CS181 Spring 2016 Practical 2: Classifying Malicious Software — Team EXT3

Robert J. Johnson — Dinesh Malav — Matthew McKenna

March 5, 2016

Abstract

Identifying and eliminating malicious software (malware) is a modern computing task of considerable importance. Properly classifying these programs in a computationally efficient manner will lead to massive savings in time and money. Our study showed how various machine learning classifiers performed at this task when presented with malicious XML executables. A random forest classifier was shown to be the best model tested, with a categorization accuracy of .81211 on the test data.

1 Technical Approach

The training data set for our examination consisted of 3086 XML executable files. Of these, roughly half contained malware. The XML files on first glance appear quite daunting, with large hexadecimal strings and semi-random calls to various processes intermingled throughout the scripts. Another foundational problem with the data analysis is the nature of the differences between the malware classes. The differences between certain classes are subtle, adding another layer of complexity to the classification process.

After initial testing of various generative and probabilistic classifiers, it was obvious that proper feature selection would be the primary manner in which we could improve the model. From the XML files, we attempted to extract certain calls in the executable that could be used by malicious software to attack a computer system. These calls were as follows:

'sleep', 'dump_line', 'open_key', 'query_value', 'load_dll', 'remove_directory', 'create_directory', 'get_computer_name', 'open_url', 'process', 'vm_protect', 'vm_allocate', 'create_mutex'

Intuitively, these were seen as calls that could manipulate a computer and this were able to be used as accurate classifiers for identifying malware. Utilization of dynamic-link libraries (DLLs) were also seen as a potentially valuable indicator in our classification, and accordingly we added features pertaining to DLL

usage. Towards the end of our analysis, we added in code to parse and count the total number of XML tags that occurred in each executable. We attempted to use a wide range of classification paradigms in our analysis. The following techniques were used at various points in our study:

Random Forest Classifier, Quadratic Discriminant Analysis, AdaBoost Classifier, Gaussian Naive Bayes, Decision Tree Classifier, Support Vector Machine, K-Nearest Neighbor Classifier

2 Results

Ultimately, our random forest classifier was the most accurate. Once we realized

Method	Accuracy
Random Forest Classifier	0.890524379025
Quadratic Discriminant Analysis	0.699172033119
AdaBoost Classifier	0.634774609016
Gaussian Naive Bayes	0.671573137075
Decision Tree Classifier	0.866605335787
Support Vector Machine	0.625574977001
K-Nearest Neighbor Classifier	0.824287028519

Random Forest Classification was the most accurate, we attempted to tweak the parameters to improve the accuracy of the model.

Parameters	Accuracy
n_estimators=100, max_features='None'	0.891444342226
n_estimators=50, max_features='log2'	0.894204231831
n_estimators=100, max_features='log2'	0.897884084637
n_estimators=125, max_features='log2'	0.896964121435
n_estimators=115, max_features='log2'	0.896044158234

3 Discussion

Another approach that we were unable to draft due to time constraints was one that analyzed the number of distinct n-grams in the executable files.

All code for this project can be found at:

$$https://github.com/HarvardCS181Practical2016$$

.