**CSE 2431**

Spring 2020

Prof.Yinqian Zhang

Project #1 – Malware Detection

Quinton Hiler

Andrew Cantor

Date Submitted: 29 March 2020 **Task1: Simulated Malware Detector**

**Input:**

Our simulated malware detector uses Python to take in the data provided by the Australian Defense Force Academy. This data is a set of system calls that are determined to either be a malware or a benign program. For the malware system calls, there are 6 different types of attack: Hydra-FTP, Hydra-SSH, Adduser, Java-Meterpreter, Meterpreter and Webshell. For each of these attacks, there are 10 folders, with each about 7 files, with each file containing the series of calls that correspond to attack. On the other side, the benign systems calls are contained in two seperate folders: “Training\_Data\_Master” containing 883 call files and “Validation\_Data\_Master” with 4373 call files. This python script brought in the data as a string containing the calls and formatted them into an array of the most common sequences of calls. In this project these sequences were formed by using a sliding window of size 3 (Ex. f = array, sequence 1 = [f[0], f[1], f[2]], sequence 2 = [f[1],f[2],f[3]] , ect.).

This produced a set of call sequences of size three and these sequences were counted until each file had an array of system call sequences paired with each sequence’s frequency within that file. After parsing the data into the frequencies, the top m% of all frequencies was found. This was to reduce the amount of frequencies to make the final data manageable given the large amount of data we were provided. Even taking only the top 30% of data provided a frequency set of about 12 thousand. To finalize the data, the top m% frequencies were used to create an array containing either all the training or the validation data. This data was formated with each row being a call file and with each column corresponding to one of the top m% sequences and each value being the frequency of said sequence in each file. The last column contained a ‘1’ if the file was malware or a ‘0’ if it was benign. These arrays were then printed to a .csv so that the machine learning code did not have to wait for the arrays to be formed everytime it was run.

**SVM machine:**

For detecting the malware, the team used a Support Vector Machine (SVM). This is a style of machine learning that is great for binary classification problems. The training data is taken and vectoried and then by training the machine with this set of data, a border between the two sets is established through multiple dimensions. By checking the border between the two sets allows the machine to predict the outcome of new inputs. Our group is new to machine learning and after some investigation into the different algorithms, we found that SVM would be both simple and effective.

The libraries and resources used in Python are Panda, Pickle, and the SVC class for the SVM machine learning algorithm. The frequency data was processed into an .csv file table in an n x m data matrix. The files processed in the previous python script were used to implement the machine for training or validation, labeled ‘training’ and ‘validation’ respectfully. Rows represented the n-samples while columnes consisted of m-1 number of attributed features, the m-th column being the label for whether the sample was malware or benign software. An SVM model was then created in python to process the training and validation data, the label being numerically binary with 1 for malware and 0 for benign. The SVM model automatically calculates and uses linear regression to organize the data accordingly once it’s been trained. After training, the algorithm can attempt to predict if a set of software system call frequencies from the validation set are malicious or benign.

Each data set was processed two ways into the SVM algorithm; with a top m = 10% and a top m = 30%. Out of the 4680 data, 307 were malware and the rest was benign software. When these were processed in the SVM, they produced a confusion matrix and a classification report. The confusion matrix is the 2 by 2 matrix in the top left of the figure and it shows the positive/negative rates of the machine. The top left matrix is the number of true positives, top right is the number of false positives while the bottom 2 are the false negative and true negative respectfully. There were 3719 true positives, and 654 false positives for the m = 10%, as shown in the top left of Figure 1. With a top m% of 10%, the precision was slightly higher than a top m% of 30%, 31% and 29% precision respectively. About 14% of the decisions made by the algorithm were false positives for both classification reports. Although the precision was low on both experiments, the recall rate and weighted average was high. This is probably due to the high amount of true negatives. Using weighted data and adding a limit curve to the algorithm could solve this issue, but SVC does not support this functionality. Between the two sets of data, the difference between the two machines is negligible. Below are the appropriate classification reports.

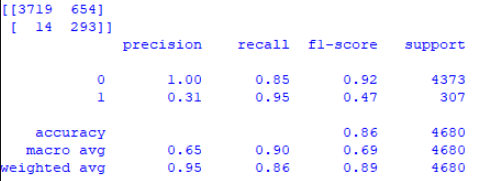
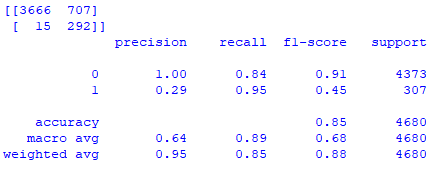


Figure 1: Classification Report m = 10%

Figure 2 : Classification Report m = 30%

The model was then saved into a .sav file for later implementation and malware detection. Pickle was used to save the SVM model for Task 2, when we used real-world malware to trace system calls and process the data into the Python code and SVM algorithm.

**Task 2 : Trace System Calls for Testing**

**System Call Tracer:**

Task 2 required that we track system calls in C of potential malware by creating a tracer. Then the data would be processed and would deliver a 1 or 0 if the process was malware or benign software respectively. Instead we opted to create a simple tracer in C that would take the processes in the same directory as the tracer and get the system calls. The system calls for each process in the directory would be recorded in a text file and then processed into our SVM algorithm using another python code to process the new data. Once the data was in the proper format for the SVM algorithm, the algorithm would predict whether the software was malware or benign.

The tracer was designed by forking a child process, then tracing the child as it executed system calls. The child would continue to process until it signalled the parent process that it had finished, prompting it to break the child process loop and end the tracer. As each system call is made, the child would write the calls into a file for later data processing. Although the tracer does not follow the conventional format, it still gets the system calls of the potential malware within the directory and organizes it into a file for later use in our Python code.

**Conclusion:**

After about a month of research and development of this malware detector, we’ve developed a competent algorithm that functions in Python. Although the precision is low with an average of 30%, the overall ratio from the total samples to false positives also proved to be low, around 14%. As mentioned before, this is due to SVM not weighing data properly, resulting in a moderate amount of true negatives. The tracer used to extract system calls works within proper implementation. The data used can be processed and put into the trained algorithm to predict whether the software is malware or benign. We did not have enough time to create or set up our own malware to extract system calls and test. The group learned much about machine learning, system call tracing, and processing system calls into data features. New skills in Python and C were developed with the various libraries used in this project. This project may be revisited in the future to finalize and polish results, the algorithm, and code.