Malware Detection using API Call Graphs

Sneha Mandan - 201401422
Aalisha Dalal - 201401433

BTech Mini Project (PC403)
Project Guide: Prof. Anish Mathuria

4th Dec, 2017

Introduction

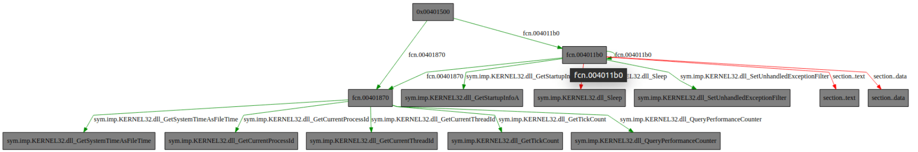
- Malware - a malicious software designed to infiltrate a computer system without the owner's knowledge or consent. e.g. Virus, worm, Trojan Horse, etc.
- Major approaches for malware classification:
  1. Signature based approach
  2. Behavioural based approach

Problem Statement

The project involves implementing a tool which will efficiently detect zero-day malware programs, based on graph theoretic properties of API call graphs and machine learning model.


sym.imp.WS2_32.dll_WSAStartup
sym.imp.KERNEL32.dll_InitializeCriticalSection
sym.imp.KERNEL32.dll_CreateThread
sym.imp.KERNEL32.dll_GetProcessHeap
sym.imp.KERNEL32.dll_HeapAlloc
sym.imp.KERNEL32.dll_Sleep
sym.imp.KERNEL32.dll_InitializeCriticalSection
sym.imp.KERNEL32.dll_CreateThread
sym.imp.KERNEL32.dll_GetProcessHeap
sym.imp.KERNEL32.dll_HeapAlloc
sym.imp.KERNEL32.dll_Sleep
0x10007deb
sym.imp.ADVAPI32.dll_InitiateSystemShutdownExW
sym.imp.USER32.dll_ExitWindowsEx
str.NtRaiseHardError
sym.imp.KERNEL32.dll_GetProcAddress
sym.imp.KERNEL32.dll_GetVersionExW
sym.imp.KERNEL32.dll_HeapFree
sym.imp.KERNEL32.dll_SetLastError
sym.imp.USER32.dll_ExitWindowsEx
str.NtRaiseHardError
sym.imp.KERNEL32.dll_GetProcAddress
sym.imp.KERNEL32.dll_GetVersionExW
sym.imp.KERNEL32.dll_HeapFree
sym.imp.KERNEL32.dll_SetLastError
sym.imp.KERNEL32.dll_GetProcAddress
sym.imp.KERNEL32.dll_GetVersionExW
sym.imp.KERNEL32.dll_HeapFree
sym.imp.KERNEL32.dll_SetLastError
sym.imp.ADVAPI32.dll_SetThreadToken
sym.imp.ADVAPI32.dll_SetThreadToken
sym.imp.KERNEL32.dll_CloseHandle
sym.imp.KERNEL32.dll_GetProcAddress
sym.imp.KERNEL32.dll_GetVersionExW
sym.imp.KERNEL32.dll_HeapFree
sym.imp.KERNEL32.dll_SetLastError
sym.imp.ADVAPI32.dll_SetThreadToken
sym.imp.KERNEL32.dll_CloseHandle

API Call Graph - Benign Program



Architecture of the Malware Detection tool

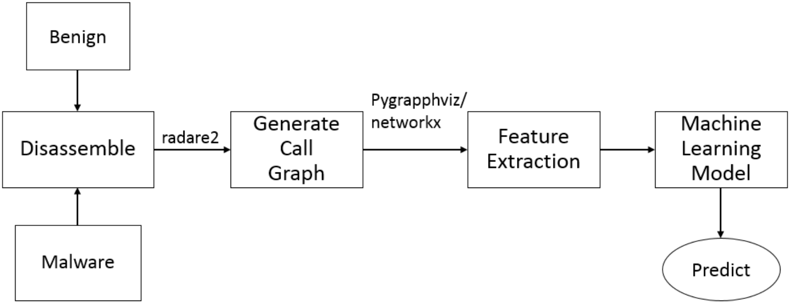


Figure: Flow chart of the Procedure

Step 1: Generating Call Graph

- In this work, we are primarily focusing on the windows executable files.
- The function call graphs can be generated from the program executables using softwares that allow reverse engineering. The tool used in this project is an open-source tool, *radare2*.
- After disassembling the entire binary file and running a statistical analysis in radare2, the generated call graphs (based on sequences of call and jump instructions) were stored in .dot file format.
- The graphs can be visualized using any graph visualization tool such as python packages "pygraphviz" and "networkx")
- The graph generated will be directed graph. The nodes represent the functions and the system calls in the program. And, the edges represents caller-callee relationship.

Dataset

- Our dataset comprises of Windows executables files.This tool can be extended to other executables files.

- Total Dataset - 90 benign files, 32 malware files
- Training dataset (70%) - 63 benign samples, 23 malware samples
- Testing dataset (30%) - 27 benign samples, 10 malware samples
- The proportion of benign and malware samples in training/test data was taken similar to that of original dataset using stratified split. To take care of the varying number of samples of each class, weight-age of each sample was taken as inversely proportional to the total number of samples of that class.

Step 2a: Feature Extraction

- Form the .dot files generated in the previous step, graph theoretic properties are determined which will be used as features to train the learning algorithm.
- *pygraphviz* and *networkx* packages of Python are used to obtain the features.

Features:

Features Based on Past Work

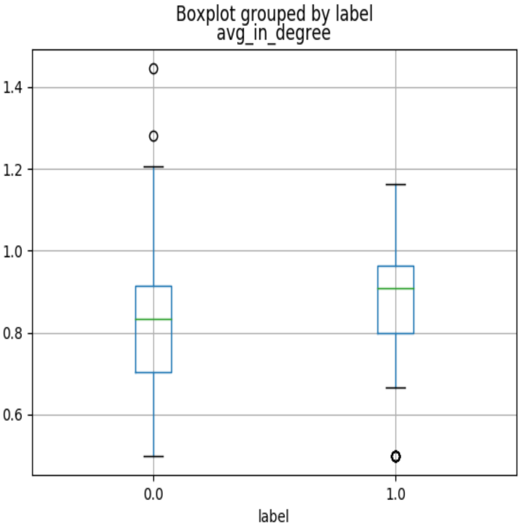
1. Average InDegree
2. Average OutDegree
3. Average Shortest Path length
4. Eigen Vector Centrality

Additional Features

1. InDegree (Maximum, Variance)
2. OutDegree (Maximum, Variance)
3. Graph Entropy
4. Density
5. Diameter

Step 2b: Feature Selection

Correlation of above features with respect to the output label was observed and only the significant features were considered for training the model.



Step 3: Using ML models

Logistic Regression Model

The model maps the input to the output by computing the optimal values of parameter θ in the below equation. Here, θ vector represent the weights assigned to each input feature.

$$Y_{pred} = g(\theta' * X + b) = h_\theta(X) \tag{1}$$

Here, g is sigmoidal function $g(x) = \frac{1}{1+exp(-x)}$

The cost of the model J(θ) for all samples can be given by,

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{N}(y_i * log(h_\theta(x_i)) + (1 - y_i) * log(1 - h_\theta(x_i))). \tag{2}$$

The optimal value of θ is obtained by iteratively applying gradient descent algorithm.

Evaluation Metrics:

- CONFUSION MATRIX

  Confusion Matrix consists of True positives (TP) - correctly classified malware instances, True Negative (TN) - correctly identified benign instances, False Positive (FP) - benign instances misclassified as malware and False Negative (FN) - malicious samples misclassified as benign.

- RECALL

  Recall rate = TP/(TP + FN)

- ACCURACY

  Accuracy = (TP + TN)/(TP + TN + FP +FN)

Results - Logistic Regression Model

Confusion Matrix for Train data:

	$Pred_{benign}$	$Pred_{malware}$
$Actual_{benign}$	50	13
$Actual_{malware}$	8	15

Recall Rate = 65.2 %

Confusion Matrix for Test data:

	$Pred_{benign}$	$Pred_{malware}$
$Actual_{benign}$	24	3
$Actual_{malware}$	6	4

Recall Rate = 40 %

- The weighted accuracy for training data was 75.58% and for test data was 75.67%.

Multi Layer Perceptron(MLP) NN model

- We have implemented a three-layer (input layer, hidden layer and output layer) MLP neural network model.
- The number of neurons in the input layer represent the number of features in the input while the number of neurons in the output layer represent the number of classes.
- The weighted matrix W_i maps the input layer to the hidden layer. The dimension of the matrix W_i is #(hidden neurons) x #(input neurons).
- The weighted matrix W_h maps the hidden layer to the output layer. The dimension of the matrix is #(output neurons) x #(hidden neurons)

$$Y = N(W_i, W_h, X) = g((W_h * g(W_i * X))) \tag{3}$$

Here, g is a bi-sigmoidal function which can be given as below,

$$g(X) = \frac{-1 + exp(X)}{1 + exp(X)} \tag{4}$$

MLP NN model

- The objective is to minimize the error function which is done by updating the weights of the matrix using the backpropagation algorithm similar to logistic regression.
- Error function:

$$\mathbf{err_i} = \begin{cases} 0 & \text{if } y_{pred} * y_{actual} > 1 \\ (y_{pred} - y_{actual}) & \text{else} \end{cases} \tag{5}$$

The total modified least square error for N samples and n classes is as below

$$\mathbf{MLSE} = \sum_{j=1}^{N} (\sum_{i=1}^{n} (err_i^2)) \tag{6}$$

Results - MLP Model

Confusion Matrix for train data:

	$Pred_{benign}$	$Pred_{malware}$
$Actual_{benign}$	59	4
$Actual_{malware}$	9	14

Recall rate = 60.8 %

Confusion Matrix for Test data:

	$Pred_{benign}$	$Pred_{malware}$
$Actual_{benign}$	27	0
$Actual_{malware}$	4	9

Recall Rate = 69.23 %

- The weighted accuracy for training data was 79.06% and for test data was 81.08%.

Summary

- Implemented signature based malware detection tool using API call graphs for zero-day malware detection. The same can be extended to real life application.
- The features derived from graph theoretic properties were selected based on significant correlation with respect to output label.
- From the accuracy and recall rate, it can be observed that MLP model performs better than Logistic Regression based model.

Future Scope

- API calls executing similar functionality can be grouped to similar class/family/package. The number of states (API groups) reduces significantly and hence can be analyzed using Markov Chains models.

References I

- Zubair Shafiq-Department of Computer Science, The University of Iowa and Alex Liu - Department of Computer Science and Engineering, Michigan State University. "A graph theoretic approach to fast and accurate malware detection."
- Mamoun Alazab, Sitalakshmi Venkataraman, and PaulWatters. " Towards understanding malware behaviour by the extraction of api calls." In Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second, pages 52–59.IEEE, 2010.
- DEEPTI VIDYARTHI G. HAMSA. Study and analysis of various approaches for malware detection and identifi-cation.
- Shanhu Shang, Ning Zheng, Jian Xu, Ming Xu, andHaiping Zhang. Detecting malware variants via function-call graph similarity. InMalicious and Unwanted Soft-ware (MALWARE), 2010 5th International Conferenceon, pages 113–120. IEEE, 2010.
- ytisf. the zoo malware database. (https://github.com/ytisf/theZoo)

References II

Parvez Faruki, Vijay Laxmi, Manoj Singh Gaur, andP Vinod. Mining control flow graph as api call-grams todetect portable executable malware. InProceedings of theFifth International Conference on Security of Informationand Networks, pages 130–137. ACM, 2012

radare (http://rada.re/r/ and https://radare.gitbooks.io/radare2book/content)

Enrico Mariconti, Lucky Onwuzurike,Emiliano De Cristofaro, Gordon Ross, Gianluca Stringhini - University College London and Panagiotis Andriotis - University of the West of England "MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models"

References: Graph Properties

1. **Average shortest path:** The average shortest path length is

$$a = \sum_{s,t \in V} \frac{d(s,t)}{n(n-1)} \tag{7}$$

   where $d(s,t)$ the shortest path from node s to node t, and n is the number of nodes in G.

2. **Diameter:** The diameter is the maximum eccentricity. The eccentricity of a node v is the maximum distance from v to all other nodes in G.
3. **Average eigenvalue:** It is the average of eigenvalues of the adjacency matrix of G.
4. **Density:** The density of graph is

$$d = \frac{m}{n(n-1)},$$

   where n is the number of nodes and m is the number of edges in G.

5. **Graph Entropy:**

$$ent = \sum_{i}^{N} \frac{c_i}{log_2 c_i} \tag{8}$$

   where c_i is the normalized degree centrality of ith node.