Data Mining Final Project Report - Spring 2015

Dargahwala, Huzefa
hydargah@indiana.edu

Gaikwad, Aniket
anikgaik@indiana.edu

Shelke, Cyril
cshelke@indiana.edu

5 May 2015

All the work herein is solely ours.

1 Introduction

Roar Zoo Corporation, a multibillion dollar zoo management company with more than 50 zoos all over the United States, is being inundated with malware attacks on its proprietary online portal used by its employees to manage the day to day monitoring activities of all the zoos in real time. This report is an exposition of this problem as presented to us, VagaBond Analytics, and the approaches we took to ameliorate it and potential deployment ready solutions for tackling the problem in real time.

Following this section, Section 2 presents an executive summary for the board of directors of the company. For readers who favor details of the project, the subsequent sections will prove illuminating to them. Section 3 delineates the problem in detail, followed by Section 4 describes our company, the team and its future. Section 5 gives an in depth analysis of the data acquisition and preprocessing step. We discuss approaches taken for Feature extraction and Feature Selection in Section 6 & Section 7 respectively. Section 8 presents models trained and the results obtained for each. Since every project has its merits and short comings, we conclude the report by mentioning ours in Section 9. . Section 10 describes the future plans of Vagabond Analytics with Roar Corporation.

2 Executive Summary

Vagabond Analytics is a software solutions company focusing on performing Data Analytics and is still in its start-up phase. Vagabond has limited resources but is blessed with an industrious staff and highly talented technical workforce. Every employee, right from the CEO to the techincal engineers to the friendly receptionist wants Vagabond Analytics to make their mark in the software solutions industry. Vagabond treats each project as an opportunity to establish their expertise and further improve its reputation. As of today, Vagabond has 5 patents to its name for coming up with innovative ways of analyzing the data and has a very satisfactory client base. Rumour has it that a couple of software bigwigs are planning a surprise swoop to buy Vagabond and have a substantial say in its share. The website of Roar Zoo Corporation has recently become the target of malware attacks. The Zoo has more than 1000 people working for it across its 20 branches all over the US. The zoo makes good profit and is will pay handsomely to any software solutions firm which can fix the software security problem. Initially, Crappy Antivirus Solutions was hired to do the fix, but Crappy did not live up to its promise and got fired and since Vagabond currently has the reputation of becoming the next big Data Analytics Company, they are immediately called onboard to find the fix and carry on from where Crappy left off. Vagabond approached the problem in a different way and treated this problem as a Data Analytics problem rather than just as a software security problem. Vagabond ran the data that it got from Roar Zoo through some complex classifiers like Random Forest and Gradient boosting to arrive at some concrete inferences. Now, given any file by Roar Zoo to check for its malicious nature, Vagabond can immediately tell if it was safe or not and if not, what family of malware does it belong to and the security threat that it poses. For Roar Zoo, the solution that Vagabond gave them was exactly what they were hoping for in their critical time of need. Even though Vagabond set back Roar Zoo Corporation

a large sum of money, Roar was more than pleased to oblige. The Return on Investment was high enough that Roar signed a contract with Vagabond Analytics to build their solution into an online solution so that such threats can be mitigated in realtime.

3 The Problem

Being such a huge company, Roar requires someway of managing its sites and resources efficiently. For this they use a online proprietary tool to manage visitors, employees, animals, etc. To have access to this tool, all the computers and mobile devices are required to have an active internet connection. The zoo industry mogul has more than enough resources to have a high speed internet connection at all locations for all devices.

Having computing devices which are online around-the-clock has its own perils. They attract attacks via malware which prove to be detrimental to the organization. The malicious programs can create back doors in the devices, install key loggers, steal data, etc.

3.1 Ad hoc Solutions

One solution is to only allow the access to the tool's website and all block other traffic. This will implicitly prevent malware attacks by only allowing access to the companies web services. A step taken in this direction, will have an adverse effect on the popularity of the zoos as visitors and employees alike would demand a free and open connection to the internet. Hence this solution is not feasible.

Roar Zoo Corp. contracted Crappy Antivirus Solutions, to design a customized antivirus for the company's in-house tool for computing devices. After 2 years of work, Crappy Antivirus Solutions concluded that all the malware that was targeted at the companies network of devices fell into one of these 9 categories.

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator ACY
9. Gatak

However, they could not stop future attacks of these types of malwares because each time the creator of the malware changes some parts of the code that changes the entire signature of the program. Hence they can not successfully identify the malware in real time. Crappy Solutions was fired because they provided no useful insights and was asked to hand over all the labeled data they had collected over their tenure at Roar Zoo Crop.

3.2 A Different Perspective

One of the employees, Hasan Kurban, from the business analytics team at Roar, suggested tackling the problem of malware classification as one of data science. After a rigorous market research of the good data science solution providers, the CEO contracted Vagabond Analytics to assist Roar with the problem at hand.

4 Vagabond Analytics

Vagabond Analytics, a Data Science startup in Bloomington IN, specializes at analyzing large amounts of data and provide knowledgeful insights to their customers. The team comprises of 3 agile minds with a dominant bend for analytics. Aniket, Cyril and Huzefa are fresh graduates from Indiana University, Bloomington who share a passion for data science. They were in a Data Mining class together and were inspired by the Professor to pursue Data Science as a career.

4.1 The Team

Aniket Gaikwad specializes in Reinforcement Learning and its applications to analysis of time series data. He believes that the markov property used in reinforcement learning also holds true for time-series data and hence it is possible for us to look at the two in the same way.

Cyril Shelke is a "gifted" Java programmer with the ability to code out complex mathematical equations in a heartbeat. His coding skills amalgamated with his mathematical ability make him a valuable asset to Vegabond.

Huzefa comes from an Applied Machine Learning background with a toolkit of multiple languages and platforms under his belt. His interest in marketing and networking with people provides a launching pad for the companies products and tools.

4.2 The Future

Vegabond Anayltics had humble beginnings and is still in its infancy stage. With many of the analytics tools and customized analytics solutions being big hits in the industry, the company is now deliberating on expansion and acquiring more employees.

The 5 year goal for the company is to obtain more traction and momentum by building more successful analytics products. The details of the future products can not be shared on grounds of Intellectual Property.

5 Data, Data, Data

As a team doing data science, one universal truth that has to be accepted is that data mining is all about data. More data is good and quality data is better. With the problem at hand, we had access to [Exact data size] of uncompressed data. Effectively managing so much data is a challenge in itself as this much data never fits in memory at once.

5.1 Data Acquisition

The first task in any analytics problem is to acquire the data and do some preliminary analytics to get a general idea of the volume and variety of data one is working with. This allows the team to brainstorm the possible approaches that can be taken to tackle the problem.

Crappy Antivirus Solutions had turned in all the data they had collected before they were relieved from their contract with Roar. The data comprised of decompiled assembly programs(asm) and byte files of those programs which were identified as malware. Each program was identified with one of the 9 malware types. This mapping of the filename and its malware type was consolidated in one csv file.

The total number of files captured and labeled by Crappy were (count test and train data). Individual files were small enough to fit in main memory of commodity hardware workstations but all of the files put together were difficult to put on a single commodity hard drive.

Hasan of the BA team at Roar, with his knowledge of Data Mining, correctly concluded that for any company that will be hired to analyze the data will require the data in 2 parts. He split the data into a training set and a test set of almost equal parts. He made the malware types of the train set available and withheld the labels of the test data. These will be used to test the correctness of the model created by the data scientists. To reduce the size of all the data, all byte and asm files were compressed using the 7z format. This consolidated data described above was given to Vegabond upon consummation of their contract.

Being Indiana University graduate students, the Vegabond team had access to the 46th fastest supercomputer in the world, when it was first launched, the Big Red II[cite]. The team was quick to unzip and upload all the train and test data on the shared scratch space. This would allow faster access to the data than on a local machine.

The protocol of operations was to write programs on the local machine and run it on a sample of 10 files. If the output of the program was favorable, the program would then be submitted as a job to the cluster to be run on the entire dataset. This approach has its pro's and con's.

The major disadvantage was the overhead of keeping the files on the server and the local machine in synchronization. Another, major drawback was the delay in obtaining the results from the supercomputer. Since our program was run as a job on the computer and the supercomputer did not always have all the required resources for the job available, it took a stochastic amount of time before job commencement. The advantage was that the local machine could be used to continue with the research.

5.2 Summary of Data

Scripts were written to perform rudimentary analysis of the data. Below are the some facts about the data:

- Total Data Size: 391 Gigabytes
- Size of Train Set: 193 Gigabytes
- Size of Test Set: 198 Gigabytes
- Number of files in Train Set: 21736
- Number of .asm files in Train Set: 10868
- Number of .byte files in Train Set: 10868
- Number of files in Test Set: 21746
- Number of .asm files in Test Set: 10873
- Number of .byte files in Test Set: 10873

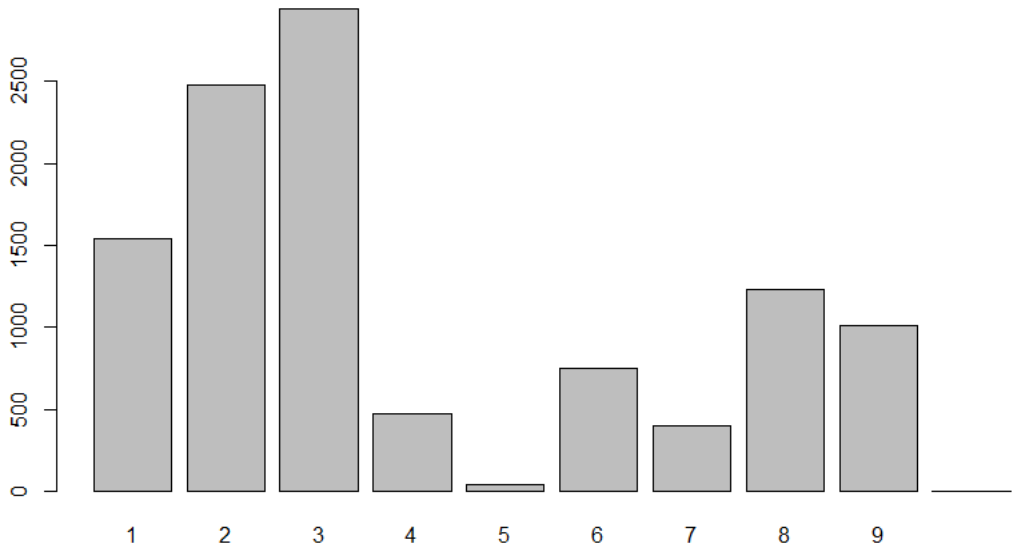

Figure 1: Proportion of class label representative files

The number of training files per class label have been visualized above. The graph suggests that not all malware occur in equal proportion. This may create a class imbalance problem and the model we learn may try to overfit the classes which occur significantly less times than others. Randomly selected byte files and asm files were analyzed by the team to assess the how the files can be interpreted and understood by humans.

The .asm files contain code from the 80386 disassembler, which is common knowledge, that does not correspond to the actual code written by the ill-willed programmer. Each line begins with a memory address where the line of code resides in memory followed by one instruction from the instruction set of the 80386 processor, the memory location or register to perform the operation at and the required operands. The memory address is preceded by directives. These signify the role that line plays in the entire program. There are 6 types of directives that we encounter[1]:

- .data : Tells the assembler that this line contains data
- .text: Line with this directive contains opcodes
- .rdata: Signifies read only data
- .bss: These are uninitialized data variables
- .idata: The OS reads these directives to make a list of all DLL's needed by the program
- .edata: These are the DLL functions that the program can provide to the OS

The asm files also contains a lot of '??' which can be the disassembler's output for an uninterpretable piece of code. This, we presume, is by design when the malicious programmers wrote the code for this program. In files we inspected, we also found a large number of "CC".

The byte files contain bytes encoded in hexadecimal format. Each of these also begin with a memory address. The rule of thumb in assembly programming is that programs are written in human interpretable asm files and "interpreted" into byte files for consumption by the machine. Usually, there is a direct mapping

from the asm files opcodes to the byte files opcode. In this case however, where we are dealing with malicious programs, the creators of the programs obfuscate the code in such a way that the mapping no longer holds and the byte files seem unrelated to the asm file. Also, this does not allow the disassembler to generate the exact program that was converted into the byte files.

Upon trying to encode the asm file of a malicious program viz. "CuB6pZlWVqT9GK3QDHdm" into byte code using an off-the shelf assembler, the output did not look similar to the bytes file of the program. This further bolstered our assumption that the source code has been obfuscated before converting it into the byte file. Hence, the asm provided is not a 100% trustworthy.

6 Feature Extraction

Features were extracted using both asm and byte files with whatever domain knowledge we could gather from various security experts. As we explained in the previous section, the asm and byte files are not similar and there is almost very little direct correlation between the 2 types of files. As the data contains both the asm and bytes files, we can use the "best-of-both-worlds" approach to obtain a model that uses both types of files by combining the prediction of 2 types of classifiers, Ones trained on the asm file features and the ones trained on the byte files.

6.1 Features of the .bytes files

Due to the unintelligible format of the .bytes file, a basic approach is to do a counting of all possible byte values and learn a classifier based on the counts. We performed the byte counts of each possible value for each file to obtain a feature vector of length 258 features, 00 to FF , which also included ?? and CC. The feature vectors were a result of byte counts on each byte file in the trainSet. A RandomForest classifier was used to do the training and we obtained the result [insert result here]

Another approach for extracting features from the bytes files was to use n-grams where n-grams$where, n \in \{1, 2, 3, \}$. n grams looks at the bytes files, as a window of n bytes in sequence. For example, the byte sequence "00 52 54 AB" has the unigrams as 00, 52, 54, AB and bigrams as 00_52, 52_54, 54_AB and so on. Feature vectors are constructed by taking a [0] array of size of all possible n-grams array and then changing the value to 1 if the corresponding n-gram is present in the all possible n-grams array.

Using unigrams is not much different than using the counts we used as out first approach. Using bigrams approach is useful as its feature vector has more variety, this is a good thing as well as bad. We hope to get better features by using bigrams but with this increase in number features we face the curse of dimensionality and the risk of overfitting. This approach is space intensive, as we obtain close to 66 thousand bigram features which causes the final feature vector file to be close to 1.8 GB.

With more than enough memory to get the entire feature vector file in memory and leveraging multi-threading, finding correlation between every 2 columns of the data was computationally intractable with the task timing out on several occasions. We then moved our focus on the features from the asm files.

6.2 Features of the .asm files

We first performed a literature survey to understand the syntax of the Assembly language programming, & how the data flows in the program. For this we went through the Assembly guide for x86 Intel processor[2].By understanding the instruction set, opcodes and system calls that are mentioned in the Assembly guide , we decided to look for the features based on three major categories i.e. size of each segment, instructions used in the .text segment of the code and system level calls used in the code. So for .asm file parsing we looked for instruction opcode in .text segment and system calls in .Text and other segments.

Also, to these features we added some superficial features like size of .text, .data, .rdata,. idata segments. After completion of execution using these parameter for the feature we get around 35K features. Now the challenge we faced during feature extraction was the same as the previous sub section ie: the size of the data. As data size was large, we decided to use parallel programming approach by using python multiprocessing library. Also, we used IU's Big Red II, supercomputer to increase the processing capacity and managed to obtain the desired features for asm files.

7 Feature Selection

The feature space of 35K was big enough to handle for any classifier. Also, upon some random sampling we found that many feature vectors were having 0 values i.e. these features do not appear in the any of the file, but were considered because of entry in the Assembly Instruction guide or appear very small number of times. So, we filtered these features with criteria of removal of features if 99% of the entry in feature are 0. With this criteria, we proposed the idea that if a feature only appears in the approximately 100 files, then it is may be due to some redundant code or non-useful feature considering it as an outlier. This approach decreased the feature space from 35K to 460 features. On these 460 features, we used Recursive Feature Extraction (RFE) to get the 50 most useful features.

RFE: Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then, features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached[**?**]. Once we got 50 features, we extracted the give data into R and started our initial analysis. By using packages in R, we removed the features which were atleast 75% correlated . Once correlated features were removed and after trying some models , we found that there were some features that had a large impact on the accuracy. So we decided to remove the features having approximately 0 variance, which further shrunk the feature space. We also tried some models with features obtained after applying the PCA algorithm on the original feature space of size 35K but the results were not good enough to be reported.

8 Model Selection and Result Analysis

With features extracted from .bytes and .asm files, we decided to use classifiers like RandomForest, Gradient boosting, Naive Bayes, & Logistic Regression. We initially tried training one model per classifier and tested its results. Upon examining the results we realized that RandomForest and Gradient boosting seems to be working good with such large feature space. So we decided to go ahead with RandomForest and Gradient boosting. Initially, we ran both RandomForest with 100 trees and Gradient boosting on full feature space (about 65K for features extracted from .bytes files and 35K for features extracted from .asm files). We got a good results for both classifiers. For features extracted from .bytes file(referred to as 'byte features' henceforth in the report), Random forest with 100 trees we obtained a log-loss of 0.588, while when executed with features extracted from .asm files (referred as 'asm features' hence forth in the report) we get very high log-loss of 5.59.

When tested with gradient boosting for byte features it gives log loss of 3.77 and for asm features log loss of 4.84. After initial naive model testing we performed the feature selection on the asm feature as explained int the above section. On applying RandomForest with 80 trees we got the log-loss as 2.2381, when modeled with 150 trees we got the log loss as 2.1771, when modeled with 250 trees we obtained the log-loss as 2.0200 and with 300 tree we obtained 2.018.

There is a improvement in our results but the improvement is not significant enough. The possible reason could be that the features we selected for the asm feature set might be faulty. As we do not have enough domain expertise in assembly language programming we considered all the instruction sets and system calls to be of equal importance. To overcome the hurdle of domain expertise, we opted to use RFE as feature selector for us, to give us possibly good results, but as per out results there no significant improvement. For asm features, we tried applying PCA with 30 principle components which retained the 99% of the variance in the data. However, the results do not show any notable improvement. So, we finally affirm that, for asm feature we have to some more feature engineering which might require some expert advice to decide of features we need to get rid of to get better results. Unfortunately, due to time constraint and monetary constraints we did not proceed with asm features but focus of the byte features which was giving comparatively better results.

After an approach with 100 tree Random tree with byte feature which gives us some good accuracy we proceed further with feature engineering as we went with asm feature files. We tried reducing the feature space by removing features having more than 90% of entries as zeros, but that did not reduce feature space

by big margin (just reduced it by 1K features), so this indicated that byte features are dense vectors. We then concurred to apply PCA, and with principal components set to 30, we obtained a result 0.8439 which was surprisingly lower than the previous approaches. After analyzing the principal components, we realized that, the variance retained by the principal components was just 50%. Also, the first principal components was retaining just 46% of the variance in the data and only 4% variance was retained by rest of the 29 principal components. When we tried Random forest with 300 trees we got a log-loss of 0.5234 with byte features. Random forest is ensemble classifier which will generate probabilities based on bagged decision trees. So, we believe that increasing number of trees in RandomForest might help in reducing log-loss due to the bagged trees introducing more bias in decision making.

9 Conclusion

We undertook several novel approaches at feature extraction and feature selection like using RFE and PCA to reduce the number of features and data transformation repectively. The approach of using n-grams on byte files was inspired from the NLP domain and applied to assembly code and proved even more useful than the features from the asm files.

There were several plans in the works to improve accuracy but we could not obtain results in time to present here. They have been mentioned in the Section 10. The project was a unique data mining problem with many twists like class imbalance and unclean data. We learnt a lot about working with large datasets and parallel programming.

10 Future Work

Firstly, we can exploit the the graph representation of the system calls in the asm files. So to compare two malware which belong to same category, it will be more convenient to use structural model. We can useensemble of different classifiers. In other words, we can used output or prediction of one class as reference input along with the class labels to next classifier, which help the next classifier to learn more information.This approach is kind of similar to unsupervised learning. Also, trying out average of 10 or more classifier output could be a one of the approach worth trying, as some classifiers have high bias while some have high variance. So, average or weighted voting could be one of the approaches to try.

The trained model can be configured to classify files arriving on the company's network in real time. This will totally eliminate the need for regular batch antivirus scans. The system can also be modified to learn to predict new types of malware by observing behaviors of files on the system.

Vagabond Analytics has signed a contract with Roar Zoo Corp to achieve the above goal with no time constraint and steady quaterly funding from Roar.

References

[1] University of Maryland. Data and text segment, http://www.cs.umd.edu/class/sum2003/cmsc311/notes/mips/dataseg.ht

[2] University of Virginia Computer Science. Guide to x86 assembly, http://www.cs.virginia.edu/ evans/cs216/guides/x86.html.