An Efficient Game Theoretical Model for Intelligent Malware Detection System

Abstract
Data safety is one of the most important concerns of information community which is threatening by many viruses, worms and threats. Malware attack is one among these threats which is a malicious code that intent to harm computers and networks. As Malware attacks are becoming increasingly, the security policy of computer is more crucial and it is so important to have a well-defined process. In this paper we defined a game in which there is a competition between malware and anti-malware for maximizing their expected utility by choosing particular courses of actions. Experimental results show that anti-malware can efficiently detect malware class.

Keywords: Malware, Analysis techniques, Data Mining, API calls, Game Theory.

1. Introduction

As more and more computer systems and the Internet communications are used for many services such as banking and financial services, the security vulnerability increases significantly.

Nowadays, data safety is a big concern for information community. One of the main topics in computer security which have been extensively investigated is Malware Detection.

Malware is any kind of software intentionally developed for malicious purposes without user's awareness. There are many kinds of malware such as Computer Virus, Worm, Trojan horse, Spyware, Rootkits, Botnet, Flooders and so on [1].

One of the most dangerous computer viruses named Melissa spread through email and shut down email systems. *I LOVE YOU*, a worm capable of replicating itself, were spread westwards through corporate email systems in 2000. In 2002, two deadly worms named *Code Red* and *Code Red 2* exploited the weakness of operating system especially Windows 2000 and Windows NT. *SQL Slammer* is another worm born in 2003 which caused a denial of service on some Internet hosts and dramatically slowed down Internet traffic.

It crashed many important servers including the bank of America's ATM service.

Sasser, a malware discovered in 2007, spreads by exploiting the system through a vulnerable network port and affects the computers which use Microsoft operating systems.

Since malwares are big disaster to exploit the security vulnerability and their daily propagation, the security society has been persuaded to consider about new approaches to confront them with novel intelligent systems.

One of the most effective methods to deal with malwares is signature-based detection which is the industrial standard for detecting malwares [2] and widely used in commercial antivirus software.

Signature of a malware is usually a bytes sequence extracted from a malware file, a regular expression, or in other formats which is hand crafted by security experts.

Although signature based methods have several advantages such as detection speed and accuracy, there are two major drawbacks dealing with such methods; the first drawback is disability to detect new threats because there is no corresponding pattern in the database. The second problem is that the discovering, analyzing and extracting the signature of malwares is time consuming process and usually needs expert knowledge domain. The difficulty of generating signature database process is by reason of the rapid proliferation of new malwares and their robustness against detection.

To overcome the main drawbacks of classic signature based methods, behavior based intelligent methods are developed. Such methods try to find the behavior malwares for further recognitions of similar malwares. Behavior based malware detection methods can fulfill the malicious activity detection during code execution by trying to trace any suspicious stir [3]. Even if a code is obfuscated, its behaviors and aims remain resolute.

Since almost all programs use application programming interface (API) calls to send their requests to the Operating System [4], one of the most attractive ways to find the behavior of a program such as malware is recognizing the pattern of its API call sequences.

As attack method, and threats are becoming increasingly, the security policy of computer is more crucial nowadays. Malware attack is one among these threats. It is very important to have well-defined process such as anti-malware to defeat these harmful threats. There is a challenge between malware writer and intelligent malware detector.

In this paper we introduce a method in which it will be made a challenge between malware and anti-malware which is described in section 3. In next section we briefly discussed data mining detection method and in section 4 we exam our method and show experimental results. At last in section 5 we introduced future work and conclude.

2. .Data Mining Detection Methods

To confront unknown malwares, behavior-based detection method is developed, Which uses its knowledge of what constitutes normal behavior to decide the maliciousness of a program. These methods mostly use data mining techniques.

Data mining methods use statistical tools and machine learning algorithms on a set of features, to recognize the malicious programs from benign ones. There exists two important phases in all of these procedures. First realizing what features are suitable for further analysis, second extracting and comparing these features to get better precision. A notable example of these features is Application Programming Interface (API) call sequences.

An API call sequence indicates flow of called API's by a program. Each program should be analyzed to extract this sequence.

2.1. Various Feature types

An n-gram is a sequence of bytes with fixed or variable length. N-gram features are extracted from the hexadecimal dump of an executable program [5].

Traditional data mining based malware detection methods apply n-grams on hexadecimal dump of a program to extract its features.

Since n-grams fail to capture the semantics of a program, assembly instruction sequences should be used instead. Instructions need disassembling the executable files. Since almost all programs use application programming interface (API) calls to send their requests to the Operating System, one of the most attractive ways to find the behavior of a program such as malware is recognizing the pattern of its API call sequences.

API call sequence gives us valuable information about malicious activities [6] by indicating the flow of called API's using a program which can be either a malicious program or a benign one. Each program should be analyzed to extract this sequence.

2.2. Problems of Current Intelligent Methods

Malware writers are very creative and genius persons. Each malware which written by these persons has new trick to confuse the detector engines.

They tries to protect their malwares against detection methods by make use of several delicate techniques. Some of these techniques are designed to fight with signature based detection such as *dead code insertion*, *variable renaming*, and *code reordering*. There are some techniques that used by malware writers to confuse the intelligent engines.

Since almost intelligent malware detection engines work with data mining techniques and need to extract features from suspicious files, these techniques try to confuse the intelligent sub system.

As mentioned API sequence is an effective feature for detecting unknown malwares by data mining methods. Malware protection methods try to call useless API's to confuse the intelligent malware detection systems.

There is a challenge between malware writer and intelligent malware detector. Malware writer wants to be write the undetectable malwares by using lowest useless API's and intelligent detector needs to change its position to detect them without reporting benign programs as malware. In the next section our method will be introduced.

3. Proposed Method

As attack method, and threats are becoming increasingly, the security policy of computer is more crucial nowadays. Malware attack is one among these threats. It is very important to have well-defined process such as anti-malware to defeat these harmful threats. Malware is a serious problem in distributed and host-based computing environments. It is very harmful external element, which enters the computer without owner's agreement. "Malware writers" began to write malware in the early 1980's [7] .To hinder detection, code packing is used by them which cause to hide the malware's real content. Malware steals your personal information from your computer and use it for wrong purposes, for example, using the credit card information for robbing. At the other point there is various anti-malware software. This software is made bearing in mind all the requirements of a computer, which will help it in removing the malware before any damage happens, prevention is better than cure. According to this issue we want assume that there is the infinitely game between malware and anti-malware.

3.1. Mechanism Design

Adversarial classification is a type of classification in which the goal of the classification task is usually to discriminate between a 'benign' and a 'malicious' class when the samples of the 'malicious' class are generated by an intelligent, adaptive adversary, which actively manipulate data to misclassified his samples innocent [8].

Let the training set S and test set T be two sets of (x; y) pairs, where x is the instance features and y is the true label of the instance. We define adversarial classification as a game between two players: anti-malware, which attempts to learn from S and tries to correctly predict the classes of instances in T, and Malware author, which attempts to make anti-malware classify positive instances in T as negative by modifying effective features of instances [9].

In this situation anti-malware will be optimal against whatever strategy that malware take into account. It means that anti-malware adapts itself automatically against malware's evolving manipulation. Experiments in spam detection illustrate that this approach can significantly improve the result of anti-malware in the standard way [9, 10]. This describes a kind of game which is the theory of rational behavior for above interactive decision problem. This is the situation of uncertainly because no participant knows for sure what the other participant is going to decide. Unlike a normal form of game played once this game is repeated, cost sensitive and extensive which model situation in which each player engages in a strategic interaction over and over., or in the other words players interact by playing a similar stage game many times [11] so malware have to notice that its current action influence on the future actions of anti-malware and vice versa.

We are talking about a game that is a competitive game because the agents' goals are in conflict and also is a mathematical representation of a situation in which a participant's gain (loss) of utility. The base concept in competitive game is that "What is good for current player is bad for his opponent and vice versa".

Our model describe an infinitely game which is mathematical method of decision-making in which an interactive and competitive situation is analyzed to determine the optimal course of action for an interested party, this game has two party.

One party is malware and the other one is anti-malware and it is played over and over again. Malware is a combination of the words "malicious" and "software". It has malicious intent and refers to software programs designed to damage computer, threats its safety and do other action on a computer system without the user's approval.

The other player is anti-malware. In our paper anti-malware refers to naïve bayse classifier [12]. According to the precise nature of the probability model, naïve bayse classifiers can be trained very efficiently in supervised learning. It assumes that a feature of the class is totally independent from all other features. In spite of its simple and naive design, they have worked quite well in many complex real-world situations.

In this *game*, both malware and anti-malware try to maximize their expected utility by choosing particular courses of actions, and their final utility payoffs or the final outcomes, are determined jointly by the strategy chosen by each of them. There is a set of all possible utility payoffs. The malware gains payoff when it defeats the anti-malware. It means that malware can fool classifier, and cause classifier classifies malware as benign. In all other situation malware doesn't gain any payoff. Anti-malware lose payoff when detect a benign as malware and malware as benign. Whenever it classifies malware and benign correctly it will obtain payoff in both situation.

3.2. Nash Equilibrium

Nash equilibrium recommends a strategy which players cannot improve upon unilaterally, that is, given that the other players follow the recommendation. Since the other players are also rational, it is reasonable for each player to expect his opponents to follow the recommendation as well [13].

As we have previously described, we have two players and each player has own set of actions and the payoffs from each combination of actions are known, so we can solve the problem by finding the Nash equilibrium. Following theorem adversarial classification game always has Nash equilibrium.

Theorem: Consider a classification game with a binary cost model for Adversary, i.e., given a pair of instances x and x', Adversary can either change x to x' (incurring a unit cost) or it cannot (the cost is infinite). This game always has a Nash equilibrium, which can be found in time polynomial in the number of instances [9].

Unfortunately, to calculate the Nash equilibrium it is necessary to have complete and accurate knowledge of the probabilities of all the instances, which with respect to this point that our approach is a subtype of adversarial classification, in practice this calculation is impossible for Adversary and Classifier because of very large action space. On the other hand, in finite domains the number of available actions is doubly exponential in the number of features. To find Nash equilibrium in general nonzero sum games the best known algorithms have worst-case exponential time in the number of actions which in our case it is triply exponential. In the last few years there are substantial works on computational tractable approaches to game theory, but they focus mainly on scaling up with the number of players, not the number of actions [9].

Further all type of equilibrium strategies either mixed or pure ones assume that the opponent play(s) the game based on its (their) optimal strategy, which is highly unrealistic in our case. When this assumption is not met standard game theory gives no guidance on how to play. We thus leave the general existence and form of Nash or other equilibria in adversarial classification as an open question.

3.3. Malware Author Strategy

In this section, we will discuss the modeling of the malware author strategy. Malware author (MA) always sends a batch of data includes positive and negative classes as benign and malware to the anti-malware (AM) then antimalware tries to predict the correct labels of this test batch.

We define a malware detection system as a game between malware author and anti-malware. If y is the true label and y’ is the estimated label by anti-malware,

Malware author utility is characterized by $U_{MA}(y,y')$, anti-malware utility with $U_{AM}(y,y')$ likewise.

Assume X is a vector variable, X=$(X_1\ldots Xi,\ldots,X_n)$, and X_i is the *i*th feature of X. Let consider a domain vector as $D=\{D_1,\ldots,D_i,\ldots,D_n\}$, where $D_i=\{d_1,..,d_j,..d_m\}$ is the set of possible value of X_i.

An instance *x* has a negative class or benign class if comes from $P(X|-)$ distribution and is named as benign software, x is labeled positive's class or as malware's class if follows up $P(X|+)$ distribution.

The aim of malware author is to modify some set of feature of *x* with minimum cost so that anti-malware will be classified it as negative then malware author gain a utility of $\Delta U_{MA} = U_{MA}(-,+) - U_{MA}(+,+)$.

The malware author supposes that anti-malware is uninformed of its presence and the anti-malware predict labels with the naïve bayse model.

The naïve bayse assumption is that features are conditionally independent given the class labels. Given this assumption, naïve bayse approximates the probability that an instance x with label y as in Equation (1) and predicts the class with the highest P (y|x).

$$P(y|x) = \frac{P(y)}{P(x)}\prod_{i=1}^{n} p(x_i|\mathrm{y}) \tag{1}$$

Recall that the malware author assumes that anti-malware predict label using the naïve bayse model. From Equation (1):

$$\log\frac{P(+|x)}{P(-|x)} = \log\frac{P(+)}{P(-)} + \sum_{x_i \in D_c} \log\frac{P(x_i|+)}{P(x_i|-)} \tag{2}$$

For simplicity we will use notations in Equation (3) and Equation (4):

$$LOC(x) = \log\frac{P(+|x)}{P(-|x)} \tag{3}$$

$$LOC(x_i) = \log\frac{P(x_i|+)}{P(x_i|-)} \tag{4}$$

Considering the naïve bayse model, If the expected utility of classifying instance x, as positive is more than negative, naïve bayse will classifies x as positive.

$$\frac{P(+|x)}{P(-|x)} > \frac{U_{AM}(-,-) - U_{AM}(+,-)}{U_{AM}(+,+) - U_{AM}(-,+)} \tag{5}$$

A Malware author does not need to do anything, if the instance is classified as negative. Suppose that x is classified as positive and Let named the logarithm of the right hand side of Equation (5) as *LogT* (U_{AM}) (log threshold) and define interval as :*interval(x) = LOC (x) - LogT* (U_{AM}).

Remember that D_i is the domain of X_i, we define the cost of modifying feature x_i to d_i as w_i (x_i, *di*) and it is obvious that the cost of changing feature of each instance is sum of the cost of modified features.

By changing features, we have the gain *Δloc*. $\Delta loc_{i,d_i}$ = *loc* (x_i) - *loc* (d_i). This gain causes anti-malware to classify instance as negative.

To find out the minimum cost require to change an instance x, for each feature x_i, for the value of D_i set ,which has the maximum *Δloc*, we find the costs, and select a set of features which sum of their cost is minimized. The new obtained instance is the nearest instance to x based on cost so that naïve bayse classifies it as negative class. After finding this instance, anti-malware only changes it if the obtained minimum cost is less than ΔU_{MA}.

3.4. Anti-malware Strategy

In this part we describe the anti-malware strategy as an optimal strategy against malware author strategy. Anti-malware make its optimal decision based on two assumption first one is that Malware author employ(s) its optimal strategy to modify the test data and second one is, the train set T which uses for learning the naïve bayse classifier is not tampered by malware author.

The goal of Anti-malware is to predict the class of each sample so that maximize its utility, But the difference is that now we want to take into account the fact that the malware author has tampered with the data.

The malware author just tampered the data with positive class, so the only part which will change in naïve bayse formula, is p(x'|+), so p (+),p(-) and p(x'|-) stay unchanged.

The probability of observing an instance x', $p(x'|+)$, is the probability that the Malware Author change an instance x to x' manipulated by probability of an instance x belongs to positive class, sum over all features.

Malware Author may change instance x or not. If it didn't change, it doesn't affect. So we change the domain into features which are tampered to x'. There are two cases where malware author leaves an instance untampered. First, when the naïve bayse predicts it negative and second, when the cost of tampering the data is more than the earned utility by the malware author.

Thus probability distribution after malware author strategy is computed using Equation (6).

$$P_{MA}(x'|+) = \left(\sum\nolimits_{x \in X'_{MA}(x')} P(x|+)\right) + N(x')P(x'|+) \tag{6}$$

$X'_{MA}(x')$ indicates all instances that are similar to new instance, except new instance. If naïve bayse predicts x’ as negative, $N(x')$ = 1, otherwise 0.

P(x’|+) is the product of each feature’s probability. P(x’|+) is the probability of feature’s produced over all features.

When anti-malware is faced with a new instance, the instance should allocate with a label indicating the class of the instance. To appoint this label by anti-malware the following actions should be done.

First, calculate the probability that the instance’s label is negative $(P^-_{x'})$. To do so, the probability of negative data in the training set multiplied with the probability of each feature in this instance (feature available in the training set).

Then calculate the probability that the instance’s class is positive$(P^+_{x'})$. To calculate this probability first the probability of positive data in training set is calculated, then it is multiplied with result of naïve bayes estimates for instance.

Table (1): Gained utility

	(+,+)	(+,-)	(-,+)	(-,-)
[illegible]	0	0	20	0
[illegible]	1	-10	-1	1

Calculating class of instances (positive or negative), obtained utilities for each positive and negative classes should be calculate (Value in Table (1) is used to calculate the utility). Utility for each class is calculated as follow:

- Utility of positive class $\mathrm{U}(+|\mathrm{x}')$: probability of being positive$(P^+_{x'})$ multiplied with the amount of utility for correctly positive detection $\mathrm{U}(+|+)$ plus the product of probability of being negative $(P^-_{x'})$ with the utility for incorrectly negative detection$\mathrm{U}(-|+)$.
- Utility of negative class $\mathrm{U}(-|\mathrm{x}')$: probability of being positive $(P^+_{x'})$ multiplied with the amount of utility for incorrectly negative detection $\mathrm{U}(-|+)$ plus the product of probability of being negative $(P^-_{x'})$ with the utility for correctly negative detection$\mathrm{U}(-|-)$.

Calculating the utility of each positive and negative class, anti-malware will attempt to identify the new instance. If the utility of positive class is more than the utility of negative class, label of instance will be positive and contrary.

4. Experimental Setup

We designed an adversarial classifier system as an intelligent malware detection model. This system presents a competitive two players game.
Since players repeat this game over and over our implemented model is in form of repeated game. At the first we briefly explained about datasets that we used in our experiments.

4.1. Dataset creation

We design and evaluate our proposed model on the PE-file format [14]. This format is chosen because large number of Windows-based malware samples in this format is available on the Internet. We collect 1000 benign Windows PE- les from a new fresh installed Windows XP and select a set of malicious samples from APA malware repository [15], which included Windows 32-bit Worms and Trojans horses. All of malwares and benign files are non-packed.

As mentioned in section 2.1, an intelligent detection system needs to extract API call sequences from executable files. In the following we introduce a mechanism that is used for feature extraction from selected PE-files.

4.1.1. Feature Extraction

There are two categories of methods to extract called API's. The first one is static analysis and the second one is dynamic analysis [16, 17].

Static analysis extracts API call sequence according to executable file's source code. First of all, it constructs control flow graph (CFG) with disassembled source codes. Second, API call sequence is obtained by traversing the CFG. This method is unable to deal with decision making points and generally makes random decision in these points. This issue is the major drawback of static methods.

Dynamic analysis captures effects of programs on the environment and ignores analyzing their source codes. This method extract called API's by running them in the controlled environment which is simulated an operating system [18]. Sandbox is a powerful tool which simulates a real environment for the executable files, so malwares rely on it and request their system calls via API calls [19]. Sandbox replies the request in virtual environment and logs them. Extracted API's from this log file are accurate and actual. These sequences use as feature set in heuristic malware detection methods. Our proposed approach is based on mining these features. Applying data mining methods on API sequences encounters with several problems. API sequence fragmentation helps us to handle these problems, which is described in the next section.

4.2. API Sequence Fragmentation

To use data mining techniques we need to construct a feature set according to extracted API call sequence set. Since the number of instructions in variant executable files are different, length of the extracted API call sequences are different.

A common solution to construct feature set is identifying distinct API's from this sequence set and put them into a new set named as feature set.

This feature set construction method is simple and fast, but, this feature set misses an important portion of behavioral information of programs. Therefore, detection method based on this feature set is not accurate enough.

Although appearance of each API indicates a part of program's behavior, order of API calls in sequence plays an important role in program's behavior.

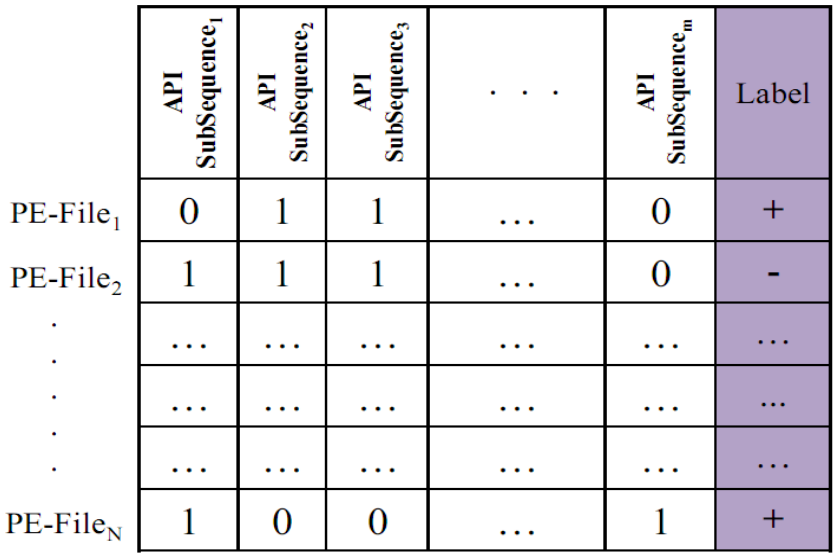
Therefore, different order of each API call in sequence infers different behavioral model. Preserving order of called API's has significant information for analyzing malware's behavior. The former feature set construction method misses the ordering of called API's. Thus, its accuracy is low. We need to consider the ordering information of API calls and store them into the feature set.

We need to use a nice approach to handle this issue. We use a method named ``N-grams'' to fragment the API call sequences without missing significant information about call ordering. Traditional data mining based malware detection methods apply n-grams on hexadecimal dump of a program to extract its features. We utilize n-grams on API call sequences to construct a perfect feature set.

The presented approach utilizes a fixed window by length *N* and moves it on an API call sequence to construct the feature set.

Each move is represented as a feature. This feature set has two outstanding attributes. The first one is keeping called API's within itself, and the second special attribute of this feature set is preserving call order of each program while the feature vector for each data-item (each executable file) has fixed length. This approach helps us to fragment all API call sequences and put their information into a tabular dataset with fixed size data-items. It is obvious that the number of features for the generated dataset is pretty high. Therefore, it is hard to perform data mining algorithms on it.

So, an appropriate feature selection need to be applied in order to reduce the size of feature set and consequently reduced the size of overall dataset. Here, we use best first selection method due to its simplicity and speed. It should be noted that the best value for N is obtained by experiment. Figure 1 shows an example of tabular form of dataset. In this table each feature is represented by a subsequence of API calls and each data-item (PE-file) is represented by a vector of absence/presence of API call subsequences.



	API $SubSequence_1$	API $SubSequence_2$	API $SubSequence_3$	. . .	API $SubSequence_m$	Label
$PE\text{-}File_1$	0	1	1	…	0	+
$PE\text{-}File_2$	1	1	1	…	0	-
.	…	…	…	…	…	…
.	…	…	…	…	…	…
.	…	…	…	…	…	…
$PE\text{-}File_N$	1	0	0	…	1	+

Figure (1): An example of tabular form of dataset

4.3. Performance Measure

In this section we present some commonly used measures for evaluation our approach. There are typical objective functions to evaluate classification algorithms. We use a quantitative measure of classification performance accuracy. In the first, we present some needed terms to introduce accuracy measure. The term True Positive (TP) shows the number of correctly recognized items which were belonging to the goal set. True Negative (TN) indicates the number of correctly recognized items which were not belonging to the goal set. False Positive (FP) represents the number of incorrectly recognized items which were belonging to the goal set. False Negative (FN) illustrates the number of incorrectly recognized items which were not belonging to the goal set.

We define “Detection Rate” as the percentage of all PE files labeled “malicious” that can receive correct label by the system, as illustrate in Equation (7).

$$\text{Detection Rate} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

False Alarm Rate is the percentage labeled normal that likewise receives the wrong label by the system, as illustrate in Equation (8).

$$\text{False Alarm Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

F-measure is a measure that combines these two measures. It computes the harmonic mean of Detection Rate and False Alarm Rate, the traditional F-measure or balanced F-score, as illustrate in Equation (9).

$$\text{F} - \text{measure} \quad = 2 * \frac{\text{Detection Rate} * \text{ False Alarm Rate}}{\text{Detection Rate} + \text{False Alarm Rate}} \tag{9}$$

"Accuracy" is the last measure that we used in our experiments. The overall accuracy of the system to detect malwares and benign files is defines as in Equation (10):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

4.4. Discussion

Nowadays one of the most essential subjects is Malware detection; in this paper we try to implement an adversarial classifier system in malware detection context. This system was defined as a repeated game between 2 players (malware author and anti malware), and the base anti-malware's classifier is naïve bayes. In our approach, the malware author modifies some features of instances considering its calculation to make it like benign instances.

In this section we face with 2 steps: First, Finding set of features which must modify. This section named as malware author strategy. As mentioned, a set of features which should be changed are chosen.

Step2: After finding the set of modifiable features, features are changed. Features are achieved by moving a window with size n, in sequence of API call of program. Features with zero means are the ones that API batch are absence in program and with one means calling the API in program .Since in this article, features are binary the scenario of changing the features is to complement them, in the other words changing one to zero and vice versa.

After these 2 steps, anti-malware is expected to classify this manipulated data correctly. Then, considering result of classification, each player allocates a utility which is as follow:

If data is malware but detected as non-malware, the anti-malware is punished by -10 points and malware author gain 20 points otherwise the malware author received 0 point as utility. In the other situation if data is non-malware but detected as malware, the utility of anti-malware is -1, and if anti-malware classifies data correctly, it gains 1 point (table (1) shows the gained utility).

Here we supposed that malware author has to pay 0.2 for changing each feature and as described earlier, here we use a repeated game in which the repetition number of game is chosen to be 200 times empirically.

Testing approach, data had to be divided into testing and training set, here the number of training and tasting data is chosen randomly. In each execute number of testing is chosen randomly and assigned data for testing set and remained ones are assigned to training set.

Here proposed method and naïve bayes is tested on nGrams. The sizes of window which we use are 1-7, 10 and 12. Then both methods are tested for different measurements and discussed.

Mean False Negative illustrates the average number of positive samples which classifier makes mistake to classify them. Table (2) shows Mean False Negative results of our method and naïve bayes, it illustrate that our method is working better than naïve bayes in most cases. For example in 'nGram_1' database, Mean False Negative is 1.865 while it is 6.84 for Naïve bayes, which means our findings, work about 6 times better than naïve bayes.

Table (2): Summary of all datasets and comparison of Mean False Negative between our method and naïve bayes

Mean False Negative	Our method	Naïve Bayes
nGrams_1	1.865	6.84
nGrams_2	2.565	3.035
nGrams_3	6.355	7.77
nGrams_4	2.46	5.86
nGrams_5	4.84	3.65
nGrams_6	80.04	82.885
nGrams_7	3.035	2.365
nGrams_10	4.465	3.355
nGrams_12	4.4	4.965

One of the other measures that we test our method is Mean False Positive, which represents the average number of negative samples that are classified as positive incorrectly. Table (3) proves that Naïve Bayes is worse than our method for all tested data bases. One illustration is 'nGram_6' data base which our method's Mean False Positive is 1.715 whereas this rate is 39.515 for naïve bayes.

As it is seen in Figure (2) our method gains the superior accuracy for all data bases. This accuracy is defined for representing that how well each method recognizes malware from benign. For instance in 'nGram_5' database, accuracy is about 0.9 by using our method whereas naïve base achieves about 0.8 accuracy.

Recall rate is same as detection Rate. And is the ability of detecting all malwares and benign from each other correctly. Figure (3) proves that all data bases achieves improved recall rate by using our method.

Table (3): Comparison of Mean False Positive between our method and Naïve Bayes

Mean False Positive	Our method	Naïve Bayes
nGrams_1	7.775	9.91
nGrams_2	2.635	9.695
nGrams_3	4.46	6.615
nGrams_4	3.545	6.385
nGrams_5	1.715	41.63
nGrams_6	1.715	39.515
nGrams_7	1.7	41.99
nGrams_10	3.295	11.505
nGrams_12	7.005	12.15

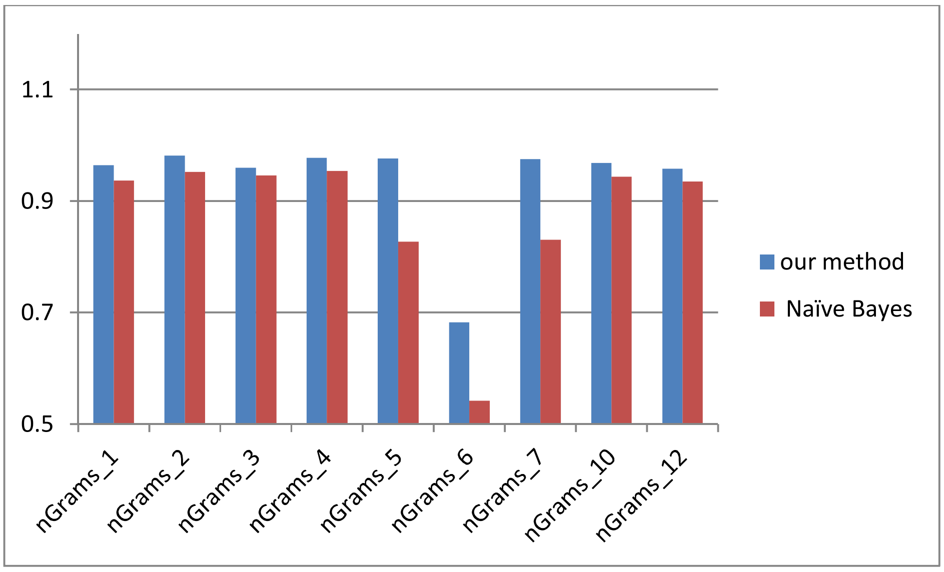


Figure (2): Comparison of accuracy which is achieved by our method and Naïve bayes

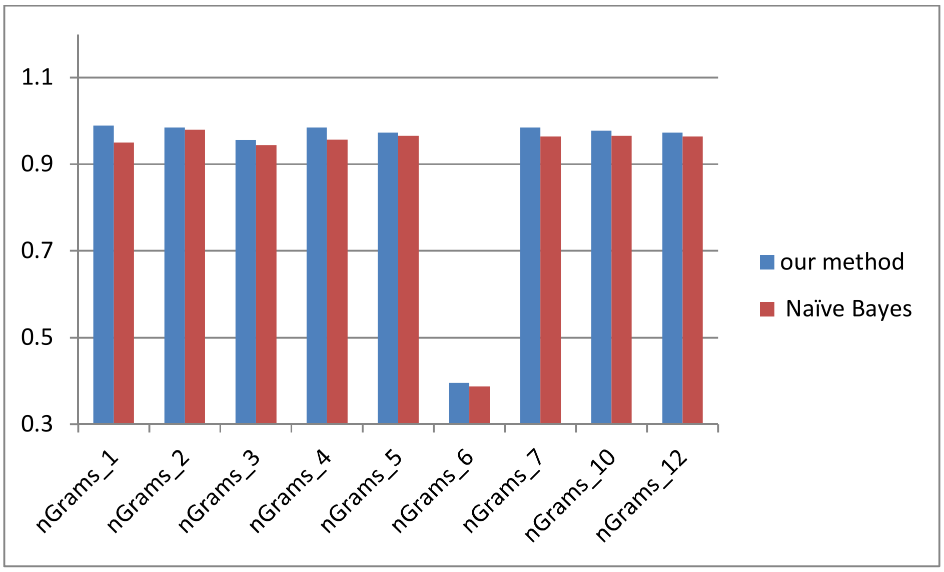


Figure (3): Evaluation of recall rate by using our method and Naïve Bayes

Precision rate is same as False Alarm Rate, and it is the ability to detect malware and benign correctly. It has been illustrated that the result of our method is better than naïve bayes (Figure (4)).

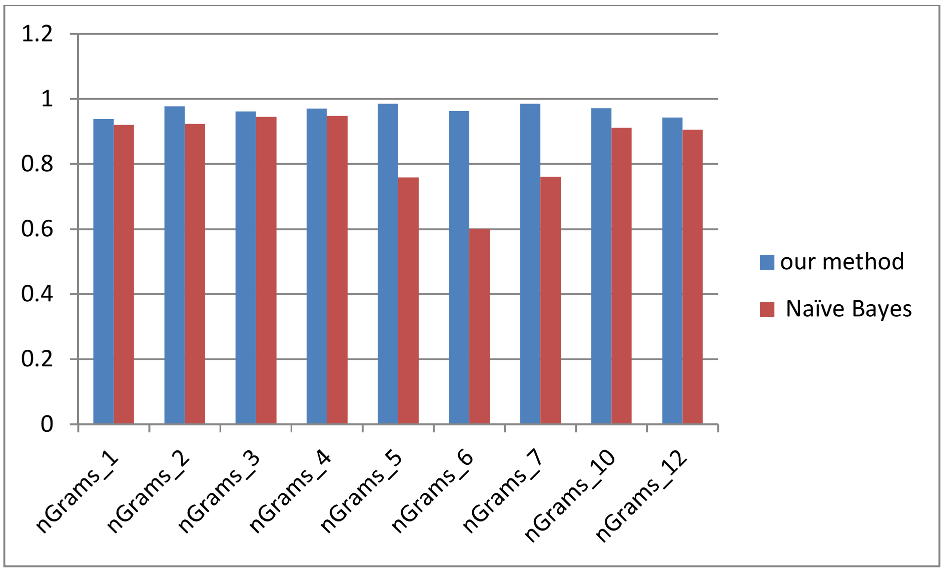


Figure (4): Evaluation of precision rate by using our method and Naïve Bayes

As mentioned F-measure is one measure of performance which takes into account both Detection Rate and False Alarm Rate and maintains a tradeoff between them. It has been seen that our method gains better results in comparison to Naïve Bayes which is shown in Figure (5).

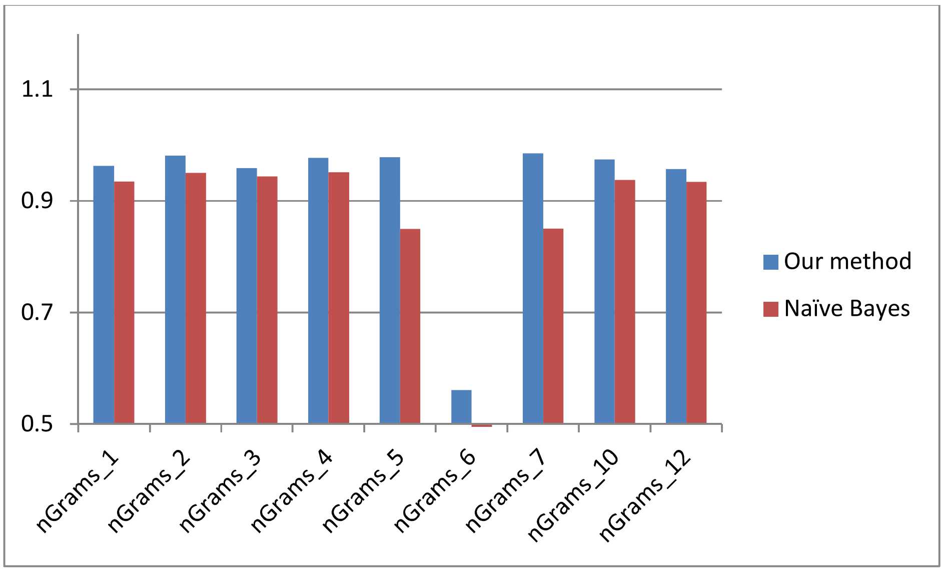


Figure (5): Showing the difference of F-measure which is achieved by Naïve Bayes and Our method

As discussed and showed our proposed method has a good results in all used measurement and overcomes naïve bayes in all situations.

5. Conclusions and Future Work

Malwares make a lot of problems for information community and is a big concern for its users. Therefore malware detection became an essential tool to defeat malwares. In this paper we try to introduce an efficient malware detection system whish its base classifier is naïve bayes and effort to detect data which are manipulated by malware author.

We test our approach with most familiar measurements and experimental results show that our approach has an efficient results and can detect malwares with an acceptable percentages and can overcomes naïve bayes in all situations. We would like to extend this model with ensemble classifiers to play against more than one adversary or different malwares types.

References

1. Idika, N. and Mathur, A.P, “A survey of malware detection techniques”, Purdue University, Citeseer, p. 48, 2007.
2. Abou-Assaleh, T. and Cercone, N. and Keselj, V. and Sweidan, R., “ Detection of new malicious code using n-grams signatures” , Proceedings of Second Annual Conference on Privacy, Security and Trust, p. 193-196, 2004.
3. Ye, Y. and Wang, D. and Li, T. and Ye, D., “IMDS: Intelligent malware detection system”, Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, p. 1043-1047, 2007.
4. Orenstein, D., QuickStudy, “Application Programming Interface (API)”, 2000.
5. M. Siddiqui, “Data mining methods for malware detection”. ProQuest, 2008.
6. C. Willems, T. Holz, and F. Freiling, “Toward automated dynamic malware analysis using cwsandbox “. IEEE Security & Privacy, p. 32-39, 2007.
7. E. Eugene Schultz, “Where have the worms and viruses gone- new trends in malware” CISSP, CISM, 2006.
8. Lowd, D and Meek, C., “Adversarial learning”. Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’05) (to appear). Chicago, IL: ACM Press, 2005.
9. Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D., “Adversarial classification”. Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ‘04), p. 99–108, Seattle, WA: ACM Press.10. 2004.
10. Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E., “A Bayesian approach to filtering junk E-mail. Learning for Text Categorization”, Papers from the 1998 Workshop. Madison, Wisconsin: AAAI Technical Re- port WS-98-05. 1998.
11. Yoav Shoham, Kevin Leyton-Brown,” MULTIAGENT SYSTEMS", 2009-2010.
12. Lewis, D.,” Naive (bayes) at forty: The independence assumption in information retrieval”. Machine Learning: ECML-98, pp. 4-15, 1998.
13. Theodore L. Turocy and Bernhard von Stengel, “Game Theory”, CDAM Research Report LSE-CDAM.2001.
14. Chartier, David, “Uncovered: Evidence that Mac OS X could run Windows apps soon”. Ars Technica, p.3-12, 2007.
15. apa.shirazu.ac.ir
16. Christodorescu, M., Jha, S., “Static analysis of executables to detect malicious patterns”. Proceedings of the 12th conference on USENIX Security Symposium-Volume 12, pp. 12-43. USENIX Association, 2003.
17. Tzermias, Z., Sykiotakis, G., Polychronakis, M., Markatos, E., “Combining static and dynamic analysis for the detection of malicious documents”. Proceedings of the Fourth European Workshop on System Security, p. 4, ACM, 2011.
18. Bayer, U., Kruegel, C., Kirda, E., “Ttanalyze: A tool for analyzing malware”. 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference. Citeseer, 2006.
19. Ian Goldberg, David Wagner, Randi Thomas, and Eric Brewer. “A Secure Environment for Untrusted Helper Applications (Confining the Wily Hacker)”. Proceedings of the Sixth USENIX UNIX Security Symposium. Retrieved 25 October 2011.