Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 1173 – 1183

doi:10.3906/elk-1601-189

A comparison of feature extraction techniques for malware analysis

Mohammad IMRAN*, Muhammad Tanvir AFZAL, Muhammad Abdul QADIR
Department of Computer Science, Capital University of Science and Technology, Islamabad, Pakistan

Received: 18.01.2016 • **Accepted/Published Online:** 06.04.2016 • **Final Version:** 10.04.2017

Abstract: The manifold growth of malware in recent years has resulted in extensive research being conducted in the domain of malware analysis and detection, and theories from a wide variety of scientific knowledge domains have been applied to solve this problem. The algorithms from the machine learning paradigm have been particularly explored, and many feature extraction methods have been proposed in the literature for representing malware as feature vectors to be used in machine learning algorithms. In this paper we present a comparison of several feature extraction techniques by first applying them on system call logs of real malware, and then evaluating them using a random forest classifier. In our experiment the HMM-based feature extraction method outperformed the other methods by obtaining an F-measure of 0.87. We also explored the possibility of using ensembles of feature extraction methods, and discovered that combination of HMM-based features with bigram frequency features improved the F-measure by 1.7%.

Key words: Malware analysis, feature extraction, machine learning

1. Introduction

The ever-increasing rate of malware attacks on computers, computer networks, and smart phones has emphasized the need for aggressive measures to counter the malware threat. As a result, malware analysis and detection has been an active area of research lately, and a multitude of techniques have been proposed in this field using concepts from a diverse range of scientific disciplines such as graph theory [1,2], machine learning [3,4], and information visualization [5,6].

Among the aforementioned methods, machine learning-based algorithms have gained special attention among the malware research community. The main reason behind using these techniques is that they enable detection of a previously unknown threat using models leaned from known malware. This is a key requirement in today's cyber world, where millions of new malware are reported every day (http://www.symantec.com/security_response/publications/threatreport.jsp), and a large number of the new malware are obfuscated from existing malware [1].

The machine learning-based techniques generally require a feature vector representation of malware, where a feature represents a particular malware attribute that can play a discriminatory role in the classification process. Extracting discriminatory attributes pertaining to malware and representing them in a way to be effectively used in a machine learning setting is a major challenge in the domain of malware analysis and detection. Termed as *feature extraction* in the machine learning paradigm, this process is a prerequisite to every malware detection technique proposed in the literature that employs a machine learning algorithm.

In this paper we report a comparison of nine feature extraction methods commonly used for malware

*Correspondence: m.imran.isd@gmail.com

detection and classification tasks, using a comprehensive dataset that contains system call logs of real malware. Specifically, we transform sequences of system calls into fixed length feature vectors by applying different feature extraction methods, and evaluate these transformations by performing malware classification using a random forest classifier on the feature vectors. There is no gold standard dataset as such for performing a comparative evaluation of malware classification techniques, and all of the feature extraction techniques have been tested on different data. The main contribution of this paper, therefore, is that it compares different feature extraction methods on the *same* data, hence providing an empirical comparative assessment of these methods. The paper also adds value to the research by evaluating a novel approach of combining the features extracted through different methods for the purpose of malware classification.

Liu et al. [7] performed a similar study about 10 years ago in which three feature extraction methods were addressed. Spanning the research of the last decade and earlier, our work covers a broader range of feature extraction methods. A recent comparison of feature extraction methods can be found in [8] but it focuses mainly on a survey of such methods and falls short of evaluating them on some benchmark data.

This paper proceeds as follows. The next section imparts some necessary background knowledge on the feature selection methods being compared in this paper. Section 3 covers related work while the methodology adopted for the experiments is explained in Section 4. Results are presented and discussed in Section 5.

2. Feature extraction methods for malware analysis

Machine learning algorithms learn the patterns from fixed length feature vectors, and therefore feature extraction is the first step before using these algorithms for malware analysis. For features that are in the form of sequences, such as sequences of code bytes, operation codes (opcodes), system calls, or API calls, the creation of a representative feature vector is a nontrivial problem, and hence various feature extraction methods have been proposed in the literature for this task. Here we shall outline the most widely used feature extraction and representation techniques applied on sequences. Let us define a finite set $\mathbb{S} = \{s_1, s_2, \ldots, s_n\}$ containing, in a specific order, all the unique symbols s_i allowed to make up a sequence. The set $\mathbb{S}$ may be considered as a *term dictionary* in the information retrieval terminology. Any arbitrary sequence containing possibly repetitive occurrences of elements of $\mathbb{S}$ in any order can then be represented as $S = (s_k)_{k \in \{1..|\mathbb{S}|\}}$.

2.1. Binary feature extraction

The baseline method of extracting features from a sequence is to identify all the distinct elements found in the sequence. The sequence can then be represented as a binary vector of the same length as the term dictionary $\mathbb{S}$ such that each feature in the vector signifies the presence or absence of the corresponding dictionary term in the sequence. The resulting feature vector can be represented as $V_{Sb} = (b_{s1}, b_{s2}, \ldots, b_{sn})$, where b_{si} is 1 if S contains at least one instance of s_i and 0 otherwise, and n is the size of $\mathbb{S}$. This method has been used for sequences of code bytes [4], API calls [9], etc.

2.2. Frequency feature extraction

In this feature extraction method, the count of occurrence of a dictionary term in the sequence is used instead of just its presence or absence [10]. Mathematically, this vector can be denoted as $V_{Sf} = (f_{s1}, f_{s2}, \ldots, f_{sn})$, where f_{si} is the frequency of the i^{th} element of $\mathbb{S}$ in S.

2.3. Frequency weight feature extraction

Frequency weighting methods such as *term frequency-inverse document frequency* (TF-IDF) have also been employed to generate feature vectors from sequences [11]. TF-IDF is a statistic widely used in information retrieval and text mining domains for calculating frequency-based weights for terms in a document in order to assess their relative importance.

Frequency weighting vector for S will be $V_{Sfw} = (fw_{s1}, fw_{s2}, \ldots, fw_{sn})$, where fw_{si} represents the frequency weight for the i^{th} element of $\mathbb{S}$ as computed over S.

2.4. Hidden Markov model

Another method of extracting features from sequences is based on hidden Markov models (HMMs) [12]. Although the authors of the cited text do not refer to their technique as a feature selection method, the process that they have adopted performs the exact task of converting a given sequence into a fixed length vector that can subsequently be used by a discriminative classifier for the classification of sequences. Imran et al. [13] have applied the same methodology for malware classification.

An HMM represents a doubly stochastic system in the form of a finite state in which the states are not visible. The progress of the state machine is observed through certain symbols that are emitted in each state. Two types of probabilities govern the state machine: the state transition probabilities determine the next state for any given state, and symbol emission probabilities indicate the possibility of each symbol being emitted in a given state. The Baum–Welch algorithm is used to compute the transition and symbol probabilities from a given sequence. Once an HMM has been trained or *learnt* from a given sequence (or a set of sequences) of symbols, an unknown symbol sequence can be evaluated for its resemblance with the model using the forward–backward algorithm [14], which returns the likelihood of the unknown sequence being generated by the HMM.

The technique presented in [12] involves training one HMM for every class of sequences. Let us suppose the input sequences belong to and are labeled with m classes. Symbols in this case are represented by the individual system calls. Let us further assume the total number of sequences to be k. After the training of HMMs has been done, there are m HMMs representing different classes of sequences. The next step is to evaluate all k sequences against the mHMMs and for each sequence its likelihood score is recorded as a feature. In this way, a feature space consisting of k feature vectors is produced where each vector is of length m. In mathematical notation, the feature vector for a sequence S would be $V_{Shmm} = (l_{S,1}, l_{S,2}, \ldots, l_{S,m})$, where $l_{S,i}$ is the likelihood score obtained by evaluating S against the ith HMM.

3. Related work

Considerable research efforts have made use of machine learning methods for the task of malware detection and classification using static, dynamic, and hybrid features. Here we briefly describe some representative research efforts that have used the methods discussed in the previous section for extracting features from sequential representation of malware.

Tian et al. [9] proposed a binary feature technique for malware detection and classification using API calls. The authors also experimented with the frequency based methods on the same data but no improvement was observed over the binary representation. In a similar approach, Devesa et al. [15] monitored API calls of malware and benign programs, and derived rules for extracting actions performed by the programs from the API call logs. The term dictionary therefore consisted of the performed actions, and a binary feature extraction was performed.

Alazab et al. [16] represented the API sequences with n-grams for values of n from 1 to 5. To keep the feature space to a manageable size, frequencies of all n-grams for a given n were computed for the whole dataset of malware and benign samples, and the 100 most frequent n-grams were selected to be included in the feature vector. Altaher et al. [17] extracted API calls from the PE executables and the frequency of each API call was considered as a potential feature. All the API calls were then ranked according to their information gain and frequencies of the top 50 API calls constituted the final feature vector.

Schultz et al. [4] used three kinds of static features extracted from DLL-related information, printable strings, and binary code of malicious files. In case of the code bytes, all the two-byte words found in malware code were combined into the term dictionary and binary feature representation was used for each malware sample. Following in their footsteps, Kolter and Maloof [18] created features using 4-grams of byte codes from executables. Since the feature space grows rapidly with increasing n, the authors used information gain to select the 500 most representative 4-grams and used them as binary features in various machine learning algorithms.

Lee et al. [19] modeled each 7-gram of system calls observed during execution of normal and intrusive executions of the Unix *sendmail* program as a feature in their intrusion detection scheme. Another feature extraction method in the domain of intrusion detection is found in [20] in which Liao and Vemuri constructed fixed length feature vectors from system call sequences by computing the TF-IDF as the weight for each unique system call in the corpus and using this weight as a feature.

Research reported in [11] made use of TF-IDF on frequencies of low-level kernel calls to represent kernel calls made by a program as a weight vector that was then used for cosine similarity computation. In recent works, Lin et al. [21] suggested computing of TF-IDF within each malware family, instead of on the whole corpus, for feature selection.

One of the few approaches employing an HMM as a feature extraction method was proposed by Annachhatre et al. [22], who used opcode sequences extracted from various compilers and virus generators to train HMMs. Opcode sequences from malware samples were then scored against the HMMs to generate the feature vectors, which were then used for clustering of malware samples. Imran et al. [13] varied this method by modeling malicious behavior instead of compiler behavior. In their approach, malware behavior was represented as a sequence of system calls used to train separate HMMs for different malware families. Known malware were evaluated against the malware family HMMs and the resulting score vectors were used to classify unknown malware using a discriminative classifier.

4. Methodology

In this section we describe the dataset used and methodology adopted for performing the comparison of different feature selection techniques proposed in the literature.

4.1. The dataset

As discussed above, various methods of capturing malware behavior have been proposed in the quest for finding the most descriptive dynamic feature. System calls have been shown to represent a program's behavior effectively and therefore we opted for sequences of system calls as the input to the feature selection methods being compared. Specifically, we exploited a subset of the system call data used by Rieck et al. [3] in the Malheur project. The original dataset includes system call logs for over 32,000 malware samples, belonging to more than 400 families. For the purposes of this paper, system call logs for 8828 malware samples were selected from 36 families. The malware families were chosen in a way so that each family had at least 100 samples. The maximum number of

samples in a family was also limited to 400 because we think it is a reasonable sample size for training machine learning algorithms.

4.2. Data preprocessing

The original dataset used by Rieck et al. was encoded in the MIST format [23], which represents a system call in the form of a string of hexadecimal numbers symbolizing the high level category of the call, the actual call, and any arguments passed to the call. A representation of the *load_dll* system call is shown in Figure 1.

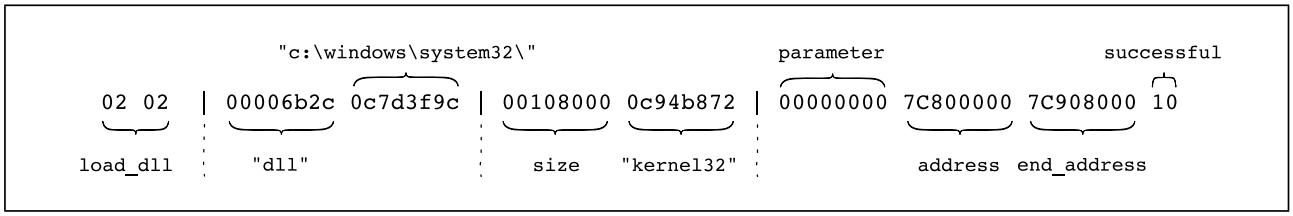


Figure 1. MIST representation of the *load_dll* system call [23].

The MIST representation was further processed in order to strip the high level category and the arguments, and to keep only the system call IDs. Using the information given in [23] about the total number of system call categories and the number of system calls within each category, we were able to code each system call as a decimal number from 1 to 120, where 120 is the total number of system calls monitored and recorded by Rieck et al. In this way, the sequence of system calls for a particular program is now represented as a sequence of numbers in the range from 1 to 120.

4.3. Feature extraction

Nine sets of feature vectors were generated against the system call sequences for comparative evaluation of the feature extraction methods, as described below. First, the feature extraction methods based on binary, frequency, and frequency weighting factors were applied on system call unigrams of all the 8828 sequences in the dataset. This resulted in three sets of feature vectors, namely unigram binary, unigram frequency, and unigram frequency weighting sets.

Since the sequential information in the input data is lost by applying the above-mentioned feature extraction methods on unigrams, the use of bigrams is suggested in the literature to preserve the order of items (system calls in this case) in the input data. For this reason, three more feature sets were obtained by applying the binary, frequency, and frequency weighting feature extraction methods on bigrams of system calls. The term dictionary for the system call bigrams was populated with 2745 unique bigrams extracted from the dataset, and therefore each of the feature vectors for the bigram schemes included 2745 features.

Some of the reviewed schemes performed feature selection based on information gain. In order for this study to cover the feature selection aspect, we performed feature ranking on the bigram frequency feature set based on information gain using Weka [24]. The top 100 highest ranked (most significant) frequencies were then selected as features in the seventh feature set, which will be referred to as the *bigram freq IG* set later in the document.

The eighth set of feature vectors was obtained by training 36 HMMs on system call sequences belonging to the corresponding malware families. Each malware sample was then scored against each of the HMMs to obtain

the feature vector composed of likelihood scores. HMM model creation and evaluation (scoring) were done in MATLAB using the HMM toolbox (http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm_download.html).

Another experiment was carried out to judge the impact of combining feature vectors obtained through different feature extraction methods. As a trial, the HMM-based feature vectors were truncated with *bigram freq IG* feature vectors to obtain the ninth and final feature set.

Figure 2 shows the number of features extracted by the nine schemes discussed above. The HMM-based feature extraction method proved to be the most concise in terms of the feature count, with only 36 features representing arbitrarily long sequences of system calls. On the other extreme, the number of bigram features exceeded 2700.

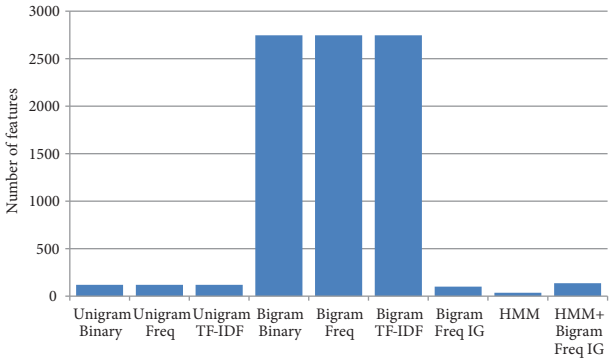


Figure 2. Number of features extracted through different feature extraction schemes.

4.4. Evaluation using a random forest classifier

For evaluating the nine sets of feature vectors obtained by applying various feature extraction methods, these sets were subjected to classification using a random forest classifier in Weka version 3.7. The random forest classifier was selected after comparing its performance against various other classification algorithms including J48 decision tree, k-NN, naïve Bayes, and support vector machine (SVM) in small-scale experiments.

The random forest classifier belongs to the ensemble learning paradigm within the domain of machine learning, in which results of multiple tree classifiers are aggregated to predict a sample's class [25]. More specifically, it leverages the concept of *bagging*, which corresponds to the process of creating tree classifiers on different samples of data and then taking a majority vote for reaching a classification decision. The randomness is involved when splitting a node in a classification tree: instead of choosing the best variable (predictor) from the whole set of variables for splitting a given node, the random forest classifier randomly creates a subset of predictors at each node and selects the best predictor from the subset. Using this simple strategy, the random forest classifier is not only able to outperform other classification algorithms such as SVMs and neural networks [25], but it also deals effectively with the problem of overfitting [26].

4.5. Classifier validation

Weka offers multiple choices of validation methods for testing how well a classifier is modeled on some given data. These validation methods include the following:

1. Using the same data for training and testing
2. Choosing separate datasets for training and testing

3. Splitting the dataset into training and testing partitions by a specific percentage
4. k-fold cross validation.

The first method is obviously not desirable because it does not judge how *generalized* a trained classifier is. Sometimes the classifier memorizes the training set instead of learning from it and therefore does not represent a generalization of the given instances. As a result such a classifier performs well on the data that has been used for training but does not classify previously unseen samples correctly [27]. The second method tests the classifier on unseen data, hence providing a better judgment about classifier performance, but requires different data files for training and testing. The third method is a simplification of the second such that it allows splitting the same data file into training and testing partitions without the need for separate files. The problem with the second and third evaluation methods is that they evaluate the classifier for just one set of training and test partitions. In such a case, the distribution of data instances in the two partitions might affect the classifier performance in a favorable or adverse manner. The k-fold cross validation method solves this problem by dividing the dataset into k partitions and performing k classification sessions such that in every session, or fold, a separate set of k-1 partitions is used for training and the remaining partition is used for testing the classifier. In this way, each partition is used k-1 times as training partition and once as test partition, hence reducing the effects of any bias in the data. The classification results of k folds are then averaged to obtain the overall classification performance of the classifier over all the data. Due to this comprehensive nature of k-fold cross validation method, we used it to validate the random forest classifier in our experiments with $k = 10$.

4.6. Evaluation metrics

Two metrics that are commonly used for reporting the performance of a classifier are *precision* and *recall*. Precision represents how many of the identified samples are correct and recall describes how many of the total samples are correctly identified. Mathematically, these two metrics are represented as

$$Precision = \frac{True\,Positives}{True\,Positives + False\,Positives} \tag{1}$$

$$Recall = \frac{True\,Positives}{True\,Positives + False\,Negatives} \tag{2}$$

Precision and recall are often combined into a unified metric, called *F-measure*, for convenience of comparisons between various classification methods. The value of F-measure is in the range from 0 to 1, where 0 depicts that no sample was identified correctly and 1 signifies a perfect classification. F-measure is defined as

$$F\_measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

The chart in Figure 3 shows the effectiveness of the feature extraction methods on the random forest classifier in terms of average F-measure across all the families. Subsequently in this document, the term F-measure represents the averaged value.

To analyze the overall classifier performance for the top feature extraction methods (bigram binary and HMM methods), we present two types of comparison graphs: a chart for the family-wise F-measure values against these two methods is given in Figure 4, while Figure 5 shows the confusion matrices for classification using the two methods respectively.

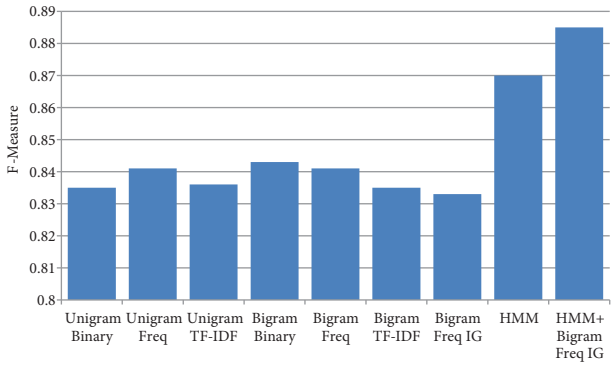


Figure 3. Comparison of feature extraction methods using a random forest classifier.

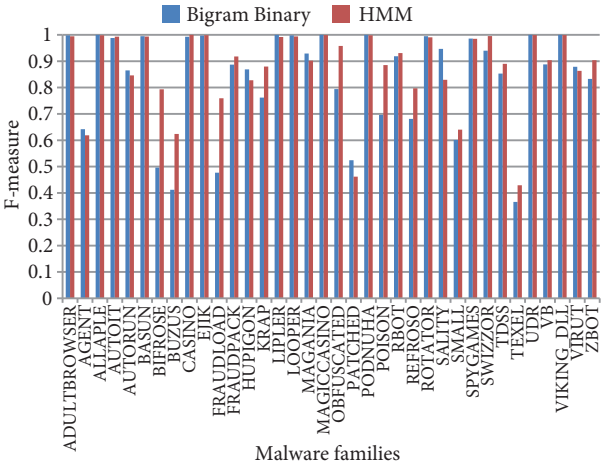
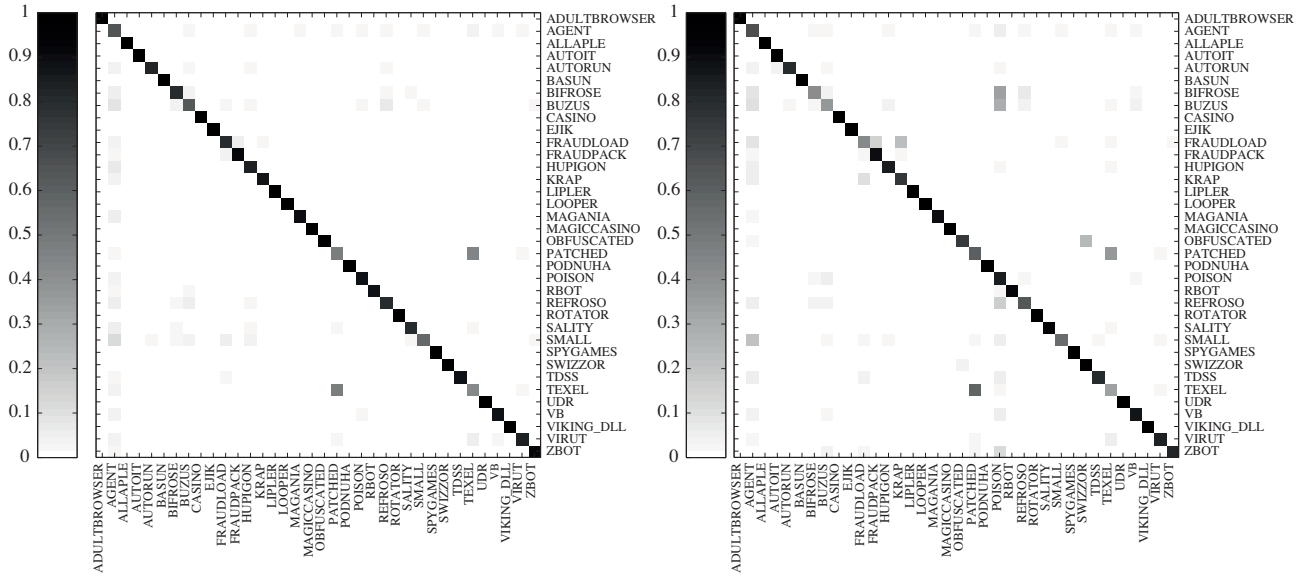


Figure 4. Family-wise F-measure for bigram binary and HMM methods.



(a) Using HMM features

(b) Using bigram binary features

Figure 5. Confusion matrices for classification.

5. Results and discussion

As is obvious from Figure 3, features extracted through the HMM were the best representatives of malware behavior among all the other individual methods. Classification using HMM features resulted in the highest F-measure value of 0.87, which was 3.2% more than the next highest F-measure of 0.843 for the bigram binary features. This result, combined with the observations made from Figure 2, signifies the expressiveness of HMM-based features since they were responsible for the best classification while being smallest in number among all other feature extraction methods.

A comparison of family-wise F-measures for bigram binary and HMM features is given in Figure 4. Although for most of the malware families the classification results for features extracted through the two

methods are close, a significant difference is observed in a few malware families such as BIFROSE, BUZUS, FRAUDLOAD, and POISON. For these malware families, the bigram binary features had scored a relatively low F-measure, but the HMM-based features proved to be more discriminative as depicted by their higher F-measure scores. Figures 5a and 5b, which show the confusion matrices for classification using the same two feature extraction methods, reveal the reason for the difference in F-measures. The confusion matrices are mostly similar, hinting that the classifier was able to discriminate between malware families on the basis of features extracted through the HMM and bigram binary methods alike. The difference is seen in BIFROSE and BUZUS samples, which were misclassified as belonging to the POSION family in the case of bigram binary features, but for HMM-based features these misclassifications were significantly less. This is an indication of the supremacy of HMM-based features because the classifier was able to differentiate between closely resembling families (BIFROSE and POISON, for example) on the basis of these features.

The feature extraction methods using bigrams were only marginally better than unigram-based methods with the exception of the binary feature method, in which case the unigram method showed slight improvement over bigram features. At first glance this observation may seem to be erroneous, since losing the sequence information in the case of unigram should result in a higher false positive rate, thus making a significant difference in the overall classification performance. A plausible explanation for the comparable results is that the system calls for performing a particular task are usually grouped together, and the sequence of calls within a short group is generally maintained across all malware families. Therefore, there do not seem to be many out of sequence system calls present in the dataset that would lead to false positives. One could, however, engineer a sequence by injecting dummy or redundant system calls, thus making a new malware look different from the known malware, which would result in a misclassification.

Applying the TF-IDF weighting scheme had an insignificant, albeit slightly negative, effect on classification performance as compared to using absolute (raw) frequencies on both unigram and bigram sets. This result suggests that the patterns of individual system call frequencies are quite specific to malware families; therefore, using the normalized and corpus-wise frequencies did not add any useful and discriminative information to the feature set.

The binary feature method remained below the frequency feature scheme in unigram-based representation but was better using bigrams, although the difference was negligible in both cases. The fact that the binary and frequency methods produced similar results further strengthens our earlier observation that system calls made by malware belonging to a specific malware family are distinct from those for other malware.

The average F-measure for the bigram frequency IG feature set, obtained after applying feature selection to the bigram frequency set, decreased to 0.833 from 0.841 for the original feature set. In other words, an approximately 1% decrease in classification performance was observed for a 96.4% decrease in the number of features. The loss in classification performance is negligible against the reduction in features, yet these two factors should be weighed before using feature selection in a given malware analysis scenario.

In the case of the combined HMM and bigram frequency IG feature vector, the average F-measure increased to 0.885, which is a gain of 1.7% as compared to the F-measure value of 0.87 for the individual HMM feature set and 5.2% more against the F-measure of 0.841 for the bigram frequency IG feature set. This is an encouraging result and paves the way for further research on using various combinations of feature vectors for the task of malware detection and classification.

The feature extraction methods used in this research have been evaluated on system calls at a somewhat middle level of granularity. A higher level, according to MIST format, would be the category level (file system, registry, communication, etc.) and a lower granularity would include system call parameters as

well. Experimenting with lower granularity may produce better results for all the schemes but will incur more computational overheads since the combination of a system call with different parameters will result in a greater number of distinct elements in the term dictionary. A possible extension of this study, thus, could include evaluating the feature extraction methods for different granularities of the input sequences.

In this paper we compared, through experimental evaluation, various feature extraction methods proposed in the malware analysis and detection domain. The feature extraction methods were implemented and applied to a dataset of system call sequences representing malware behavior. Features extracted through the HMM produced the best classification results on the random forest classifier. It was also observed that using bigrams of system calls did not have a significant edge over unigrams, and representing a sequence with binary features was comparable to using the frequency or frequency weighting representation. Another important finding was that better classification can be achieved by combining features extracted through different feature extraction methods.

References

[1] Elhadi AAE, Maarof MA, Barry BI, Hamza H. Enhancing the detection of metamorphic malware using call graphs. Comput Secur 2014; 46: 62-78.

[2] Hu X, Chiueh T, Shin KG. Large-scale malware indexing using function-call graphs. Proc of CCS'09 2009; 611-620.

[3] Rieck K, Trinius P, Willems C, Holz T. Automatic analysis of malware behavior using machine learning. J Comput Sec 2011; 19: 639-668.

[4] Schultz MG, Eskin E, Zadok E, Stolfo SJ. Data mining methods for detection of new malicious executables. IEEE Proc of S&P 2001; 38-49.

[5] Nataraj L, Karthikeyan S, Jacob G, Manjunath B. Malware images: visualization and automatic classification. Proc of VizSec'11 2011; 4.

[6] Saxe J, Mentis D, Greamo C. Visualization of shared system call sequence relationships in large malware corpora. Proc of VizSec'12 2012; 33-40.

[7] Liu A, Martin C, Hetherington T, Matzner S. A comparison of system call feature representations for insider threat detection. Proc of IAW'05 2005; 340-347.

[8] Ranveer S, Hiray S. Comparative analysis of feature extraction methods of malware detection. Int J Comput App 2015; 120.

[9] Tian R, Islam R, Batten L, Versteeg S. Differentiating malware from cleanware using behavioural analysis. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on; 2010; New York, NY, USA: IEEE. pp. 23-30.

[10] Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Info Sci 2013; 231: 64-82.

[11] Marian T, Weatherspoon H, Lee KS, Sagar A. Fmeter: Extracting indexable low-level system signatures by counting kernel function calls. In: Middleware 2012 Springer; 2012. pp. 81-100.

[12] Bicego M, Murino V, Figueiredo MA. Similarity-based classification of sequences using hidden Markov models. Pattern Recog 2004; 37: 2281-2291.

[13] Imran M, Afzal MT, Qadir MA. Similarity-based Malware Classification using Hidden Markov Model. Proc of CyberSec2015 2015; 129-134.

[14] Rabiner L, Juang BH. An introduction to hidden Markov models. ASSP Mag 1986; 3: 4-16.

[15] Devesa J, Santos I, Cantero X, Penya YK, Bringas PG. Automatic Behaviour-based Analysis and Classification System for Malware Detection. In: ICEIS (2); 2010; pp. 395-399.

[16] Alazab M, Layton R, Venkataraman S, Watters P. Malware detection based on structural and behavioural features of API calls. Edith Cowan University, 2010.

[17] Altaher A, Ramadass S, Ali A. Computer virus detection using features ranking and machine learning. Aus J Bas App Sci 2011; 5: 1482-1486.

[18] Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. Proc of KDD'04 2004; 470-478.

[19] Lee W, Stolfo SJ, Chan PK. Learning patterns from unix process execution traces for intrusion detection. In: AAAI Workshop on AI Approaches to Fraud Detection and Risk Management; 1997; pp. 50-56.

[20] Liao Y, Vemuri VR. Using Text Categorization Techniques for Intrusion Detection. In: USENIX Security Symposium; 2002; pp. 51-59.

[21] Lin C-T, Wang N-J, Xiao H, Eckert C. Feature selection and extraction for malware classification. J Info Sci Eng 2015; 31: 965-992.

[22] Annachhatre C, Austin T, Stamp M. Hidden Markov models for malware classification. J Comput Virol Hack Tech 2014; 1-15.

[23] Trinius P, Willems C, Holz T, Rieck K. A malware instruction set for behavior-based analysis. 2009.

[24] Garner SR. Weka: The waikato environment for knowledge analysis. The University of Waikato, 2007.

[25] Liaw A, Wiener M. Classification and regression by Random Forest. R News 2002; 2: 18-22.

[26] Breiman L. Random forests. Mach Learn 2001; 45: 5-32.

[27] Bascil MS, Temurtas F. A study on hepatitis disease diagnosis using multilayer neural network with Levenberg Marquardt training algorithm. J Med Syst 2011; 35: 433-436.