Capstone Project

Dileep Tiku

Machine Learning Engineer Nanodegree June 11th, 2016

Definition

Project Overview

Computer Malware poses a security risk for businesses throughout the world. With the advent of Industrial internet of things besides compromising information technology assets, malware can impact operational technology which has potential of causing harm to society at large. Using advanced machine learning algorithms, computer malware can be identified using behavioral analysis of the software. This will allow organizations to use the malware detection model, to assist in advanced threat protection not possible only by traditional methods of malware detection, which are signature based. In this project, I created a model that analyzes the Windows API calls generated by executing a software in a controlled environment and then classifies the software as malware or benign. The model is trained by using the malicious training sets from CSMining[1] website.

Problem Statement

Develop a behavior analysis model that can identify whether a software is malware or not.

- Identify the algorithm to process the Windows API Calls list in the CSMining data.
- Apply the algorithm to extract the features from API Calls for classification
- Train a classifier to use the features extracted from the API Calls.
- Test the classifier trained to use the extracted features from API Calls.

Metrics

Accuracy is a common metric for binary classifiers; it takes into account both true positives and true negatives with equal weight.

$$accuracy = correct\ predictions/all\ predictions$$

1 http://csmining.org/index.php/malicious-software-datasets-.html

This metric was used when evaluating the classifier because false negatives compromise the security of an entity and false positives are responsible for wasting a lot of time for the scarce security personnel in an organization.

Analysis

Data Exploration

The CSMining dataset used for this project is a collection 388 software traces. The software traces include APICall lists generated by the execution of malware and benign software. Out of a total of 388 traces, 320 are from malware execution and 68 are from benign software execution. The 320 traces generated from different types of malware including worms, trojans and viruses. The API calls are repeated in the dataset for completeness of the calls.

A subset of the Windows APU/System-Calls which are considered informative for differentiating a malware from a benign software are logged by API monitors when a designated program is running in the system[1]

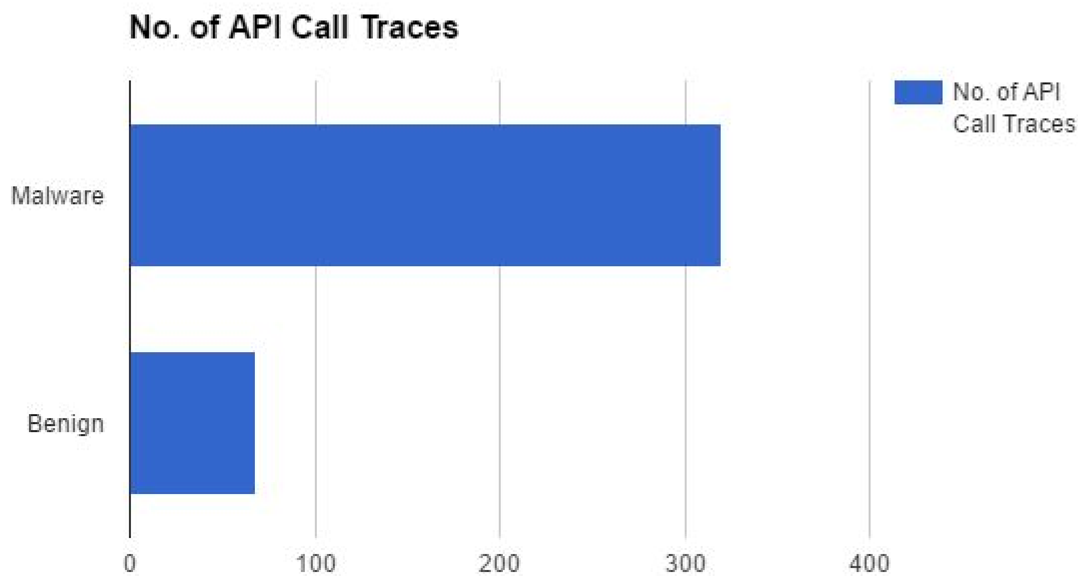
A partial list of API Calls from one malware run taken from one record in the CSMining dataset

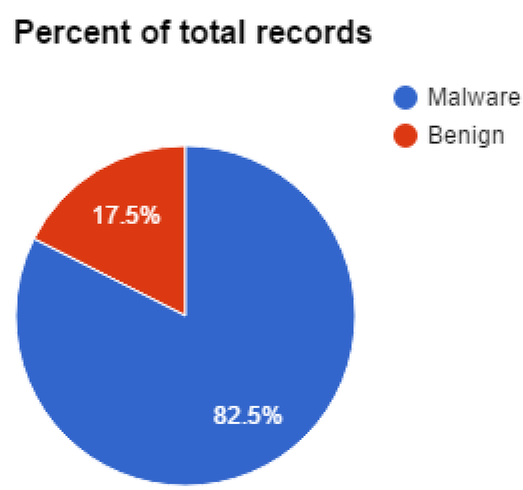
LoadLibraryW HeapAlloc HeapAlloc HeapFree HeapAlloc HeapFree HeapFree NtOpenKey LoadLibraryW GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetCurrentThreadId NtOpenKey NtQueryValueKey HeapAlloc HeapAlloc HeapFree HeapFree HeapAlloc HeapAlloc HeapFree HeapFree NtOpenKey GetProcessHeap HeapAlloc NtOpenKey HeapAlloc NtOpenKey NtQueryValueKey NtQueryValueKey HeapFree HeapAlloc HeapAlloc NtOpenKey NtQueryValueKey HeapAlloc HeapAlloc RegOpenKeyExW LocalAlloc HeapAlloc HeapAlloc NtOpenKey HeapAlloc GetStartupInfoA GetFileType GetFileType GetFileType GetEnvironmentStringsW HeapAlloc FreeEnvironmentStringsW GetCommandLineA GetCommandLineW GetCommandLineW HeapAlloc HeapFree HeapAlloc VirtualQuery GetModuleFileNameW VirtualQuery NtOpenKey HeapAlloc GetProcessHeap HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc GetProcessVersion RegOpenKeyExA RegQueryValueExA RegCloseKey RegOpenKeyExA RegQueryValueExA RegCloseKey GetProcessHeap HeapAlloc HeapAlloc HeapAlloc HeapAlloc GetProcessVersion HeapAlloc HeapAlloc HeapAlloc HeapAlloc GetProcessVersion HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc NtOpenKey NtQueryValueKey NtQueryValueKey NtQueryValueKey NtOpenKey NtQueryValueKey NtQueryValueKey GetModuleFileNameW NtOpenKey HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapFree HeapAlloc HeapAlloc HeapAlloc RegOpenKeyExA HeapAlloc HeapAlloc RegOpenKeyExA HeapAlloc HeapAlloc RegOpenKeyExA NtOpenKey RegOpenKeyExW RegNotifyChangeKeyValue HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc NtOpenKey HeapAlloc RegOpenKeyExA GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetProcAddress LocalAlloc LocalAlloc LocalAlloc LocalAlloc HeapAlloc HeapAlloc HeapAlloc LocalAlloc HeapFree HeapFree HeapFree LocalFree NtOpenKey RegOpenKeyExW

RegQueryValueExW RegCloseKey GetModuleFileNameW GetProcAddress HeapAlloc HeapAlloc HeapFree HeapFree HeapAlloc GetProcAddress HeapAlloc LoadLibraryW GetProcAddress HeapFree HeapAlloc LoadLibraryW GetProcAddress HeapFree HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc NtOpenKey HeapAlloc NtOpenKey GetProcessVersion GetProcessHeap HeapAlloc HeapAlloc HeapAlloc NtOpenKey GetCurrentProcessId GetCurrentThreadId GetProcAddress GetProcAddress VirtualAlloc VirtualAlloc VirtualFree VirtualAlloc VirtualFree VirtualAlloc VirtualFree VirtualAlloc VirtualFree VirtualFree GetProcAddress GetCommandLineA GetStartupInfoA GetCurrentThreadId GetModuleFileNameA GetModuleFileNameA RegOpenKeyExA RegOpenKeyExA RegOpenKeyExA LoadLibraryExA LoadLibraryExA HeapAlloc LocalAlloc VirtualAlloc LocalAlloc VirtualAlloc GetProcAddress HeapAlloc HeapAlloc HeapAlloc HeapAlloc HeapAlloc GetModuleFileNameA GetModuleFileNameA RegOpenKeyExA RegQueryValueExA RegQueryValueExA RegCloseKey RegOpenKeyExA RegQueryValueExA RegQueryValueExA RegCloseKey RegOpenKeyExA RegQueryValueExA RegQueryValueExA RegCloseKey RegOpenKeyExA RegQueryValueExA RegQueryValueExA RegCloseKey RegOpenKeyExA RegQueryValueExA RegQueryValueExA RegCloseKey CreateFileA GetCommandLineA GetCommandLineA GetCommandLineA GetCommandLineA GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess GetCurrentProcess

Exploratory Visualization

The API Call list for some records is exceptionally long. The number of API Calls for a single software run is approximately equal to 1800.

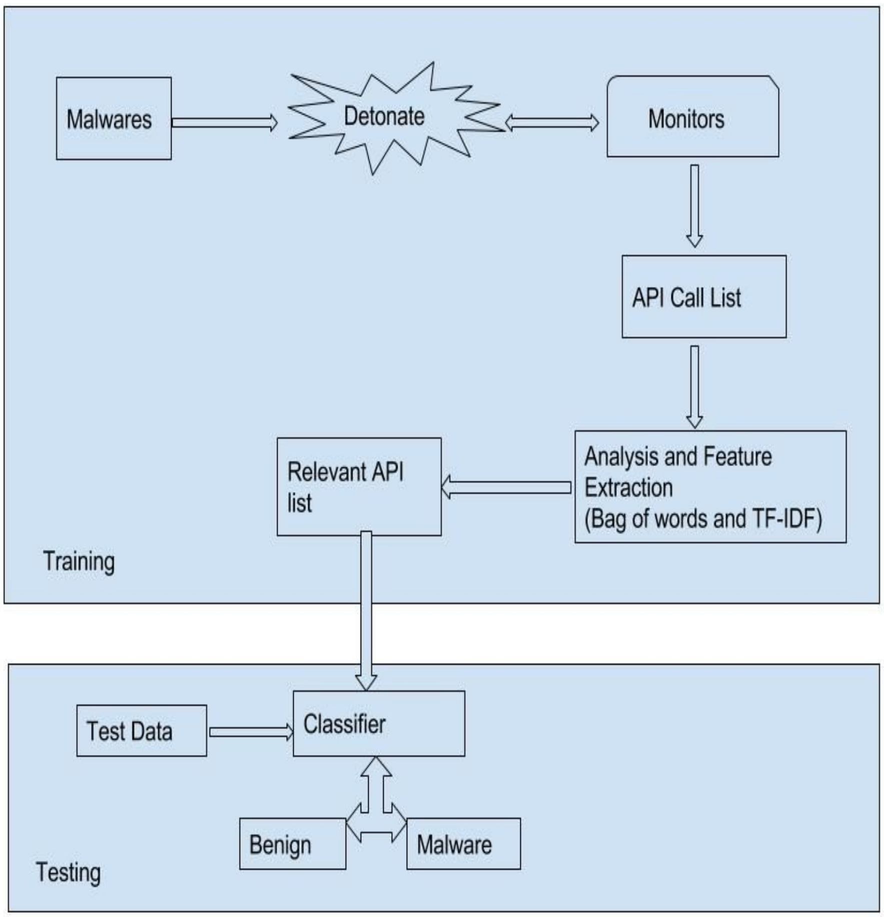

No. of API Call Traces
No. of API Call Traces
Malware
Benign
0
100
200
300
400


Percent of total records
Malware
Benign
17.5%
82.5%

Algorithms and Techniques

As seen in the figure 1 below, the API Call list had to be analyzed such that relevant features could be extracted. In order to extract the relevant features

Figure 1.



Tf-idf transformer was used to analyze the words for their importance. TF-IDF (term frequency - inverse document frequency) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus[1]. Stochastic Gradient Descent method was used to execute linear SVM. Numerous studies (Joachmis 1998, Dumais et al. 1998, Drucker et al. 1999) have shown the superiority of SVM over other machine learning methods for text categorization problems.

Benchmark

To create an initial benchmark for the classifier, I used MultiNomial Naive Bayes algorithm. Naive Bayes algorithm was able to achieve approximately 84% accuracy. My goal was to use an algorithm that will get a higher accuracy than Mutinomial Naive Bayes.

Methodology

Data Preprocessing

Since API Call list is textual data, the text analysis algorithms were used to analyze them. Following pre-processing steps were done before the data was sent through the classifier for the final classification.

1) Vectorizer

Scikit learn's feature extraction module offers a CountVectorizer function, which converts the text to numerical vectors. This function combines bag of words generation, tokenization, and filtering of stop words. The countvectorizer also uses sparse matrices to save space in the memory. The bag of words representation indicates that n-features are the distinct words in the corpus. In this case the bag of words is the list of distinct API Calls from all the records. CountVectorizer supports the count of individual API Calls. It produces a vocabulary list, where the index value of the API Call is linked to its frequency.

2) TF-IDF Transformer

Even though occurrence count of an API call would give the indication of type of software, some of the API Calls appear multiple times in a single software execution. To get a better sense of the API Calls occurring during the execution of the software, term frequency is used to get the ratio of the count of a specific API Call and the total number of API Calls from the run. Since some of the API Calls are seen both in malware and benign software runs, inverse document frequency is used to lower the importance of API calls seen in many records. The TF-IDF transformer transforms the data by calculating term frequency and inverse document frequency.

Implementation

I evaluated several approaches before finalizing on the usage of countvectorizer and tfidf transformer for feature extraction. Before finding these built in functions in scikit learn, I was considering writing the algorithms from scratch. Writing the algorithms from scratch would have taken a very long time.

For the classifier, I considered using different sizes of n-grams for the textual analysis of API Calls, but the size of ngram = 1 provided the best results. This finding was in line with research done in this field by other people like Veeramani R and Nitin Rai[1] .

I chose to use SGDClassifier because it is highly efficient and easy to use. Support Vector Machines are a good fit for textual analysis problems because the data generated is sparse and most text classification problems are linearly separable[2].

Refinement

The model can be further refined to identify various classes of malware instead of categorizing as just malware. Different classes of malware are viruses, trojans, rootkits, sypware, trojan horse, bot etc. Besides the Windows API Call analysis, the model can include the communication behavior analysis of the software considering that the purpose of many malware types is to send precious information of an entity to a command and control center. Neural network can be used to perform the analysis of the malware behavior.

1

---

[1]International Journal of Scientific & Engineering Research Volume 3, Issue 3, March -2012 1
ISSN 2229-5518

2 Text Categorization with Support Vector Machines: Learning with Many Relevant Features Thorsten Joachims, https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf

Results

Model Evaluation

tp: true positive fp: false positive	fn: false negative tn: true negative	**Predicted Class**	
		Malware	Benign
	Malware	tp	fn
Actual Class	Benign	fp	tn

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

Accuracy with Mutinomial Naive Bayes = 84.6%
Accuracy with SVM classifier = 89.7%

SVM classifier was able to predict with better accuracy than the Multinomial Naive Bayes due to the nature of this problem.

Metrics from SGD Classifier

	precision	recall	f1-score	support
Benign	0.83	0.42	0.56	12
Malware	0.90	0.98	0.94	66
avg / total	0.89	0.90	0.88	78