with various configurations and different datasets (result of these are present in further Sections 4.3.3, 4.3.2, 4.3.3 and 4.3.5).

4.3. Result analysis

Various experiments are conducted by training classifiers with Raw Feature (RF) set (that have only PE headers field value as feature) and Integrated Feature (IF) set (which has a few selected PE fields' value and derived features).

Accuracy, Precision, Recall, F-measure and Receiver Operating Characteristic (ROC) curve are used as performance metrics. These performance metrics with other primitive metrics are explained with formulae in Table 4.

In this subsection result of each experiments with analysis is presented.

4.3.1. Train-test split

The aim of this experiment is to study the performance of the proposed integrated feature set by splitting the dataset into two parts with 70 : 30 ratio for training and testing respectively. Train-test split is a method of splitting the dataset into two parts, training, and testing. It takes the percentage as splitting threshold. Training dataset is used for training the machine learning algorithm while testing dataset tests the classifier's performance. We use 70 : 30 ratio for splitting our raw and integrated feature and hence 70% of the dataset used to trained classification algorithms and 30% is used for testing.

The result of various classifiers' performance on different metrics with train-test split is summarized in Table 5. It can be observed that NB (accuracy 50 – 60%) performance is minimum while performance of Random Forest (accuracy **97.47%** and **98.78%** on raw and integrated feature set respectively) is best

Table 4
Different metrics for measuring performance and comparison.

Metric	Formulae	Description
True Positive (TP)	count()	Total malware which predicted as malware
False Positive (FP)	count()	Total benign which predicted as malware
True Negative (TN)	count()	Total benign which predicted as benign
False Negative (FN)	count()	Total malware which predicted as benign
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Rate of correct predictions
Precision	$\frac{FP}{TN+FP}$	Proportion of predicted malware which are actual malware
Recall/TPR	$\frac{TP}{TP+FP}$	Proportion of actual malware which are predicted malware
F-measure	$2 \times \frac{precision \times recall}{precision + recall}$	Harmonic mean between precision and recall
ROC area/AUC	2D curve of TPR/FPR	Area Under the ROC

[10] https://www.av-test.org/en/antivirus/home-windows/.
[11] https://github.com/erocarrera/pefile [Accessed May 27, 2015].

Table 5
Classifiers result on train-test (70–30) split.

Classifier	Accuracy		Precision		Recall		F1-score	
	RawF	IntF	RawF	IntF	RawF	IntF	RawF	IntF
LR	77.06	78.12	0.81	0.80	0.77	0.78	0.76	0.77
LDA	91.71	92.45	0.92	0.93	0.92	0.92	0.92	0.92
RF	97.43	98.78	0.97	0.99	0.97	0.99	0.97	0.99
DT	96.47	97.12	0.96	0.97	0.96	0.97	0.96	0.97
NB	56.04	50.09	0.74	0.77	0.56	0.58	0.48	0.51
kNN	94.73	90.79	0.95	0.91	0.95	0.91	0.95	0.91

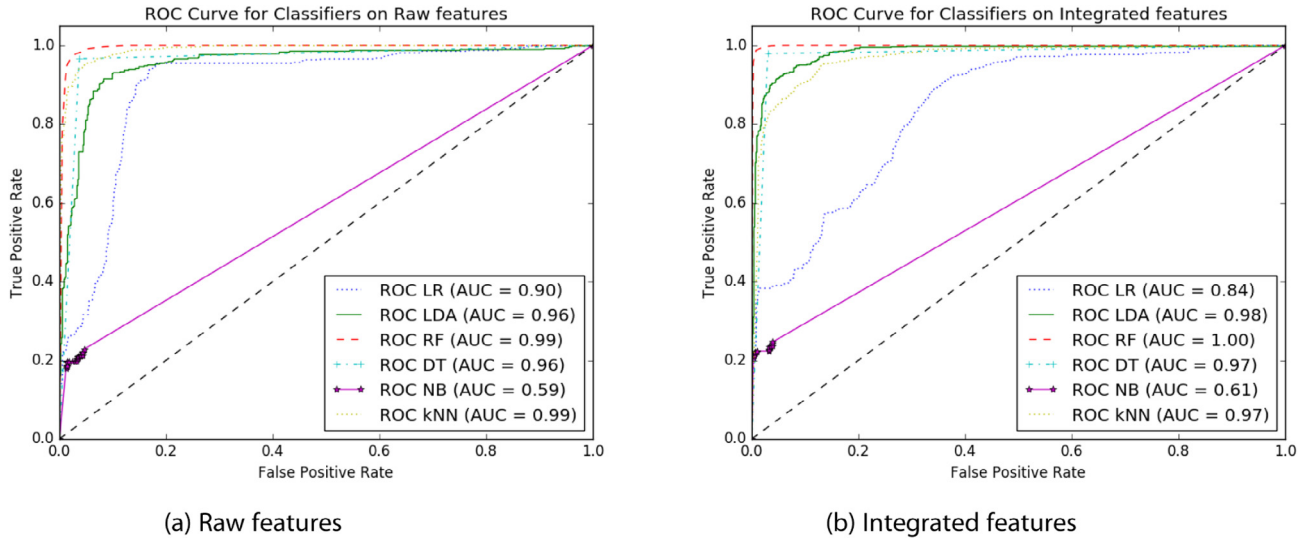

Fig. 4. ROC curves for different classifiers with train-test split method.

among all classifiers on both datasets, and the integrated feature set is performing better ($accuracy + 1.31\%$) than the raw feature set. Other classifiers also show better performance on all metrics except kNN which gives better result on raw features than the integrated feature. Fig. 4 shows the ROC curve for classifiers trained and tested on raw and integrated feature set. ROC and Area Under Curve (AUC) values also indicate similar performance as other metrics, proving that the integrated feature set have better AUC value than the raw feature set. Random forest performance is best among all other classifiers.

4.3.2. 10-fold cross validation

The aim of this experiment is to empirically validate the performance generalization of proposed integrated feature set independent from the dataset utilized for the current experimental setup. Train-and-test method of validation has two main limitations, first, in a single train-and-test experiment, the holdout estimate of error rate will be misleading if we get a split which does not represent each class sample proportionally. Second, it reduces the training samples which would degrade the performance of the classifier. To overcome these limitations, a method more robust for validation, cross-validation method is used. Cross-validation is represented as $k/n - fold$ where k or n represent the number of folds the dataset will be split and the number of times the training and testing will be done. For a *k-fold cross-validation*, the dataset will be split in k folds and for k times the $k-1$ fold dataset will be used for training and the one fold left out will be used for testing. The final result is given as the average of all k folds.

We have performed a 10-fold cross-validation of all classifiers on raw and integrated features. For comparison, we have measured the accuracy of each classifier for both raw and integrated features. Table 6 shows the 10-fold cross-validation output of all the classifiers.

Table 6
Comparison of classifiers' accuracy with 10-fold cross-validation.

Classifier	Accuracy		Difference
	Integrated	Raw	
LR	0.600	0.742	−0.142
LDA	0.923	0.874	+0.049
RF	0.984	0.977	+0.007
DT	0.974	0.960	+0.014
NB	0.583	0.578	+0.005
kNN	0.899	0.928	−0.029

It can be observed that accuracy of all classifiers is decreased by nearly 1% than the accuracy achieved under test-split validation and the difference column clearly shows that integrated feature set is performing better than raw features. Four classifiers (LDA, RF, DT, and GaussianNB) have higher accuracy rate while other two classifiers (LR and kNN) have a very close accuracy of the raw feature set. Fig. 5 shows the box plot output of classifier's accuracy collected during 10-fold cross validation. It is observed that Random forest and Decision Tree have high accuracy and low variation where others have more variation with low accuracy.

Among the classifiers, with the same configuration, Random forest consumed the highest training time on both raw and integrated features set, **9.51** and **8.25** seconds respectively, and Decision Tree has taken lowest training time for both raw and integrated features set, **0.83** and **0.6** seconds respectively. It is also observed that training time is less for integrated features set, for example, both RF and DT have taken less time than raw feature set.

4.3.3. Testing with new test dataset

The aim of this experiment is to test the performance of the proposed integrated feature with new samples which represent the

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

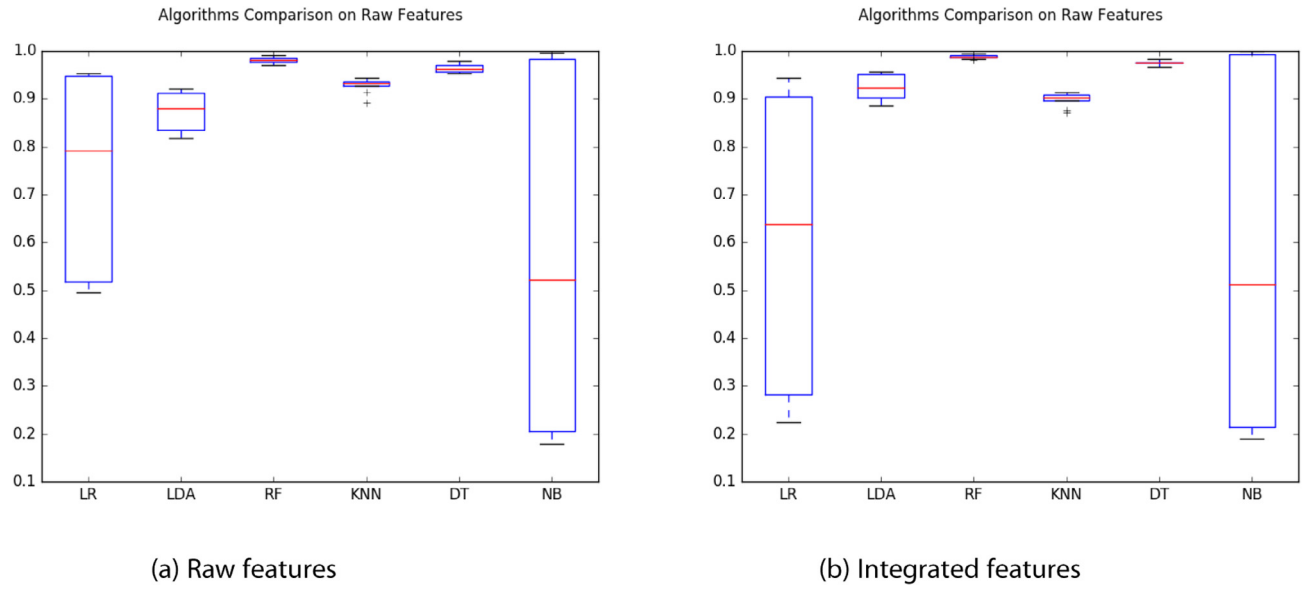

Fig. 5. Box plot for different classifiers' accuracy with 10-fold cross validation.

Table 7
Type of malware samples and count in test dataset.

Malware Type	Virus	Worm	Trojan	Bot	Spyware	Downloader	Backdoor	Total
Initial Count	20	20	20	9	20	20	20	129
After extraction	5	18	19	7	20	13	20	102

Table 8
Classifiers performance on test dataset.

Classifier	Accuracy		Precision		Recall		F1-score	
	RawF	IntF	RawF	IntF	RawF	IntF	RawF	IntF
LR	73.48	70.00	0.77	0.77	0.73	0.70	0.75	0.72
LDA	81.82	88.46	0.85	0.90	0.82	0.88	0.83	0.89
RF	74.24	89.23	0.87	0.93	0.74	0.89	0.76	0.90
DT	74.24	83.85	0.86	0.90	0.74	0.84	0.76	0.85
NB	28.79	31.54	0.83	0.84	0.29	0.32	0.21	0.26
kNN	79.55	76.15	0.88	0.83	0.80	0.76	0.81	0.78

real world scenario where classifiers get *unknown* and *zero-day* samples. Test dataset is created with samples which are not included in training dataset and is used to validate or test the performance of trained classifier. After testing and comparing the performance of classifiers on the raw and integrated feature set with train-test and cross-validation method, we have also performed experiments to validate the efficiency of the integrated feature set with new test dataset. We have collected unique 129 malware samples (which are not present in earlier collected samples) of different types and mixed with 30 benign samples randomly selected from earlier collected samples.

Table 7 shows the count of each malware type that is included in the test dataset. All the samples are downloaded from *openmalware* public malware repository[12] and the same label given by the aforementioned archive is adopted. The MD5 hash comparison is done to ensure that none of the training dataset sample is present in the new test dataset. After pre-processing and feature extraction step, the final test dataset has 102 malware and 30 benign samples in total.

[12] http://oc.gtisc.gatech.edu:8080 [Accessed May 27, 2016].

Aforementioned selected learning algorithms are trained with the feature set having 5180 samples from both malware and benign class, and their performance is validated with new test dataset that has 132 samples. The result of various metrics on integrated and raw features set with test dataset is summarized in Table 8.

It can be observed from Table 8 that raw feature set based classifiers performed poorly on the new test dataset. The raw feature set has an accuracy rate ranging between 73% – 81% for classifiers reported low to high. The performance of integrated feature-based classifier also reduces and the best classifier Random forest has only 89.23% accuracy which is 9.17% less than the 10 folds cross-validation result. With comparison to raw, integrated feature set performance is better. Random forest and Decision tree had only 74.24% accuracy on raw features, while 89.23% and 83.85% are the accuracy rate achieved with Random forest and Decision tree respectively on integrated features. It shall also be inferred from the Table 8 that classifiers such as LR & kNN have performed better with the *RawF*. However, the metric values fall in 70 % range which is considered below average as per the current trend of machine learning based classifiers. At the same time, the classifiers such as LDA, RF, and DT which performed better with *IntF* have

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

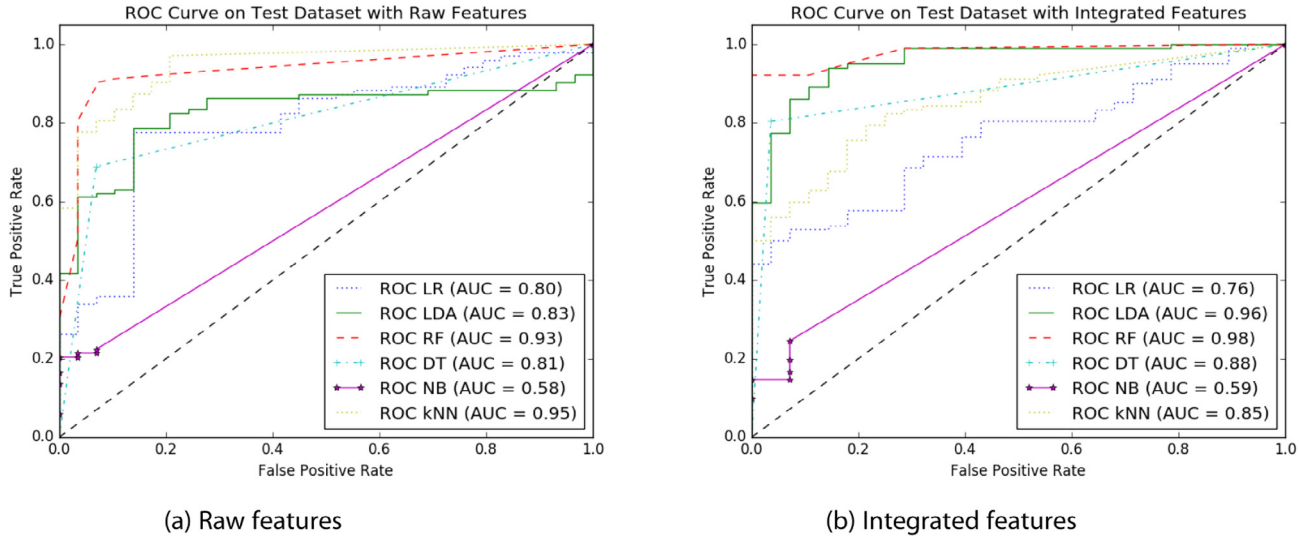

Fig. 6. ROC curves of classifiers on Test dataset.

Table 9
Partial AUC for FPR range [0, 0.1] and [0, 0.2].

Classifier	FPR = 0.1		FPR = 0.2	
	RawF	IntF	RawF	IntF
LR	0.03	0.04	0.12	0.09
LDA	0.07	0.09	0.17	0.18
RF	0.10	0.10	0.19	0.20
DT	0.06	0.07	0.16	0.17
NB	0.02	0.02	0.04	0.05
kNN	0.09	0.07	0.19	0.17

Table 10
Comparing the proposed work with earlier work.

Works	DT	RF
Proposed work	97.4	98.4
Bai et al. (2014)	98.7	98.9
Markel and Bilzor (2014)(F-score)	97.0	–

comparatively higher margin than their *RawF* counterparts (Accuracy improvement(%): 6.64, 14.99 and 9.61 for LDA, RF and DT respectively). One of the potential reasons for this reduced performance shall be the presence of samples belonging to new malware families which are originally not part of the training dataset. These *zero-day* samples shall be analyzed in detail to confirm the aforementioned performance degradation that can be carried out as an independent study which we could not do due to the challenges such as samples clustering based on malware families for both training and testing dataset.

Fig. 6 shows the ROC curve and AUC value of different classifiers on raw (Ref. Fig. 6a) and and integrated (Ref. Fig. 6b) feature set. It is clearly evident that integrated feature set is performing better than raw feature set and have 5% more AUC value for Random forest and other classifiers.

4.3.4. Comparing performance on partial AUC

The purpose of this experiment is to test the performance of the proposed integrated feature set with specific FPR intervals. Partial AUC is suitable for this requirement as it provides information on the effectiveness of a classifier in a given range of FPR. McClish (1989), Thompson and Zucchini (1989) and Jiang et al. (1996) have proposed and explained the functionality and capability of partial AUC. Partial AUC metric suits the malware classification task as its usage notes the classifier performance within low FPR area. Earlier, the use of partial AUC are limited to test classifier built for medical domains such as for classifying diagnosis result and disease detection, but due to the high damage caused even by a single cyber security attack, researchers today focus on the performance of classifiers within very low FPR area.

Table 11
Top 10 Features from Raw and Integrated feature set selected with tree method.

Raw		Integrated	
Features	Value	Features	Value
Characteristics	0.202	FH_Char12	0.213
ImageBase	0.107	OH_DLLChar2	0.082
DLL Charaticteristics	0.079	fileinfo	0.080
MajorSubSystemVersion	0.064	OH_DLLChar0	0.072
SubSystem	0.056	FH_Char0	0.064
MajorOSVersion	0.037	SubSytem	0.031
MinorSubSystem	0.030	E_data	0.027
e_Ifanew	0.029	E_file	0.025
SizeOfStackRes	0.028	OH_DLLChar6	0.0248
CreationYear	0.026	FH_Char3	0.0241

From Table 9, it shall be inferred that in FPR interval $[0, 0.1]$ integrated feature set is performing better than raw feature set with the exception of the *kNN* classifier. Similarly in FPR interval $[0, 0.2]$ integrated feature performance is better than raw feature set except for *LR* & *kNN* classifiers. In both intervals, integrated feature set is able to achieve maximum value (0.1 and 0.2 respectively) with the *RF* classifier and so with this result, it can be suggested to use for malware classification task.

4.3.5. Comparison with previous works

In the previous four experiments, it is found that the proposed integrated feature set is performing better than the raw features set.

To further strengthen the accuracy performance of proposed integrated features set, it is important to compare the proposed work with earlier works based on PE file and its header information. To the best of our knowledge, except (Markel and Bilzor, 2014), no other malware classification work has been carried out based on only headers' field values. David et al. (2016) have discussed the structured analysis of various fields of PE header and stated that considering these features enhance the malware detection rate. There are few other works close to the proposed work but

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

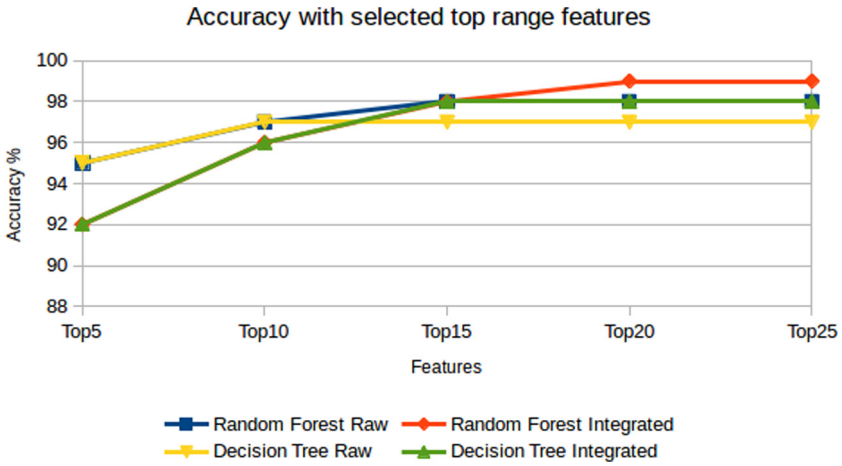

Fig. 7. Decision Tree and Random forest accuracy on top N features.

most of them have used PE header's based feature as a complementing feature and have used them with other features set such as API & DLL calls, byte-n-gram, and opcode-n-gram.

Bai et al. (2014) have used API & DLL calls along with various raw values of PE headers. Comparison with (Bai et al., 2014) work will not state clear merits or demerits of the proposed work but can be tested for approximate results.

Bai et al. (2014) work is reproduced and compared with the proposed integrated feature set. The selection of previous work is based on the similarity of work i.e. learning algorithms and the feature set used.

Table 10 summarizes the comparison result of the proposed work with the previous works. From Table 10, it can be observed that Random forest with proposed feature set has an accuracy of 98.4% which is only 0.5% less than the accuracy of Bai et al. (2014) 98.9%. The improvement of 0.5% might be due to the use of API & DLL calls along with PE headers' field values.

4.3.6. Testing with selected features

Feature selection is the process of selecting a subset of relevant features for use in model construction[13]. Generally, feature selection is performed before training and testing, and it reduces the feature set dimension as it selects only features with high discriminative value. As the focus of the proposed work is to test the classifiers' performance with raw and integrated feature set, feature selection before training and testing is avoided on both feature sets.

In the proposed work, we are interested in knowing the importance ranking of features in raw and integrated feature sets which further helps to understand the difference and importance of features among both sets. Among model based feature selection methods (wrapper methods) tree based methods are easier to apply and without much tuning, it can also model non-linear relations.

We have used Extra Trees Classifier with 250 trees and all other default settings (*scikit-learn* implementation) to get feature importance for raw and integrated feature set. It is an ensemble classifier and by default *Gini impurity* is used to measure the quality of a split. Table 11 lists out the top 10 features from raw and integrated feature set. It is observed that in top 10 from the integrated feature set, 3 features are derived features, 6 are boolean represented by *Characteristics* and *DLLCharacteristics* raw features, *SubSystem* is common in both which clearly indicates the importance of derived features.

To verify, the suitability and efficiency of the proposed integrated feature set, five subsets of features are selected from raw and integrated feature set. Selections are made by considering aforementioned feature ranking and five subsets are created as top5, top10, top15, top20 and top25 having top 5, 10, 15, 20 and 25 features respectively. On these five feature set, Random forest and Decision tree classifiers are trained with 10-fold cross validation and their accuracy is recorded and compared. Fig. 7 shows the accuracy of RF and DT on raw and integrated feature set. From Fig. 7, it is evident that the performance of raw feature is better till top 15 features above which accuracy is constant. More features do not improve accuracy any further whereas integrated feature set have lower or equivalent performance below top 15 features but get improved and have greater accuracy over top 15 than raw feature set. The reasons for this behavior is very obvious, *Characteristics* and *DLLCharacteristics* is in top 5 features in raw dataset whereas it is separated in various boolean features in integrated feature set. Hence, features after top 15 provide constant accuracy for the raw feature set. In the case of integrated feature set, more than 15 top features improve accuracy and overtake the accuracy of the raw feature set because of other informative features apart from *Characteristics* and *DLLCharacteristics* based boolean features and the top 20 features of integrated features set have 6 out of 14 derived features.

5. Conclusions

Detection of malicious PE file from benign is very important as PE file format is the highly used file format, used in *Windows OS*. The proposed work has provided a novel *derived* feature engineering technique that improves the performance of machine learning based classifier for malicious PE file detection. The proposed technique used static analysis technique to extract the features which have low time and resource requirement than dynamic analysis. It has also compared the performance of the proposed integrated feature set with the raw feature set by training and testing different classes of machine learning algorithms and found that the performance of classifiers improved significantly with the new integrated feature set. To record a robust and stable performance metric from classifiers, validation is performed with four different methods, train-and-test split, 10-fold cross validation, new test dataset and Partial AUC. With all four methods, integrated feature set is performed better than the raw feature set. The performance of the proposed integrated feature set is also compared with the work of Bai et al. (2014) who used PE headers' fields' values along with DLL & API calls. The accuracy of the integrated feature set is

[13] https://en.wikipedia.org/wiki/Feature_selection.

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

only 0.5% less than the Bai et al. (2014) work, and the probable reason might be the use of DLL & API calls.

Contributions of the proposed work are as listed below:

- An integrated feature set is created, which has raw and derived features, based on different header fields' values of PE file. The proposed integrated feature set improves the classification accuracy of machine learning classifiers.
- The proposed work also provides an empirical evidence for usability of different headers' fields' value of PE file as useful discriminative features. Experiments' result clearly suggest that header fields' raw values and values derived from these can be used to classify malware programs from benign programs.

As a tangible outcome, the proposed work provides a pre-processed dataset having 2722 malware and 2488 benign program samples.[14] Extracted and processed raw & integrated feature set is also available for comparing future work with the proposed work.[15] The proposed dataset is pre-processed following most of the required metric for malware research suggested in Firdausi et al. (2010). The dataset will help future researchers of this field to take research on the further level by having such dataset in advance.

5.1. Limitations and Future works

In the result and analysis section (Ref. Section 4.3), we have empirically observed that it is possible to build malware classifier using header fields' value alone as the feature and achieved a good classification accuracy.

The real-world scenario is unpredictable and can be different than experimental environment so to protect a sensitive system from malware, it is not advisable to use only headers' values based classifier. For example, a carefully crafted malware would have a benign header and malicious payload hidden in the body of PE file. Hence, the proposed work is not suggesting the stand-alone use of such classifier and also does not compete with other features like (API & DLL call, byte-n-gram, and opcode-n-gram) based classifiers. Instead, the proposed work is an attempt to complement existing classifiers with a high accuracy and low computation based classifier built on PE headers' values. The proposed work also shows improvement in accuracy using derived features in conjunction with a subset of existing raw features over the accuracy of only raw features. Currently, the proposed work is limited with PE file format but as the proposed feature engineering method is very generic, it can be extended to other file formats such as *image*, *pdf*, *audio* and including mobile OS such as *Android* and *iOS*.

In current work, malware and benign samples' application type information is not taken into consideration. The impact of application type on classification discrimination can be considered as important future directions. The classification of PE file representing different application type such as multimedia, document processing, device drivers, etc. shall be studied with both malware and benign samples belonging to aforementioned application type. For future study, the performance of a new feature set after combining proposed integrated feature with another feature set can be studied and an in-depth comparison with earlier works would be performed. Another file format can be studied in future and with feature engineering, the scope of similarly derived features can be explored.

Appendix A. Raw features used in the proposed work

See Table A.12.

[14] Available on email request (to minimize misuse).

[15] www.github.com/urwithajit9/clamp.

Table A.12
List of header's fields used as raw features.

Header	Fields
DOS_HEADER(6/19)	e_cblp, e_cp, e_cparhdr, e_maxalloc, e_sp, e_lfanew
FILE_HEADER(1/7)	NumberOfSections
OPTIONAL_HEADER (8 + 13)	MajorLinkerVersion, MinorLinkerVersion, SizeOfCode, SizeOfInitializedData, SizeOfUninitializedData, AddressOfEntryPoint, BaseOfCode, BaseOfData, MajorOperatingSystemVersion, MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, CheckSum, MajorSubsystemVersion, MinorSubsystemVersion, Subsystem, SizeOfStackReserve, SizeOfStackCommit, SizeOfHeapReserve, SizeOfHeapCommit, LoaderFlags

References

Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R., 2004. N-gram-based detection of new malicious code. Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International, vol. 2. IEEE, pp. 41–42.

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. ACM, pp. 183–194.

Altaher, A., ALmomani, A., Anbar, M., Ramadass, S., et al., 2012. Malware detection based on evolving clustering method for classification. In: Sci. Res. Essays 7 (22), 2031–2036.

Bai, J., Wang, J., Zou, G., 2014. A malware detection scheme based on mining format information. Sci. World J. 2014.

Baldangombo, U., Jambaljav, N., Horng, S.-J., 2013. A static malware detection system using data mining methods. arXiv preprint arXiv:1308.2831.

Belaoued, M., Mazouzi, S., 2015. A real-time pe-malware detection system based on chi-square test and pe-file features. In: IFIP International Conference on Computer Science and its Applications. Springer, pp. 416–425.

Bilar, D., 2007. Opcodes as predictor for malware. Int. J. Electron. Secur. Digital Forensics 1 (2), 156–168.

David, B., Filiol, E., Gallienne, K., 2016. Structural analysis of binary executable headers for malware detection optimization. Journal of Computer Virology and Hacking Techniques, 1–7.

Egele, M., Scholte, T., Kirda, E., Kruegel, C., 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. (CSUR) 44 (2), 6.

Firdausi, I., Erwin, A., Nugroho, A.S., et al., 2010. Analysis of machine learning techniques used in behavior-based malware detection. In: Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on. IEEE, pp. 201–203.

Idika, N., Mathur, A.P., 2007. A Survey of Malware Detection Techniques, 48. Purdue University.

Jiang, Y., Metz, C.E., Nishikawa, R.M., 1996. A receiver operating characteristic partial area index for highly sensitive diagnostic tests. Radiology 201 (3), 745–750.

Liao, Y., 2012. Pe-header-based malware study and detection. Retrieved from the University of Georgia: http://www.cs.uga.edu/l~iao/PEFinalReport.pdf

Markel, Z., Bilzor, M., 2014. Building a machine learning classifier for malware detection. In: Anti-malware Testing Research (WATeR), 2014 Second Workshop on. IEEE, pp. 1–4.

Markel, Z.A., 2015. Machine Learning Based Malware Detection. Tech. rep., DTIC Document.

McClish, D.K., 1989. Analyzing a portion of the roc curve. Med. Decis. Making 9 (3), 190–195.

Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., Elovici, Y., 2008. Unknown malcode detection using opcode representation. In: Intelligence and Security Informatics. Springer, pp. 204–215.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Perdisci, R., Lanzi, A., Lee, W., 2008. Classification of packed executables for accurate computer virus detection. Pattern Recognit. Lett. 29 (14), 1941–1946.

Pietrek, M., 1994. Peering inside the pe: a tour of the win32 (r) portable executable file format. Microsoft Syst. J.-US Ed., 15–38

Qin, J., Yan, H., Si, Q., Yan, F., 2010. A trojan horse detection technology based on behavior analysis. In: 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM). IEEE, pp. 1–4.

Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., Hamze, A., 2010. Malware detection based on mining api calls. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, pp. 1020–1025.

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003

Santos, I., Brezo, F., Nieves, J., Penya, Y.K., Sanz, B., Laorden, C., Bringas, P.G., 2010. Idea: Opcode-sequence-based malware detection. In: International Symposium on Engineering Secure Software and Systems. Springer, pp. 35–43.

Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J., 2001. Data mining methods for detection of new malicious executables. In: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. IEEE, pp. 38–49.

Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C., 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Inf. Secur. Tech. Rep. 14 (1), 16–29.

Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M., 2009. Pe-miner: Mining structural information to detect malicious executables in realtime. In: International Workshop on Recent Advances in Intrusion Detection. Springer, pp. 121–141.

Shankarapani, M.K., Ramamoorthy, S., Movva, R.S., Mukkamala, S., 2011. Malware detection using assembly and api call sequences. J. Comput. Virol. 7 (2), 107–119.

Shannon, C., 1948. A mathematical theory of communication. Bell Syst. Tech. J. 27. 379–423 and 623–656. Mathematical Reviews (MathSciNet): MR10, 133e.

Thompson, M.L., Zucchini, W., 1989. On the statistical analysis of roc curves. Stat. Med. 8 (10), 1277–1290.

Treadwell, S., Zhou, M., 2009. A heuristic approach for detection of obfuscated malware. In: Intelligence and Security Informatics, 2009. ISI'09. IEEE International Conference on. IEEE, pp. 291–299.

Vinod, P., Laxmi, V., Gaur, M.S., 2011. Scattered feature space for malware analysis. In: International Conference on Advances in Computing and Communications. Springer, pp. 562–571.

Walenstein, A., Hefner, D.J., Wichers, J., 2010. Header information in malware families and impact on automated classifiers. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. IEEE, pp. 15–22.

Wang, T.-Y., Wu, C.-H., Hsieh, C.-C., 2009. Detecting unknown malicious executables using portable executable headers. In: INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on. IEEE, pp. 278–284.

Yan, G., Brown, N., Kong, D., 2013. Exploring discriminatory features for automated malware classification. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 41–61.

Ye, Y., Wang, D., Li, T., Ye, D., Jiang, Q., 2008. An intelligent pe-malware detection system based on association mining. J. Comput. Virol. 4 (4), 323–334.

Yonts, J., 2012. Attributes of Malicious Files. SANS Institute InfoSec Reading Room.

Please cite this article in press as: Kumar, A., et al. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University – Computer and Information Sciences (2017), http://dx.doi.org/10.1016/j.jksuci.2017.01.003