Malware Classification through HEX Conversion and Mining

A.Pratheema Manju Prabha
PG Scholar, Dept of CSE,
Dr. M.G.R. Educational and Research Institute,
Maduravoyal, Chennai – 600 095.

P.Kavitha
Assistant Professor, Dept of CSE,
Dr. M.G.R. Educational and Research Institute,
Maduravoyal, Chennai – 600 095.

ABSTRACT
The malicious codes are normally referred as malware. Systems are vulnerable to the traditional attacks, and attackers continue to find new ways around existing protection mechanisms in order to execute their injected code. Malware is a pervasive problem in distributed computer and network systems. These new malicious executables are created at the rate of thousands every year. There are several types of threat to violate these components; for example Viruses, Worms, Trojan horse and Malware. Malware represents a serious threat to confidentiality since it may result in loss of control over private data for computer users. It is typically hidden from the user and difficult to detect since it can create significant unwanted CPU activity, disk usage and network traffic. In existing systems, new malicious programs can be detected by automatic signature generation called as F-Sign for automatic extraction of unique signatures from malware files. This is primarily intended for high-speed network traffic. The signature extraction process is based on a comparison with a common function repository. The data mining framework employed in this research learns through analyzing the behavior of existing malicious and benign codes in large datasets. We have employed robust classifiers, namely Naïve Bayes (NB) Algorithm, k−Nearest Neighbor (kNN) Algorithm, and J48 decision tree and have evaluated their performance. This involves extracting opcode sequence from the dataset, to construct a classification model and to identify it as malicious or benign. Our approach showed 98.4% detection rate on new programs whose data was not used in the model building process.

Keywords
Malware, F –Sign, Naïve Bayes Algorithm, J48 Algorithm, k−Nearest Neighbor Algorithm.

1. I NTRODUCTION
Malware is short for malicious software. The basic definition of malware (malicious software) may be presented as follows: piece of software code that works for the attacker. It causes severe damage to private users, commercial companies, and governments. The recent growth in high-speed Internet connections provides a platform for creating and rapidly spreading the new malware. Several analysis techniques for detecting malware have been proposed. They are classified as to whether they are static or dynamic. In dynamic analysis (also known as behavioral-based analysis), detection is based on information collected from the operating system at runtime [3] (i.e., during the execution of the program), such as system calls, network access and files, and memory modifications. In static analysis, the detection is based on information extracted explicitly or implicitly from the executable binary/source code. The main advantage of static analysis is in providing rapid classification. Since antivirus Programs that have the potential to violate the privacy and security of a system. According to the Symantec Internet Threat Report [1], 499,811 new malware samples were received in the second half of 2007. Detection of malware is important to a secure distributed computing environment.

The technique used in commercial anti malware systems to detect an instance of malware is through the use of malware signatures. Malware signatures attempt to capture invariant characteristics or patterns in the malware that uniquely identifies it. String based signatures have remained popular in commercial systems due to their high efficiency, but can be ineffective in detecting malware variant [12][3][8][7].

The byte level content is different because small changes to the malware source code can result in significantly different compiled object code. In this paper we describe malware variants with the umbrella term of polymorphism. Polymorphism describes related malware sharing a common history of code. Code sharing among variants can be derived from autonomously self mutating malware, or manually copied by the malware creator to reuse previously authored code.

Malwise automatically identifies and unpacks the malware as necessary. The results demonstrate that the system finds high similarities between malware families using both approximate and exact matching. Additionally, our work performs in close to real-time analysis is done with quantitative analysis on efficiency.

1.1 Existing Approaches and Motivation
A malware's control flow information provides a characteristic that is identifiable across strains of malware variants. Approximate matchings of flowgraph based characteristics can be used in order to identify a greater number of malware variants. Detection of variants is possible even when more significant changes to the malware source code are introduced. Control flow [9][11][18] has proven effective, and fast algorithms have been proposed to identify exact isomorphic whole program control flow graphs and related information, yet approximate matching of program structure has shown to be expensive in runtime costs.

Poor performance in execution speed has resulted in the absence of approximate matching in end host malware detection. To hinder the static analysis necessary for control flow analysis, the malware's real content is frequently hidden using a code transformation known as packing. Packing is not solely used by malware. Packing is also used in software protection schemes and file compression for legitimate software, yet the majority of malware also uses the code packing transformation. In one month during 2007, 79% of identified malware was packed [1].

Unpacking is a necessary component to perform static analysis and to reveal the hidden characteristics of malware. In the problem scope of unpacking, it can be seen that many instances of malware utilize identical or similar packers. Many of these packers are also public, and malware often employs the use of these public packers. Many instances of malware also employ modified versions of public packers.

Malware detection has been investigated extensively, however shortcomings still exist. For modern malware classification approaches, a system must be developed that is not only effective against polymorphic and packed malware, but that is also efficient. Unless efficient systems are developed, commercial Antivirus will be unable to implement the solutions developed by researchers. We believe combining effectiveness with real-time efficiency is an area of research which has been largely ignored. In this paper we present an effective and efficient system that employs dynamic and static analysis to automatically unpack and classify a malware instance as a variant, based on similarities of control flow graphs [19].

Machine-learning methods, including the K-nearest neighbor, Support Vector Machine, and decision tree methods are used to classify executables. Basic common Techniques used for detecting malware can be categorized as shown in the figure 1.

1.2 Malware Types

Viruses are malware that infects other files and make them perform some unwanted and harmful function. In other words, a virus copies itself into another file. When the file is executed, the virus functions will also be executed.

Worms are self-propagating malware. This category spreads through networks by for example exploiting known vulnerabilities in commonly used operating systems.

Trojan horses are programs with a disguised intent, by concealing malicious pay load. Trojans may emulate the behavior of an arbitrary program such as an authentication through a login shell and retrieve an user's login credentials.

Root kits are software with the main purpose of staying concealed and undetected by anti-virus software and end-users. This type of malware was originally intended to provide root-account on UNIX-like systems.

Backdoors are malware used to bypass authentication and/or security measures. When a system has been compromised by one of the previous described types of malware, a backdoor can be installed to allow easier access later on.

1.3 Malware detection

Every program which wants to achieve its goal always takes action. No matter how crafty the malicious code is in disguise, it always has some different, relatively peculiar action which is called suspicious behavior. Behavior identification is becoming the direction of anti -virus. As Windows operating system is widely used, it rapidly catches the malware' eye and becomes the mainly growing environment and attacking object of computer vicious code. Currently most of the malicious programs are under Win32 environment. The popular vicious code for the nonce always use API function provided by Windows operating system to implement their functions, aiming at the size of code predigestion and the effect mightiness. The computer vicious program always infect normal program, and carry out their malicious purpose when the infected program is running.

String scanning is the most primitive approach to detect malware. It searches for sequences of strings (bytes) that are typical for a specific malware. Anti-virus companies organize these string sequences as signatures in databases and a local anti-virus application must download the latest signature updates to have the latest means for detecting new malware.

Wildcards is a method that allows the scanner to skip bytes or a range of bytes, for example skip bytes represented with the '?' character. Malware with early-generation obfuscation techniques can be detected with wildcards.

Algorithmic scanning methods are techniques used when the standard algorithm (such as string scanning) of the anti-virus cannot deal with a specific malware. Under this category we find filtering techniques that only scans certain files that are more exposed to infections, for example to apply boot virus signatures to boot sectors. Another technique is decryptor detection that focuses on detecting the decryption component in malware that applies encryption.

Code emulation uses a virtual machine that simulates a CPU and memory management system in order to execute the malicious executable. This technique mimics the instruction set of the CPU by using virtual registers and flags. Additionally, the functionality of the operating system must be emulated in such a way that it supports system APIs, files etc. To detect malware with this method the emulator analyzes each of the instructions that are run in the virtual machine.

Heuristic analysis is useful when detecting new malware. This technique looks for certain instructions/commands within an executable that are not found in "benign" executables. However, its biggest disadvantage is that they often find false positives.

1.4 Malware Analysis

Malware analysis is techniques that enable us to study and obtain information about a malware's behavior [17]. These techniques are also known as reverse engineering of malware. Commonly used approaches are static (code) analysis that studies the malware without executing it, and dynamic (behavioral) analysis which study malware as they execute. Even though both methods may accomplish the same goal of studying how malware works, the tools and skills required are different [13].

Static analysis is done by analyzing the source code of the malware to study how it functions. Typically, static analysis use reverse engineering tools such as disassemblers, debuggers and compilers. After applying these tools on the malware executable, the investigator or malware analyst can study the source code to gain knowledge on how the malware operates. For example how it infects systems and propagates.

The easiest way of doing a dynamic analysis is to run the malware and see what hap-pens. Note that this approach is not without problems, since you may end up destroying all information on your system or letting the malware propagate if the sacrificed host is connected to the Internet. A popular technique is to use a sandbox, which is a controlled environment for running software.

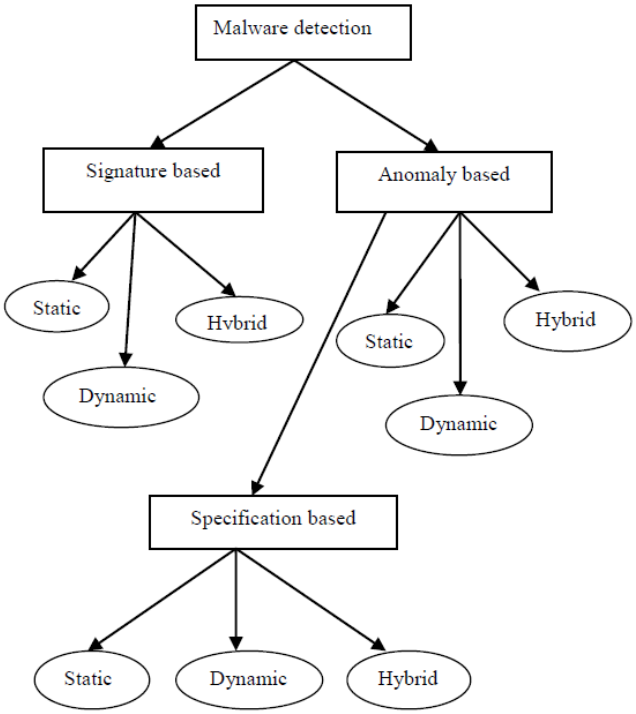

Figure 1: Detection of Malware

2. RELATED WORK

There are two main approaches for the detection of malware: static analysis and dynamic analysis. Static analysis examines the binary code to determine properties of this program without running it. This technique was first used by compiler developers to optimize the code. It is also used in reverse engineering and for program understanding. It is not long since it was used for the malware detection.

Dynamic analysis mainly consists in monitoring the execution of a program to detect malicious behaviour. In the Windows operating system, user applications rely on the interface provided within a set of libraries, such as KERNEL32.DLL, NTDLL.DLL and USER32.DLL in order to access system resources including files, processes, network information and the registry. This interface is known as the Win32 API [7]. Applications may also call functions in NTDLL.DLL known as the Native API. The Native API functions perform system calls in order to have the kernel provide the requested service. In our previous works (Alazab 2010; Alazab et al. 2010; Alazab, Venkatraman & Watters 2010) we have demonstrated how to extract and analyse these API call features including hooking of the system services that are responsible to manage files[2][3][6][14]. The extracted calls are confined to those that affect the files. Various features related to the calls that create or modify files or even get information from the file to change some value and information about the DLLs that are loaded by the malware before the actual execution are considered for the analysis.

The analysis of computer system performed offline is called static analysis, which has been employed in this research to study the patterns of the API calls within binary executables by reverse engineering the code. Static analysis provides a better understanding of the anomalous behavior patterns of the code since we adopt a methodology to perform a deep analysis into the code program and their statistical properties. The existing techniques and methods exhibit false positives as they do not perform sufficient statistical analysis to determine if the anomaly was, actually malicious (Jacob et al., 2008; Symantec Enterprise Security, 2011) [1][15]. Therefore, in this research, static anomaly-based detection analysis is adopted to perform introspection of the program code with the goal of determining various dynamic properties of API function calls that are extracted from these codes in an isolated environment.

In general, malware signatures can be classified as vulnerability-based, exploit-based, and payload-based. A Vulnerability-based signature describes the properties of a certain bug in the system that can be maliciously exploited by the malware. Vulnerability-based signatures do not attempt to detect every malicious code exploiting the vulnerability, and therefore, can be very effective when dealing with polymorphic malware. However, a vulnerability-based signature can be generated only when the vulnerability is discovered.

In 2005, studies reported in (Malan & Smith 2005) that a temporal consistency element was added to the system call frequency to calculate the frequency of API system call sequences [8][11][9]. Similarity measures were calculated using edit distance and Measuring Similarity with Intersection. The first measure was on ordered sets of native API system calls, while the second one was on unordered sets. Both similarity measures based on API gave the probabilities of two peers. The drawback is that they had considered only native API call features.

3. PROBLEM DEFINITIONS AND OUR APPROACH

While the battle between malware authors and anti-virus producers are continuing, our motivation is to find the statistical method to classify the malware.

Two approaches are employed to generate and compare flowgraph signatures. Two flowgraph matching methods are used to achieve the goal of either effectiveness or efficiency.

Exact Matching: An ordering of the nodes in the control flow graph is used to generate a string based signature or graph invariant of the flowgraph. String equality between graph invariants is used to estimate isomorphic graphs.

Approximate Matching: The control flow graph is structured in this approach. Structuring is t he process of decompiling unstructured control flow in to higher level, source code like constructs including structured conditions and iteration. Each signature representing the structured control flow is represented as a string. These signatures are then used for querying t he database of known malware using an approximate dictionary search. A similarity between flow graphs can subsequently be constructed using the edit distance.

Data mining process:

Data mining is the process of generating patterns and comparing the patterns with target resource and identifies their characteristics. In this spyware detection process, we make use of classification, association and regression techniques to mine the files and WebPages. The notion of using data mining for this purpose is that, data mining is capable of identifying the features of a data that is completely new to the system. This detection is performed on the basis of similar data set that is present in the system in the form of training data. When a collection of data with certain characteristics is provided, the system will be able to classify the new data or predict the nature of the new data entering the system based on the features of the training data set. In this case, the classification and feature detection is to identify whether the data is spyware or legitimate software. The resources that are vulnerable to spyware threat are identified and

the resource is discarded by the system. This process requires a basic training data that is used to generate the patterns of legitimate software and spyware.

Data mining is a discipline which is an intersection of different fields such as statistics, machine learning, data management and databases [16]. Often data mining is associated with knowledge discovery which is an interactive and iterative process used to find and structure information from large data sets [14].

There are two terms in data mining that is worthy of noticing; namely, descriptive modeling and predictive modeling. A descriptive model presents the most important aspects of the data, which is mainly a summary of the data that enables us to gain further knowledge. An example that falls in this category is cluster analysis that groups data objects based on their feature similarities. On the other hand, predictive models are designed to predict or forecast the outcome of a data mining process based on previously known characteristics of the observed data. Typical examples of predictive modeling are classification algorithms that assign a class label to an observed object based on feature measurements, and regression that predicts values of new input to the algorithm.

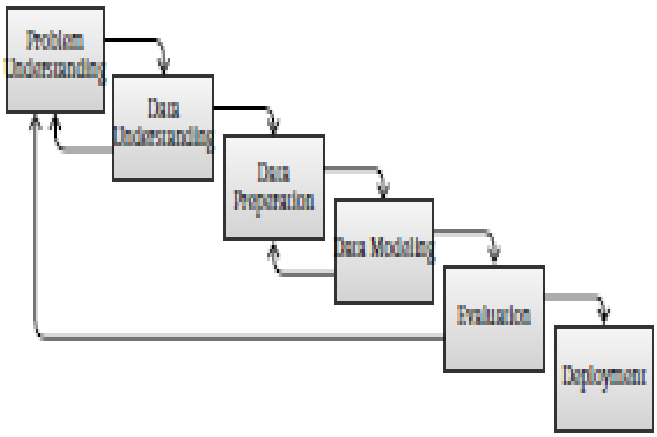

Figure 2: Data Mining Process

Figure 2 shows a popular data mining process called Cross Industry Standard Process for Data Mining (CRISP-DM) [7, 14]. The following list will give a short description of the 6 different phases of the process:

Problem understanding focuses on project objectives to further convert this know-ledge into a data mining problem.

Data understanding starts with collection of initial data. This is done to gain initial knowledge about the data that is going to be analyzed.

Data preparation is the phase where you construct the dataset from the collected data. This phase will include aspects such as feature extraction and feature selection.

Modeling starts with selecting various modeling methods. Some methods require certain representation of the data set (e.g. discrete features values). Thus, it may be necessary to take a step back to the data-preprocessing phase.

Evaluation phase focus on evaluating the previous used model. Depending on the objectives, the different evaluation criteria may be related to performance, ac-curacy etc.

Deployment is the last phase where the model is implemented and utilized.

Compared to the machine learning example presented in the previous related work the approaches are clearly similar. However, as stated by Witten and Frank [18], the process of data mining is a more practical approach. Therefore, simply put, data mining employs learning in a practical manner.

The Naive Bayes (NB) Algorithm:

The Naive Bayes algorithm is one classification method based on conditional probabilities that uses a statistical approach to the problem of pattern recognition. Literature reports that it is the most successful known algorithms for learning to classify text documents, and further it is fast and highly scalable for model building and scoring reference [6][9].

The idea behind a Naive Bayes algorithm is the Bayes Theorem and the maximum posteriori hypothesis. Bayes Theorem finds the probability of an event occurring given the probability of another event that has occurred already. Among data mining methods, Naive Bayes algorithm is easy to implement and is an efficient and effective inductive learning algorithm for machine learning.

Naive Bayes classifiers can handle an arbitrary number of independent variables whether continuous or categorical. Given a set of variables, $X = \{x_1, x_2, x_3 \ldots, x_d\}$, we want to construct the posterior probability for the event C_j among a set of possible outcomes $C = \{c_1, c_2, c_3 \ldots, c_n\}$. In a more familiar language, X is the predictor and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C|x_1, \ldots, x_d) = \frac{p(C)\ p(x_1, \ldots, x_d|C)}{p(x_1, \ldots, x_d)}.$$

Where $p(C_j|x_1, \ldots, x_d)$ is the posterior probability of class membership, i.e., the probability that X belongs to C_j.

In practice we are only interested in the numerator of that fraction, since the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The "naive" conditional independence assumptions come into play: assume that each feature x_i is conditionally statistical independent of every other feature x_j for $j \neq i$. This means that

$$p(x_i|C, x_j) = p(x_i|C)$$

For $i \neq j$, and so the joint model can be expressed as

$$p(C, x_1, \ldots, x_d) = p(C)\ p(x_1|C)\ p(x_2|C)\ p(x_3|C)\ \cdots$$
$$= p(C) \prod_{i=1}^{d} p(x_i|C).$$

This means that under the above independence assumptions, the conditional distribution over the class variable C can be expressed like this:

$$p(C|x_1, \ldots, x_d) = \frac{1}{Z} p(C) \prod_{i=1}^{d} p(x_i|C)$$

where Z (the evidence) is a scaling factor dependent only on $x_1, \ldots, x_d$, i.e., a constant if the values of the feature variables are known.

Finally, we can label a new case F with a class level C_j that achieves the highest posterior probability:

$$\text{classify}(F_1, \ldots, F_d) = \underset{c}{\text{argmax}}\ p(C = c) \prod_{i=1}^{d} p(x_i = F_i|C = c).$$

Among data mining methods, Naive Bayes algorithm is easy to implement and is an efficient and effective inductive learning algorithm for machine learning. Figure 3 provides the overall

accuracy rate for malware detection achieved through our experiments using Naive Bayes with k cross validations, k= {2,3,4,5,6,7,8,9,10}.

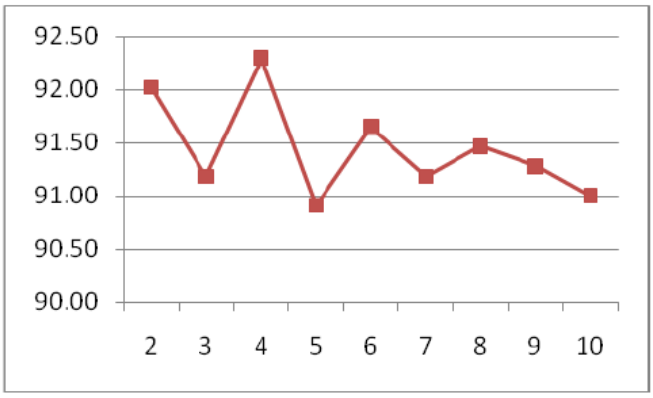

Figure 3: Performance of Naïve Bayes

J48 Algorithm:

J48 classifier is a C4.5 decision tree used for classification purposes. In order to classify a new item, the classifier first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly. This feature that is able to tell the most about the data instances for classifying them the best is said to have the highest information gain.

Algorithm:

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set $S = s_1, s_2, \cdots$ of already classified samples. Each sample $s_i = x_1, x_2, \cdots$ is a vector where $x_1, x_2, \cdots$ represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, \cdots$ where $c_1, c_2, \cdots$ represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) [2][4] that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller sub lists.

This algorithm has a few base cases.

A. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.

B. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.

Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value

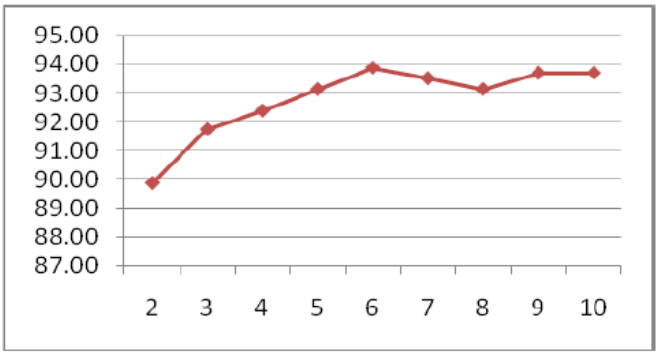

Figure 4: Performance of J48

Among the possible values of this feature, if there is any value for which there is no ambiguity, that is, when the data instances falling within its category have the same value for the target variable, then that branch is terminated and the target value arrived is assigned to it.

4. PERFORMANCE EVALUATION

The classification algorithms require training data to train the formulated models, and testing data to test those models. Validation of the models is achieved by making a partition on the database of malware and benign for carrying out the experiments. The cross-validation is a technique used for evaluating the results of a statistical analysis by generating an independent dataset for Malware and benign. The most common types of cross-validation are repeated random sub-sampling validation and K-fold cross-validation [4][8] (Hand, Mannila, & Smyth, 2001). For this research study of Malware and Benign classification, K-fold cross-validation has been selected for validation as it is commonly adopted for many classifiers[1][9][12] (Witten and Frank, 2010; Bhattacharyya, etal 2011).

In k-fold cross-validation the data is first partitioned into k sized segments or folds. Then, k iterations of training and validation are performed such that within iteration a different fold of the data is held-out for validation while the remaining k-1 folds are used for learning. The advantage of K-Fold cross-validation is that all the examples in the dataset are eventually used for both training and testing. Also, all observations are used for both training and validation, and each observation is used for validation exactly once.

The following metrics are used to evaluate our method with an existing system

True positive (TP): benign programs are correctly identified

True negative (TN): malicious programs are correctly identified.

False positive (FP): benign programs are wrongly identified as malicious.

False negative (FN): malicious programs are incorrectly classified as benign.

The performance of our methodology was evaluated using the true positive rate, false positive rate which are defined as follows,

True positive rate (TPR): percentage of benign programs correctly identified.

TPR= (TP/TP+FN)

False Positive Rate (FPR): percentage of malicious programs wrongly identified.

FPR= (FP/TN+FP)

500 virus file and 300 benign file are given as input. From which the accuracy of true positive rate (TPR) of our proposed methodology is higher than existing system and false positive rate (FPR) of our proposed methodology is lower than existing system.

5. EXPERIMENTAL WORK

In order to perform our experiments, we collect significantly large malware database as stated in the system design section. To obtain more accurate results we count in the subfamilies that contain maximum number of samples in our dataset. In this manner, experiments are carried out 1056 samples belonging to ten families, five of them have two subfamilies, and therefore there exists 15 subfamilies in our dataset shown in Table 1.

There are two main parameters in the experimental setup: the first parameter is the size of the n-grams and the second parameter is the number of the list size which is constituted by ranking the n-grams according to their df values in the subfamilies. The size of the n-grams, denoted by n, allows us to decide how long in bytes the n-gram will be.

Subfamily Name	Instance Number	Subfamily Name	Instance Number
Win32-Vobfus.Y	13	Win32-Sality.AT	64
Win32-Alureon.H	19	Win32-Small.AHY	69
Win32-Ramnit.F	19	Win32-Renos.NS	95
Win32-Virut.BG	19	Win32-Sality.AM	100
Win32-Alureon.CT	22	Win32-Renos.LT	137
Win32-Agent.ACF	23	Win32-Vobfus.gen!D	183
Win32-Viking.CR	30	Win32-Ramnit.B	200
Win32-Vobfus.AH	42		

Table1: Number of the Instances for each Subfamily

A. Data Collection

Data set consists of 100 binaries out of which 90 are benign and 10 are Malware binaries. This hosts information about different types of malicious software.

B. Byte Sequence Generation

We have opted to use byte sequences as data set features in our experiment. These byte sequences represent fragments of machine code from an executable file. We use xxd, which is a UNIX-based utility for generating hexadecimal dumps of the binary files. From these hexadecimal dumps we may then extract byte sequences, in terms of *n*-grams of different sizes.

C. Dataset Generation

Two ARFF databases based on frequency and common features were generated. All input attributes in the data set are represented by Hexadecimal codes. These ranges are represented by either 0 to 9, A to F.

D. Classification

A Naive Bayes classifier is a probabilistic classifier based on Bayes theorem with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. This of course, is seldom true for real-life applications. Nevertheless, the algorithm has shown good performance for a wide variety of complex problems. J48 is a decision tree-based learning algorithm. During classification, it adopts a top-down approach and traverses a tree for classification of any instance. Moreover, Random Forest is an ensemble learner. In this ensemble, a collection of decision trees are generated to obtain a model that may give better predictions than a single decision tree.

In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods. In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations.

6. CONCLUSION

Data mining techniques perform better than traditional techniques such as signature-base detection and Heuristic-based detection.

In this paper, we have proposed a two phase analysis technique to detect malicious code injection attack by using static analysis and classification model constructed by frequency of occurrence of opcode extracted from a dataset. Since we are using the two phase analysis technique, files with obfuscated code is detected in first phase by static analysis and there is no need of the second phase. Files without obfuscated code are detected in second phase by classification model which classifies them as malicious or benign.

The precision of detection of the algorithm has been validated by the training and testing of abundant sample space. The technique is a promising method to detect the win32 virus.

The proposed system is efficient as it uses filter approaches to be able to successfully detect malware with a smaller feature set.

The system is signature-free and does not require knowledge or detailed study about the API sequence of execution to classify a malware.

7. REFERENCES

[1] Symantec, "Symantec internet security threat report: Volume XII," effectiveness and efficiency of our work is in [9]. In their Symantec 2008.

[2] F-Secure. (2007, 19 August 2009). F-Secure Reports Amount of variants such as the Netsky family of malware u sing the Malware Grew by 100% during 2007.

[3] K. Griffin, S. Schneider, X. Hu, and T. Chiueh, "Automatic Generation of String Signatures for Malware Detection," in *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009* , Saint- Malo, France, 2009.

[4] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *4th Virus Bulletin International Conference* , 1994, pp. 178-184.

[5] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *International*

Conference on Knowledge Discovery and Data Mining , 2004, pp. 470-478.

[6] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology,* vol. 1, pp. 13-23, 2005.

[7] M. Gheorghescu, "An automated virus classification system," in *Virus Bulletin Conference* , 2005, pp. 294-300.

[8] Y. Ye, D. Wang, T. Li, and D. Ye, "IMDS: intelligent malware detection system," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* , 2007.

[9] E. Carrera and G. Erdélyi, "Digital genome mapping–advanced binary malware analysis," in *Virus Bulletin Conference* , 2004, pp. 187-197.

[10] T. Dullien and R. Rolles, "Graph-based comparison of Executable Objects (English Version)," in *SSTIC* , 2005.

[11] I. Briones and A. Gomez, "Graphs, Entropy and Grid Computing: Automatic Comparison of Malware," in *Virus Bulletin Conference* , 2008 pp. 1-12.

[12] S. Cesare and Y. Xiang, "Classification of Malware Using Structured Control Flow," in *8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010)* , 2010.

[13] G. Bonfante, M. Kaczmarek, and J. Y. Marion, "Morphological Detection of Malware," in *International Conference on Malicious and Unwanted Software, IEEE* , Alexendria VA, USA, 2008, pp. 1-8.

[14] R. T. Gerald and A. F. Lori, "Polymorphic malware detection and identification via context-free grammar homomorphism," *Bell Labs Technical Journal,* vol. 12, pp. 139-147, 2007.

[15] X. Hu, T. Chiueh, and K. G. Shin, "Large-Scale Malware Indexing Using Function-Call Graphs," in *Computer and Communications Security* , Chicago, Illinois, USA, pp. 611-620.

[16] Henchiri.O, Japkowicz.N (2006), ―A Feature Selection and Evaluation Scheme for Computer Virus Detection , Data Mining, ICDM '06. Sixth International Conference on Digital Object Identifier: 10.1109/ICDM.2006.4 Publication Year: 2006 , Page(s): 891 – 895

[17] Moskovitch.R, Feher.C, Tzachar.N, Berger.E, Gitelman.M, Dolev.S, and Elovici.Y (2008) ―Unknown Malcode Detection Using OPCODE Representation , ISI 2008, June 17-20, Taipei, Taiwan.

[18] Bozagac.C.D, ―Application of Data Mining based Malicious Code Detection Techniques for Detecting new Spyware , White paper, Bilkent University 2005.

[19] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in Computer Virology,* vol. 7, pp. 233-245, 2011.

[20] Moskovitch.R, Feher.C, Tzachar.N, Berger.E, Gitelman.M, Dolev.S, and Elovici.Y (2008) ―Unknown Malcode Detection Using OPCODE Representation , ISI 2008, June 17-20, Taipei, Taiwan.