Malware Detection

Introduction: -

Malware is short for malicious software, meaning software that can be used to compromise computer functions, steal data, bypass access controls, or otherwise cause harm to the host computer, its applications or data. It is designed to gain access to computer systems, normally for the benefit of some third party, without the user’s permission. Malware is usually introduced into a network through phishing, malicious attachments, or malicious downloads, but it may gain access through social engineering or flash drives as well. It's crucial that users know how to recognize the different types of malware in order to help protect yourself, and your business systems, from being compromised. Malware detection is one of the crucial computer security challenges due to the tremendous growth of new malware and variants of existing malware. Commercial antivirus scanners are unable to detect new malware since a known signature is not present in the virus database. Machine Learning has shown great promise in addressing this issue and is widely used for this purpose.

Problem Statement: -

In this assignment, you are going to build your own malware detection system. Here you are going to use only Windows executables. You are required to use the python utility pefile and linux command line utility strings to extract the static features from each PE file. Setup: Since the dataset contains malware executables, don't try to open the samples in the dataset. Set up an Ubuntu virtual machine (higher than 16.04) and write the python code for the below. Perform the below tasks to implement the malware detection model.

1. Download the dataset given as zipped file "malware_dataset.zip". The dataset consists of 3 folders 'malware' with 443 samples, 'benignware' with 400 samples and 'testdata' with 50 samples
2. Extract static features from each benign and malicious executable using python utility pefile and linux command line utility strings to represent the sample as an array of numbers. Assign the label as '1' for malware and '0' for benignware. This step also includes research to design good features that will help your machine learning system make accurate inferences. (Maximum number of features: 50)

3. Split dataset into non-overlapping training and test sets, in which the training set consists of 70% of the data (an arbitrarily chosen proportion) and the test set consists of the remaining 30%.

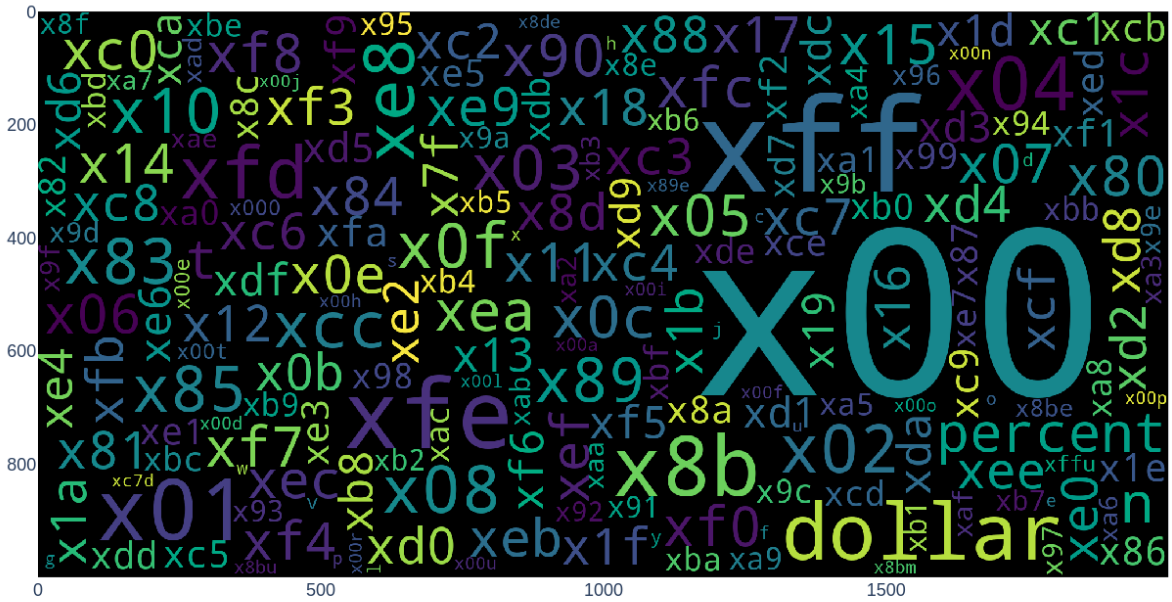
4. Train the below 2 machine learning classifiers to recognize malware using the features we have extracted. Use the inbuilt functions from sklearn for implementing the models and calculating metrics • K-Nearest Neighbors • Random Forest

5. Test your model and calculate the prediction accuracy, false-positive rate and confusion matrix.

6. Predict whether the executables in the dataset folder 'testdata' is malware or benignware and write the results to a csv file in the name "testlabel.csv". Format is given below. Mention '1' in "Category" column if file is malware, else 0.

Word cloud: - A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is.

Common words in comments



Pre-processing of Text:-

1. Delete all the duplicates rows
2. Removing html tags
3. Removing Punctuations
4. Performing stemming
5. Delete all the duplicates rows

```
In [7]: data.head()
```

Out[7]:

	class	filepath	contents
0	1	../malware/Backdoor.Win32.Agent.bhin_088e.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
1	1	../malware/Backdoor.Win32.Agent.bhin_0c2a.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
2	1	../malware/Backdoor.Win32.Agent.bhin_1119.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
3	1	../malware/Backdoor.Win32.Agent.bhin_12c5.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
4	1	../malware/Backdoor.Win32.Agent.bhin_138b.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...

```
In [8]: data.tail()
```

Out[8]:

	class	filepath	contents
840	0	../benignware/wuauclt.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
841	0	../benignware/wupdmgr.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
842	0	../benignware/xcopy.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
843	0	../benignware/xpnetdiag.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...
844	0	../benignware/zClientm.exe	b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\x...

What is Natural Language Processing (NLP) ?

Natural Language processing or NLP is a subset of [Artificial Intelligence (AI)](), where it is basically responsible for the understanding of human language by a machine or a robot.

One of the important subtopics in NLP is Natural Language Understanding (NLU) and the reason is that it is used to understand the structure and meaning of human language, and then with the help of computer science transform this linguistic knowledge into algorithms of Rules-based machine learning that can solve specific problems and perform desired tasks.

Apply tf-idf NLP algo to convert text data to vector –

TF-IDF : - Term Frequency Inverse Document Frequency

EX- If we have totel N documents

N=5 r = documents(comments)

r1 : w1 w2 w3 w2 w5
r2 : w1 w3 w4 w5 w6 w2
r3 : w1 w4 w7 w3 w4
r4 : w1 w6 w9
r5 : w2 w1 w3 w5 w8

step 1 : - Find all distinct(unique) words and count like this-

	w1	w2	w3	w4	w5	w6	w7	w8	w9
r1	1	2	1	0	1	0	0	0	0
r2	1	1	1	1	1	1	0	0	0
r3	1	0	1	2	0	0	1	0	0
r4	1	0	0	0	0	1	0	0	1
r5	1	1	1	0	1	0	0	1	0

TF : -

$$TF(w_i , r_j) = \frac{\text{number if times } w_i \text{ occurd in } r_j}{\text{totel number if words in } r_j}$$

EX-

$$TF(w2 , r1) = \frac{2}{5} = 0.4$$

Note :- $0 <= tf(w_i , r_j) <= 1$ always

IDF :-

Dc = r1 : r2 : r3 : ... rN :

Totel we have N documents

DATA CORPAS D_c = { r1 , r2 , r3, - - - - - - - ,rN }

$$IDF(W_i , D_c) = \log\left(\frac{N}{n_i}\right)$$

where N = number of documents
n_i = number of documents which contain W_i

Ex-

IDF(W1 , Dc) = log (5/5) = 0
IDF(W2 , Dc) = log (5/3) = 0.22
IDF(W9 , Dc) = log (5/1) = 0.69

$$\boxed{TF\text{-}IDF(w_i) = TF(w_i, r_j) * IDF(w_i, D_c)}$$

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
```

```
tf_idf_vect = TfidfVectorizer(max_features=182054)
final_tf_idf = tf_idf_vect.fit_transform(data['contents'].values)
```

```
final_tf_idf.get_shape()
```

```
(844, 182054)
```

```
features = tf_idf_vect.get_feature_names()
len(features)
```

```
182054
```

Split data 70:30 ratio randomly: -

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)

Applying different ML algorithms to analyse which one is best for this dataset: -

Apply RandomForest to binary classification :-

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_jobs=-1)
rf_clf.fit(X_train, y_train)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    309.0  281.0       1.0      590.0         590.0
_______________________________________________
Confusion Matrix:
 [[309   0]
 [  0 281]]

Test Result:
================================================
Accuracy Score: 95.28%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.955556    0.94958  0.952756    0.952568      0.952756
recall       0.955556    0.94958  0.952756    0.952568      0.952756
f1-score     0.955556    0.94958  0.952756    0.952568      0.952756
support    135.000000  119.00000  0.952756  254.000000    254.000000
_______________________________________________
Confusion Matrix:
 [[129   6]
 [  6 113]]
```

Apply K-NN to binary classification: -

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(X_train, y_train)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 96.78%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.970779    0.964539  0.967797    0.967659      0.967807
recall       0.967638    0.967972  0.967797    0.967805      0.967797
f1-score     0.969206    0.966252  0.967797    0.967729      0.967799
support    309.000000  281.000000  0.967797  590.000000    590.000000
_______________________________________________
Confusion Matrix:
 [[299  10]
 [  9 272]]

Test Result:
================================================
Accuracy Score: 93.31%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.940299    0.925000  0.933071    0.932649      0.933131
recall       0.933333    0.932773  0.933071    0.933053      0.933071
f1-score     0.936803    0.928870  0.933071    0.932837      0.933086
support    135.000000  119.000000  0.933071  254.000000    254.000000
_______________________________________________
Confusion Matrix:
 [[126   9]
 [  8 111]]
```

Conclusion :-

I got best accuracy and precision value using RandomForest algo.

Algorithmen	Accuracy	Precision	Recall	F1-score
RandomForest	95.28%	0.94958	0.94958	0.94958
K-NN	93.31%	0.925000	0.932773	0.928870