Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Hybrids of support vector machine wrapper and filter based framework for malware detection

Shamsul Huda [a,*], Jemal Abawajy [b], Mamoun Alazab [c], Mali Abdollalihian [d], Rafiqul Islam [e], John Yearwood [a]

[a] School of SITE, Federation University, Australia
[b] School of Information Technology, Deakin University, Australia
[c] Australian National University, Australia
[d] Mathematical and Geospatial Sciences Department, RMIT University, Australia
[e] Charles Sturt university, Australia

HIGHLIGHTS

- A signature-free malware detection approach has been proposed.
- A hybrid wrapper–Filter based malware feature selection has been proposed.
- Proposed hybrid approach can take advantages from both filter and wrapper.
- Models have also been validated by statistical model selection criteria such as Chi Square and Akaike information criterion (AIC).

ARTICLE INFO

Article history:
Received 19 November 2013
Received in revised form
28 April 2014
Accepted 1 June 2014
Available online xxxx

Keywords:
Malware detection
API call statistics
Hybrid wrapper–filter heuristics

ABSTRACT

Malware replicates itself and produces offspring with the same characteristics but different signatures by using code obfuscation techniques. Current generation Anti-Virus (AV) engines employ a signature-template type detection approach where malware can easily evade existing signatures in the database. This reduces the capability of current AV engines in detecting malware. In this paper we propose a hybrid framework for malware detection by using the hybrids of Support Vector Machines Wrapper, Maximum-Relevance–Minimum-Redundancy Filter heuristics where Application Program Interface (API) call statistics are used as a malware features. The novelty of our hybrid framework is that it injects the filter's ranking score in the wrapper selection process and combines the properties of both wrapper and filters and API call statistics which can detect malware based on the nature of infectious actions instead of signature. To the best of our knowledge, this kind of hybrid approach has not been explored yet in the literature in the context of feature selection and malware detection. Knowledge about the intrinsic characteristics of malicious activities is determined by the API call statistics which is injected as a filter score into the wrapper's backward elimination process in order to find the most significant APIs. While using the most significant APIs in the wrapper classification on both obfuscated and benign types malware datasets, the results show that the proposed hybrid framework clearly surpasses the existing models including the independent filters and wrappers using only a very compact set of significant APIs. The performances of the proposed and existing models have further been compared using binary logistic regression. Various goodness of fit comparison criteria such as Chi Square, Akaike's Information Criterion (AIC) and Receiver Operating Characteristic Curve ROC are deployed to identify the best performing models. Experimental outcomes based on the above criteria also show that the proposed hybrid framework outperforms other existing models of signature types including independent wrapper and filter approaches to identify malware.

* Corresponding author. Tel.: +61 353276217.
E-mail addresses: s.huda@ballarat.edu.au (S. Huda), jemal.abawajy@deakin.edu.au (J. Abawajy), mamoun.alazab@anu.edu.au (M. Alazab), mali.abdollahian@rmit.edu.au (M. Abdollalihian), mislam@csu.edu.au (R. Islam), j.yearwood@federation.edu.au (J. Yearwood).

http://dx.doi.org/10.1016/j.future.2014.06.001
0167-739X/

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

1. Introduction

Malicious software (Malware) affects the secrecy and integrity of data as well as the control flow and functionality of a computer system which we combat every day [1]. There is no single technique [2–5], but most Anti-Virus (AV) engines use two main approaches: (1) signature-based and (2) anomaly-based approaches for malware detection. The signature-based detection [6,7] methods are very efficient to detect known malware [7]. However, the signature generation process for construction of the database for the AV engine involves manual processing and requires strict code analysis. Most of the malwares [5] have in-built process that can generate new variants each time it is executed and a new signature is generated. Therefore, signature based approaches fail to detect unknown malwares [5] which are not in the database. In contrast, anomaly-based detection approaches [7,8] use API call sequences instead of byte sequence matching through optimal sequence alignment. Although anomaly-based detection approaches use the knowledge of normal behavior patterns and perform better than the signature based approach. But these approaches [7,8] ignore the frequency of API calls in the sequences and suffer from the same problem as normal signature approaches and become similar to signature based approach resulting in a more false positives outcome [9]. Windows Application Program Interface (API) function calls [10–12,10] have been used in statistical N-gram modeling techniques [11,12] for detection. However these approaches [11,12] use simple wrapper classification methods [13] which did not explore the ways of selecting the best set of APIs from a large set of APIs. To find an optimal subset of API that can discriminate malware from benign is essential and difficult which also can be transformed into a feature selection problem. Usually given an m-dimensional API based malware dataset, a detection algorithm needs to find an optimal API subset from the 2^m subsets of the APIs. Therefore finding an optimal API subset is computationally expensive [14] feature selection problem. The performance of a detection algorithm depends on its evaluation criterion as well as search strategies.

The filter based models for best subset selection [15] are computationally cheap due to its evaluation criteria. However, feature subsets selected by filter may result in poor prediction accuracies, since they are independent from the induction algorithm. In contrast, the wrapper models [16] face huge computational overhead due to the use of the induction algorithm's performance criteria as its evaluation criteria. Some researchers have proposed [17] hybrid of genetic algorithm (GA) and filter heuristic where GA framework works as a subset generation process and filter heuristic improves local search. Despite significant researches on evaluation criteria and search strategies, current generation feature selection approaches lack the work that can combine the merits of wrapper and filter approaches. To the best of our knowledge, there is no complete malware literature that reveals with a suitable approach to find the most significant set of APIs from enormous number of API sets and can exploit the merits of both wrapper and filter approaches. This shows a clear and strong motivation for this work in the context of API feature selection for malware detection.

In this paper, we propose a framework that attempts to identify malware by using its malicious activities characterized by the Application Program Interface (API) calls and a novel hybrid wrapper–Filter feature selections techniques. At first, a large number of malware datasets with obfuscated and unknown malware are collected from many sources including the honeynet project, VX heavens [18]. The hybrid frame work proposes a novel automated method to extract the API call behaviors from malware dataset using sophisticated unpacking, disassembling and mapping analysis techniques. Then we propose two hybrid approaches using the hybrids of Support Vector Machines Wrapper heuristic and Maximum-Relevance–Minimum-Redundancy Filter heuristics for malware detection from the API call statistics. The novelty of our proposed malware detection approaches is that these techniques combine the knowledge about the intrinsic nature of malicious activities of the malware with the wrapper score in order to select the most significant set of API features. This is achieved by injecting the filter's ranking score (computed using the intrinsic characteristics from API call statistics) in the wrapper selection process and different search strategies. We have also used binary logistic regression to compare and assess the efficacy of the proposed approaches based on different goodness of fit criteria. Our contribution also includes the following hitherto unreported in the literature:

(1) Development of a fully automated framework for malware detection to compute API call statistics from malware and benign programs.
(2) Development of two novel hybrid API feature selection approaches based on the hybrid of Support Vector machine wrapper heuristics and maximum-relevance–minimum-redundancy filter heuristics that can find an optimal set of APIs in order to detect the malware from their malicious behavior.

The rest of the paper is organized as follows. The next section introduces some related background literature and limitations of current malware detection techniques. Section 3 discusses Filter and wrapper approaches and a mathematical derivation for wrapper heuristic based on Support Vector Machine (SVM). The proposed framework for malware detection using hybrids of Support Vector Machine wrapper heuristics and Maximum-Relevance–Minimum-Redundancy filter heuristics with API Call statistics has been described in Section 4. Section 5 describes the malware datasets. Section 6 presents experimental results, statistical validation and discussion about the results. Conclusions of this study are presented in the last section.

2. Related work

2.1. Code obfuscations and current malware detection approaches

Code obfuscation modifies the program code to produce offspring copies which have the same functionality with different byte sequence so that the new code is not recognized by antivirus scanner. Obfuscation techniques such as, packing [19] is used by malware authors as well as legitimate software developers in order to compress and encrypt the Portable Executable (PE) or Dynamic Link Library (DLL) in secondary memory for changing the byte sequence in the PE. This results different byte sequences in the newly produced packed PE. A second technique, polymorphism [19] uses encryption and data appending/data pre-pending in order to change the body of the malware. It also changes decryption routines from one infection to another as the encryption keys change. Finally, metamorphism [20] is used to transform the code without encryption in order to evade detection by static signature-based virus scanners. Several works [21] propose to use program graph mining techniques for combating (polymorphic) malwares. However, these works either employ subgraph matching or vector-space modeling to learn classifiers for malware detection. These methods are either not scalable (e.g., subgraph matching) or not adaptable to dynamic feature space such as API. Sung et al. [8] present a signature-based malware detection technique, with emphasis on detecting obfuscated (or polymorphic) malware and mutated (or metamorphic) malware. Tian et al. [22] present a method for classifying Trojans based on function lengths, and show that function length plays an important role in classifying malware and if combined with other features may result in a better method of malware classification. Signatures matching techniques in [21,22,8,20] to detect malware requires that signatures to be generated

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

by human experts by disassembling the file and selecting pieces of unique code. Therefore, Anti-malware scanners will not be able to detect the malware created by code obfuscation techniques as they generate new signatures each time they are executed which may not exist in AV engine database. Hence, there is a need to build signature-free methods.

2.2. Windows API calls for malware detection

In the Windows operating system, user applications rely on the API interface within a set of libraries, such as KERNEL32.DLL, NTDLL.DLL and USER32.DLL in order to access system resources including files, processes, network information and the registry. Various features related to the API calls are loaded by the malware before the actual executions occur. In the literature, Bailey et al. [23] propose a method for behavioral-patterns (e.g., files written, processes created) of malwares into groups that reflect similar classes of behaviors. Ahmed et al. [24] propose a technique by using statistical features which are extracted from spatial (arguments) information available in Windows API calls. Sami et al. [25] also propose an approach for detecting malwares based on mining API calls from PE-files. Lu et al. [26] propose a hierarchical framework to classify the network into different application communities based on payload signatures and a new cross-association clustering algorithm, and then analyze the frequent characteristics of flows to distinguish malicious channels created by bots from normal traffic generated by human beings. Tian et al. [27] propose an approach to distinguish Trojan and virus families based on strings from library code. Sung et al. [8] propose an anomaly based detection approach using API call sequences through the similarity measure techniques between sequences including Euclidian distance, Sequence alignment. Optimal sequence alignment proposed by Sung et al. [8] is similar to a signature based approach since it ignores the frequency of API call which may fail to detect unknown malware as other approaches. Kolter and Maloof [28] tested several classifiers including, IBk, naive Bayes, decision trees, boosted naive Bayes with API call sequence. Monitoring a large number of APIs for detecting malware is one of the most crucial problems in the antivirus engines. Therefore, it is essential to develop an approach in order to find a compact and significant set of APIs from a set of the large number of APIs.

3. Filter and wrapper approaches for malware detection

Filter approaches [15,3] use API statistics as the training data. A subset generation process with empty or full API set is used to generate the subsets. The generated subsets are evaluated using filter heuristics such as co-relation measure, Principal Component Analysis and Mutual Information [15,3]. The final subset is justified using a wrapper classification algorithm. In the wrapper approach, subsets are evaluated by the predictive accuracies of a trained classifier. Therefore wrapper approaches such as Support Vector Machine (SVM) [29,28,8], Artificial Neural Networks (ANN) [30,28] are more significant than the filter approaches. Adaptive Neuro-Fuzzy Inference Systems (ANFIS) [31,32] also can be used for wrapper. However SVM performs better than ANFIS [32] and ANFIS does not provide any feature reduction heuristics [31]. Different search strategies [33] such as sequential backward elimination (SBE), sequential forward elimination (SFE) [33] or bidirectional search approaches are used in the subset generation process. In the worst cases, growth of subset generation in the wrapper may increase in the order $O(2^m)$ as the dimension of dataset m increases. In most cases, the algorithm is trained repeatedly with the training data for each subset of APIs. This makes the algorithms computationally very expensive. Comparisons of different search strategies have been made in [33]. However, the subsets in the wrapper approaches are evaluated by the predictive accuracies of the trained wrapper and therefore are more significant than those in the filter approaches which only depend on APIs redundancy or relevance. To the best of our knowledge, none of the above approaches [30,28,8,27,21,22,20] used either the wrapper heuristics or hybrid heuristics in order to determine the most suitable set of APIs from a very large set of APIs for malware detection. This shows a clear literature gap in the context of API selection for malware detection.

3.1. Support vector machine (SVM) based wrapper heuristics

Support Vector Machine (SVM) is proposed by Vapnik et al. [29] and known as a popular wrapper classifier. Let us denote the training examples and its corresponding class label pair as (x_i, y_i) where $y_i \in C$ and the set of discrete values for classes: $C = \{c_1, c_2, c_3 \ldots c_m\}$. Consider a binary classification problem where a positive training example is denoted as $+1$ and a negative example is denoted as a -1. If the set of positive examples is P_+ and the set of negative examples is P_-, for a linearly separable case we can find a linear discriminant plane as mentioned in Fig. 1 which is furthest from both positive and negative example sets $\{P_+$ and $P_-\}$. The planes in Fig. 1 which are on the border lines of both sets are called supporting planes and can be defined as below.

$$\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \approx (constant). \tag{1}$$

The points which are on these supporting planes can be defined as the support vectors. The aim is to find $\mathbf{W}$ and b such that $\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \geq k$ for positive examples where $k = \min_i |\mathbf{x}_i + b|$ and $\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \leq k$ for negative examples for $\forall x_i$. Then for positive examples, the supporting plane is

$$\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \geq k \tag{2}$$

for negative examples, the supporting plane is

$$\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \leq -k. \tag{3}$$

After normalizing and re-scaling, these can be written as

$$\begin{aligned} &\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \geq +1; \quad \forall y_i \in P_+ \\ &\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b \leq -1; \quad \forall y_i \in P_-. \end{aligned} \tag{4}$$

The above equations can be simplified as

$$y_i(\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b) \geq +1. \tag{5}$$

The geometric distance between these two planes is: $2/\|\mathbf{W}\|^2$. For better discrimination we want to maximize the distance between these two planes, therefore this can be transformed into a constraint-based maximization problem as below:

$$\max_{\mathbf{w},b} 2/\|\mathbf{W}\|^2 \tag{6}$$

which is bounded by the constraints: $y_i(\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b) \geq 1$. For linearly inseparable cases, a penalty based objective function is considered with an error variable which is also added to each constraint and then the optimization problem is transformed into a minimization problem

$$\min_{\mathbf{w},b} (1/2)\|\mathbf{W}\|^2 + \varpi \sum_{i=1}^{D} \xi_i \tag{7}$$

which is bounded by the constraints: $y_i(\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b) + \xi_i \geq 1$, $\xi_i \geq 0$ and ϖ is a constant. By using lagrangian Multipliers γ_i we get the primal lagrangian function for optimization as below:

$$(1/2)\langle \mathbf{W} \cdot \mathbf{W} \rangle - \sum_{i=1}^{D} \gamma_i [y_i(\langle \mathbf{W} \cdot \mathbf{x}_i \rangle + b) - 1]. \tag{8}$$

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

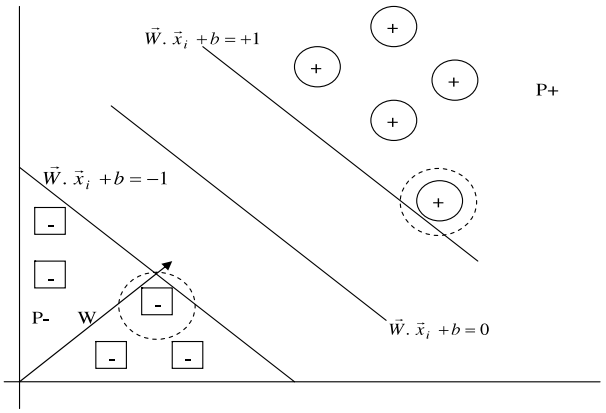


Fig. 1. Support Vector machine based wrapper heuristics.

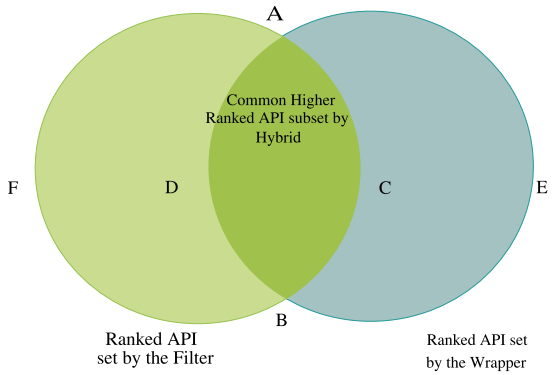


Fig. 2. Venn diagram for combined heuristics.

Taking partial derivative of (8) with respect to $\mathbf{W}$, b, γ_i and equating the results to zero, we get

$$\mathbf{W} = \sum_{i=1}^{D} y_i \gamma_i \mathbf{x}_i \quad \text{and} \quad \sum_{i=1}^{D} y_i \gamma_i = 0. \tag{9}$$

The above results can be used in primal lagrangian function (8) which gives corresponding optimization function for a dual problem as below:

$$\sum_{i=1}^{D} \gamma_i - (1/2) \sum_{i,j=1}^{D} y_i y_j \gamma_i \gamma_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \tag{10}$$

subject to the constraints: $\sum_{i=1}^{D} y_i \gamma_i = 0$ and $\gamma_i \geq 0$. The partial derivatives with respect to the lagrange multipliers will be zero at the extreme. The resulting decision hyperplane is obtained by using lagrange Multipliers based solution of (10) and is described as below:

$$\mathbf{W} = \sum_{i=1}^{D} y_i \gamma_i \mathbf{x}_i. \tag{11}$$

This shows that $\mathbf{W}$ is a function of the training vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_D$ where $\mathbf{x}_D = \mathbf{x}_{1D}, \mathbf{x}_{2D} \ldots, \mathbf{x}_{mD}$ and $m =$ Total number of features. The influence of a feature on the resulting hyperplane can be evaluated by taking the partial derivatives of (6) or (7) with respect to $\mathbf{x}_{ji}$ where $i = 1, 2, \ldots, D$ and $j = 1, 2, \ldots, m$ as below:

$$\sum_{i} \left| \frac{\partial \|\mathbf{W}\|^2}{\partial \mathbf{x}_{ji}} \right| = \tau |\mathbf{w}_j| \tag{12}$$

where τ is a constant. The above derivation (12) shows that feature with higher $|\mathbf{w}_j|$ has more influence in determining the width of the margin of the resulting hyperplane. However wider margin of the resulting hyperplane ensures less number of training examples falling on the wrong side of the hyperplane. This reduces the misclassification rate. It concludes that a significant feature for better discrimination will have a higher $|\mathbf{w}_j|$.

4. Proposed methodology: a hybrid wrapper–filter based framework for malware detection

4.1. Motivation for hybridization

Filter approaches can extract knowledge of the intrinsic pattern from real data. However filter approaches [15,3] do not use any performance criteria based on predictive accuracies. Subsequently, there is no guarantee that the final subset of API features makes a better prediction and would be the most informative subset for malware detection. In contrast, the wrapper approaches [16] use a predetermined induction algorithm to find the most informative set of API features. Use of the predictive-accuracy based evaluation criteria in the wrapper ensures optimal performance from the selected API subset. However repeated execution of the induction algorithm (in the worst case an exponential search space) in the search process incurs a high computational cost in the wrapper approach. In this paper, we propose two hybrid approaches using the Support Vector Machines Wrapper (SVM) heuristic and Maximum-Relevance–Minimum-Redundancy Filter heuristics. The proposed hybrid approaches introduce a filter heuristic in the wrapper stage that combine the complementary properties from both approaches. The knowledge about the intrinsic characteristics of malicious activities of the malware is computed by the filter approaches using the API call statistics. This filter heuristic score is injected into the wrapper backward elimination process and hybridized with the wrapper heuristics which can take advantages from both approaches. Therefore the hybrids can find more significant API features than either wrapper or filter alone. The idea behind these approaches can be explained by the Venn-diagram (Fig. 2). If the two subsets (*ACBF* and *ADBE*) of the APIs are separately ordered/ranked according to their score, then the common higher ranked subset (ACBD) is the most significant subset recommended by both algorithms. If the scores of both algorithms are normalized on the same scale and combined, then the subsets with the higher combined scores provide the common higher ranked subsets. A Backward Elimination (BE) search strategy based on the combined score along with the wrapper evaluation criteria can determine the most significant API features. The performance of the combined score can be affected by the performance of the incorporated filter for a particular wrapper approach in the hybrid. However, different filter approaches can be combined to find a suitable hybrid for a particular wrapper heuristic and vice-versa. In the proposed method, we have combined mutual information based Maximum Relevance and Minimum Redundancy filter heuristics with SVM based wrapper heuristics. We will use other wrapper approaches in future work. The following sub-sections describe the different heuristics and steps of the proposed hybrid framework.

4.2. Extraction of application program interface (API) statistics

To extract the API call lists from Portable Executables (PE), a Python language based automated system has been developed with the three main steps as: (1) Unpack and Disassemble the binary executable, (2) Extract API calls and important machine-code features from the disassembly program and (3) Map the API calls with MSDN library and analyze the malicious behavior and prepare the API call list statistics. These steps have been presented in part-1:feature extraction of Fig. 3.

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

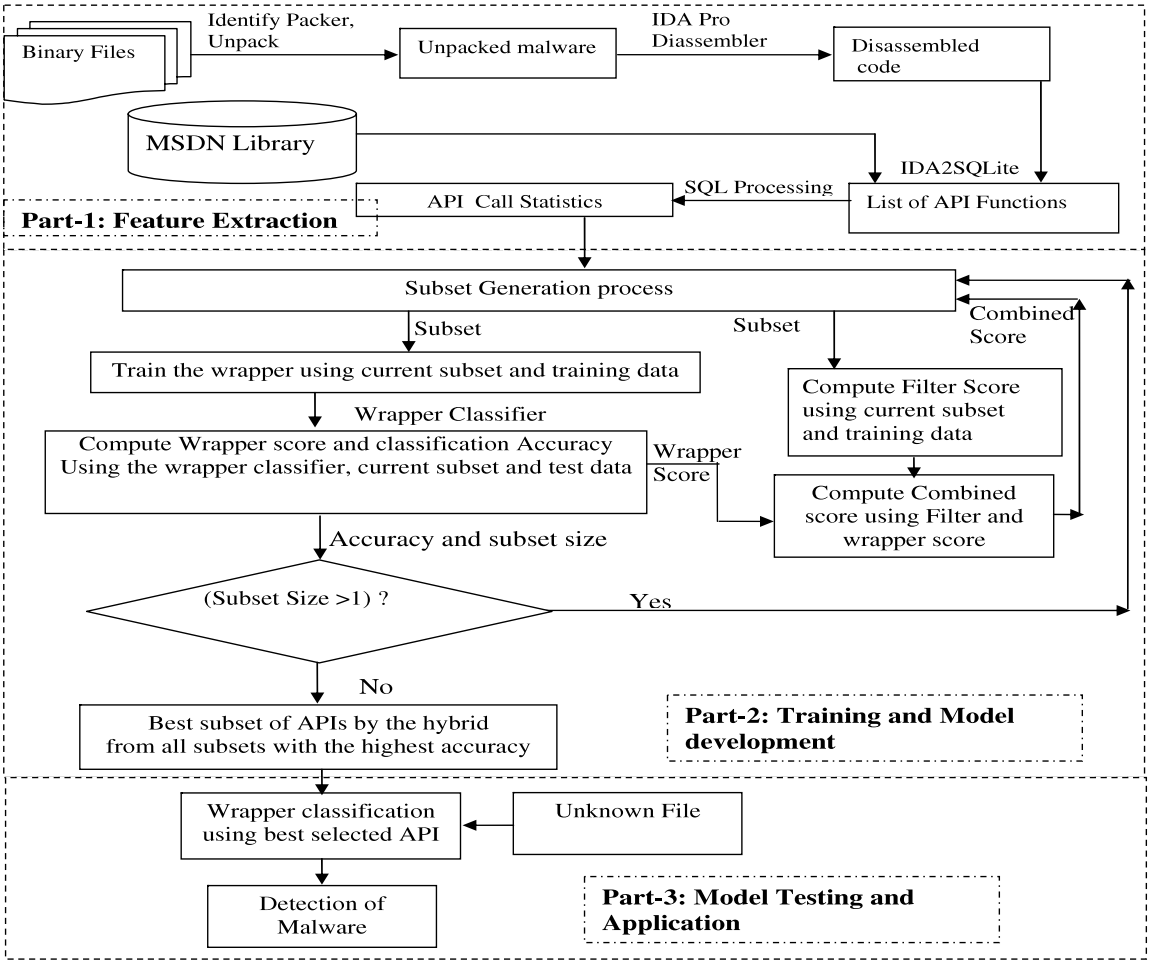


Fig. 3. A hybrid Wrapper–Filter based framework for Malware detection.

4.2.1. Unpack and disassemble the binary executable

Researchers have been trying to build semi-automated tools for automatically unpacking malware, such as PolyUnpack [34], Renovo [35], OmniUnpack [36] and Eureka [37]. PolyUnpack extracts the hidden code through process execution and uses the Windows debugging API to single-step. Renovo supports multiple layers of unpacking. However, OmniUnpack [36] uses a coarse-grained execution tracking approach at the page-level protection mechanism available in hardware. Eureka, is similar to OmniUnpack except that Eureka tracks execution at the system call level. By studying these semi-automated tools, we observe that none of them are completely meeting the purpose of analyzing the behavior of malware by extracting API call features. All dataset files collected are pre-processed for anomaly testing. In order to translate a program into an equivalent high-level-language program based on the binary content, a disassembly tool is used in this paper for static analysis including interactive Disassembler Pro (IDA Pro) [38] since it can disassemble all types of non-executable and executable files (such as ELF, EXE, and PE). Also, we have selected the IDA Pro as a component of the automation process that automatically recognizes API calls for various compilers and can be further extended with our Python programs and compiled plugins, resulting in incredibly powerful implementation with flexible levels of analysis and control. IDA Pro loads the selected file into memory to analyze the relevant program portion to create an IDA database whose components are stored in four files:.id0 that contains the content of a B-tree-style database,.id1 that contains flags describing each program byte,.nam that contains index information related to program locations, and.til that is used to store information concerning local type definitions to a given database. IDA Pro generates the IDA database files into a single IDB file (.idb) by disassembling and analyzing the binary of the file. Our fully-automated system using Python programming language generates.idb automatically from the set of malware samples.

4.2.2. Extraction of API calls

IDA Pro [38] provides access to its internal resources via an API that allows users to create plug-ins to be executed by IDA Pro [38]. We have used SQLite [39], a software library that implements a self-contained transactional SQL database engine. Our Python system automatically runs and creates the plugin to use SQLite [39] with IDA Pro for generating the database (.db). We have developed an interface for accessing the database file (.db) so that the results from the assembly code of the malware stored in the database can be used for better binary analysis as in Fig. 3. IDASQLitplugin [38] generates eight tables (Blocks, Functions, Instructions, Names, Maps, Stacks, Segments, Target Binaries), each of them contains different information about the binary content. Function table contains all the recognizable API system calls and non-recognizable function names and the length (start and the end location of each function). Instructions table contains all the operation code (OP) and their addresses and block addresses. Maps table contains the function address and source of block address and the destination of the function address. Names table contains function addresses, the name of the function and the type of the function. Stacks table contains function address, the stack name, and the start and the end address. Segments table contains information that describes

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

each segment in an executable file, segment name (Code, Data, BSS, *_idata*, *_tls*, *_rdata*, *_reloc*, and *_rsrc*) and the segment length. Finally, target binaries contain the file name, path name, MD5, and start and the end of analyses.

4.2.3. API call mapping and feature analysis

We downloaded the Windows APIs from the Microsoft Developer Network (MSDN) [40] and implemented in Python the required processes to match the API from MSDN and the API calls generated in the database (.db) for the malware sample sets. In addition, to list all the API calls that are associated with malcode and to analyze the features, we have considered the machine opcodes such as Jump and Call operations as well as the function type (import or function). For the analysis of malware behavior, we have considered features such as frequency of call, call sequence pattern and actions immediately preceding or after call. Some actions that lead to invalid memory reference or undefined register or invalid jump target help in refining the extracted features for analysis. This develops a fully-automated system that integrates well with IDA Pro and SQlite [39] using Python programming to perform all the three steps described and presented in Fig. 3.

4.3. Maximum relevance and minimum redundancy (MRMR)

Relevant features can provide more information about the class variable than irrelevant features [15,3]. Therefore mutual information based maximum relevance (MR) [15,3] is a suitable heuristic for selecting the most relevant APIs. If S is a set of APIs; $\{F_i|F_i \in S : i = 1, 2, 3 \ldots\}$ and Malware class variable is c, the maximum relevance (MR) can be defined as (13). Here c denotes class values of a particular sample.

$$\textit{Maximum Relevance } (F_i, c) = \frac{1}{|S|} \sum_{F_i \in S} I(F_i; c) \tag{13}$$

$I(F_i; c)$ is the mutual information between F_i and class c which is defined as

$$I(F_i; c) = H(F_i) - H(F_i|c) \tag{14}$$

$H(F_i)$ is the entropy of F_i with the probability density function p. If F_i takes discrete values from set of values $V = v_1, v_2, v_3 \ldots v_l$, then,

$$H(F_i) = -\sum_{v_l \in V} p(v_l) \log(p(v_l)). \tag{15}$$

Let $H(F_i|c)$ be the conditional entropy between F_i and c then,

$$H(F_i|c) = -\sum_{v_l \in V} \sum_{c_m \in C} p(v_l, c_m) \log(p(c_m|v_l)) \tag{16}$$

where class variable c takes the discrete values from the set C. Maximum relevance (MR) [15] can select features that are highly relevant to class. However MR may contribute to redundancy. When two features are highly dependent on each other, the corresponding class discriminative ability of the two features would not be affected much if one of them were removed. Therefore, to avoid the redundancy in MR, a redundancy function is incorporated as:

$$\textit{Minimum Redundancy } (F_i, c) = \frac{1}{|S|^2} \sum_{F_i, F_j \in S} I(F_i; F_j), \tag{17}$$

where $I(F_i; F_j)$ is the mutual information between the features: F_i and F_j.

4.4. Hybrid of maximum relevance (MR) and SVM score (MR–SVMS)

The proposed MR–SVMS uses Support Vector Machine as the classification algorithm in the wrapper stage. The detail steps of computing hybrid score has been described in part-2:Training and model development of Fig. 3 and Algorithm 1. An n-fold cross-validation approach has been used in MR–SVMS to train the wrapper. In each fold, we compute the SVM score for every API by (11). Then after training of all folds, the SVM score is averaged as (18):

$$SVMS(F_i)_{average} = \frac{1}{n}(SVMS(F_i)_1 + SVMS(F_i)_2 + \cdots + SVMS(F_i)_n). \tag{18}$$

While computing the combined score in the proposed MR–SVMS, the relevance of APIs in the current subset is computed from the individual score which is scaled to the maximum individual relevance of the subset. Thus relevance of an API in a subset in the hybrid approach is defined by

$$\textit{Relevance } (F_i) = \frac{I(F_i; c)}{\max_{F_i \in S} I(F_i; c)}. \tag{19}$$

The combined score of the filter's and wrapper's heuristic in the proposed MR–SVMS is computed by

$$\textit{Combined Score } : MR\_SVMS(F_i) = \frac{I(F_i; c)}{\max_{F_i \in S} I(F_i; c)} + SVMS(F_i)_{average}. \tag{20}$$

Algorithm 1 Hybrid Wrapper–Filter MR–SVMS or MRMR–SVMS approach for Malware detection

```
input← D(F1, F2, ...Fm) Training data with m APIs
output← S_BEST an optimal subset of APIs
begin
1. Let S ←whole set of m APIs F1, F2, ...Fm
2. Let S0 ←Initial set of APIs which records all
generated subsets with corresponding accuracies
of subset
//Apply a backward elimination (BE) search
strategy
for N = 1 to m − 1 do
  4. Current set of APIs S_current ← S
  5. Compute filter score by (19) or (21)
  for fold = 1 to n do
    7. Train the SVM with feature set S_current
    8. Compute SVM score of all APIs
    9. Compute Accuracy
  end for
  10. Compute average accuracy of all folds for
  S_current
  11. Compute average SVM score of S_current by (18)

  12. Compute combined score for every API in
  S_current by (20) to (22) for hybrid MR--SVMS
  13. Rank the APIs in S_current using combined score
  in descending order
  14. S0 ← S0 ∪ S_current
  16. Update the current API set S_current by
  removing the API with lowest score
end for
17. S_BEST ←Find the subset from S0 withe highest
accuracy
18. Return S_BEST
end
```

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

4.5. Hybrid of maximum relevance–minimum redundancy (MRMR) and SVM score (MRMR–SVMS)

Similar to MR–SVMS, MRMR–SVMS uses SVM as the classification algorithm in the wrapper stage. An n-fold cross-validation approach has been used in MRMR–SVMS to train the wrapper. In each fold, we compute the SVM score for every API by (11). Then after training of all folds, the SVM score is averaged as (18). The detail steps of computing MRMR–SVMS score has been described in part-2:Training and model development of Fig. 3 and Algorithm 1. An incremental search method [41] is used to compute the Maximum Relevance and Minimum Redundancy (MRMR) score as below. Maximum Relevance and Minimum Redundancy score is the difference of maximum relevance score of a candidate features in the candidate set and redundancy score between the corresponding feature with a feature in the goal set.

$$MRMR(F_i, c) = \max_{F_i \in S - S_{l-1}} \left[\frac{1}{|S|} \sum_{F_i \in S} I(F_i\ c) - \frac{1}{l-1} \sum_{F_j \in S_{l-1}} I(F_i\ F_j) \right]. \quad (21)$$

Since the Maximum Relevance–Minimum Redundancy (MRMR) score is a difference of feature score which is relative to the search iteration, while computing the combined score in the hybrid, an equivalent weighted score of MRMR score is computed for each feature. The features are ordered according to their ranks in the MRMR incremental search method [41]. Then equivalent weighted score is computed from their ranking on a unity scale. Orders of the feature ranking are incremental integers starting from one to total number of features in the dataset (see Table 1) where top-ranked has a maximum score of one. The combined score of the filter's and wrapper's heuristic in the proposed MRMR–SVMS is computed as:

$$Combined\ Score\ : MRMR\_SVMS(F_i) = Scaled\ MRMR(F_i, c) + SVMS(F_i)_{average}. \quad (22)$$

4.6. Search strategies and subset generation in MR–SVMS/MRMR–SVMS

The hybrid approach uses a Backward Elimination (BE) search strategy to generate a subset of APIs. The detail steps of BE process has been presented in Algorithm 1. Initially hybrid starts with the full set. Subset generation in BE is guided by the wrapper–filter hybrid heuristic score. The combined score computation follows the steps of Sections 4.3 and 4.4. When the number of APIs in BE process is significantly reduced compared to the total, the filter score component is weighted less than the wrapper score as:

$$MR\_SVMS(F_i)\_final = \left(u * \frac{I(F_i; c)}{\max_{F_i \in S} I(F_i; c)} \right) + \left(v * SVMS(F_i)_{average} \right) \quad (23)$$

$$MRMR\_SVMS(F_i)\_final = \left(u * MRMR\_SVMS(F_i) \right) + \left(v * SVMS(F_i)_{average} \right) \quad (24)$$

where $1 \leq u, v \geq 0$.

4.7. Wrapper step in MR–SVMS

The proposed hybrids (MRMR–SVMS and MR–SVMS) use a support vector machine in the wrapper stage. An n-fold cross validation approach has been applied in the training. The evaluation criterion of API subset is based on the average prediction accuracy over n-fold of the wrapper. In Algorithm 1, steps-(1–10) compute the average accuracy over n-folds for the current subset of APIs. Steps-(11–13) compute the hybrid score and ranks the APIs based on their combined score. Steps-(14–16) generate new subset based on the APIs ranking and keep the records of evaluated API subsets and their respective accuracy. The BE processes in both MRMR–SVMS and MR–SVMS update the scores of MRMR, MR and SVM as well as the combined score in every iteration. The combined score guides the subset generation. The BE continues until the number of APIs in the current subset is reduced to one. The subset with highest accuracies or close to the highest accuracies with a fewer APIs is chosen as the final subset.

Table 1
Dataset description.

Type	Qty	Max. Size (KB)	Min. Size (KB)	Avg. Size (KB)
Benign	15,480	109,850	0.8	32,039
Virus	17,509	546	1.9	142
Worm	10,403	13,688	1.6	860
Rootkit	270	570	2.8	380
Backdoor	6,689	1,299	2.4	685
Constructor	1,039	77,662	0.9	1,193
Exploit	1,207	22,746	0.5	375
Flooder	905	16,709	1	1,397
Trojan	13,201	17,810	0.7	1,819

4.8. Evaluating goodness of fit of the model

The final models have been verified using statistical binary logistic regression techniques [42], chi-square [43] and Akaike's Information Criterion (AIC) [44]. In logistic regression analysis, deviance is used in lieu of sum of squares calculations [42]. Deviance (D) is a measure of the lack of fit to the data and is calculated by comparing a given model with the saturated model — a model with a theoretically perfect fit [43]. This computation is called the likelihood-ratio test and is defined by

$$D = -2 \ln \frac{\text{likelihood of fitted model}}{\text{likelihood of the saturated model}}. \quad (25)$$

The results of the likelihood ratio (the ratio of the fitted model to the saturated model) will produce a negative value, so the product is multiplied by negative two times its natural logarithm to produce a D value with an approximate chi-squared distribution [43]. Smaller D value (leading to a smaller chi-square statistics) indicate a better fit as the fitted model deviates less from the saturated model. When assessed upon a chi-square distribution, non significant chi-square values indicate very little unexplained variance and thus, good model fit. Conversely, a significant chi-square value indicates that a significant amount of the variance is unexplained.

The Akaike's Information Criterion (AIC) [44] was first introduced by [44] to measure a model fitting accuracy. For the logistic regression model, it is defined by:

$$AIC = -2 * \log\ (likelihood) + 2 * k \quad (26)$$

where k is the number of estimated parameters. In itself, the value of the AIC for a given dataset has no meaning. It becomes interesting when it is compared to the AIC of a series of models specified a priori, the model with the lowest AIC being the 'best' model among all models specified for the data as they reflect a trade-off between the lack of fit and the number of parameters in the model.

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

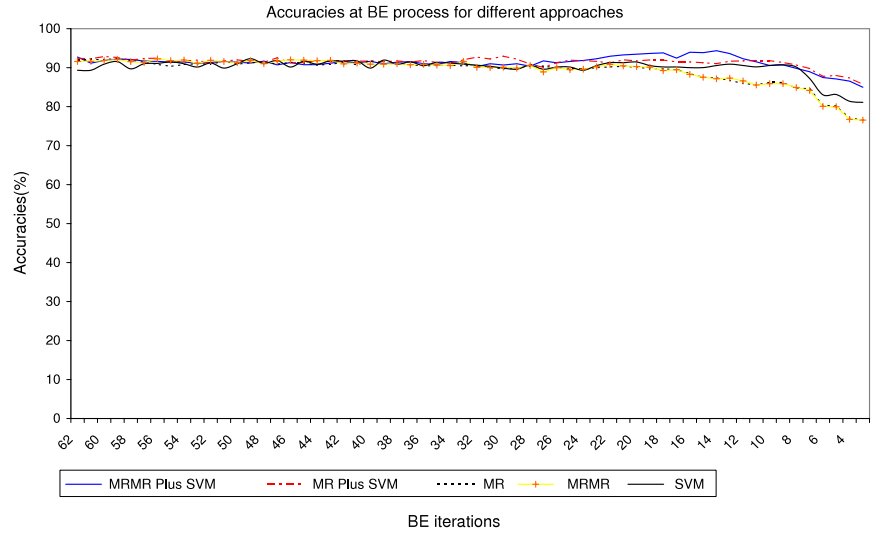


Fig. 4. Accuracies in (%) at different iterations of BE process for SVM, MRMR, MR, MR with SVM and MRMR with SVM.

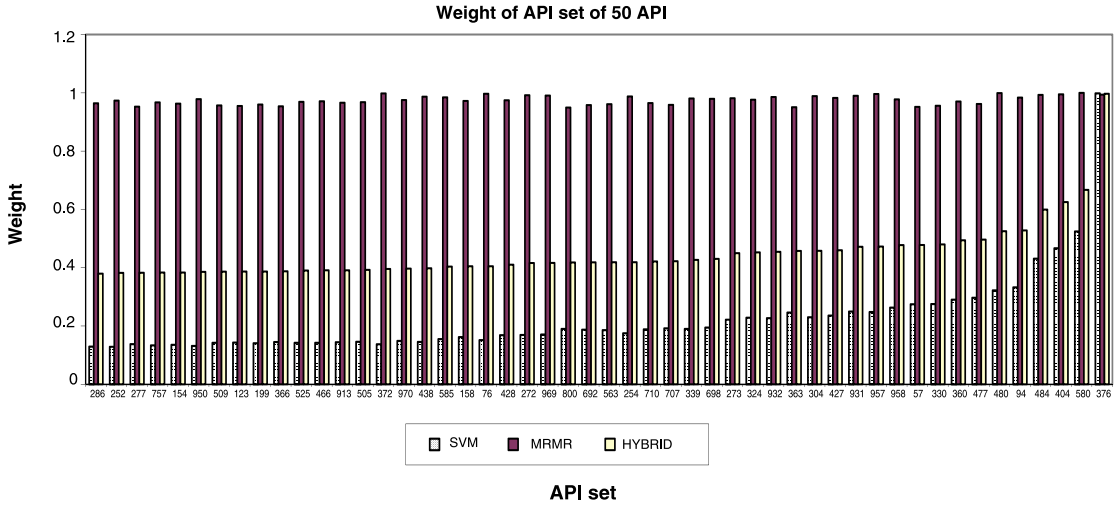


Fig. 5. API selection procedure in the hybrid using the score of API set with 50 API.

5. Dataset

We have gathered 66,703 executable files in total consisting of 51,223 recent Malware datasets and the remaining being benign datasets as shown in Table 1. Such large malware datasets with obfuscated and unknown malware used in this paper have been collected from honeynet project and VX heavens [18]. The API call statistics was prepared by following feature generation procedure described in Fig. 3. The 15,480 benign datasets include: application software such as Databases, educational software, mathematical software, image editing, spreadsheet, word processing, decision making software, internet Browser, email and system related software and programming language software. Both (Malware and Benign) have been uniquely named according to their MD5 hash value.

6. Experimental results and discussion

The proposed approaches (MR–SVMS and MRMR–SVMS) are tested using a 10-fold cross validation and are executed for 10-trials. Then the results are compared with independent filters (MR and MRMR) and wrapper (SVM) which are also executed for 10-fold cross validation and are executed for 10-trials. In the BE process, 2/3 of the iterations uses ($u = u' = 1$) and the last 1/3 of the iterations uses ($u = 0.3, u' = 0.7$). The average accuracies from 10 trials were considered for the assessment of the final accuracies which are summarized in Fig. 4. The detail are given in the Appendix in Table 4. As shown in the Appendix in Table 4, the wrapper approach, SVM, achieves an accuracy of (96.84%) based on all APIs, the filter approaches (MRMR and MR) achieve accuracy (96.13% and 95.98%) accordingly. The hybrids of wrapper and filter (MRMR–SVMS and MR–SVMS) start with 972 APIs where the accuracies (96.5% and 96.8%) are achieved. Table 4 in the Appendix provides accuracies for successive BE iterations for all algorithms. At each iteration, the scores of APIs for filter and wrapper are computed and then hybrid score is computed. BE process generates a total of 972 subsets only instead of 2^{972} for the worst case. Some of the sample comparative score graphs for subsets are presented here to demonstrate the selection process of API subset in the BE of the proposed hybrid approaches. In Fig. 5, a score chart for the set with 50 APIs is presented. Fig. 5 shows that wrapper-SVM provides lowest score for API-252, whereas Filter-MRMR provides lowest score for API-800. However hybrid MRMR–SVMS computes lowest score for API-286. Therefore, the hybrid eliminates the API-286 at the 50th iteration. In the next iteration of BE process for MRMR–SVMS, Fig. 6, the hybrid re-computes all scores and shows that the API-950 attains the lowest score for SVM, the API-800 attains the lowest score for MRMR. The API-757 has the lowest

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

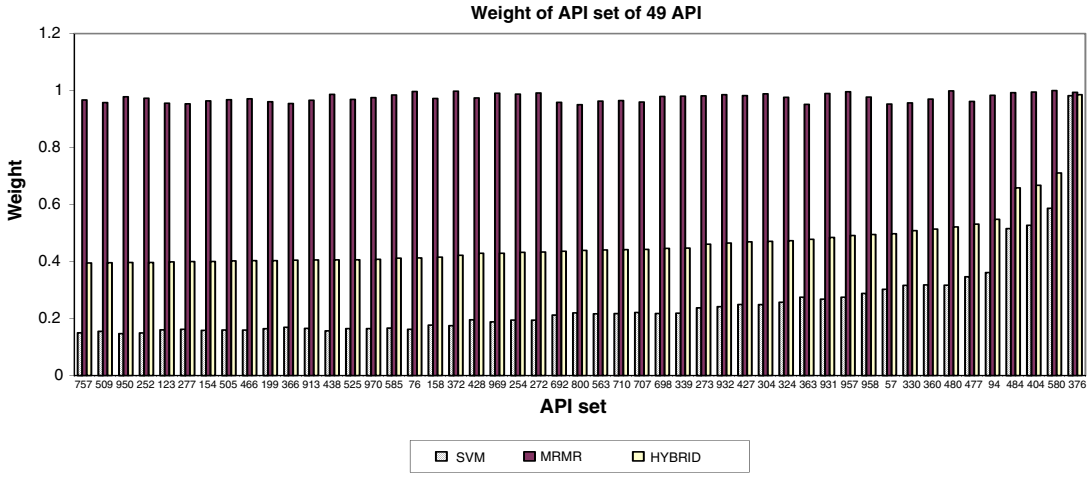
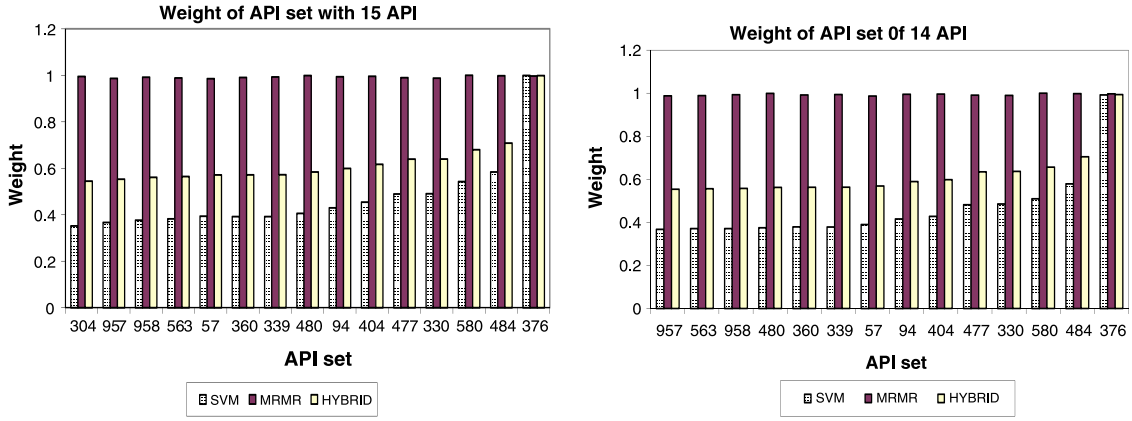
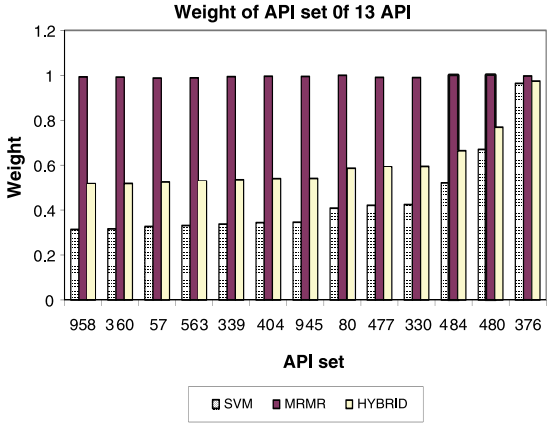


Fig. 6. API selection procedure in the hybrid using the score of API set with 49 API.



(a) 15 API.

(b) 14 API.



(c) 13 API.

Fig. 7. API selection procedure in the hybrid using the score of API set with different number of APIs.

combined score. Therefore the MRMR–SVMS eliminates API-757 in this iteration. The BE process in the hybrid (MRMR–SVMS) continues. The subset generation for APIs at different stages have been more presented in (Fig. 7(a)–(c)). The accuracies for different subset of APIs for (MRMR–SVMS) have been presented in the Appendix in Table 4 and also in Fig. 4 and the receiver operating characteristics (ROC) curves have been presented in Figs. 8 and 9. Since the total number of APIs is very large and our intention is to find a smaller subset of APIs, we consider a set of 50 APIs for comparison. Therefore the final subset is considered from the last 50 iterations of BE process for all algorithms. From Appendix, Table 4 and Fig. 4, it is seen that MR–SVMS achieves 92.944% for 30 APIs which is higher than both MRMR, MR and the wrapper SVM. MRMR achieves 91.614% for 33 APIs, MR achieves 91.833% for 40 APIs and SVM achieves 91.953% for 39 APIs. The MRMR–SVMS achieves the highest accuracy (94.362%) with 14 APIs which is higher than others. The APIs includes in the set 14 APIs have been listed in Table 2 which lists the 14 set APIs for SVM, MR, MRMR MR–SVMS as well. Both hybrid approaches (MRMR–SVMS and MR–SVMS) perform better than the independent filters (MRMR

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

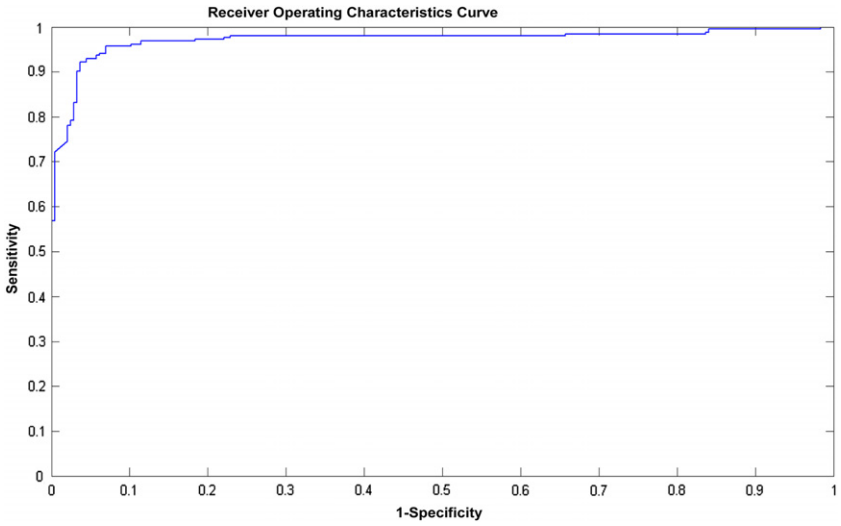


Fig. 8. Receiver operating characteristics curve (ROC) for MRMR–SVMS.

Table 2
APIs for final subset (14 set) for MRMR–SVMS and also APIs for (14 set) for other algorithms.

MRMR–SVMS	MR–SVMS	MR	MRMR	SVM
57	57	57	65	57
94	94	94	203	94
330	232	304	330	232
339	304	323	360	304
360	323	324	387	323
376	375	363	480	339
404	376	404	482	363
477	404	427	516	375
480	427	428	580	404
484	477	477	638	427
563	484	580	641	477
580	580	710	689	563
957	931	878	752	580
958	958	958	957	958

Table 3
Statistical validation (14 set) for MRMR–SVMS, MR–SVMS, MRMR, SVM and MR.

Model criteria	MRMR–SVMS	MR–SVMS	MR	MRMR	SVM
AIC	819.588	965.451	1068.684	845.527	981.114
Chi Square	789.588	935.451	1038.684	815.527	951.114

and MR) and the wrapper SVM and find smaller subset of APIs. However MRMR–SVMS achieves the highest accuracy (94.362%) with a compact set of 14 APIs. MRMR–SVMS also shows an accuracy of 96.042% with 291 APIs. But this is a set of very high number of APIs compared to 14 APIs.

6.1. Statistical validation of models

Binary logistic regression is deployed to assess the goodness of fit of the final models of the proposed approaches in terms of their APIs selection and accuracies. The statistical selection criteria for the best approach are the values of Chi Square and AIC. The logistic regression model is fitted to the individual 14 predictors (APIs) selected by different approaches. Table 3 presents the summary output under the different techniques. Based on the criteria listed in Section 4.8, we can conclude that the best performing APIs are the one selected by MRMR–SVMS. This set of APIs has led to the minimum AIC and Chi Square values. The ROC curve also confirm this statement as the area under the ROC curve for MRMR–SVMS is much closer to the upper left corner of the graph indicating that the hybrid has the best prediction ability. The logistic regression results in Table 3 also confirm that the proposed approaches outperform the existing models including independent filter and wrapper approaches.

6.2. Computational performances and search space complexity

The hybrid algorithms run a backward elimination (BE) process where each iteration involves the computational time in training the SVM, the computation of MR, MRMR, SVM and hybrid scores. At the beginning, when all APIs are used, the time for training and computing scores (MR, MRMR, SVM and hybrids) will be the highest. Subsequent computation for aforementioned scores will take less time. Considering the initial computational cost as a constant-maximum value including the cost for n-fold cross-validation, the rest of the wrapper search process in the BE iteration for MR, MRMR, SVM, MRMR–SVMS and MR–SVMS depends on the total number of BE iterations. For our proposed hybrid approaches, the total number of BE iterations is equal to the dimensions of the API set as mentioned in Algorithm 1. Therefore hybrids (MRMR–SVMS and MR–SVMS) generate $(m-1)$ subsets of features which demonstrate a search space complexity of a linear-function of the dimensions of the set of APIs. Thus hybrid approaches reduce the search space complexity. This also shows that the hybrid approaches have linear computational time complexity which is a function of the dimensions of the set of API set. The experimental platform was 3.2 GHz Intel Core Duo CPU with 2 GB of RAM. The computational time for MR, MRMR, SVM and hybrids is 2.98, 3.08, 2.43 h. MR–SVMS and MRMR–SVMS take longer time (3.38, 3.68 h) than either MR or MRMR or SVM.

7. Conclusion

With obfuscation techniques such as packer, polymorphism and metamorphism, recent malwares are able to evade from current detection methods. Consequently, security researchers and the anti-virus industries are facing a herculean task in extracting payloads hidden within packed executable. There is an anticipation that API statistics could be extracted as features and can be used to identify malware. To the best of our knowledge, in this paper, we have proposed for the first time in the malware detection domain, a hybrid framework for Malware detection using the hybrid of filter and wrapper approaches. The filter approaches MR and MRMR face drawback in evaluating the best subset and

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

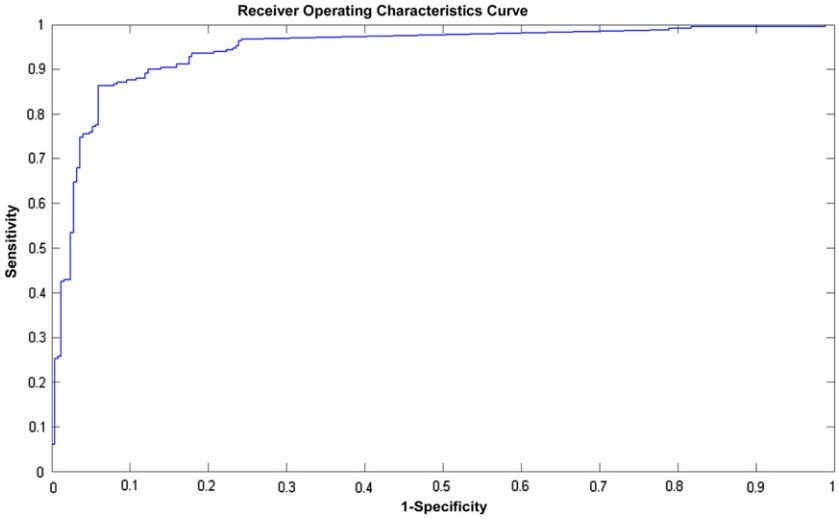


Fig. 9. Receiver operating characteristics curve (ROC) for MR–SVMS.

Table 4
Accuracies for different approaches: MRMR–SVMS, MR–SVMS, MRMR, MR, SVM in different iterations of backward elimination process.

No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM	No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM
972	96.506	96.815	96.135	95.98	96.846	157	93.537	92.053	91.991	94.249	89.425
971	96.846	96.722	95.949	96.042	96.289	156	93.878	92.084	92.177	93.135	89.703
970	96.877	96.908	96.011	96.135	96.382	155	93.506	92.641	92.022	93.754	89.703
969	96.815	96.228	96.135	96.166	96.289	154	94.434	92.146	91.991	93.723	88.126
968	96.475	96.475	96.073	96.135	95.547	153	93.383	91.775	91.651	94.465	88.776
967	96.444	96.691	96.289	96.599	96.104	152	93.754	91.96	91.528	94.001	88.033
966	96.691	96.939	96.259	96.599	96.104	151	93.785	92.301	92.022	93.939	89.796
965	96.877	96.444	96.259	96.413	96.568	150	94.218	92.331	91.991	94.125	89.054
964	96.413	96.197	96.197	96.073	96.382	149	93.908	92.177	91.837	94.465	89.054
963	96.63	96.66	96.166	96.32	97.31	148	93.878	92.579	91.528	93.939	88.683
962	96.289	96.259	96.011	96.444	96.197	147	94.28	92.331	92.022	93.816	90.816
961	96.568	96.444	96.66	96.166	96.197	146	93.692	92.486	92.115	93.97	89.332
300	95.826	95.949	91.218	95.671	90.631	145	93.537	92.393	91.775	93.135	88.683
299	95.547	95.733	91.342	95.795	90.353	144	93.692	91.96	91.682	93.692	89.981
298	95.887	95.516	92.239	95.176	89.796	143	93.816	92.115	91.682	93.259	88.776
297	95.733	95.362	91.558	95.207	88.312	142	93.908	92.424	92.022	93.321	90.631
296	95.949	95.516	91.806	96.011	88.126	141	94.156	91.991	91.342	93.723	88.59
295	95.455	95.455	91.929	95.795	90.074	140	93.785	91.899	91.651	93.785	90.074
294	95.3	95.64	91.466	95.887	90.909	139	93.723	92.362	91.033	93.939	87.941
293	95.733	95.887	91.589	96.011	89.796	138	93.476	91.868	90.909	93.63	87.941
292	95.826	95.269	91.435	95.702	88.033	137	93.105	92.424	91.28	93.445	89.518
291	96.042	95.578	91.249	95.114	90.724	136	93.908	92.764	91.033	93.939	90.909
290	95.485	95.764	92.084	95.949	90.909	135	93.97	92.177	91.249	93.63	88.961
289	95.671	95.764	91.373	95.578	90.353	134	93.568	92.795	90.878	93.197	88.312
288	95.485	95.795	91.095	95.609	90.724	133	92.672	92.641	90.693	93.908	90.26
287	95.64	95.733	92.115	95.764	89.425	132	93.105	92.331	91.064	93.414	89.981
286	95.671	95.887	92.239	95.145	90.26	131	93.476	92.486	91.744	93.506	88.683
285	95.733	95.485	91.249	95.826	90.074	130	93.383	92.641	91.682	92.919	90.631
284	95.702	95.671	91.342	95.3	90.167	129	92.857	92.424	91.249	93.074	90.353
283	95.702	96.011	91.713	95.609	90.26	128	92.95	92.764	90.538	92.764	90.445
282	95.702	95.764	91.713	95.98	90.538	127	92.95	93.383	90.878	93.135	90.631
281	95.516	95.424	91.156	95.857	89.518	126	92.95	93.135	91.342	93.352	89.332
280	95.64	95.331	92.27	95.485	89.703	125	92.733	92.95	90.971	92.795	88.776
279	95.671	95.609	91.651	95.362	90.445	124	92.641	92.641	91.033	92.672	89.61
278	95.269	95.516	91.929	95.671	90.353	123	93.043	92.95	91.558	92.764	89.703
277	95.702	95.578	91.744	95.393	89.889	122	93.012	92.208	91.126	93.445	87.848
276	95.455	95.455	91.929	95.702	89.796	121	93.352	92.115	91.404	92.795	89.054
275	95.485	95.578	91.589	95.609	90.631	120	92.61	91.991	90.847	93.166	89.239
274	94.96	95.578	91.497	96.011	91.095	119	93.105	92.053	90.878	93.043	89.796
273	95.485	95.578	91.899	95.671	90.538	118	93.105	91.62	91.342	93.383	89.796
272	95.053	95.083	91.466	96.073	90.353	117	92.795	91.528	91.373	92.795	88.033
271	95.145	94.774	91.187	95.733	90.631	116	92.27	91.466	90.847	92.95	89.332
270	95.331	95.547	91.837	95.918	90.167	115	92.424	91.62	90.476	92.826	89.239
269	95.949	95.516	91.064	95.331	90.26	114	92.733	91.249	91.435	92.579	88.683
268	96.135	95.114	92.022	95.795	90.631	113	92.826	91.62	91.435	92.826	89.703
267	95.887	94.682	91.806	95.238	90.074	112	93.043	91.435	91.187	92.795	88.59
266	95.857	94.898	91.528	95.949	90.724	111	93.259	90.6	91.497	92.672	88.776
265	95.207	95.083	91.991	95.516	90.538	110	92.888	90.971	91.96	93.29	89.239

(*continued on next page*)

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

Table 4 (*continued*)

No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM	No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM
264	95.826	95.393	91.744	95.455	88.961	109	93.043	91.404	91.96	93.197	89.981
263	95.887	95.053	91.899	96.104	90.167	108	92.486	91.713	91.218	92.826	89.518
262	95.795	95.083	91.218	95.455	90.631	107	92.641	91.466	91.156	93.197	90.26
261	94.867	94.867	91.528	95.764	90.26	106	92.919	91.218	91.095	92.795	88.497
260	95.176	95.114	91.589	95.609	88.868	105	92.919	91.218	91.064	92.764	88.776
259	95.516	94.774	91.187	95.3	89.796	104	92.95	91.187	90.909	93.012	88.961
258	96.073	94.774	91.682	95.826	90.353	103	92.857	90.785	90.167	92.826	91.002
257	95.207	95.053	91.651	95.702	88.683	102	92.764	90.847	91.497	92.424	89.981
256	95.393	95.053	91.899	95.547	90.26	101	92.579	90.847	90.909	93.074	88.033
255	95.918	94.774	91.744	95.887	89.796	100	92.888	91.064	90.94	92.95	90.074
254	95.671	94.836	91.651	95.795	89.425	99	92.764	91.156	93.166	91.095	89.796
253	95.671	94.187	91.837	95.3	89.796	98	92.641	91.404	92.239	90.94	89.518
252	95.485	94.836	91.929	95.485	89.889	97	92.641	91.095	92.424	91.373	90.167
251	95.516	95.022	91.837	95.547	90.538	96	92.486	91.929	91.96	90.94	90.816
250	95.393	95.207	90.94	95.547	90.816	95	92.177	90.785	92.455	90.538	90.074
249	95.516	94.651	91.002	95.733	90.631	94	92.424	90.631	92.239	91.033	92.022
248	96.011	94.62	91.466	95.393	89.703	93	92.548	91.311	91.713	91.156	89.796
247	95.331	94.527	91.528	95.918	91.187	92	92.641	90.693	92.239	91.466	90.538
246	95.238	94.341	91.497	95.393	89.703	91	91.991	91.806	91.991	92.084	89.981
245	95.609	94.218	91.373	95.455	89.703	90	92.053	91.218	92.517	91.435	90.816
244	95.362	94.496	91.775	95.609	89.703	89	92.177	91.528	92.826	92.208	90.445
243	95.485	94.156	92.208	95.269	89.518	88	92.146	91.558	92.084	91.929	91.558
242	95.826	94.465	91.899	95.671	90.816	87	92.362	91.528	91.868	90.909	90.445
241	95.64	94.898	91.713	95.238	89.518	86	92.826	91.466	92.177	90.94	91.744
240	95.485	94.465	91.651	96.011	89.518	85	91.868	91.806	92.486	91.466	90.724
239	95.702	94.249	91.651	95.455	89.518	84	92.084	92.301	91.899	92.177	91.651
238	95.176	94.125	91.868	95.671	90.074	83	92.053	91.311	92.177	91.62	91.558
237	95.022	94.125	91.991	95.331	91.837	82	92.115	91.929	92.239	91.62	90.724
236	95.609	93.785	92.641	95.331	90.26	81	91.929	91.991	92.331	91.466	90.538
235	95.547	94.094	92.208	95.764	90.538	80	91.806	92.331	91.528	91.373	91.28
234	95.702	94.341	91.558	95.393	91.002	79	91.126	92.424	91.342	91.837	91.466
233	95.3	93.723	91.96	95.918	89.054	78	91.064	92.981	91.404	92.053	91.558
232	95.547	94.187	92.208	95.64	89.889	77	90.785	93.074	91.373	91.837	90.724
231	95.485	94.125	91.991	95.114	90.167	76	91.589	92.733	91.064	91.806	90.538
230	95.3	93.939	91.775	95.609	89.796	75	90.291	92.331	90.724	92.084	91.558
229	95.362	94.682	92.424	95.857	88.961	74	91.064	92.084	91.033	91.558	90.816
228	95.64	94.28	91.713	95.393	90.26	73	90.012	92.301	90.662	92.053	90.631
227	95.393	93.878	91.837	95.764	90.074	72	91.806	92.888	90.569	91.373	91.187
226	94.991	93.939	91.899	95.3	88.961	71	90.816	92.517	90.94	92.301	90.816
225	95.516	93.908	92.053	95.455	89.796	70	91.837	92.331	91.651	91.96	90.074
224	95.547	94.094	92.146	95.578	89.981	69	92.084	92.455	91.466	91.187	92.301
223	95.176	93.939	91.837	95.516	89.61	68	92.517	92.517	91.806	91.651	90.167
222	95.207	94.341	91.806	95.702	89.889	67	92.548	92.579	91.991	92.239	91.651
221	94.991	93.939	91.373	95.269	90.724	66	92.733	92.517	91.837	92.084	90.167
220	95.516	94.372	92.022	95.424	90.724	65	91.899	91.558	91.929	91.466	91.002
219	95.022	93.847	91.589	95.331	90.353	64	92.95	91.806	91.528	91.435	91.095
218	94.836	93.939	91.744	95.207	89.796	63	91.62	92.239	92.486	91.806	90.538
217	95.053	93.754	92.393	95.671	89.054	62	92.733	92.424	92.084	91.589	89.332
216	94.774	94.341	91.929	95.238	89.518	61	91.156	92.27	92.115	91.62	89.332
215	94.743	93.847	91.775	95.516	89.889	60	91.868	92.888	92.084	91.837	90.909
214	94.991	93.63	91.991	95.145	88.961	59	92.239	92.61	92.331	92.084	91.558
213	94.991	93.63	92.331	94.62	88.868	58	92.115	91.929	91.806	91.558	89.703
212	95.269	94.372	92.084	95.083	90.074	57	91.713	92.362	91.435	91.311	91.002
211	94.898	93.847	91.218	95.053	89.518	56	91.62	92.455	90.847	92.27	91.095
210	95.3	93.847	91.744	95.331	90.074	55	91.435	91.435	90.353	91.806	91.466
209	94.929	93.414	91.929	95.238	88.59	54	91.435	91.744	90.816	91.96	91.002
208	94.712	93.29	91.899	95.362	89.796	53	91.156	91.929	91.002	91.095	90.167
207	95.022	93.445	91.806	94.96	89.425	52	91.342	91.033	90.878	91.899	91.187
206	94.929	93.414	92.27	95.114	89.61	51	91.62	91.713	91.651	91.589	89.889
205	94.589	93.352	92.022	95.362	90.353	50	91.404	92.022	90.909	91.28	91.002
204	94.96	93.228	91.713	95.485	88.868	49	91.187	91.589	91.249	91.837	92.301
203	94.743	93.816	92.888	94.434	89.518	48	91.682	91.373	91.497	91.033	91.095
202	95.053	93.29	91.373	95.022	90.445	47	90.724	92.579	90.909	91.713	91.837
201	95.207	93.074	91.868	94.743	89.332	46	91.311	91.218	91.373	92.022	90.167
200	95.022	93.383	92.424	95.022	88.961	45	90.754	91.342	91.218	91.929	91.651
199	94.589	93.506	91.435	94.403	90.074	44	90.785	91.868	90.631	91.806	90.909
198	94.589	93.537	91.929	95.176	89.703	43	91.156	91.682	90.878	91.868	91.744
197	94.991	93.383	91.528	94.867	90.631	42	91.589	91.713	91.311	90.971	91.651
196	94.465	93.692	92.672	94.898	90.538	41	91.28	91.651	90.724	91.156	91.744
195	93.816	93.785	92.455	94.187	88.59	40	91.62	91.713	91.833	90.909	89.889
194	94.156	93.043	92.331	94.62	88.312	39	91.002	91.528	91.002	90.878	91.953
193	94.991	93.908	91.96	94.249	90.538	38	91.404	91.744	91.126	91.033	90.909
192	94.527	93.321	91.899	94.558	89.054	37	91.558	91.435	90.751	90.754	91.466
191	94.712	93.63	92.362	94.434	88.683	36	91.033	91.837	90.445	90.847	90.538
190	94.651	93.506	91.868	94.496	90.445	35	90.816	91.311	90.569	90.65	91.373
189	94.743	93.847	92.208	94.682	90.816	34	91.589	91.156	90.631	90.538	91.187

(*continued on next page*)

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

Table 4 (*continued*)

No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM	No of API	MRMR + SVM	MR + SVM	MR	MRMR	SVM
188	94.743	93.105	92.424	95.114	90.167	33	91.28	92.022	90.445	91.614	90.909
187	94.465	93.135	92.27	94.712	90.631	32	90.291	92.713	90.631	90.074	90.631
186	94.465	93.135	91.806	95.114	89.332	31	91.002	92.218	90.012	90.043	90.353
185	94.558	93.352	92.301	94.589	89.703	30	90.693	92.944	89.734	90.136	89.889
184	93.723	92.733	92.177	94.31	89.518	29	91.002	92.187	89.858	89.858	89.61
183	94.434	92.981	92.177	94.434	89.239	28	90.383	90.94	90.507	90.569	90.631
182	94.867	92.579	92.301	94.28	88.219	27	91.744	90.064	90.383	88.899	89.61
181	94.682	92.888	91.744	94.434	89.425	26	91.249	91.28	89.951	90.012	90.167
180	93.506	93.135	92.084	94.187	88.59	25	91.6	91.929	89.641	89.487	90.167
179	94.712	92.95	92.517	94.805	89.332	24	91.847	91.806	89.641	89.61	89.332
178	94.31	93.352	91.806	94.651	89.703	23	92.28	91.62	90.012	90.229	90.538
177	94.465	92.703	92.301	94.589	90.26	22	92.93	91.187	90.198	90.847	91.28
176	94.372	92.826	92.022	94.187	88.683	21	93.27	92.022	90.353	90.476	91.28
175	94.187	92.857	92.672	94.434	89.332	20	93.455	91.744	90.105	90.291	91.466
174	94.372	92.888	92.27	94.31	89.054	19	93.651	91.991	89.796	90.136	90.538
173	94.249	93.043	92.517	94.063	89.332	18	93.806	91.96	89.672	89.27	90.187
172	94.125	92.301	91.868	93.754	89.332	17	92.455	91.486	89.487	89.487	90.187
171	94.527	92.084	92.424	93.908	90.631	16	93.96	91.486	88.497	88.281	90.002
170	93.878	92.239	92.022	94.032	89.61	15	93.86	91.27	87.508	87.539	90.022
169	94.28	92.301	92.27	94.403	89.61	14	94.362	91.022	87.415	87.168	90.538
168	94.372	91.96	92.053	94.28	89.147	13	93.62	91.651	86.735	87.322	90.909
167	94.156	92.362	92.27	94.31	88.219	12	92.28	91.744	85.962	86.611	90.558
166	93.754	92.301	92.331	94.558	89.425	11	91.497	91.837	85.591	85.56	90.208
165	94.063	91.775	92.331	93.908	87.662	10	90.569	91.806	86.24	85.9	90.558
164	93.723	91.868	92.331	93.506	89.981	9	90.724	91.342	86.209	85.9	90.724
163	94.249	92.084	92.084	93.878	89.61	8	89.827	90.724	84.756	84.91	90.167
162	94.249	93.074	92.424	94.063	89.981	7	88.961	89.765	84.632	84.168	87.239
161	94.001	92.331	92.022	93.878	89.054	6	87.477	87.755	80.365	80.087	83.054
160	94.125	91.682	92.61	94.063	89.703	5	87.106	88.002	80.21	79.994	83.126
159	94.001	91.929	91.713	93.878	88.219	4	86.549	87.353	77.087	76.747	81.384
158	93.599	91.991	91.744	93.568	89.054	3	84.972	85.683	76.592	76.562	81.106

wrapper suffers from computational overhead for repeated induction process. The goal of this paper is to exploit the strengths of each of these two approaches towards the development of a hybrid framework for efficient malware detection. Several important performance measures including accuracy, compact feature set, Chi Square and Akaike information criterion (AIC) and area under ROC curve were employed to assess the efficacy of our proposed models and to compare with the independent filters and wrappers. The experimental results show that the hybrid approaches MRMR–SVMS and MR–SVMS provide higher accuracies than other approaches. The novelty of proposed hybrid wrapper–filter models (MRMR–SVMS and MR–SVMS) lies on the fact that the proposed MRMR–SVMS/MR–SVMS achieve the highest accuracy with a very compact subset of significant APIs through a signature-free approach with a comparatively lower computational and search space complexities. The proposed signature-free approaches are also able to detect the malware variants which evade detection from signature-based approaches. The statistical goodness of fit criteria also confirm this conclusion.

One of the major contributions of this paper is the development of a fully-automated signature-free method to unpack, deobfuscate and reverse engineer the binary executable without any need for manual inspection of assembly codes. Thus proposed approaches are able to find the API statistics without any manual intervention. The novelty of the proposed hybrid framework is that it integrates the knowledge (from the intrinsic characteristics of Malwares) obtained by the filter into the wrapper approach and combines the wrapper's heuristic score with the filter's ranking score in the wrapper stage of the hybrid. To the best of our knowledge, the approach is new and has not been explored yet in the context of feature selection and malware detection. The combined heuristics in the hybrid take the advantages of the complementary properties of the both filter and wrapper heuristics and efficiently guide the wrapper to find an optimal and compact API subsets. In this paper, we consider maximum-relevance and minimum-redundancy filters with a SVM wrapper. However, choice of filter may affect overall performances of the hybrid. Therefore, a suitable filter can be selected by using different filters with the wrapper and then evaluating them using the wrapper evaluation criteria. We consider these approaches of model selection for filters as a future work.

Table 5
Acronyms and their meaning.

Acronyms	Meaning
AV	Anti-Virus
API	Application Program Interface
AIC	Akaike's Information Criterion
ROC	Receiver Operating Characteristic Curve
SVM	Support Vector Machine
PE	Portable Executable
DLL	Dynamic Link Library
ANN	Artificial Neural Networks
ANFIS	Adaptive Neuro-Fuzzy Inference Systems
SBE	Sequential backward elimination
SFE	Sequential forward elimination
MRMR	Maximum Relevance and Minimum Redundancy
MR	Maximum relevance
MR–SVMS	Maximum Relevance and SVM score
MRMR–SVMS	Maximum Relevance–Minimum Redundancy and SVM score
BE	Backward Elimination

Appendix

See Tables 4 and 5.

References

[1] S. Ghosh, E. Turrini, Cybercrimes: A Multidisciplinary Analysis, Springer Verlag, 2010.
[2] I. You, K. Kim, Malware obfuscation techniques: a brief survey, in: International Conference on Broadband, Wireless Computing, Communication and Applications, 2010, pp. 297–300.
[3] P. Singhal, N. Raul, Malware detection module using machine learning algorithms to assist in centralized security in enterprise networks, International Journal of Network Security & Its Applications (IJNSA) 4 (2012).
[4] A. Stabek, P. Watters, R. Layton, The seven scam types: Mapping the terrain of cybercrime, in: Cybercrime and Trustworthy Computing Workshop, Ballarat, Victorial, Australia, 2010, pp. 41–51.

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

[5] S. Treadwell, M. Zhou, A heuristic approach for detection of obfuscated malware, in: Proceedings of the IEEE International Conference on Intelligence and Security Informatics, IEEE Press Piscataway, NJ, USA, 2009, pp. 291–299.
[6] K. Tang, M.T. Zhou, Z. Z -H, An enhanced automated signature generation algorithm for polymorphic malware detection, Journal of Electronic Science and Technology of China (JESTC) 8 (2010) 114–121.
[7] P. Desai, A highly metamorphic virus generator, International Journal of Multimedia Intelligence and Security (IJMIS) 1 (2010) 402–427.
[8] A.H. Sung, J. Xu, P. Chavez, S. Mukkamala, Static analyzer of vicious executables (save), in: 20th Annual Computer Security Applications Conference, Tucson, Arizona. USA, 2004, pp. 326–334.
[9] B. Birrer, R. Raines, R. Baldwin, M. Oxley, S. Rogers, Using qualia and hierarchical models in malware detection, J. Inf. Assur. Secur. 4 (2009) 247–255.
[10] S. Choi, H. Park, H.i. Lim, T. Han, A static api birthmark for windows binary executables, J. Syst. Softw. 82 (2009) 862–873.
[11] C. Wang, J. Pang, R. Zhao, W. Fu, X. Liu, Malware detection based on suspicious behavior identification, in: In proceedings of First International Workshop on Education Technology and Computer Science, Wuhan, Hubei, China, 2009, pp. 198–202.
[12] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, L. W, Eureka: a framework for enabling static malware analysis, in: Jajodia, S., Lopez, J. (Eds.), Computer Security - ESORICS 2008. Springer Berlin / Heidelberg. volume 5283, 2008, pp. 481–500.
[13] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, Y. Elovici, Detecting unknown malicious code by applying classification techniques on opcode patterns, Security Informatics 1 (2012) 1–22.
[14] R. Kohavi, G. John, Wrappers for feature subset selection, Artificial Intelligence 97, 273–324.
[15] H., B.D. Wang, F. Murtagh, Axiomatic approach to feature subset selection based on relevance, IEEE Trans. Pattern Anal. Mach. Intell. 21 (1999).
[16] C.N. Hsu, H.J. Huang, S. Dietrich, The annigma-wrapper approach to fast feature selection for neural nets, IEEE Trans. Syst. Man Cybern. B 32 (2002) 207–212.
[17] J.S.L. II-Seok Oh, B.R. Moon, Hybrid genetic algorithms for feature selection, IEEE Trans. Pattern Anal. Mach. Intell. 26 (2004).
[18] VX-Heavens-Online-Resource, Vx heavens. (2011, 2/3). Available at http://vx.netlux.org/.
[19] E. Eilam, Reversing: Secrets of Reverse Engineering, first ed., Wiley Publishing, 2005.
[20] M, R. Chouchane, A. Lakhotia, Using engine signature to detect metamorphic malware, in: 2006 Proceedings of the 4th ACM Workshop on Recurring Malcode, ACM, 2006, pp. 73–78.
[21] M. Eskandari, S. Hashemi, A graph mining approach for detecting unknown malwares, J. Vis. Lang. Comput. 23 (2012).
[22] R. Tian, L. Batten, S. Versteeg, Function length as a tool for malware classification, in: Proceedings of the 3rd International Conference on Malicious and Unwanted Software : MALWARE, 2008, pp. 69–76.
[23] M. Bailey, J. Oberheide, J.e.a. Andersen, Automated classification and analysis of internet malware, in: Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID '07), 2007, pp. 178–197.
[24] F. Ahmed, H. Hameed, M. Shafiq, M. Farooq, Using spatio-temporal information in api calls with machine learning algorithms for malware detection, in: Proceedings of the 2nd ACM workshop on Security and Artificial Intelligence, 2009, pp. 5–62.
[25] A. Sami, B. Yadegari, H.e.a. Rahimi, Malware detection based on mining api calls, in: Proceedings of the 2010 ACM Symposium on Applied Computing, 2010, pp. 1020–1025.
[26] W. Lu, M. Tavallaee, A. Ghorbani, Automatic discovery of botnet communities on large-scale communication networks, in: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, 2009, pp. 1–10.
[27] R. Tian, L. Batten, R.e.a. Islam, An automated classification system based on the strings of trojan and virus families, in: Proceedings of the 4rd International Conference on Malicious and Unwanted Software: MALWARE, 2009, pp. 23–30.
[28] J.Z. Kolter, M.A. Maloof, Learning to detect and classify malicious executables in the wild, J. Mach. Learn. Res. 7 (2006) 2721–2744.
[29] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
[30] D. Stopel, Z. Boger, R. Moskovitch, Y. Shahar, Y. Elovici, Application of artificial neural networks techniques to computer worm detection, in: Proceedings of the International Joint Conference on Neural Networks, Vancouver, 2006.
[31] M. Bhattacharya, A. Das, Genetic algorithm based feature selection in a recognition scheme using adaptive neuro fuzzy techniques, International Journal of Computers Communications & Control (IJCCC) V (2010) 458–468.
[32] H. Ankishan, Comparison of svm and anfis for snore related sounds classification by using the largest lyapunov exponent and entropy, Computational and Mathematical Methods in Medicine 2013 (2013).
[33] F.J. Torczon, P. Pudil, M. Hatef, J. Kittler, Comparative study of techniques for large-scale feature selection, in: Pattern Recognition in Practice IV, Multiple Paradigms, Comparative Studies and Hybrid Systems, 1994, pp. 403–413.
[34] P. Royal, M. Halpin, D. Dagon, R. Edmonds, Polyunpack: automating the hidden-code extraction of unpack-executing malware, in: Proc. of 22nd Annual Computer Security Applications Conference, ACSAC, 2006.
[35] M.G. Kang, P. Poosankam, H. Yin, Renovo: a hidden code extractor for packed executables, in: Proceedings of the 2007 ACM Workshop on Recurring Malcode, 2007.
[36] L. Martignoni, M. Christodorescu, S. Jha, Omniunpack: Fast, generic, and safe unpacking of malware, in: Twenty-Third Annual Computer Security Applications Conference, ACSAC 2007, 2007.
[37] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, L. W, Eureka: a framework for enabling static malware analysis, in: Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security, 2008.
[38] IDA Pro Disassembler and Debugger, IDA Pro, 2010.
[39] SQLite-Online-Resource, Sqlite. Available at www.sqlite.org.
[40] Microsoft-MSDN, Msdn..
[41] H. Peng, C. Ding, F. Long, Minimum redundancy-maximum relevance feature selection, IEEE Intelligent Systems 20 (2005) 70–71.
[42] W.N. Greene, Econometric Analysis, Prentice-Hall, 2003.
[43] D.W. Hosmer, S. Lemeshow, Applied Logistic Regression, second ed., Wiley, ISBN: 0-471-35632-8, 2000.
[44] A. H, Information theory and an extension of the maximum likelihood principle, in: Proc. of 2nd Int. Symp. on Information Theory, Akademiai Kiado, 1973, pp. 267–281.

Shamsul Huda is a Research fellow/Lecturer in School of Science, Information Technology and Engineering (SITE), Federation University, Australia since 2009. He has published more than 30 journal and conference papers in well reputed journals including IEEE Transactions and Elsevier Science. His main research area is Dynamic pattern recognition modeling and estimation, optimization approaches to data mining information and network security and big data mining. Earlier to join in the University of Ballarat, he has worked as an Assistant professor in the Computer Science Department in Khulna University of Engineering and Technology (KUET), Bangladesh.

Jemal Abawajy is a full professor at school of Information Technology, Faculty of Science, Engineering and Built Environment, Deakin University, Australia. He is currently the Director of the Parallel and Distributing Computing Laboratory. He is a Senior Member of IEEE Computer Society; IEEE Technical Committee on Scalable Computing (TCSC); IEEE Technical Committee on Dependable Computing and Fault Tolerance and IEEE Communication Society. He has served on the editorial-board of numerous international journals and currently serving as associate editor of the International Journal of Big Data Intelligence and International Journal of Parallel, Emergent and Distributed Systems. He has also guest edited many special issues. He is the author/co-author of five books, more than 250 papers in conferences, book chapters and journals such as IEEE Transactions on Computers and IEEE Transactions on Fuzzy Systems. He also edited 10 conference volumes.

Mamoun Alazab is a Research Fellow at the Australian National University and the co-founder of the ANU Cybercrime observatory. He comes from an Information Security (Computer Science) background with previous research experience in the 'technology' aspects of cybercrime research. Some examples include cybercrime/cyber security, Internet and spam regulation, reverse engineering, malware analysis, computer forensics, and network intrusion detection.

Mali Abdollahian is an internationally recognized researcher in the field of Industrial Quality Control, Quality Assurances and Process Capability Analysis. She has more than 23 years research experience in monitoring service and industrial processes. She has more than 90 refereed international journal and conference publications in the areas of quality improvement, Quality assurance, process capability analysis, production control and inventory management and failure analysis.

Her research work involves monitoring mean and variability of multi-stream processes in areas such as manufacturing and automotive industry, clinical area, food industry and water and air pollution processes which led to a higher level of on-line maintenance and repair without requiring system shut-down. This would result in a cost effective plant maintenance strategies. She has been acting as an International Technical Committee member for a number of World Congresses and International Industrial Conferences.

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001

Rafiqul Islam is a Lecturer in the School of Computing and Mathematics at Charles Sturt University, Australia. He has expertise in Network & Information Security, Cyber-Security, Sensor Network and Machine Learning. He received his Ph.D. and postdoctoral experience from Deakin University, Australia. Previously he was a lead researcher for an ARC Linkage Project (LP) on Malware analysis and classification, in collaboration with industrial partners Computer Associates (CA) and RMIT University. He also worked on several ARC funded projects as a lead researcher. Currently he is working on several projects as Chief Investigator (CI) and is supervising five Ph.D. students. He published more than 80 refereed research articles, more than 10 book chapters, and he received 'Best Paper Award' for his ChinaComm 2010 conference paper. He is also an editorial board member for an international journal and General Chair and Technical Program Committee (TPC) member of various international conferences. He is a member of the IEEE and a founder member of ATT.

John Yearwood is the Executive Dean of the Faculty of Science and Technology. He was Director of The Centre for Informatics and Applied Optimization (CIAO). He holds a Bachelor of Science with first class Honours and Diploma of Education from Monash University, a Master of Science from the University of Sydney. He was awarded his Doctor of Philosophy from RMIT University Australia. He has a significant publication record, having published two books and over 200 refereed journal, book chapter and conference articles and has supervised Masters and many Ph.D. research students through to completion. He is the Editor-in-Chief of the Journal of Research and Practice in Information Technology, and a reviewer for Australian Journal of Information.

Please cite this article in press as: S. Huda, et al., Hybrids of support vector machine wrapper and filter based framework for malware detection, Future Generation Computer Systems (2014), http://dx.doi.org/10.1016/j.future.2014.06.001