IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 9, September 2017

Malware detection using Machine Learning Algorithms

Mohammad Danish Khan[1], Mohd Tanveer Shaikh[2], Rafia Ansari[3], Mahenoor Suriya[4], Sonalii Suryawanshi[5]

Student, Computer Department, Rizvi College of Engineering, Mumbai, India[1,2,3,4]

Asst. Prof., Computer Department, Rizvi College of Engineering, Mumbai, India[5]

Abstract: Current antivirus software's are effective against known viruses, if a malware with new signature is introduced then it will be difficult to detect that it is malicious. Signature-based detection is not that effective during zero-day attacks. Till the signature is created for new (unseen) malware, distributed to the systems and added to the anti-malware database, the systems can be exploited by that malware. But Machine learning methods can be used to create more effective antimalware software which is capable of detecting previously unknown malware, zero-day attack etc. We propose an approach that learns from the header data of PE32 files. We examine various features of the PE32 header and check those which are suitable for machine learning classifier. We hypothesize that machine learning classifiers can tell apart the difference between malware and benign software. Various machine learning methods such as Support Vector Machine (SVM), Decision tree, Logistic Regression and Naive Bayes will be used

Keywords: Malware, detection, Feature extraction, machine learning, Classifier, SVM, Decision Tree, Naïve Bayes, Header Data.

I. INTRODUCTION

Malicious code is "any code added, changed, or removed from a software system to intentionally cause harm or subvert the system's intended function" [1]. A new malware specimen emerged every 4.6 seconds, in 2016. In the first quarter of 2017 this only takes 4.2 seconds. In 2016, 6,834,446 new malware specimens were counted. This is an increase of 32.9%. On average this is 780 per hour. And this trend is continued in the first quarter of 2017. The 1,852,945 new malware specimens are 72.6% above the figure a year before. With an average of 858 per hour this is 10.0% above the average of 2016. So as the number of malware is increasing drastically, it is causing great inconveniences and also leading to economic losses [2].

Traditional anti-virus programs are mostly based on the decade's old "Signature-Based Methodology". Only "known" malware can be identified using signature based malware detection. All objects have attributes that can be used to create a unique signature. The digital signature is determined by scanning an object. When an anti-malware program discerns an object as malicious, its signature is added to a database of known malware. Hundreds of millions of signatures that identify an object as malicious are contained in these databases. Unfortunately, it is hard to keep up with the increasing number of new versions of malicious code appearing day-by-day.

These newly released forms of malware can be distinguished from benign files using Supervised Machine Learning Classification. Also, it helps to overcome few of the drawbacks of signature-based malware detection. Three types of features that have been commonly used in experiments are n-grams over machine code instructions, API call sequences, and PE32 header data. When using machine learning, firstly a labeled dataset must be built for training, to classify files as malicious or clean. Specific characteristics of the file are used to derive features of the file and then each file is designated as either benign or malicious. A model is generated by analyzing training records using learning algorithms, this model maps the association of file features and labels. The classifier/ learning algorithm is used to predict whether the new file is benign or malicious.

In our approach, we will be using 32-bit Portable Executable PE32 header data to detect malicious executable files. The PE32 is the format for executable files for 32-bit windows operating system. The header data of PE32 contains many fields which reports the metadata of the file (such as how long the executable section of the code is, in what year the file was made. etc.) and the file structure of the executable. A statistical comparison between the header data of a benign file and a malicious PE32 file was published by Yonts [3] in 2012, which exhibited that malicious and benign programs usually differed in certain elements of the header data.

Since many of the header features are innate to the structure of a program, it is difficult for an attacker to maneuver the program without affecting its function. Thus, the attacker will not be able to exploit the detection rules by simply modifying the header data alone. New malware can be detected by an antivirus program based on Machine Learning if the structure of the novel malware is similar to a known malware.

In the proposed system we will use following libraries of python;

- NumPy
- sciPy
- matplotlib
- panda
- Scikit-learn

II. RELATED WORK

An Several attempts have already been made to detect malware. Various approaches have been used in various research papers. Maloof and Kolter [4] had collected 1971 benign and 1651 malicious executables in Windows PE format for training and testing the classifier. They used Hexdump utility to convert each executable to hexadecimal format. Top 500 n-grams with n-gram size of 4 were selected as features. Wakaito Environment for Knowledge Acquisition (WEKA) Java Implementation: IBk, naive Bayes, a support vector machine (SVM), and a decision tree were implemented on the n-grams. Boosted J48 outperformed all other methods in their studies by detecting 98% of the new malware. IBk and SVM also performed well in their experiments.

Schultz et al. [5] used data mining techniques to detect whether a new (unseen) executable is a malware or not with good accuracy. The data set contained 4266 programs of which 3265 were malicious binaries and 1001 were clean programs. Binary profile was extracted from each example of the data set and from these binary profiles features were extracted. Features extracted were static properties of the binary and the files were not executed. System resource information, strings and byte sequences were also extracted from executables as different types of features. To extract resource information from executables GNU's Bin–Utils was used. There were three types of features: 1. The list of DLLs 2. The list of DLL function calls 3. The number of different function calls from each DLL.5-fold cross validation was used by authors. RIPPER, an inductive rule based learner was used to learn rules from the training dataset. RIPPER with DLL functions used showed an accuracy of 89.36%. Naïve Bayes and Multi-Naïve Bayes algorithm were also used to detect malware. Multi-Naïve Bayes performed better with a detection rate of 97.76 which is more than double the rate of signature based algorithms.

In 2012, Yonts [3] studied the possibility and utility of detecting malware based on header data of PE32 files. Yonts collected 2.5 million malicious PE32 files for the dataset. Data Mining was used to determine malicious indicators. The detection rules distinguished well between malicious and benign files by using the attributes individually. The author tried to find the effect of an individual attribute in distinguishing between malicious and benign but didn't use the attributes altogether to build a machine learning classifier.columns.

III.DATA COLLECTION

The database will consist of clean and malicious P32 executables. The malicious are obtained from a collection of viruses at VX Heavens website [7]. We obtained benign executables from the folders of computers running Windows 7, Windows 8 and Windows 10. Additional executables were downloaded from SourceForge. We used python module pefile to extract features from PE32 executables. 28 different header data attributes were extracted as features from PE32 executables. 5-fold cross validation was usedused.

IV.WORKING PRINCIPLE

In our proposed system the data set consists of PEF32 file. The working is divided into two phases: training and testing. Python PE module is used to extract features, that is, the header data of the file in training phase the features are generated and saved in a database. This database is used to train the classifier.

We have used 5-fold cross validation. Dataset is divided into 5 parts. 4 parts were used for training and 1 part for testing. Then the process was repeated 5 times taking different new part for testing each time. We extract features from the executables and create test file. This test file is provided to the classifier along with the database created in the training phase. Then the classifier predicts whether the executable is malicious or benign. If its malicious then it is deleted from the system.

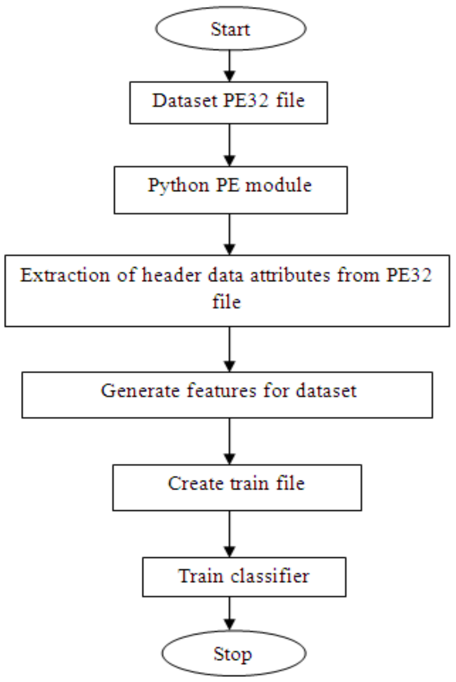
Start
Dataset PE32 file
Python PE module
Extraction of header data attributes from PE32 file
Generate features for dataset
Create train file
Train classifier
Stop

Fig 1. Flow chart of training phase

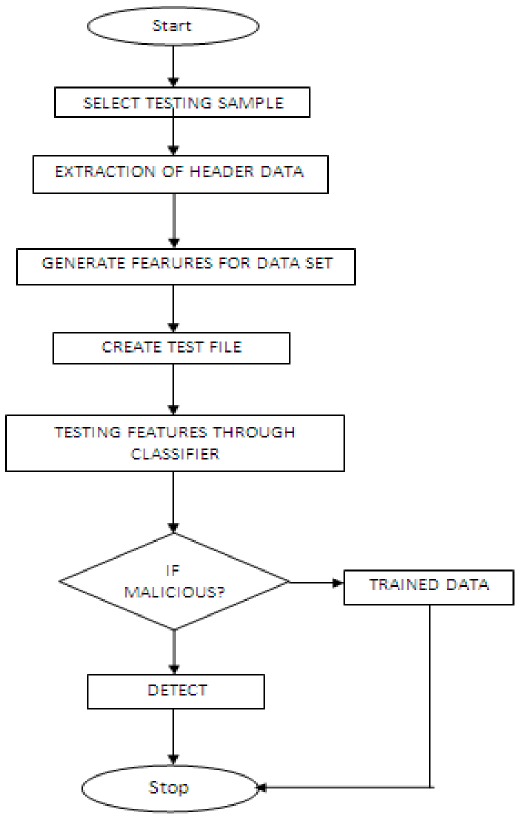
Start
SELECT TESTING SAMPLE
EXTRACTION OF HEADER DATA
GENERATE FEARURES FOR DATA SET
CREATE TEST FILE
TESTING FEATURES THROUGH CLASSIFIER
IF MALICIOUS?
TRAINED DATA
DETECT
Stop

Fig 2. Flow chart of testing phase

V. CLASSIFICATION METHODOLOGY

An algorithm that implements classification, mostly in a concrete implementation, is known as a classifier. The term "classifier" often refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category. “Our approach is to use techniques of machine learning, data mining and text classification. We discuss each of these methods in the below sections. Since the task is to detect malicious executables, we say the malicious class as the positive class and benign class as the negative class. All classifier techniques are discussed in detail. Each experiment was broken into a set of trials in which we sampled the database for training and test data, trained a classifier, and measured classifier performance on test data. Our sampling algorithm generates non-intersecting training and test samples from a given dataset. We specify the desired number of records as well as the malware prevalence, which is the percent of records in the sample that correspond to malicious files. Specifically, we used the scikit-learn Python module to implement:

A. SVM

SVMs introduced in COLT-92 by Boser, Guyon and Vapnik. A Support Vector Machine (SVM) is a unique classifier formally defined by a separating hyper plane. Basically, in SVM different lines or graph is used to discriminate or categorized between different classes or examples Separation of classes. That’s what SVM does. It draws lines to separate classes like kernel logistic regression, SVMs are a form of kernel linear classifier. However, the SVM uses an objective which more explicitly encourages good generalization performance. SVMs do not comfortably within a probabilistic framework and as such we describe them here only briefly. Since SVM was used by Kolter and Maloof [4] and they got good results, taking to reference of those points it might give us good results too. In the SVM literature it is common to use +1 and -1 to denote the two classes. For a hyper plane defined
by weight w and bias b, a linear discriminant is given by
$w^T x + b$ **{**>= 0 class +1 ,< 0 class -1
Generally we use svm when there is not too much of training data set.

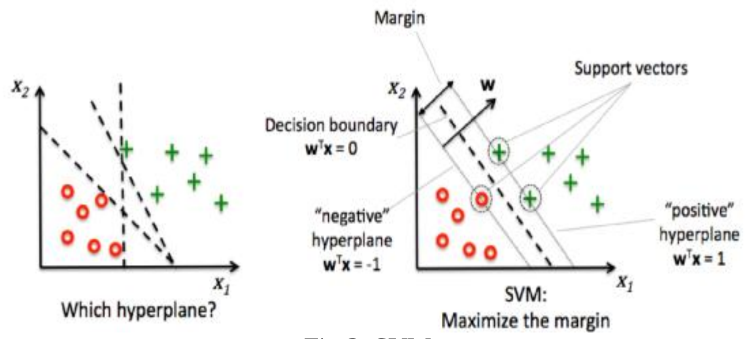

Fig 3. SVM

B. Decision tree

Tables Decision tree is like a flow chart diagram consisting of root node, leaf node and branches. Root node is the starting point of the tree and leaf nodes are the condition based results; branches are arrows connecting to node shows flow from question to answer. Internal nodes corresponding to attribute and leaf nodes corresponds to class labels Decision trees helps obtaining the result in faster and in an efficient way. In this we will use CART algorithm that is introduced by Leo Breimean to refer decision tree algorithm that can be used for classification and regression Trees. CART has following types of decision trees,
Classification trees: In this target variable is categorical and tree is used to identify the class
Regression tree: target variable is continuous and tree is used to predict its value.

The performance element is uses the attribute and their instance to traverse root node to leaf node and it predicts the class of leaf node and learning element is used to draw new trees based on their attribute. It creates node, branches and children for the attributes and its values, removes the attribute from further consideration, and distributes the examples to the appropriate child node. And this process repeats until all node contain same class. Since Kolter [4] used the above method and got the good result hence we will use this method too.

C. Naive Bayes

Next classifier we will discuss is naïve base classifier, itis likelihood computes the program is malicious given that feature in the program. Schultz et al. [5] achieved good detection rate using this classifier. This method uses both string and byte sequence data to compute binary malicious given on its feature. Nigam et al. (Nigam, McCallum, Thrun, & Mitchell, 1998)Performed a same experiment when they classified text documents according to which newsgroup they initiated from. In this method each executable feature treated as text document and classified based on that. The assumption is taken as binaries contain same feature such as signature, machine instruction etc.

Naive Bayes is a probabilistic that is used in information retrieval and text classification. It stores as its concept description the prior probability of each class, P(Ci), and the conditional probability of each attribute value given the class, P(vj|Ci). It approximates these quantities by counting in training data the frequency of occurrence of the classes and of the attribute values for each class. Then, assuming conditional independence of the attributes, it uses Bayes' rule to compute the posterior probability (the statistical probability that a hypothesis is true calculated in the light of relevant observations.) of each class given some unknown instance, returning as its prediction the class with the highest such value.

$$C = \operatorname{argmax} P(C_i) \prod P(v_j|C_i)$$

VI. CONCLUSION

We have studied various papers in which they extracted features from executables and used classifier to detect if its malicious or benign. In traditional system signatures of malware were created and these signatures were matched with executables to find out if its malware. The problem with traditional system was that whenever new malware is introduced in the system, to detect that it is malware signature of that malware needs to created and updated in the anti-malware software database, till the time the signature is distributed or process completed, the system is at high risk and malware can exploit the system during that time. So, to protect the system during such time in we propose a system in which the classifier extracts the features from the new (unseen) file and detects if its malicious or benign.

REFERENCES

[1] G. McGraw and G. Morisett, "Attacking malicious code: A report to the Infosec Research Council," IEEE Software, 2000.
[2] R. Benzmüller, "Malware trends 2017," GData Security Blog, 10 04 2017.
[3] J. Yonts, "Attributes of Malicious Files," SANS Inst. InfoSec Read. Room, 2012.
[4] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," in Proceedings of the International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 2004.
[5] M. G. Schultz, E. Eskin, F. Zadok and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001, Oakland, CA, 2001.
[6] K. Nigam, A. McCallum, S. Thrun and T. Mitchell, "Learning to Classify Text from Labeled and Unlabled Documents," AAAI, 1998.
[7] VX-Heavens: http://vxer.org/