On the Robustness of Machine Learning Based Malware Detection Algorithms

Weiwei Hu and Ying Tan
Key Laboratory of Machine Perception (MOE), and Department of Machine Intelligence
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871 China
{weiwei.hu, ytan}@pku.edu.cn

***Abstract*—With the rapid popularity of the Internet, a large amount of new malware is produced every day, while the traditional signature based malware detection algorithm is unable to detect such unseen malware. In recent years, many machine learning based algorithms have been proposed to detect new malware, and several of these algorithms are able to achieve quite good detection performance when supplied with plenty of training data. However, most of these algorithms just focus on how to improve the classification performance, while the robustness is not taken into consideration. This paper performs a detailed analysis on the robustness of four well-known machine learning based malware detection approaches, i.e. the DLL and API feature, the string feature, PE-Miner and the byte level N-Gram feature. We proposed two pretense approaches under which malware is able to pretend to be benign and bypass the detection algorithms. Experimental results show that the performances of these detection algorithms decline greatly under the pretense approaches. The lack of robustness makes these algorithms unable to be used in real world applications. In future works of machine learning based malware detection, researchers have to take the problem of robustness seriously.**

I. Introduction

Malware is the program which damages the computer systems or steals sensitive information from computers. There are huge amounts of new malware appearing every day and they pose a great threat to the computer security. According to the statistics provided by AV-TEST [1], the numbers of new malware appeared in 2014 and 2015 both exceed 140 million. Effective malware detection algorithms are required to protect computer systems.

Signature based approach is the most popular method used by antivirus software. Human experts extract a small piece of binary code from a malicious program and regards the piece as the unique signature of the program. If an incoming program contains the signature, the antivirus software will identify it as malware. Signature based approach is very efficient to detect known malware with a low false positive rate. However, for new malware the signature usually cannot be extracted in time and in such case the antivirus software is unable to detect it.

Due to the ability to predict the attributes of unseen instances, machine learning has been applied to detect new malware by classifying programs into benign programs or malware. Machine learning based malware detection algorithms usually consist of two processes, i.e. feature extraction and classification. The feature extraction process converts the original programs to feature vectors. Then the classification process will train a binary classifier on the feature vectors in the training set, and the classifier is able to predict whether a test feature vector is malicious or not.

There are a lot of feature extraction algorithms proposed by different researchers in recent years. The most popular algorithms among them are the dynamic link library (DLL) and the application programming interface (API) features, the string feature, PE-Miner and the byte level n-gram feature.

The DLL, API and string features were proposed by Schultz et al. [2]. The absence or presence of a DLL, API or string is converted to a binary feature value. Shafiq et al. proposed an algorithm called PE-Miner [3] which uses 189 fields from the PE header, section headers and the import table as features. Inspired by the word level n-gram feature used in text classification, Kolter et al. extracted n-gram features from sequences of program bytes and selected the most relevant features by information gain to detect malware [4][5].

Many other malware detection models are based on these features. For example, Dahl et al. [6] and Pascanu et al. [7] used API feature in their neural network models for malware detection. An ensemble of different features such as DLL, API, string and PE fields was used by Saxe et al. [8] in their deep learning model.

Most of the feature extraction algorithms just regard malware detection as a regular machine learning problem and mainly aim at improving the final detection performance. However, these algorithms usually ignore an important aspect of malware detection, i.e. the robustness. If malware authors know how these algorithms work, they would adopt some pretense techniques accordingly in order to bypass these algorithms. Therefore, the robustness of malware detection algorithms should be evaluated seriously. If an algorithm is able to detect malware with a high accuracy but can be easily bypassed by some simple pretense techniques, it cannot be used in real world applications.

This paper performs a detailed analysis on the robustness of four well-known feature extraction algorithms for malware detection. Two pretense approaches which generate fake benign features for malware are proposed to avoid being detected by these algorithms. A series of experiments are conducted to get the actual detection performance of these algorithms under the pretense approaches.

Using pretense to bypass malware detection algorithms is similar to the idea of adversarial examples [9] in deep learning.

978-1-5090-6182-2/17/$31.00 ©2017 IEEE

Szegedy et al. applied some small perturbation to real images, in order to make a trained neural network unable to classify them correctly [10], and the images after perturbation are called adversarial examples. Grosse et al. generated adversarial examples from a neural network based malware detection model for Android malware and showed that the neural network has a high misclassification rate on adversarial examples [11].

II. DLL and API Features

A. Introduction

The behaviors of malware are usually carried out by calling system DLLs and APIs. Therefore, DLL and API can be used as the discriminative features to detective malware.

The dimension of a DLL feature vector is equal to the number of system DLLs; each element in the feature vector corresponds to a system DLL. If a program calls a DLL, the corresponding element in the feature vector is set as 1, otherwise it is set as 0. The API feature vector can be constructed in the same way.

B. Characteristics of DLL and API Features

1) Datasets: The dataset used in this paper consists of 9998 benign programs and 9183 malware samples, with a total size of 9.12GB. The benign programs are collected from the system executables of Windows operating systems and a variety of application software. The malware samples come from VX Heaven virus collection [12].

2) Characteristics: In this paper we use two scores to measure the characteristics of a binary feature, i.e. information gain and malicious tendency.

Information gain measures the relevance between a feature and the class [13] which is often used for feature selection. A feature with larger information gain usually contributes more to the final classification performance.

Malicious tendency reflects the extent to which a feature tends to appear in malware, which is defined as $P_M(f) - P_B(f)$, where $P_M(f)$ represents the proportion that the f-th feature appears in malware samples and $P_B(f)$ represents the proportion that the f-th feature appears in benign programs. A larger malicious tendency means the feature is more likely to appear in malware and therefore it is potentially related to malicious behaviors.

In the experiments we used 22 system DLLs and there are 4214 distinct system APIs in these DLLs. For each DLL and API feature, its information gain and malicious tendency are plotted in scatter graphs, as shown in Figure 1 and Figure 2.

It can be seen that for both DLL and API features, the features with large information gain mostly have negative malicious tendency. That is, the features which contribute a lot to the classification mainly come from benign programs. Only a small number of features have positive malicious tendency and their information gain are quite small. The features tend to appear in malware usually contribute less to classification.

In such conditions, the classifier will mainly make use of benign features to judge whether a program is benign or malware. It is hard for the classifier to learn the patterns of malicious behaviors.

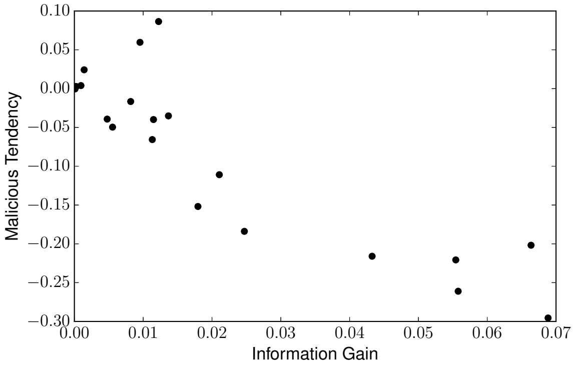

Fig. 1. Information gain versus malicious tendency of DLL features

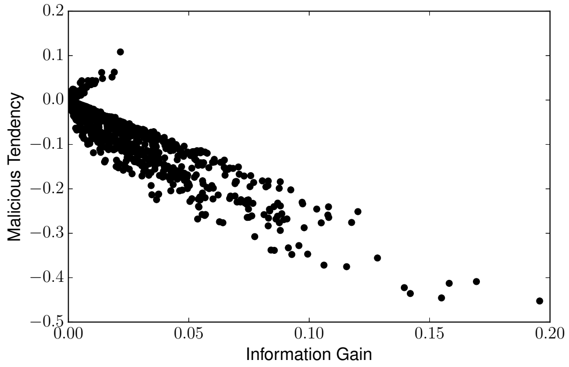

Fig. 2. Information gain versus malicious tendency of API features

Malware authors can take advantages of this characteristic to bypass the malware detection algorithm, by artificially adding some unnecessary benign behaviors to malware. For example, benign programs usually use GUI to interact with users, while malware usually hides itself from users. The GUI related DLLs and APIs will appear frequently in benign programs and infrequently in malware. If a program uses GUI it is more probable to be classified as a benign program. If a malware author intentionally calls GUI related DLLs and APIs, the malware will have benign characteristics and the malware detection algorithm will probably recognize the malware as benign.

C. Pretense for DLL and API Features

In this section we propose two approaches for malware to pretend to be benign, and show how the performance of the malware detection algorithm declines under such pretense.

The pretense is performed by modifying malware's feature value, but there is a restriction on the modification. If a feature value is 0, we can easily modify it to 1 by calling the corresponding DLL or API. Adding an irrelevant DLL or API does not influence the function of the program. If a feature value is 1, it is not easy to modify it to 0 (i.e. remove the DLL or API from the program) without damaging the integrity of the program. Although a few of DLLs or APIs can be removed by replacing them with similar ones, the replacement may involve many human efforts and removing such a small number of DLLs or APIs only has small influence on the final classification performance. In this paper we want to develop automatic pretense methods which can be applied to a variety of malware without human involvement. Therefore, when modifying a feature value, we set up a restriction that we can only change 0 to 1 and cannot change 1 to 0.

For a malware sample, we will manually modify its feature values according to a chosen benign program, in order to make it look like the benign program and avoid the detection of the malware detection algorithm. For each dimension of the feature vector, if the malware sample's value is 0 and the benign program's value is 1, the malware sample's value will be modified to 1.

1) Choosing a Benign Program for Malware: For a malware sample, we proposed two approaches to choose a benign program to modify its feature values, i.e. the random approach and the max pretense approach. The random approach just chooses a benign program randomly from the training set, while the max pretense approach uses some strategies to make the pretended feature vector like benign to the maximum extent.

We use the DLL feature as an example to introduce the max pretense approach. The set of DLLs called by the malware sample is denoted as M, while the set of DLLs called by the i-th benign program in the training set is denoted as B_i. $|B_i - M|$ is the number of DLLs that called by the i-th benign program but not called by the malware. The larger $|B_i - M|$ is, the more benign DLL features the pretended malware will have, and the more likely that the pretended malware will bypass the malware detection algorithm. Different DLL features are usually not of equal importance, and therefore for each DLL feature f we assign a weight to it using its information gain $IG(f)$. Then the objective to be maximized becomes $\sum\limits_{f \in B_i - M} IG(f)$.

The pretense method should also make sure that the chosen benign program and the malware have enough common characteristics. If the benign program and the malware are quite distinct from each other, their combination will look unnatural. After the malware detection algorithm collected a large amount of such pretended malware samples, it will learn the pattern of such unnatural combination of two distinct sets of DLLs and use this pattern to classify unnatural combination as malware. If the benign program and the malware have lots of common characteristics, their combination will look more like a normal program and this will make the learning of the combination pattern uneasy. The common DLLs called by the benign program and the malware $B_i \cap M$ can represent their common characteristics. After assigning each DLL feature a weight using information gain we get the second objective to be maximized $\sum\limits_{f \in B_i \cap M} IG(f)$.

The two objectives can be converted to one objective by adding them up, i.e. $\sum\limits_{f \in B_i - M} IG(f) + \sum\limits_{f \in B_i \cap M} IG(f) = \sum\limits_{f \in B_i} IG(f)$. This sum only needs to add up the information gain of appeared DLLs in the benign program and is independent of the malware. Thus it can be calculated for all the benign programs in the training set in advance and does not need to be calculated again for each malware sample. This process will significantly reduce the time complexity of pretense. We also tried to use a weighted sum of the two objectives and the two weights are tuned on a validation set to improve the pretense performance. The weighted sum did not show significant better pretense performance than the unweighted sum. Besides, weighted sum is not independent of the malware so that it cannot be calculated in advance, resulting in a much higher time complexity than unweighted sum. Therefore, in this paper we just add the objectives up without weights.

If we directly choose the benign program with the largest $\sum\limits_{f \in B_i} IG(f)$, all malware samples will use the same benign program. When the malware detection algorithm has enough pretended malware samples it will learn this pattern and classify the samples with this benign program's features as malware. Therefore, the pretense method should keep the diversity of benign program's distribution. We will penalize the benign program that has already been used by other malware samples using a penalty factor λ. If the i-th benign program has already been used by $T(i)$ malware samples, for current malware we will use Formula 1 to choose the optimal benign program, i.e. the i^*-th benign program.

$$i^* = \arg\max_i \Big(\sum_{f \in B_i} IG(f) - \lambda T(i) \Big) \qquad (1)$$

2) Experiment Settings: For each pretense approach, two experiments are conducted. In the first experiment the pretense approach is only applied to malware in the test set. A classifier is trained on the original training set without pretense and the detection performance on the pretended malware in the test set is reported. If the pretense approach becomes popular, a lot of pretended malware samples will appear on the Internet. After collecting enough pretended samples the malware detection algorithm is able to train a classifier to distinguish pretended malware from benign programs. Therefore, in the second experiment the pretense approach is firstly applied to malware in the training set and then applied to malware in the test set. A classifier is trained on the pretended malware in the training set and the benign programs in the training set. The classifier is able to learn some patterns of the pretended malware, e.g. the unnatural combination of features from two programs. Then the detection performance on the pretended malware in the test set is reported.

When the pretense approach is applied to the training set or the test set, the benign programs used to modify malware's features are both chosen from the training set. Generally speaking, the benign programs in the test set can be regarded as unseen samples. We can only use existing benign programs (i.e. benign programs in the training set) to modify malware's features.

The random forest with 100 trees is used as the classifier for all the experiments in this paper. Five-fold cross validation is used to get the final results. To tune the parameter λ in Formula 1, the training set is split into a smaller training set and a validation set and line search is performed to search the λ with the lowest accuracy on the validation set.

3) Experimental Results: The experimental results of DLL and API features are presented in Table I and Table II respectively. TPR is abbreviated to true positive rate and FPR is abbreviated to false positive rate. "no-pretense" represents that the pretense is applied to neither the training set nor the test set. "rand-tst" represents that the random approach is only applied to malware in the test set, while "rand-tra-tst" means pretense is applied to both the training set and the test set. "max-pret-tst" and "max-pret-tra-tst" represent the usage of the max pretense approach for different subsets.

TABLE I
THE PRETENSE RESULTS OF DLL FEATURE

	TPR	FPR	Accuracy	AUC
no-pretense	88.62%	41.86%	72.73%	81.27%
rand-tst	42.25%	41.86%	50.53%	53.20%
rand-tra-tst	47.90%	23.04%	63.05%	67.54%
max-pret-tst	2.83%	41.88%	31.65%	23.66%
max-pret-tra-tst	30.38%	33.12%	49.41%	52.04%

When there is no pretense in the training set and the test set, the DLL feature results in a high detection rate (i.e. TPR) and a high false alarm (i.e. FPR). When the random approach is applied to the test set, the detection rate on the pretended malware decreases by 52.3%. If the pretended malware samples are added to the training set the detection rate just increases a little. When the max pretense approach is applied to the test set, the detection rate is only 2.83%, which means the malware detection algorithm does not work at all. Even after the classifier is trained on the pretended malware, the detection rate only reaches 30.38%, with a false alarm of 33.12%.

TABLE II
THE PRETENSE RESULTS OF API FEATURE

	TPR	FPR	Accuracy	AUC
no-pretense	94.34%	9.23%	92.48%	97.58%
rand-tst	22.02%	9.09%	57.93%	64.19%
rand-tra-tst	53.25%	7.82%	73.54%	78.75%
max-pret-tst	0.05%	9.15%	47.38%	32.40%
max-pret-tra-tst	34.08%	13.32%	61.50%	69.12%

When pretense is not applied to the training set and the test set, the API feature has better detection performance than DLL feature because API has more detailed information than DLL. When only applying pretense to the test set, the detection rate decreases from 94.34% to 22.02% and 0.05% for the two pretense approach respectively. After the classifier is trained on the pretended malware, the detection rate still cannot rise to an acceptable level.

It can be seen that DLL and API features are not robust to these pretense approaches and the pretended malware can avoid the detection of the classifier easily.

III. String Feature

The executables usually contain many printable strings in their binary contents. Schultz et al. [2] used these strings as the features to detect malware.

After analyzing the string features with large information gain we found that the strings mainly consist of DLL and API names in the import section and the user-defined string constants. 46% of the top 400 strings in information gain are DLL or API strings.

For user-defined string constants, it is very easy to add irrelevant strings to a program and hide an existing string by some obfuscation operations such as XOR and bit shift. Given a malware sample and a benign program, we can modify the malware's feature vector to make it exactly the same as the benign program's feature vector by hide the features that only appear in the malware and add irrelevant strings that only appear in the benign program.

Therefore, we only need to consider the DLL and API strings, while the pretense analysis given in the previous section can be generalized to these strings. We conclude that string features are also not robust enough and the malware detection algorithm can be easily bypassed.

IV. PE-Miner

A. Introduction

On windows operating systems, the executables are usually in PE format [14]. The PE format has many fields in the head of an executable such as time date stamp, number of sections and virtual addresses of sections. PE-Miner [3] regards these fields along with DLLs as the features to detect malware.

B. Characteristics of PE-Miner

We found that the meanings of many PE fields are not related to the malicious behaviors of malware. For example, the feature with the largest information gain is time date stamp. Benign programs usually use normal time date stamp while a lot of malware samples use abnormal time date stamp such as zero value (i.e. '1970-01-01 00:00:00'). Actually time date stamp has nothing to do with a program's behaviors. If the malware authors know the abnormal time date stamp will be used as a feature to detect malware, they can just use normal time date stamp in their malware.

Many PE fields can be modified unrestrictedly without crashing the functions of the program, such as time date stamp, linker version and image version. Some address related fields such as address of entry point can be modified with little restriction. The new value should be kept in a valid address range and the compiler should rearrange the corresponding code or data to the new address. For example, if we changed the address of entry point, the compiler should move the code piece at the entry point to the new address.

Other PE fields can only be modified in the increasing direction such as number of sections, size of code and virtual sizes of sections. For example, the size of code can be increased by inserting irrelevant code, but decreasing the size of code may crash the program. Although it is possible to decrease the size of code by manually removing some redundant code, this will involve many human efforts and cannot be applied to a variety of malware. Therefore, we only consider to increase these PE fields in the pretense approach.

There are 57 PE fields which can only be increased. For each of these fields, we use information gain and class difference to measure its characteristics. Before calculating class difference all fields are standardized to have zero mean and unit variance. Class difference for the f-th field is defined as $F_M(f) - F_B(f)$, where $F_M(f)$ is the field's average value in malware samples and $F_B(f)$ is the field's average value in benign programs. The information gain and class difference of the 57 fields are shown in Figure 3.

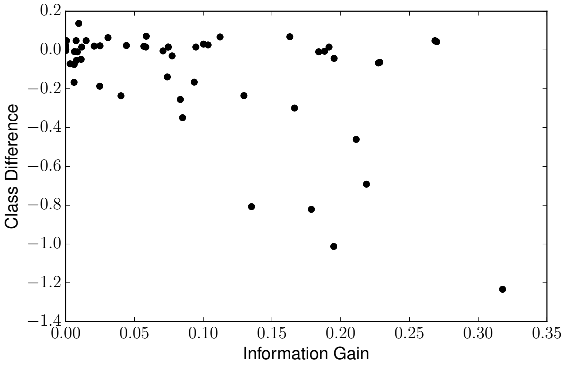

Fig. 3. Information gain versus class difference of the 57 PE fields which can only be increased

The points with positive class difference are not far from the horizontal zero line while many points have negative class differences and some negative points stay quite far from the horizontal zero line. Negative class difference means the average field value of malware is smaller than that of benign programs'. There is enough space for malware authors to increase these fields with negative class differences to make malware look like benign.

C. Pretense for PE-Miner

A benign program is firstly chosen to modify malware's feature value. PE-Miner uses PE fields and DLLs as features. For the PE fields which can be modified unrestrictedly, the malware's value will be modified to the benign program's value. For PE fields which only be modified in the increasing direction, if the malware's value is smaller than the benign program's value the malware's value will be modified to the benign program's value. After a PE field is modified, we may need to rearrange the corresponding content of the executable to make sure the format of the executable is still valid. For example, if we increase the number of sections, we should add some irrelevant sections to the executable. DLL features can be modified in the same way as the previous section.

The fields used by PE-Miner include the virtual addresses and the virtual sizes of 15 data directories and 3 sections. If the virtual sizes are increased some data directories or sections may overlap in the virtual address space. If such overlap happens, the posterior data directory or section should be shift backwards to eliminate the overlap. The new virtual addresses of sections after shifting should be aligned with the field "section alignment". The file pointers to the sections and the sizes of the sections on disk may also suffer from overlap when the sizes are increased. In such case the sections in the file should also be shifted to eliminate the overlap.

For PE-Miner we also use the random approach and the max pretense approach to choose the benign program. The max pretense approach only considers the DLLs used in PE-Miner and the 57 PE fields which can only be modified in the increasing direction. The 57 PE fields are continuous while Formula 1 is defined for binary DLL and API features. For continuous variables the objective should be redefined. The binary DLL features in PE-Miner can be viewed as special continuous variables and they also can only be increased. Therefore, the DLL features are merged into these PE fields so that we can use a unified model to deal with these two kinds of features.

Like the max pretense approach for the binary DLL and API features, we also define two objectives to be maximized for continuous variables. The f-th field value of the malware is denoted as v_f^m, and the f-th field value of the i-th benign program is denoted as v_{if}^b. The set $\{f|v_f^m < v_{if}^b\}$ represents the fields on which the malware has smaller value than the i-th benign program. The size of this set reflects how much benign characteristic the i-th benign program will bring to the pretended malware. It is the first objective of the max pretense approach. The set $\{f|v_f^m = v_{if}^b\}$ represents the fields on which the malware and the i-th benign program hold the same value, and it can be used to represent the common characteristics of the two programs. Maximizing its size will make the combined program look like a normal program and the pattern of unnatural combination will be weakened. After adding the two objectives we get the unified objective $|\{f|v_f^m <= v_{if}^b\}|$. After assigning each field a weight using information gain we can use Formula 2 to choose the benign program for current malware.

$$i^* = \arg\max_i \Big(\sum_{f|v_f^m <= v_{if}^b} IG(f) - \lambda T(i) \Big) \qquad (2)$$

The experimental results are shown in Table III. Except the last two rows, the abbreviations in this table are the same as Table I and Table II.

When there is no pretense, PE-Miner is able to achieve a relative high detection rate of 97.5%. Under the random approach, the classifier trained on the dataset without pre-

TABLE III
THE PRETENSE RESULTS OF PE-MINER

	TPR	FPR	Accuracy	AUC
no-pretense	97.50%	3.07%	97.20%	99.57%
rand-tst	2.76%	3.16%	51.80%	69.46%
rand-tra-tst	87.18%	0.23%	93.74%	98.18%
max-pret-tst	0.23%	3.14%	50.60%	50.28%
max-pret-tra-tst	69.76%	5.76%	82.52%	89.33%
max-pret-tst-wos	0.13%	3.05%	50.60%	47.54%
max-pret-tra-tst-wos	49.60%	14.33%	68.40%	73.52%

tense can only detect 2.76% pretended malware. While after adding the pretended malware to the training set, it will learn the unnatural combination pattern of pretended malware and detect 87.18% of them. The max pretense approach tries to weaken the unnatural combination pattern, and after the classifier learns from the pretended malware, the detection rate is 69.76%, which is much smaller than that of the random approach.

In the pretense process if data directories or sections overlap after increasing their sizes they will be shifted backwards. This will cause new pattern in the pretended malware, i.e. the shift pattern, and the classifier will learn this pattern to detect such kind of malware. The suffix "wos" in the last two rows of Table III means "without shift". Of course it is impossible not to shift the overlapped data directories or sections but we just use these two rows to show the effect of shift pattern. The detection rate drops from 69.76% to 49.6% if we do not shift the overlapped data directories or sections, which means the shift pattern contributes a lot to the detection rate of "max-pret-tra-tst" and the classifier only learns limited malicious pattern from malware.

The detection performance declines a lot under the max pretense approach, which means PE-Miner is also not a robust malware detection algorithm.

After learning from pretended malware, the achieved detection rate owes more to the combination pattern and the shift pattern. Actually the PE fields mainly contain some rough statistics of the executables such as the sizes of sections and the number of symbols, and these fields are not closely related to the malicious behaviors of malware. It is possible to develop more meticulous pretense approach to weaken the effect the combination and shift patterns and to make PE-Miner hard to detect such malware. We leave this pretense approach as future works.

V. BYTE LEVEL N-GRAM FEATURE

A. Introduction

A sliding windows with length N is used to extract N-Gram features from the byte sequence in a program. N is a hyper-parameter of the malware detection algorithm which is usually tuned on a validation set. In most papers the final chosen value of N is 4. In such case there usually will be hundreds of millions of N-Gram features and feature selection is required to select a small amount of discriminative features. We use information gain to select the 2000 most discriminative features in all the experiments of this section. The feature value of an N-Gram is 0 or 1 according to its absence or presence in a program.

B. Characteristics of N-Gram Feature

N-Gram feature is binary, and we can use the same method as DLL and API features to analyze it. The information gain and the malicious tendency of the top 2000 features are shown in Figure 4.

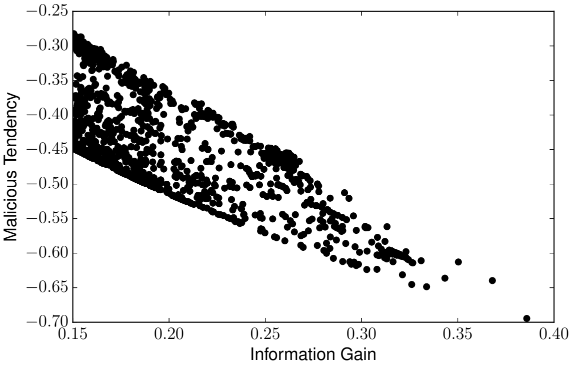

Fig. 4. Information gain versus malicious tendency of the top 2000 N-Grams features

We can see that all the top N-Grams have negative malicious tendency, which means these features tend to appear in benign programs. After manually checking the contents of these features, we found that many of these features come from the normal string resources in the benign executables, while malware seldom uses these strings. The malware detection algorithm can be easily bypassed by adding these irrelevant strings to malware.

C. Pretense for N-Gram Feature

To apply pretense to a malware sample, we choose a benign program to inject its full binary content to the binary malware. This is identical to modify the malware's feature value from 0 to 1 when the benign program's feature value is 1 and the malware's feature value is 0. The modification of feature value is not just on the top 2000 features, but on all the hundreds of millions of features before feature selection. After changing the feature values, their information gain will also change. Therefore, when we train a classifier on pretended malware, we should re-calculate all features' information gain and select the most discriminative features again.

The random approach and the max pretense approach used here are the same as DLL and API features, since N-Gram feature is also binary. When using max pretense approach for N-Gram feature, B_i in Formula 1 is the set of all N-Grams appeared in the i-th benign executable.

The experimental results of N-Gram feature are shown in Figure IV.

The classifier trained on non-pretended dataset is able to detect 97.65% non-pretended malware, but it only detects

TABLE IV
THE PRETENSE RESULTS OF N-GRAM FEATURE

	TPR	FPR	Accuracy	AUC
no-pretense	97.65%	4.44%	96.56%	99.27%
rand-tst	2.56%	4.24%	51.24%	51.41%
rand-tra-tst	96.73%	2.25%	97.26%	99.56%
max-pret-tst	0.00%	4.49%	49.88%	34.80%
max-pret-tra-tst	40.92%	15.37%	63.75%	76.49%

2.56% pretended malware when using the random approach and cannot detect any malware when using the max pretense approach. When pretended malware samples of random approach come into the training set, the classifier detects 96.73% of them. This is because the combination with a random chosen benign program is usually unnatural and the unnatural combination pattern is learned by the malware detection model so that the model can use this pattern to detect such malware. While the max pretense approach is able to significantly weaken the combination pattern, which results in a detection rate of 40.92%. Most pretended malware of the max pretense approach still cannot be detected even after the model learns their characteristics.

Therefore, the widely used N-Gram feature is very vulnerable to the well-designed pretended approach and anti-virus software must be cautious enough to use this feature.

VI. Conclusion and Discussion

In this paper we proposed two approaches to analyze the robustness of machine learning based malware detection algorithms. It is shown that the most used feature extraction algorithms, i.e. DLL, API, string, PE-Miner and N-Gram, have poor detection performance under the proposed pretense approaches. These malware detection algorithms cannot be applied in real world applications unless effective mechanism is proposed to deal with pretense.

When the max pretense approach is only applied to the test set, the detection rate of these feature extraction algorithms range from 0% to 3%, which means that the malware detection algorithms do not work at all on the new pretended malware. Even though the detection rate will increase after antivirus vendor collects enough pretended malware and adds them to the training set, the collection will take time and when the detection rate increases malware authors may have already turn to other pretense approaches. When the malware detection algorithm learns to deal with these new pretense approaches, malware authors will develop other new pretense approaches again. Adding pretended malware to the training set is not a solution to the detection of pretended malware because the patterns of pretended malware keep changing and the classifier cannot generalize to new pattern. Malware detection algorithm should learn the inherent malicious characteristics of malware and drop the benign patterns to avoid that malware samples use the benign patterns to bypass detection. This will make the malware detection algorithm more robust against pretense.

VII. Acknowledgment

This work was supported by the Natural Science Foundation of China (NSFC) under grant no. 61375119 and the Beijing Natural Science Foundation under grant no. 4162029, and partially supported by National Key Basic Research Development Plan (973 Plan) Project of China under grant no. 2015CB352302.

References

[1] AV-TEST, "Malware statistics & trends report," 2016. [Online]. Available: http://www.av-test.org/en/statistics/malware/

[2] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 38–49.

[3] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in realtime," in *Recent advances in intrusion detection*. Springer, 2009, pp. 121–141.

[4] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 470–478.

[5] ——, "Learning to detect and classify malicious executables in the wild," *The Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[6] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3422–3426.

[7] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1916–1920.

[8] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 11–20.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[11] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[12] VX-Heaven, "Virus collection (vx heaven)," 2016. [Online]. Available: https://vxheaven.org/vl.php

[13] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *ICML*, vol. 97, 1997, pp. 412–420.

[14] Microsoft, "Microsoft portable executable and common object file format specification," 2013. [Online]. Available: http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff_v83.docx