Unknown Malicious Identification: an Improved Solution Based on Bayesian Analysis

Ying-xu. LAI, Zeng-hui. LIU

***Abstract*—The detection of unknown malicious executables is beyond the capability of many existing detection approaches. Machine learning or data mining methods can identify new or unknown malicious executables with some degree of success. Feature set is a key to apply data mining or machine learning to successfully detect malicious executables. In this paper, we present an approach that conducts an exhaustive feature search on a set of malicious executables and strives to obviate over-fitting. To improve the performance of Bayesian classifier, we present a novel algorithm called Half Increment Naïve Bayes(HIB), which selects the features by carrying an evolutional search. We also evaluate the predictive power of a classifier, and we show that our classifier yields high detection rates and learning speed.**

***Index Terms*—unknown malicious detection; Half Increment Naïve Bayes; classification**

I. Introduction

As network-based computer systems play increasingly vital roles in modern society, a serious security risk is the propagation of malicious executables. Malicious executables include viruses, Trojan horses, worms, back doors, spyware, Java attack applets, dangerous ActiveX and attack scripts. Identifying malicious executables quickly is an important goal, as they can cause significant damage in a short time. Consequently detecting the presence of malicious executables on a given host is a crucial component of any defense mechanism.

Traditional malicious executables detection solutions use signature-based methods, in that they use case-specific features extracted from malicious executables in order to detect those same instances in the future [1]. Security products such as virus scanners are examples of such application. While this method yields excellent detection rates for existing and previously encountered malicious executables, it lacks the capacity to efficiently detect new unseen instances or variants. Due to detect malicious accurately is a NP problem [2-3], heuristic scanners attempt to compensate for this lacuna by using more general features from viral code, such as structural or behavioral patterns [4]. Although proved to be highly effective in detecting unknown malicious executables, this process still requires human intervention.

Recently, attempts to use machine learning and data mining for the purpose of identifying new or unknown malicious executables have emerged. Schultz et al. examined how data mining methods can be applied to malicious executables detection [5] and built a binary filter that can be integrated with email servers. Kolter et al. used data mining methods, such as Naïve Bayes, J48 and SVM to detect malicious executables [6]. Their results have improved the performance of these methods. Bayes or improved Bayes algorithm has the capability of unknown malicious detection, but it spends more time to study. A new improved algorithm (half-increment Bayes algorithm) is proposed in this paper.

In this paper, we are interested in applying data mining methods to malicious executables detection, and in particular to the problem of feature selection. Two main contributions will be made through this paper. We will show how to choose features which are most representative properties. Furthermore, we propose a new improved algorithm and will show that our method achieve high learning speed and high detection rates, even on completely new, previously unseen malicious executables.

The rest of this paper is organized as follows: Section 2 is a brief discussion of related works. Section 3 gives a brief description of Bayesian algorithm. Section 4 presents details of our methods to obtain high learning speed. Section 5 shows the experiment results. Lastly, we state our conclusions in Section 6.

II. Related Work

At IBM, Kephart et al. [7] proposed the use of Neural Networks to detect boot sector malicious binaries. Using a Neural Network classifier with all bytes from the boot sector malicious code as input, it had shown that 80%-85% of unknown boot sector malicious programs can be successfully identified with low false positive rate (<1%). The approach for detecting boot-sector virus had incorporated into IBM's Anti-Virus software. Later, Arnold et al. [8,9] applied the same techniques to win32 binaries. Motivated by the success of data mining techniques in network intrusion system [10,11], Schultz et al. [5] proposed several data mining techniques to detect different types of malicious programs, such as RIPPER, Naïve Bayes and Multi-Naïve Bayes. The authors collected 4,301 programs for the Windows operating system and used

Manuscript received April 25, 2008. This work was supported in part by the Beijing Committee of Education under Grant KM20081005030 and the Youthful Scholar in Beijing University of Technology under Grant 97007011200603.

Yingxu. LAI. Author is with the College of Computer and Science, Beijing University of Technology, Beijing 100124, China (phone: 86-10-67396063; e-mail: laiyingxu@ bjut.edu.cn).

Zenghui. LIU. Author is with the Science and Technology Engineering Faculty, Beijing Vocational College of Electronic Science, Beijing 100029, China (phone: 86-10-84613274; e-mail: zenghuiliu@ sohu.com).

MacAfee Virus Scan to label each as either malicious or benign. The authors concluded that the voting naïve-Bayesian classifier outperformed all other methods. In a companion paper [12] the authors developed an Unix mail filter that detect malicious Windows executables based on the above work. Kolter et al. [6] also used data mining methods, such as Naïve Bayes, J48 and SVM to detect malicious codes. The authors gathered 1,971 benign and 1,651 malicious codes and encoded each as a training example using n-grams of byte codes as feature, boosted decision trees outperformed other methods with an area under the ROC curve of 0.996. (applied detectors to 291 malicious executables discovered, and boosted decision trees achieved a TP rate of 0.98 for a desired FP rate of 0.05.) Zhang et al. [13] used SVM and BP neural network to virus detection, the D-S theory of evidence was used to combine the contribution of each individual classifier to give the final decision. It showed that the combination approach improves the performance of the individual classifier significantly. Zhang et al. [14,15] established methods based on fuzzy pattern and K-nearest neighbor recognition applying to detect malicious executables for the first time.

There are other methods of guarding against malicious code. Static analysis, where analysis of program is done without executing it, is attempted in [12,13]. Dynamic analysis which combines testing and debugging to detect malicious activities by running a program includes wrappers [14], sandboxing [15], etc. Inspired by the phenomenon in the mid-1990s, some computer-virologist proposed an immune strategy on single-chipped computers, which uses virus protection software to simulate immune phenomenon [16,17]. These approaches are not based on data mining, although one could imagine the role such techniques might play.

III. Naive Bayesian and application

The goal of our work was to improve a standard data mining technique to compute accurate detectors for new binaries. We gathered a large set of programs from public sources and separated the problem into two classes: malicious and benign executables. We split the dataset into two subsets: the training set and the test set. The data mining algorithms used the training set while generating the rule sets. We used a test set to check the accuracy of the classifiers over unseen examples.

In a data mining framework, features are properties extracted from each example in the data set—such as strings or byte sequences. A data mining classifier trained with features can use to distinguish between benign and malicious programs. We used strings that are extracted from the malicious and benign executables in the data set as features.

We propose an exhaustive search for strings. Typically there are many "disorder-words" when files are read as ASCII code, like "autoupdate.exe飛撞膜岹?software*abbbbbb膜岁download", etc. The string selection program extracts consecutive printable characters from files. To avoid yielding a large number of unwanted features, characters are filtered before they are recorded. Our feature selection involves an extraction step followed by an elimination step.

A. String Extraction and Elimination

In string extraction step, we scan files and read character one by one, record consecutive printable characters (like English letter, number, symbolic and etc.), and construct lists of all the strings. The character string extraction algorithm is shown in Algorithm 1, by adjusting string length. The length of the string is specified as a number of characters. The shorter the length, the more likely the feature is to have general relevance in the dataset. But a short length will yield a larger number of features.

Algorithm 1. String extraction (*B,V, len*)

Input: A non-empty set of benigns, *B*.
Input: A non-empty set of malicious executables, *V*.
Input: The word set, *W*, as a filter.
Input: A non-zero length, *len*.
Output: A set features, *S1*, representing common characteristics of malicious executables in *V*
Output: A set features, *S2*, representing common characteristics of benign executables in *B*
1: for (each malicious V_i in *V*) do
2: for (each character in V_i) do
3: record a character found in *W*, otherwise discard off.
4: record consecutive printable characters in a buffer.
5: record all strings of length less than *len* found in V_i
6: build the set of features *S1* of strings with length less than *len*
7: return *S1*
8: for (each virus B_i in *B*) do
9: for (each character in B_i) do
10: record a character found in *W*, otherwise discard off.
11: record consecutive printable characters in a buffer.
12: record all strings of length less than *len* found in B_i
13: build the set of features *S2* of strings with length less than *len*
14: return *S2*

Extracted strings from an executable are not very robust as features because some strings are unmeaning, like "abbbbbb", so we select a subset of strings as feature set by an eliminate step.

Many have noted that the need for a feature filter is to make use of conventional learning methods[13,14], to improve generalization performance, and to avoid over-fitting. Following the recommendation of those, the glossary filter criterion is used in the paper to select a subset of strings.

The glossary is a computer glossary which includes 7,336 items:

- Computer words, such as function name, API name.
- Abbreviations on computer networks, like "QQ", "msn", etc.
- Postfixs, like ".dll", presenting dynamic link libraries.

B. Naive Bayes

The naive Bayes classifier computes the likelihood that a program is malicious given the features that are contained in the program. We treat each executable's features as a text

document and classified based on that. Specifically, we want to compute the class of a program given that the program contains a set of features *F*. We define *C* to be a random variable over the set of classes: benign, and malicious executables. That is, we want to compute *P(C|F)*, the probability that a program is in a certain class given the program contains the set of features *F*. We apply Bayes rule and express the probability as:

$$P(C|F)=\frac{P(F|C)\times P(C)}{P(F)} \quad (1)$$

To use the naïve Bayes rule we assume that the features occur independently from one another. If the features of a program F include the features $F_1, F_2, F_3 ..., F_n$, then equation (1) becomes:

$$P(C|F)=\frac{\prod_{i=1}^{n} P(F_i|C)\times P(C)}{\prod_{j=1}^{n} P(F_j)} \quad (2)$$

Each $P(F_i \mid C)$ is the frequency that string F_i occurs in a program of class *C*. *P(C)* is the proportion of the class *C* in the entire set of programs.

The output of the classifier is the highest probability class for a given set of strings. Since the denominator of (1) is the same for all classes we take the maximum class over all classes *C* of the probability of each class computed in (2) to get:

$$\text{Most Likely Class}=\max_{C}\left(P(C)\prod_{i=1}^{n} P(F_i \mid C)\right) \quad (3)$$

In (3), we use max_C to denote the function that returns the class with the highest probability. Most Likely Class is the class in *C* with the highest probability and hence the most likely classification of the example with features *F*.

IV. Increment Naïve Bayes

A. Naive Bayes(NB)

To train the classifier, we record how many programs in each class contained each unique feature. We use this information to classify a new program into an appropriate class. We first use feature extraction to determine the features contained in the program. Then we apply equation (3) to compute the most likely class for the program.

The Naïve Bayes algorithm requires a table of all features to compute its probabilities. This method requires a machine with one gigabyte of RAM, because the size of the binary data was too large to fit into memory.

To update the classifier, when new programs are added to the training set, we update feature set at first, and apply equation (3) to compute the most likely class for the program again. So it is time-consuming to update the classifier by NB algorithm.

B. Multi-Navie Bayes(MNB)

To correct NB algorithm problem the training set is divided to smaller pieces that would fit in memory. For each set we train a Naïve Bayes classifier. Each classifier gives a probability of a class *C* given a set of strings *F* which the Multi-Naïve Bayes uses to generate a probability for class *C* given *F* over all the classifiers.

For each classifier, the probabilities in the rules for the different classifiers may be different because the underlying data the each classifier is trained on is different. The prediction of the Multi-Naïve Bayes algorithm is the product of the predictions of the underlying Naïve Bayes classifier.

$$P(C|F)=\prod_{k=1}^{n} P_k(C|F) \quad (4)$$

When new programs are added to the training set, these new programs as a subset and train a Naïve Bayes classifier over the subproblem. Based on the equation (4), we update the probability. The Multi-Naïve Bayes algorithm does not need to compute its probabilities over all the training set, but the accuracy of the classifier will be worsen.

C. Half Increment Bayes(HIB)

The above NB and MNB algorithms obtain feature set over all training set or subset at first. Once the final feature set was obtained, we represent our positive and negative data in the feature space by using, for each feature, "1" or "0" to indicate whether or not the feature is present a given executable file. The probability for a string occurring in a class is the total number of times it occurred in that class's training set divided by the total number of times that the string occurred over the entire training set.

We can derive a method purely from the NB algorithm for increment update. In our method, feature set is increased with studying of classifier. That is, composed there are k_1 string features extracted from the first sample, so k_1 strings are elements of set *F*. If there are S_2 strings extracted from the second sample, k_2 elements are not found in *F*, these elements should be added in feature set *F*. Set *F* will includes $k_1 + k_2$ elements. Classifier is trained based on the evolutional feature set.

Claim 1: We can obtain the class-conditional probability $P(F^{(n+1)}|C)$ over *n*+1 samples by that of former *n* samples and the (*n*+1)th sample, that is $P(F_i^{(n+1)}|C_j)=\frac{\left[P(F_i^{(n)}|C_j)\times n+P(x^{(n+1)}|C_j)\right]}{n+1}$.

Where, $P(F^{(n+1)}|C)$ is the class-conditional probability over *n*+1 samples, $P(F^{(n)}|C)$ is class-conditional probability over *n* samples, $P(x^{(n+1)}|C)$ is class-conditional probability over the (*n*+1)th sample.

Proof:

Composed there are *a* features obtained from *n* samples, that is $F^{(n)}=\{F_1, F_2 ..., F_a\}$, then

$$P(F^{(n)}|C)=\prod_{i=1}^{a} P(F_i|C_j) \text{ (j=1,2)} \quad (4$$

$$P(F_i|C_j)=\frac{P(F_iC_j)}{P(C_j)}=\frac{count(F=F_i \wedge C=C_j)}{count(C=C_j)} \quad (5$$

Where, $count(F=F_i \wedge C=C_j)$ is sample number for $F=F_i$ and class $C=C_j$, $count(C=C_j)$ is sample number for class $C=C_j$.

If there are n samples with $C = C_j$ in training set, then the class-conditional probability $P\left(F^{(n)}\middle|C\right)$ for n samples is:

$$P\left(F_i^{(n)}\middle|C_j\right)=\frac{count((F=F_i^{(n)})\wedge(C=C_j))}{n} \quad (6)$$

If the (n+1)th sample for class $C = C_j$ was added, there are two cases.

Case 1: the strings in the (n+1)th sample are all found in F, then

$$P\left(F_i^{(n+1)}\middle|C_j\right)=\frac{count((F=F_i^{(n)})\wedge(C=C_j))}{n+1} \quad (7)$$

Case 2: there are b strings in the (n+1)th sample are not found in F, then these strings are added in F, that is $F^{(n+1)}=\{F_1,F_2...,F_a,F_{a+1},\cdots,F_{a+b}\}$. For those new b probabilities, due to $P\left(F_i^{(n)}\middle|C_j\right)=0 \quad a<i\le a+b$, then

$$P\left(F_i^{(n+1)}\middle|C_j\right)=\frac{1}{n+1} \quad a<i\le a+b \quad (8)$$

We rewrite (7) and (8) as (9):

$$P\left(F_i^{(n+1)}\middle|C_j\right)=\frac{\left[P\left(F_i^{(n)}\middle|C_j\right)\times n+P\left(x^{(n+1)}\middle|C_j\right)\right]}{n+1} \quad (9)$$

Therefore, information of n+1 samples can obtained from those of former n samples and the (n+1)th sample. □

Claim 2: For NB and HIB, based on same training set, same feature sets can be obtained by same string extraction and elimination methods, that is F_N=F_H. Where F_H is feature set obtained by half increment algorithm, F_N is feature set obtained by naïve bayse algorithm.

Proof:

Composed there are a features obtained from n samples, that is $F_N=\{F_1,F_2...,F_a\}$, Set F_N is made of strings that extracted from n samples and eliminated based on computer dictionary.

F_H is increased as training samples. When all of training samples are all extracted, Set F_H is made of strings that extracted from n samples and eliminated based on computer dictionary. So F_N=F_H. □

Based on Claim 1 and 2, we can obtain Theory 1.

Theory 1: Most Likely Classes computed by half increment classification and Naïve Bayes classification are same, that is $C_1 = C_2$. Where C_1 is the most likely class obtained by naïve Bayes, and C_2 is the most likely class obtained by half-increment classification.

Proof: Composed a features were obtained from n samples, that is $F_N=\{F_1,F_2...,F_a\}$, Based on equation (3),

$$C_1=\max\left(P(C)\prod_{i=1}^{a}P(F_i\mid C)\right) \quad (10)$$

When a new sample is added, $F_N=\{F_1,F_2...,F_a,F_{a+1},\cdots,F_{a+b}\}$ the most likely class computed by naïve Bayes is

$$C_1=\arg\max P(C)\times\prod_{i=1}^{a+b}P\left(F_i\middle|C\right) \quad (11)$$

As F_N=F_H, based on equation (9)

$$C_2=\arg\max P(C)\times\prod_{i=1}^{a+b}P\left(F_i\middle|C\right) \quad (12)$$

Therefore, the $C_1 = C_2$.

The HIB algorithm is shown in Algorithm 2.

Algorithm 2. Half Increment Bayes Algorithm (D)

Input: A non-empty training set, D.
Input: A Feature set, F.
Output: A classifier
1: for (each sample D_i in D) do
2: for (each string s_j)
3: if found in F go to 4, otherwise record unrepeated Strings and update set F.
4: Training classifier
5: return classifier

D. Complexity

Based on former algorithm, time-consuming of NB and HIB are made of two parts:

(1) extract unrepeated strings from samples.

(2) fix on feature set and training set, build up classifier.

For step (1), time-consuming of two algorithms are same. But for step (2), they are different.

For HIB, feature set is increased with studying of classifier. That is, composed there are S_1 strings in the first sample, and k_1 strings are unrepeated, so k_1 strings are elements of set F. If there are S_2 strings in the second sample, the time-consuming of computing whether k_1 elements are found in S_2 or not is $T_{H2}=O(S_2\times k_1)$. If k_2 elements are not found in F, these elements should be added in feature set F. Set F will includes $k_1 + k_2$ elements. For n samples, the all time-consuming of fix on set F is

$$T_H=O(S_2\times k_1+S_3\times(k_1+k_2)+...+S_n\times(k_1+k_2+...+k_r)) \quad (13)$$

For NB, the set F are obtained before classifier study. Composed there are K elements in F, the time consuming to indicate whether or not the feature is present a given executable file is

$$T_N=O(K\times(S_1+S_2+...+S_n)) \quad (14)$$

Based on Claim 2, F_N=F_H, $K=k_1+k_2+...+k_r$.

Based on equation (13),

$$T_H=O(K\times(S_2+S_3+...+S_n)-S_2\times(k_2+...+k_r)-...-S_{n-1}\times k_r) \quad (15)$$

If $k_2=k_3=...=k_r=0$, or $K=k_1$, $T_H=T_N$, the time-consuming of two algorithm are same. But malicious executables include viruses, Trojan horses, worms, back doors, spyware, Java attack applets, dangerous ActiveX and attack scripts, a sample cannot include all feature elements.

V. Experimental Results

A. Experimental Design

Our experiment are carried out on a dataset of 2995 examples consisting of 995 previously labeled malicious executables and 2000 benign executables, collected from desktop computers using various versions of the Windows operating system. The malicious executables are taken from

an anti-Virus software company.

For each run, we extract strings from the executables in the training and testing sets. We select the most relevant features from the training data, apply elimination method, and use the resulting classifier to rate the examples in the test set.

B. Performance Analysis

1) Time-consuming

In our experiments, we used VC++ implementation of the NB, MNB and HIB classifiers. In this section, we evaluate the performance of NB, MNB and HIB three algorithms in comparison.

For HIB algorithm, increasing rule of feature elements is based on order of studied samples. Figure 1 is the curve between number of studied samples and number of feature elements. In Figure 1, the slope of the curve is steep at some points. When classifier begins to study some type sample, the slope of the curve is steep, that is, the number of feature elements increase quickly. After classifier has studied some samples, the slope of the curve is smooth.

Figure 2 is the curve of three algorithm's time-consuming. The slope of NB algorithm is initially much steeper than the HIB and MB algorithms. HIB algorithm has better efficiency.

The ROC curves in Figure 3 show a more quickly growth than the NB and MNB until the false positive rate climbed above 4%. Then the three algorithms converged for false positive rates greater then 6% with a detection rate greater then 95%.

2) Comparison Against Benchmark Model

To evaluate our system we are interested in several quantities:

1. True Positives (TP), the number of malicious executable examples classified as malicious code
2. True Negatives (TN), the number of benign programs classified as benign
3. False Positives (FP), the number of benign programs classified as malicious code
4. False Negatives (FN), the number of malicious codes classified as benign.

The Detection Rate is defined as $\frac{TP}{TP+FN}$, False Positive Rate as $\frac{FP}{TN+FP}$, and Overall Accuracy as $\frac{TP+TN}{TP+TN+FP+FN}$.

We compare our method with a model used in previous research [7,15]. The results, displayed in Table 1, indicate that a virus classifier can be made more accurate by using features representative of general viral properties, as generated by our feature search method. With up to 97.9% overall accuracy, our system outperforms NB and MNB algorithms and achieves better results than some of the leading research in the field, which performs at 97.11%.

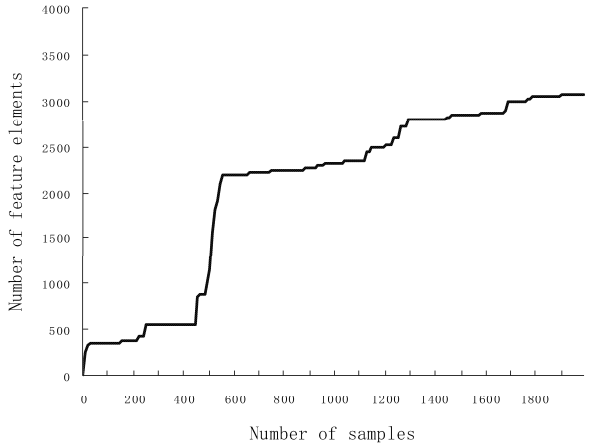

Figure 1: Curve between number of feature elements and number of samples

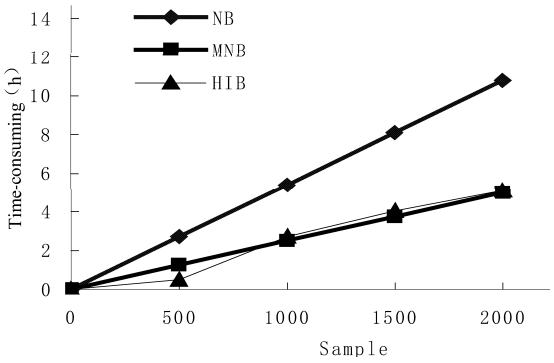

Figure 2: NB, MNB and HIB curves of Time-consuming

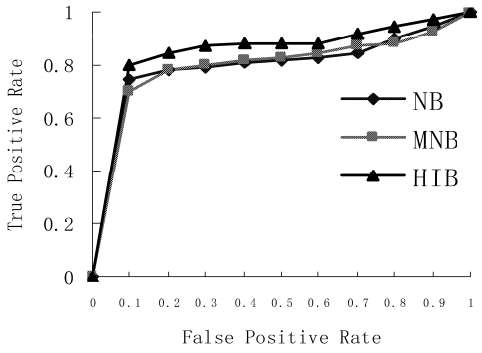

Figure 3: NB, MNB and HIB ROC

Table 1 Experimental results using our method and traditional methods

Method	Feature	Classifier	Accuracy
Our method	Strings	NB	96.8%
Our method	Strings	MNB	93.3%
Our method	Strings	HIB	97.9%
Schultz	String	NB	97.11%
Schultz	Bytes	MNB	96.88%
Kolter	4-gram	SVM	93%
Henchiri	8-gram	J48	93.65%

VI. CONCLUSION

The naïve Bayes classifier is widely used in many classification tasks because its performance is competitive

with state-of-the-art classifiers, it is simple to implement, and it possesses fast execution speed. In this paper, we discussed the problem of how to classify a set of query vectors from the same unknown class with the naïve Bayes classifier. Then, we propose the method HIB algorithm and compare it with naïve Bayes and multi-naïve Bayes. The experimental results show that HIB algorithm can take advantage of the prior information, can work well on this task. Finally, HIB algorithm was compared with a model used in previous research [7,15]. Experimental results reveal that, HIB can reach a higher level of accuracies as 97.9%. HIB's execution speed is much faster than MNB and NB, and HIB has low implementation cost. Hence, we suggest that HIB is useful in the domain of unknown malicious recognition and may be applied to other application.

ACKNOWLEDGMENT

Our thanks to Dr. Ling Qiu for his valuable comments.

REFERENCES

[1] G. McGraw, G. Morrosett, "Attacking malicious code: a report to the infosec research council", *IEEE Trans. Software*, vol. 2, Aug. 1987, pp. 740–741 2000.

[2] F. Cohen, "Computer Virues Theory and Experiments", *Computers &Security*, vol. 6, Jan. 1987, pp. 22–35.

[3] D.Spinellis, "Reliable Identification of Bounded-Length Viruses Is NP Complete", *IEEE Transactions on Information Theory*, vol. 49, Jan. 2003, pp. 280–284.

[4] MacAfee. Homepage-MacAfee.com. Online Publication,2000. http://www.macafee.com

[5] M.G. Schcltz, E. Eskin, E. Zadok, S.J.Stolfo, Data mining methods for detection of new malicious executables, *In: Proceedings of IEEE Symposium in Security and Privacy*, 2001.

[6] J.Z. Kolter, M.A.Mloof, Learning to detect malicious executables in the wild, *In: proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining*, ACM Press, New York, 2004, pp. 470–478.

[7] J. O. Kepahart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Teauro, S.R. White, Biologically inspired defenses against computer viruses. *In: Proceedings of IJCAI '95*, Montreal, 1995, pp. 985–996.

[8] W. Arnold, G.Tesauro, Automatically generated Win32 heuristic virus detection, *In: Proceedings of the 2000 International Virus Bulletin Congerence,* 2000.

[9] G. Tesauro, J.O. Kephart and G.B.Sorkin. Neural networks for computer virus recognition, *IEEE Expert*, vol.11, Apr. 1996, pp.5–6.

[10] W. Lee, S. Stolfo, K. Mok, A Data Mining Framework for Building Intrusion Detection Models, *IEEE Symosium on Security and Privacy*, 1999.

[11] W. Lee, S.J. Stolfo, P.K. Chan, Learning patterns from UNIX processes execution traces for intrusion detection, *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management,* AAAI Press, 1997, pp. 50–56.

[12] M.G. Schcltz,, E. Eskin, E. Zadok, M. Bhattacharyya, S.J. Stolfo, MEF: Malicious Email filter, A Unix mail filter that detects malicious windows executables, *In: Proceeding of USENIX Annual Technical Conference,* 2001.

[13] B. Zhang, J. Yin, J. Hao, Unknown computer virus detection based on multi-naive Bayes algorithm. *Computer Engineering*, vol.32, Oct. 2006, pp. 18–21

[14] B. Zhang, J. Yin, J. Hao, Using fuzzy pattern recognition to detect unknown malicious executables code, *Lecture Notes in Computer Science,* vol. 36, Mar. 2005, pp. 629–634.

[15] B. Zhang, J. Yin, D. Zhang, J. Hao, Unknown computer virus detection based on K-nearest neighbor algorithm, *Computer Engineering and Applications*, vol.6, 2005, pp.7–10.

[16] P. R. Kerchen, J. Crossley, G. Elikinbard, R. Olssopn, Static analysis virus detection tools for unix systems. *In: 13th National Computer Security Conference,* 1990.

[17] J.O. Kephart, W.C.Arnold, Automatic extraction of computer virus signatures, I*n: 4th virus Bulletin International Conference,* 1994, pp. 78–184.

[18] O. Henchiri, N. Japkowicz, A feature selection and evaluation scheme for computer virus detecting. *In: Proceedings of the 6th International Conference on Data Mining (ICDM'06)* ,2006.

[19] T.A. Assaleh, N. Cercone, V. Keselj, R. Sweidan, Detection of new malicious code using N-grams signatures, *In: Proceedings of the second annual conference on privacy, Security and Trust*, 2004, pp. 193–196.

[20] D.S. Reddy, S.K. Dash, A.K. Pujari, New malicious code detecting using variable length n-grams. *In: Proceedings of ICISS*, 2006, pp. 276–288.

[21] H. Huang, C. Hsu, Bayesian Classification for Data From the Same Unknown Class, *IEEE Transactions on System, MAN, and Cybernetics—Part B: Cybernetics*, vol. 32, Feb. 2002, pp. 137–145.

[22] D. Zeng, S. Zhang, Z. Cai, S. Jiang, L .Jiang, Augmented Naive Bayes Based on Evolutional Strategy, *In : Proceedings of the Sixth International Conference on Intelligent Systems Design and Applicacion(ISDA'06),* 2006.