Malicious Detection Based on ReliefF and Boosting Multidimensional Features

Yang Xia Luo
School of Information, Xi'an University of Finance and Economics, 710100, China
Email: yxluo8836@163.com

Abstract—Aiming at the problem of large overhead and low accuracy on the identification of obfuscated and malicious code, a new algorithm is proposed to detect malicious code by identifying multidimensional features based on ReliefF and Boosting techniques. After a disassembly analysis and static analysis for the clustered malicious code families, the algorithm extracts features from four dimensions: two static properties (operation code sequences and bytecode sequence) and two features (system call graph and function call graph) which combines the semantic features to reflect the behaviour characteristic of the malware, and then selects important feature vectors based on Relief. Finally, ensemble learning is carried out, and the decision result is boosted based on weighted voting according to accuracy for a different feature analysis. It has been proven by experiment and comparison that the algorithms have a much higher accuracy of the testing dataset with low overhead.

Index Terms—Malicious code detection, multidimensional features, Boosting, ReliefF

I. INTRODUCTION

With the rapid development of network interconnection and widely using computers and digital devices for almost everything in everywhere worldwide, the confidentiality and integrity of electronic information is increasingly under serious threats [1]. Information security grows to be an important research field with much more concern. The appearance and increasing of malicious codes cause immeasurable losses and difficulties for numerous commercial companies, government institutions and people's daily lives. Therefore, investigation and development of techniques of detecting malicious code gradually become an important research direction in the field of information security.

Existing methods of detecting malicious code include signature-based detecting, checksum approach, behaviour monitoring, characteristic code detecting, artificial intelligence based detecting, etc. Since they apply different theories, and result in different overheads, each has its own advantages and disadvantages. Concerning different types of detecting objects of malicious code, major studies of static detecting methods involve: Assaleh [2] extract n-gram from byte sequence of a program as an effective information to classify malicious codes and apply the classifying algorithm using K nearest neighbor to detect computer viruses. Canali [3] analyze the system-call graph of malicious codes and identify malicious codes on the basis of detecting threshold of graph nodes. KANG B [4] use the control flow graph of a program as its characteristic attribute, and describe malicious behaviors of programs with Linear Time Logic (LTL) rule. Shankarapani et al. [5] design a static method to extract API calls from PE documents, and judge malicious code by finding the degree of difference between the suspicious file's call sequence and normal API call sequence using a detection engine. Most existing researches are based on a kind of attributes, combine statistical methods and artificial intelligence based methods, and apply traditional n-gram extraction techniques. Because most software applies polymorphic and metamorphic techniques, traditional detection methods have a limitation that only achieves a low detecting rate on transformed codes. Moreover, dynamic detection techniques [6]-[8] need to conduct virtual execution of the codes in a virtual machine to get the corresponding execution behaviour and path of codes, which result in slow detection, overhead and extensive usage of resources.

In obfuscated code identification, it is difficult to identify malicious code with lower system overhead, and improve the detection rate and the accuracy rate while reducing the false alarm rate. This paper presents a research on multi-attribute feature detection. On the basis of static feature analysis of malicious code family and the "behavioural" feature in static mode, we propose to select characteristic vector based on ReliefF, and carry out weighted voting according to accuracy rate, thus boost the detection rate of different classifiers of multidimensional features, and solve the problem of large overhead and low accuracy in malicious code detection, and then conduct ensemble learning through weighted voting, and finally give the results of obfuscated malicious code identification.

Our key contributions can be summarized as follows: (1) selecting feature vectors based on Relief, and improve the time and space efficiency of classification through feature dimension compression. (2) promoting detecting...
results based on Boosting, and performing weighted voting according to accuracy rates of different classifiers to improve recognition for obfuscated and malicious codes. (3) based on variable-length n-gram feature selection, carrying out expert voting to calculate frequency of a feature in order to improve traditional n-gram [9] and n-perm [10] segmentation and statistics methods.

The other parts of this paper are organized as follows: the second section provides some background knowledge about malicious code detection and feature selection and boosting. The Section 3 introduces important algorithms and the framework of obfuscated malicious code detecting method based on ReliefF and Boosting. in the Section 4, the algorithm is proposed and analyzed. the Section 5 contains experimental result verification and comparisons with other works. Finally, conclusion and outlook are given.

II. BACKGROUND KNOWLEDGE

A. Obfuscating Transform and Malicious Code Detection

Let P represent a program set and an obfuscating process (o) can be considered as a form of program transformation o: p→p′ (p, p′ ∈ P). For example, renaming or reordering is an obfuscating process. Denote the set of all obfuscating transforms as O, (o∈ O). A malicious code detector D is to detect whether a program p is infected by a malicious code m (m∈ P), written as D : P×M→{0,1}, where

\[ D(p,m) = \begin{cases} 1, & \text{if } p \text{ is infected with } m \\ 0, & \text{otherwise} \end{cases} \]  

(1)

If a program p is infected by a malicious code m, it is defined as m→p. If all programs, which are checked by a malicious code detector, are infected, the detector is said to be sound, in other words, there are no false positive tuples. If a malicious code detector can detect all infected programs, it is said to be complete, in other words, there are no false negative tuples. Its formal definition is as follows [11].

Definition 1: Soundness and completeness of detection

If a malicious code detector is complete, for an obfuscating transform o (o∈ O), if and only if \( \forall M \in P, O(M) \rightarrow P \Rightarrow D(P,M)=1 \). If a malicious code detector is sound, for an obfuscating transform o ∈ O, if and only if \( \forall M \in P, D(P,M) = 1 \Rightarrow O(M) \rightarrow P \).

B. Feature Selection

Assume that every training set \( R_1 \) is composed of attribute vectors \( A_1, A_2, \cdots, A_n, A_{n+1} \), where \( A_1, A_2, \cdots, A_n \) are input attributes and \( A_{n+1} \) is type attributes. Most existing trainings use all attributes, wishing to create a mapping from \( A_1 \times A_2 \times \cdots \times A_n \) to \( A_{n+1} \). However, all input attributes may not be useful or be less useful, in predictive classification of unknown data. In the process of feature selection, in order to find the most effective features and improve learning efficiency, select the first \( n' (n'<n) \) characteristic vectors based on weights, and compress high-dimensional feature space \( R_{tn} \) into low-dimensional space \( S_{tn} \).

C. Boosting

Boosting is a complementary classifier based on weights. A given sample set S is divided into training sets \( S_1, S_2, \cdots, S_t \) according to attribute type. and each of t sub-training sets is endowed a weight. The weight distribution of each training set is assigned to be \( 1/t \) in initialization, and training is conducted on all training sets according to this distribution.

![Fig. 1. Boosting based on weighted voting](image)

After each of the trainings, the weight distribution of training sets is updated on the basis of training results, and then, the sample weights are increased or decreased in conformity to weight strategy to make the samples obtain more concerns in the next iteration. In the next iteration, training is carried out according to the new sample weights. After repeating iteration for T times, a learning machine series is acquired finally. When testing, each learning machine has a particular weight. The final estimated value is acquired by weighted voting, as shown in Fig. 1.

III. DETECTION FRAMEWORK FOR MALICIOUS CODE BASED ON SELECTING AND BOOSTING

The obfuscated and malicious code detecting framework, based on feature selection and classification boosting, consists of the following main steps: clustering the malicious code family samples, obfuscating transform every type of code families, disassembling and statically analyzing samples before and after obfuscation, and normal code samples. Extracting specific attribute types of each sample to form training sets \( R_1, R_2, \cdots, R_{tn} \). out of each training set, select the first \( n' (n'<n) \) vectors based on ReliefF, to generate learning machine \( S_1, S_2, \cdots, S_{tn} \) for classification learning. performing weighted voting according to accuracy rates, to boost the judging results of classifiers.
Sample training mainly accomplishes the following three procedures: disassembling and static analysis, feature selection and classifier construction, feature selection. On code to be detected, three steps are performed: disassembling and static analysis, feature selection, weighted voting for detection result. Main modules of the system are shown in Fig. 2.

Fig. 2. The detection frameworks based on selection and boosting

Before feature extraction, extracting object, i.e., feature type should be determined. Presently the extracted feature types include source character string, operation code, bytecode, API call, control flow graph, PE file header, etc. Execute obfuscating transform on code [12]: renaming obfuscating attack will affect the type name of the source code, adding useless predicate attack will influence API frequency and order. smooth type name of the source code. adding useless predicate code [12]: renaming obfuscating attack will affect the PE file header, etc. Execute obfuscating transform on operation code, bytecode, API call, control flow graph, etc.

A. Feature Extraction and Classifier Construction

Feature extraction involves the four feature types mentioned above. First, disassemble the executable file with IDA pro [13] (if shelled, unshell it first), then extract the assembly code, analyze statistically various feature sets, and store in vector (A1, A2, ..., An). Feature selecting algorithms are as follows.

1) Variable-length n-gram feature extraction algorithm

The algorithm mainly accomplishes extraction and frequency statistics of operation code sequence and bytecode sequence. Using (opcode sequence, frequency) and (bytecode sequence, frequency) as characteristic vector, gather statistics of n sequences in each type of samples to form the training set R1 and R2.

Use variable-length n-gram feature extraction to avoid splitting meaningful sequences. Considering two different sequences string1={push call xor mov xor pop}, string2={01 E8 B8 01 E8 B8 B8 01}, suitable breakpoints need to be found first, and continuous sequence between two consecutive breakpoints is an extracting feature. The expert voting algorithm is selected as our segmentation algorithm. To improve the frequency computation and feature generation, Trie data structure is adopted to implement the expert voting algorithm. Its basic principle is: denoting depth of Trie by d, read d-1 bytes as a sub-sequence each time in the byte sequence. Each sub-sequence is inserted into Trie. If this sub-sequence has not appeared in Trie, its frequency is 0, otherwise, if this sub-sequence already exists, add its frequency by one. Each node in Trie has two parameters, frequency (f) and entropy (e), where m is the number of subtrees owned by the parent node x₀, entropy of leaf node is 0, the frequency of a node in Trie is

\[ P(x_0) = \frac{f(x_0)}{f(parent(x_0))} \]  

(3)

The entropy of a node is

\[ e(parent(x_0)) = -\sum_{i=0}^{m} P(x_i) \log P(x_i) \]  

(4)

For instance, string2={01 E8 B8 01 E8 B8 B8 01}, when depth d=4, use the two experts (frequency and
entropy) to vote on possible breakpoints, and give higher scores at breakpoints. After the slide window has traversed the whole string, some local maxima can be found according to the scores and corresponding positions are breakpoints. Consequently, the sequence splitting result of string2 is ((01 E8 B8) (01 E8 B8) (B8 01)).

When constructing classifiers, SVM algorithm is adopted for training learning. Every sample is a n-dimensional vector. Construct m secondary classifiers, where m is the number of types of malicious codes. Then there exists \((x_1, y_1) \cdots (x_m, y_m) \in \mathbb{R} \times \{ \pm 1 \}\). Find the optimal partitioning plane \(w \cdot x + b = 0\) between the two types. The \(i\)th classifier decision function is \(f_i(x)\). Use malicious code from type1 as positive examples, and malicious code from non type1 as negative examples. When training, to adjust the size of the sliding window, and optimize the objective function according to

\[\max_W (a) = \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i=1}^{m} a_i a_j y_i y_j k(x_i, x_j)\]  
(5)

With constraint \(\sum_{i=1}^{m} a_i y_i = 0\). Lagrange multiplier \(a_i \in [0, C]\), where C is a positive constant.

Decision function is defined as

\[f(x) = \text{sgn}(w \cdot x + b) = \text{sgn}(\sum_{i=1}^{m} a_i y_i k(x_i, x) + b)\]  
(6)

The kernel function \(k(x_i, x_j)\) satisfies Mercer condition, as an inner product of some transformed space. In training, use the radial base kernel function \(k(x_i, x) = \exp(- \frac{|x_i - x|^2}{\delta^2})\), where \(\delta\) is an argument. Use quadratic programming to solve equation (5), to find optimal Lagrange coefficient \(a\). threshold \(b\) is found by solving equation (6).

2) Extracting algorithm based on similarity of directed graphs

This algorithm mainly accomplishes analysis and statistics of system calls (System_Call) and function calls (Function_Call). Calling graph reflects semantic characteristics of code to some extent. They differ only semantically, yet having similar structures of directed graph called. Use tuples (system feature graph, similarity) and (function feature graph, similarity) as characteristic vector, find the common subgraph of each malicious code family. then calculate the similarity between each sample and common subgraph of each malicious code family, to form training sets \(R_3\) and \(R_4\).

Structure of a directed graph is a quadruple \(G = (N, E, \alpha, \beta)\), where \(N\) represents node junction \((n \in N)\), \(E\) is set of edges. there exists \(E \subseteq N \times N\), and node \(n1 \rightarrow n2 \in E\) representing dependency of calls. \(\alpha: N \rightarrow S\) represents function to assign calls to nodes. \(\beta: E \rightarrow T\) represents function to assign calls to edges. For each node, there is a precursor node \(n_i\) pre and a successive node \(n_i\) suc, and use \(n_i\) pre_num and \(n_i\) suc_num to represent number of precursor and successive nodes of node \(n_i\) respectively. Assume y malicious code families \(\{f_1, f_2, \cdots, f_y\}\), and each family contains \(K\) samples. Extract \(K\) directed graphs \(G_1 \cdots G_k\), and find the common subgraph \(G'_1 \cdots G'_k\) of the \(k\) directed graphs as the characteristic graph. Malicious codes of the same family enjoy a relatively large similarity, so there exists a common graph. while normal codes are greatly different, so it is impossible to extract common subgraph from the samples, thus, their similarity is set to 0.

Match each sample’s directed graph \(G_k\) with all common directed subgraphs \(G'_y\) (Feature graph). Suppose the node number of \(G_k\) and \(G'_y\) is \(n_x\) num = \(a\) and \(n_y\) num = \(b\) respectively, scoring function is defined as

\[\text{Sim}(G_k, G'_y) = \sum_{i=1}^{a} \sum_{j=1}^{b} \varphi_{ij} \varphi_{ij} = \begin{cases} 1, & \text{same}(G_k, G'_y) \\ 0, & \text{diff}(G_k, G'_y) \\ \mu, & \text{simi}(G_k, G'_y) \end{cases}\]  
(7)

\(\text{same}(G_k, G'_y)\): For \(\forall x, x \in [1, a]\), \(\exists y, y \in [1, b]\), makes \(n_{xi}.\)pre = \(n_{yj}.\)pre \& \& \(n_{xi}.\)suc = \(n_{yj}.\)suc , indicating all precursor and successive nodes are identical. \(\text{diff}(G_k, G'_y)\): For \(\forall x, y\), \(\exists x, y \in [1, b]\), makes \(n_{xi}.\)pre \(!=\) \(n_{yj}.\)pre \& \& \(n_{xi}.\)suc \(!=\) \(n_{yj}.\)suc , indicating all precursor and subsequent nodes are different. \(\text{simi}(G_k, G'_y)\): indicates precursor and subsequent nodes are partly identical. \(\forall x, y\), \(\exists x, y \in [1, b]\), makes \(n_{xi}.\)pre \(!=\) \(n_{yj}.\)pre \& \& \(n_{xi}.\)suc \(!=\) \(n_{yj}.\)suc . Define degree of similarity between two graphs as the ratio of common node number and maximum number of the two graphs, noted as parameter

\[\mu = \frac{|\text{mcs}(n_x\text{num}, n_y\text{num})|}{|\text{max}(n_x\text{num}, n_y\text{num})|}\]  
(8)

\(\text{Mcs}(\cdot)\) is the node number of the maximum common subgraph, \(\mu \in (0, 1)\). The final classified type of each malicious code is the type of maximum similarity match.

\[\text{Label Class}(G_k) = \max \{ \text{Sim}(G_k, G'_y) \}\]  
(9)

In classification training, set similarity threshold, adjust it, until the classification result of every malicious code is obtained. The attributes of constructed the training set are now to be selected.

B. Feature Selection Based on ReliefF

In the above training sets \(R_1, R_2, R_3, R_4\), based on ReliefF, from \(n\) characteristic vectors in each group, select according to their weights the first \(\frac{2}{3}n\) attributes which play important roles in classification, and get the learning machine \(S_1, S_2, S_3, S_4\) after dimension reduction. Suppose \(w\) is the feature’s weight vector need to be acquired, and \(\text{diff}(F, I_1, I_2)\) represents the difference between instance \(I_1\) and \(I_2\) with respect to feature \(F\), if the selected attributes are discrete, then

\[\text{diff}(F, I_1, I_2) = \begin{cases} 0 & \text{value}(F, I_1) = \text{value}(F, I_2) \\ 1 & \text{otherwise} \end{cases}\]  
(10)

If the attributes are continuous, then
Algorithm 1: Malicious code attributes selection based on weights.

Input: for each training instance a vector of feature values \( X \) and the class value
Output: the vector \( w \) of estimation of the qualities of features

Process:
Set all weights \( W[x] = 0 \).
For \( i = 1 \) to \( m \) do
   Randomly select an instance \( R \).
   Find \( k \) nearest hit \( H \).
   For each class \( C \in \text{class}(R) \) do
      From class \( C \) find \( k \) nearest misses \( M_j(C) \)
      For \( F_{\text{index}} : = 1 \) to \# all features do
         \[ W[F_{\text{index}}] = W[F_{\text{index}}] + \frac{\sum_{i=1}^{m} \text{diff}(F_{\text{index}}, R, H_i)}{(m \cdot k)} - \frac{\sum_{C \in \text{class}(R)} \frac{P(C)}{1-P(C)} \sum_{i=1}^{k} \text{diff}(F_{\text{index}}, R, M_j(C))}{(m \cdot k)} \]
         \[ s_i' = \frac{2}{3} S_i \] // using \( 2/3 \) of the most important features based on \( W[A_i] \)
   end for
end for
end for

where \( m \) is the number of loops to obtain stable results. Search for the nearest two neighbours of instance \( R \). If \( H \) and \( R \) differ with respect to characteristic \( F \), this difference will harm the classification (\( H \) and \( R \) are of the same type), so add the sign "-" before this term. If \( M \) and \( R \) differ with respect to characteristic \( F \), this difference is favourable for classification (\( M \) and \( R \) are of different types), so add the sign "+" before this term. The range of values in result \( W \) is \([-1, 1]\), and a higher value indicates that feature plays a more important role in the classification. Select the important attributes for classification by means of dimension reduced vectors, to enhance the learning efficiency of classification. We now perform boosting on multiple independent classifiers.

C. Boosting Based on Accurate Rate

The classifiers \( S_1, S_2, S_3, S_4 \) are independent from each other. Extract features of the operation code sequence, bytecode sequence, system call, and function call respectively. Integrating the static characteristics and the program's behavioural characteristics in static analysis, conduct weighted voting on multiple classifiers according to accuracy of classification.

The classifiers are \( S_1, S_2, \ldots, S_y \), with each classifier's weight coefficient being \( w_1, w_2, \ldots, w_t \), respectively, and the sum of the weights satisfies \( \sum_{i=1}^{t} w_i = 1 \). For samples of different types, the detecting accuracy of each classifier is \( p_1, p_2, \ldots, p_t \). In order to improve maximally the detecting result of malicious code, \( w_i \) is determined by accuracy rate \( p_i \), and satisfies \( w_i \propto \log \frac{p_i}{1-p_i} \). All classifying rules take part in decision, and each classifying rule along with its weight constitutes the decision function.

Suppose there are \( y \) families of malicious code, denoted as \{ \( f_1, f_2, \ldots, f_y \) \} respectively, and each family contains \( k \) samples. Use various classifiers to vote on the detecting result of samples. If a classifier \( S_i \) determines that sample \( x \) belongs to the family \( f_j \), record the vote as \( \text{vote}_{ij} = 1 \), and the others are recorded as \( 0 \). Then weight the voting result according to accuracy rate:

\[ \sum_{i=1}^{t} (\text{vote}_{ij} \times w_i) = \sum_{i=1}^{t} (\text{vote}_{ij} \times \log \frac{p_i}{1-p_i}) \]

and sample \( x \) is determined to be in the family \( f_j \). If its integrated decision result takes the maximum value.

Decision function is of the form:

\[ F(x) = \max_{i=1,2\ldots t} \left( \sum_{j=1}^{y} (\text{vote}_{ij} \times \log \frac{p_i}{1-p_i}) \right) \]

IV. ALGORITHM ANALYSIS

A. Efficiency Analysis of Feature Selection

We now analyze the impact of attribute vector selection on time complexity. Assume \( y \) types of malicious code families, take \( k \) samples from each one type, the number of attribute vectors of training samples is \( n \), after feature selection, vector number is \( n' (n' < n) \), training time for a sample unit characteristic vector is \( t \). Consider only the impact of vector set on classification efficiency, then the required time before characteristic selection is \( T(R_n) = \sum_{i=1}^{y} \sum_{j=1}^{t} n t \), and the required time after characteristic selection is \( T(S_n) = \sum_{i=1}^{y} \sum_{j=1}^{t} n t \). We employ \( n' = \frac{2}{3} n \) in the experiment, hence the improvement of time efficiency owing to feature selection is

\[ T = T(R_n) - T(S_n) = \sum_{i=1}^{y} \sum_{j=1}^{t} n t - \sum_{i=1}^{y} \sum_{j=1}^{t} n t = \frac{1}{3} y k n t \]

It is shown that, with the reduction of vector dimension, the time complexity decreases linearly. But the amount of malicious code families and samples in each family is large, so the coefficient of the decreasing function is also large, and the decreasing speed is high.

B. Analysis of Detection Accuracy

Based on boosting of weight \( w_i \propto \log \frac{p_i}{1-p_i} \), we now analyze the impact of integrated learning of multiple classifiers on the detection accuracy. Firstly, we explain the reason to set weight value \( w_i \), and then we prove that the integrated learning based on this weight is superior to the single or double classifier approach.

Assume multiple classifiers \( S_i \) and \((i \in [1, t])\) independent with each other, there are \( y \) families of malicious codes, denoted as \{ \( f_1, f_2, \ldots, f_y \) \} respectively, and each family contains \( K \) samples. If a classifier \( S_i \) determines that sample \( x \) belongs to the family \( f_j \), record the vote as \( \text{vote}_{ij} = 1 \), and the others are recorded as \( 0 \). Use \( X = [x_1, x_2, \ldots, x_t] \), to represent the vector of...
detecting results when classifiers \( S_1, S_2, \ldots, S_k \) determine that sample \( x \) belongs to the family \( f_j \). Suppose the attributes are conditionally independent, the Bayes optimal decision function of \( t \) classifiers.

\[
P(x|f_j) = \log P(f_j) \prod_{k=1}^{t} P(x_k|f_j)
\]

\[
= \log P(f_j) + \log \prod_{x_i \neq f_j} \left( \frac{1 - p_i}{1 - p_i} \right)
\]

\[
= \log P(f_j) + \log \prod_{x_i \neq f_j} (1 - p_i)
\]

\[
= \log P(f_j) + \sum_{x_i \neq f_j} \log (1 - p_i)
\]

Because \( \sum_{i=1}^{t} \log (1 - p_i) \) is not related to a classifier's decision, the function can be written as \( P(x|f_j) = \log P(f_j) + \sum_{x_i \neq f_j} \log \frac{1 - p_i}{1 - p_i} \).

By comparing to decision function

\[
F(x) = \sum_{i=1}^{t} (\text{vote}_{ij} \times w_i)
\]

We find that \( w_i \propto \log \frac{1 - p_i}{1 - p_i} \). Assume three classifiers with weights \( w_1 \geq w_2 \geq w_3 \), respectively, which satisfy \( \sum_{i=1}^{3} w_i = 1 \). From Pigeonhole Principle, we get \( w_1 \geq \frac{1}{3} \geq w_2 \geq w_3 \) or \( w_1 \geq w_2 \geq \frac{1}{3} \geq w_3 \) (the two are analogous, so the only one is proved below).

For \( w_1 \geq \frac{1}{3} \geq w_2 \geq w_3 \), there are two cases:

(a) when \( w_1 \geq w_2 + w_3 \), classification accuracy is

\[
p_1(1 - p_2)(1 - p_3) + p_1p_2(1 - p_3) + p_1p_3(1 - p_2) + p_1p_2p_3 = p_1,
\]

and the classification accuracy rate is the accuracy rate of a single type under optimal weights.

(b) when \( W_1 < W_2 + W_3 \), we have \( \log \frac{1 - p_2}{1 - p_2} + \log \frac{p_2}{1 - p_2} > \log \frac{1 - p_3}{1 - p_3} \). Thus \( \frac{p_2}{1 - p_2} = \frac{p_3}{1 - p_3} \). Compare multi-classifier of optimal selection to the single or double classifier and get the difference of accuracy rate \( p_1 \):

\[
\text{sub}= p_1p_2(1 - p_3) + p_1p_3(1 - p_2) + p_2p_3(1 - p_1) + p_1p_2p_3 - p_1 = \prod_{i=2}^{3} (1 - p_i) \left( \prod_{i=2}^{3} \left( \frac{p_i}{1 - p_i} - \frac{p_i}{1 - p_i} \right) \right) > 0.
\]

Consequently, multi-classifier based on accuracy rate weighted votes obtains better detecting result than the cases of single or double attribute type.

V. EXPERIMENTS AND COMPARISONS

Choose 450 malicious codes and 400 normal codes as training sample set. All malicious codes are from the website http://vx.netlux.org, and normal codes are all retrieved from a computer with newly installed Window XP. Malicious code families chosen in our experiments count to 90, and 5 samples are selected from each family. When extracting features, change the size of slide window \( n \), and extract and train the features and frequencies of operation code (OperCode_Sequ) and bytecode (ByteCode_Sequ). Based on the distance between the sample's directed graph and common subgraph, calculate the similarity of system-call (System_Call) and function call (Function_Call) graphs. Devise different threshold intervals to divide different malicious code families. Training sets are divided into two groups: one is sets of normal samples and malicious code samples, and the other is malicious code samples of each family and malicious code samples from other families. Since the classifier exhibits low performance in cross validation, we increase the number of samples through obfuscating transform on samples of the same family, so as to balance the training data.

Suppose the number of code samples to be tested is \( N \), then \( N = TP + TN + FP + FN \). Herein, \( TP \) (True Positive) is number of malicious codes that have been classified correctly, \( FP \) (False Positive) is number of normal codes recorded as malicious codes wrongly, \( TN \) (True Negative) is number of normal codes correctly classified, and \( FN \) (False Negative) is number of malicious codes recorded as normal codes wrongly. Three evaluation indexes are used: accuracy rate \( R_{\text{Accuracy}} = (TP + TN) / N \), namely proportion of correctly classified codes in all samples under test. detecting rate \( R_{\text{TP}} = TP / (TP + FN) \), namely proportion of correctly classified malicious codes in all malicious codes within all samples under test. false positive rate \( R_{FP} = FP / (TN + FP) \), namely the proportion of normal codes wrongly classified as malicious codes in all normal code samples.

| Table 1: Detection Results Based on Different Segmenting Methods (N=3, trie_d=4) |
|-----------------------------|------------------|------------------|------------------|
| Algorithms / Metric         | feature number   | R_accuracy       | R_TP             | R_FP             |
| n-gram                      | 8,109,318        | 0.7065           | 0.7118           | 0.1521           |
| n-perm                      | 5,609,562        | 0.7391           | 0.7492           | 0.1242           |
| variable-length n-gram      | 7,351,725        | 0.8014           | 0.8094           | 0.0873           |
| V-L n-gram after ReliefF    | 5,018,475        | 0.8315           | 0.8324           | 0.0704           |

We analyze the impact of different segmenting methods on detecting results, test respectively and compare the impacts of n-gram [9] and n-perm [10] and variable-length n-gram (in this paper) on classification accuracy. Adjust the value of the sliding window \( n=1,2,3,4,5,6 \) and trie depth \( d=3,4,5 \), to make training results satisfy the objective function in section3.1.1. In training, while the sliding window size increases, the accuracy rates of these three segmenting methods vary from small to large and then from large to small. Within the range, when selecting the optimal parameters \( n=3, \) \( trie_d=4 \), these detecting data satisfying optimal objective function from Table 1. Table II shows that the detection results based on different thresholds. In classification training, adjusting these thresholds until the classification result of every malicious code is obtained according to the Section 3.1.2. then we compare the
average efficiency of the detection algorithm of multidimensional feature to the other two single-dimensional algorithms, OperCode_Sequ and system_call in Fig. 3. Finally, the multi-dimensional test results compared with the existing anti-virus software, shown in the Fig. 4 after testing 850 samples (450 malicious code and 400 normal code).

| TABLE II: DETECTION RESULTS BASED ON DIFFERENT THRESHOLDS (Threshold 1 INDICATES PARAMETERS OF FUNCTION CALL GRAPH AND THRESHOLD 2 INDICATES PARAMETERS OF SYSTEM CALL GRAPH) |
|---------------------------------|-----------------|-------|-------|
| (Threshold1, Threshold2)        | R_accuracy      | R_TP  | R_FP  |
| (0.3, 0.2)                      | 0.6807          | 0.6951| 0.1546|
| (0.5, 0.4)                      | 0.7524          | 0.7894| 0.1153|
| (0.7, 0.6)                      | 0.8683          | 0.8861| 0.0675|
| (0.9, 0.8)                      | 0.7911          | 0.8092| 0.0952|

From Fig. 3, the detection results of multidimensional features have obvious advantages than test results of single attribute algorithm, but the training time and testing for a long time, in practice, should make a trade-off in terms of accuracy and efficiency. This algorithm has a high accuracy rate and a low false positive rate in detecting polymorphic malicious code, and can detect the obfuscated code that cannot be detected by the commercial anti-virus software (Norton, Rav2013) shown in Fig. 4.

VI. CONCLUSION

To solve the problem of low detecting rate in malicious code detection, this paper proposes a low-overhead method to improve the detecting rate. Using static analysis of codes and static "behavioural" characteristics, we identify obfuscated malicious codes by classification learning of malicious code families. This method improves classification efficiency by feature selection based on ReliefF, and increases detecting accuracy by weighted voting according to accuracy. Consequently solves key problems in feature extraction, such as feature sequence segmentation and measurement of feature graph. It is an improvement of existing algorithms. In the next stage of research, we will apply clustering [14] and Rough Set to expand the selected range of feature types, and solve the problem of the correlation between features. Additionally, combine existing static analysis and dynamic program slicing, and deeply analyze and learning behaviours of malicious codes, so as to further increase the accuracy of detection.

ACKNOWLEDGEMENTS

This work was supported in part by Natural Science Foundation of Shaanxi Province of China (No.2014 JM2-6100) and Scientific Research Program Funded by Shaanxi Provincial Education Department (15JK1274).
REFERENCES


Yang-Xia Luo was born in Shaanxi Province. Female. Xian Institute of finance and information Institute, Associate Professor. She received the Ph.D. degree from the Information College of Northwestern University. Research on Data Mining and Pattern recognition. E-mail: yxluo8836@163.com.