Evading the Machine Learning Detector: A Virus Perspective

Group 10
Yuan Xiao, Xiaokuan Zhang

04/27/2017

Abstract

1 Introduction

1.1 Problem Description

There are a bunch of anti-virus software on market nowadays, such as Norton and McAfee. They can detect malicious apps/binaries and prevent them from jeopardizing the user's data. However, in our project, we are not aiming at improving the detection rate of anti-virus software; instead, we want to study from a virus' perspective: how should a virus disguise itself to evade the detection? To answer this question, we designed a novel evasion scheme and conducted experiments the test its efficiency on a dataset from UCI Machine Learning Repository. We will introduce them in more detail in Section **??**.

1.2 Classification Algorithms

In this report, we use LibSVM [1] as our machine learning algorithm, which implements Support Vector Machine (SVM). To compare with our feature selection scheme, we use Weka's `weka.attributeSelection.CfsSubsetEval` module as the feature selection method. It implements correlation-based feature selection. We will present the detail of algorithms in Section 3.

1.3 Hypothesis

1.4 Originality

1.5 Result Summary

Our results show that our scheme works well and can truly fool the detector.

2 Background

The author of the dataset we used generated the dataset based on the n-gram feature set construction from Masud et al.'s paper [5]. In this paper, they proposed a new way to construct features for binary files, called N-gram Features. An n-gram may be either a sequence of n bytes or n assembly instructions, or n DLL function calls, depending on whether they want to extract features from binary or assembly program, or DLL call list. For example, the 3-grams corresponding to the 4 bytes sequence "a1b2c3d4" are "a1b2c3" and "b2c3d4". They constructed Binary/Assembly/DLL N-gram feature set, and proposed a hybrid model to combine the three feature sets. Then, they choose best 500 features for each set based on entropy gain. Later on, they use LibSVM to test their models. The hybrid model works best for both datasets (Accuracy:96.30%, 96.15%).

There is an older paper about detecting malicious executables by Schultz et al. [6]. In their paper, they constructed DLLs, GNU Strings, and Byte Sequence features. The DLL and Byte Sequence features are similar to the way of constructing our dataset, but they did not use N-gram feature. They further tested their features using different machine learning algorithms, i.e., RIPPER, Naive Bayes, and Multi-Naive Bayes. The best one (Multi-Naive Bayes) can achieve 97.76% detection rate.

3 Methodology

3.1 Tools

3.1.1 LibSVM

LibSVM [1] is an open-source software that implements Support Vector Machine (SVM) algorithm [2]. It is one of the most popular tools when it comes to SVM implementation. It is the main tool that we use to test the performance our evasion scheme.

3.1.2 Weka

Weka [3] is a collection of machine learning algorithms for data mining tasks. It has a nice GUI interface and can accept different file inputs (e.g, .arff, .csv, etc.). In our project, our main task is to find important features to add or delete (feature selection). Therefore, we use Weka as a tool for feature selection and compare the performance with our feature selection scheme.

3.2 Algorithms

3.2.1 SVM

SVM [2] is a supervised learning model that analyzes data for classification or regression. Provided with a set of training examples, which are marked with their belonging categories, a SVM algorithm performs to build a model so as to recognize and assign testing examples to the predicted categories.

3.2.2 Correlation-based Feature Selection

We choose the `weka.attributeSelection.CfsSubsetEval` module as the feature selection method to compare with our scheme. It implements correlation-based feature selection, which evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them [4].

4 Experiments

The whole experiment is made up of four steps. First, data preprocessing. Second, model training with LibSVM and testing with unmodified test set. Thus we have a criterion for evaluation. Third, data modification in the test set and testing using trained libSVM model. Fourth, performance comparison. The methods for model training and data modification are already described in section 3 and the rest parts will be discussed in the this section.

4.1 Original Dataset

The dataset includes instances of malicious executables (computer virus) as well as non-malicious executables (normal programs). The features are extracted from real-world malicious and non-malicious executables. The dataset was published in March 2016 in UCI Machine Learning Repository. Within one year, the web hits of the dataset has already reached over 200,000. It is obvious that the dataset arouses great interest in the machine learning community.

The dataset consists of 373 instances, of which 301 are malicious and 72 are benign. Each instance has 500 hex features and 30 DLL features. Notice that on the UCI webpage it claims that there are 13 DLL features but we found from the raw data that there are actually 30 of them. All the attributes are binary, meaning a certain feature exists or not.

However, the primary weakness of the dataset is that the sample number is relatively low. However, the goal of our project is not to exhaustively find the best machine learning model to classify malicious and non-malicious executables. Thus we think the low sample amount is tolerable. In addition, after searching online, the UCI dataset is the only dataset available that do not require extracting features by ourselves.

4.2 Data Preprocessing

The raw data is generally LibSVM-format conformant. The class attribute at the beginning of an instance marks whether it is benign or malicious. -1 stands for a malicious instance and +1 means benign. However, at the end of each instance there is an additional -1. We need to remove it before passing it to LibSVM as input.

In addition, we randomly seperate the dataset into two sets. One consists of 80% of the dataset as the training set while the rest as the test set.

4.3 Work Flow

After data preprocessing, we pass the training set to LibSVM to train a classification model. With the different cores, we will have different models. The following implementations apply to all of the models.

Given a trained classification model, we apply it to the test set first and get the test results as a criterion for later evaluation. With the unmodified test set, we expect the performance of the model to classify malicious instances to be high enough. Otherwise, the model itself does not make sense and it is pointless to fool an inaccurate classification model.

After that, we apply different attribute modification methods to the test set. Notice that only malicious instances are modified since our only goal is to disguise a malicious executable as a benign one. With the different modified test sets, we re-do the testing phase with unmodified classification model. Then we compare the new test results with those of original test set and see whether any malicious instances successfully escape the detection.

In the end, we compare the escape results of different modification methods to find which of them perform best and how to achieve a best performance. Meanwhile, we also do comparison of such modifications with existing anti-antivirus techniques for real-world feasibility discussion.

4.4 Results

4.5 Result Analysis

5 Discussion

5.1 Real-world Feasibility

In this section we briefly discusses the feasibility of our modification towards the malicious executables and compare them with existing anti-antivirus methods deployed by real-world virus.

In reality, it is much more difficult to remove features than to add features. The features to remove mostly denote the malicious behavior of a virus and are thus hard to take out. In contrast, we can easily add features as dummy code that will never be executed.

Our modification schemes in fact serves as a combination of two existing anti-antivirus methods. The first is to disguise as popular file formats such as .pdf or .docx or programs such as calc.exe or notepad.exe. The second is polymorphic virus. It mutates on each copy by adding different types of NOP instructions.

References

[1] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[3] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[4] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[5] Mohammad M Masud, Latifur Khan, and Bhavani Thuraisingham. A hybrid model to detect malicious executables. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1443–1448. IEEE, 2007.

[6] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.