Malware Detection

Import Required Packages

In [141]:

```
import json
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, VotingClas
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot as plt
```

Running the python script *'prepare_dataset.py'*, We have decompiled all the APKs available in dataset and get the as much as features that we can grab and are relevant to detect malware or phising Apps.

In android app development, The main part of getting into in someone's privacy is allowing the dangerous permissions that application ask to the user. Such as permission like Contacts, Storage, Get Running Tasks etc. By using these permission various apk grabs the confedential data from the app and may missuse it or sell it.

So We have full list of permissions available in android os so loading them and creating matrix for permission in a way that we can feed it into classifier.

In [142]:

```
ALL_PERMISSIONS = open('permissions.txt', 'r').readlines()
ALL_OPECODES = []
ALL_STRINGS = []
```

In [143]:

```
print('Listing 1st 20 permissions')
ALL_PERMISSIONS[0:20]
```

```
Listing 1st 20 permissions
```

Out[143]:

```
['android.permission.ACCESS_ALL_DOWNLOADS\n',
 'android.permission.ACCESS_BLUETOOTH_SHARE\n',
 'android.permission.ACCESS_CACHE_FILESYSTEM\n',
 'android.permission.ACCESS_CHECKIN_PROPERTIES\n',
 'android.permission.ACCESS_CONTENT_PROVIDERS_EXTERNALLY\n',
 'android.permission.ACCESS_DOWNLOAD_MANAGER\n',
 'android.permission.ACCESS_DOWNLOAD_MANAGER_ADVANCED\n',
 'android.permission.ACCESS_DRM_CERTIFICATES\n',
 'android.permission.ACCESS_EPHEMERAL_APPS\n',
 'android.permission.ACCESS_FM_RADIO\n',
 'android.permission.ACCESS_INPUT_FLINGER\n',
 'android.permission.ACCESS_KEYGUARD_SECURE_STORAGE\n',
 'android.permission.ACCESS_LOCATION_EXTRA_COMMANDS\n',
 'android.permission.ACCESS_MOCK_LOCATION\n',
 'android.permission.ACCESS_MTP\n',
 'android.permission.ACCESS_NETWORK_CONDITIONS\n',
 'android.permission.ACCESS_NETWORK_STATE\n',
 'android.permission.ACCESS_NOTIFICATIONS\n',
 'android.permission.ACCESS_NOTIFICATION_POLICY\n',
 'android.permission.ACCESS_PDB_STATE\n']
```

Now loading **data.json** file which is generated through **prepare_dataset.py** that contains all the **static** and **dynamic** features such as OpCodes, Scan Info, VirusTotal Analysis, String, Api Calls, Permissions etc of all the android apps.

This below line may take some time as it is loading the 38.6mb large json dataset of APKs.

In [144]:

```
with open('data.json', 'r') as jsonFile:
    data = json.load(jsonFile)

print("Printing the availlable features of 1st APK in data.json")
data[0:1]
```

```
   'android.app.IntentService.onDestroy': 1,
   'android.app.IntentService.onStartCommand': 1,
   'android.content.SharedPreferences': 2,
   'java.lang.Boolean': 224,
   'java.lang.Boolean.booleanValue': 13,
   'java.lang.Character': 3,
   'java.lang.Exception': 4,
   'java.lang.Integer': 84,
   'java.lang.Integer.intValue': 16,
   'java.lang.Integer.valueOf': 67,
   'java.lang.Long': 6,
   'java.lang.Long.longValue': 5,
   'java.lang.Object': 22,
   'java.lang.Object.equals': 3,
   'java.lang.Object.toString': 14,
   'java.lang.String': 31,
   'java.lang.String.charAt': 3,
   'java.lang.String.contains': 2,
   'java.lang.String.endsWith': 2,
   'java.lang.String.equalsIgnoreCase': 6,
```

Now we have very complex dataset of APKs which includes numerical and nonnumerical dataset. But the classifier only understands only numerical dataset. So we are converting nonnumerical dataset into categorical dataset.

The following function is defined to get permission matrix in which we are marking them as binary. **1** for the permission used in app and **0** for not used.

In [145]:

```
def get_permission_matrix(permissions):
    perm_vector = np.zeros(len(ALL_PERMISSIONS))
    for permission in permissions:
        for i in range(len(perm_vector)):
            if ALL_PERMISSIONS[i].strip() == permission:
                perm_vector[i] = 1
            else:
                perm_vector[i] = 0
    return perm_vector
```

So we just finished by creating on method for creating permission matrix which is defined static in android application. Lets create matrix for dynamic features that are created from the dynamic methods available in the android app. Such as moveObj() = 'move-object' that used to move object from one directory to another or to delete that object.

To list all the operational code used in android app.

In [146]:

```
def generate_opcode_vector(dictOpCodes, ALL_OPECODES):
    opcode_vec = []
    for opcode in ALL_OPECODES:
        try:
            opcode_vec.append(dictOpCodes[opcode])
        except Exception as err:
            opcode_vec.append(0)
    return opcode_vec

# generating all opcodes
for datarow in data:
    ALL_OPECODES += list(datarow['Opcodes'].keys())

ALL_OPECODES = sorted(list(set(ALL_OPECODES)))
print('TOTAL OPCODES ' + str(len(ALL_OPECODES)))
```

```
TOTAL OPCODES 218
```

Listing some of operation codes of android apps that we grabbed.

In [147]:

```
ALL_OPECODES[50:70]
```

Out[147]:

```
['div-float',
 'div-float/2addr',
 'div-int',
 'div-int/2addr',
 'div-int/lit16',
 'div-int/lit8',
 'div-long',
 'div-long/2addr',
 'double-to-float',
 'double-to-int',
 'double-to-long',
 'fill-array-data',
 'fill-array-data-payload',
 'filled-new-array',
 'filled-new-array/range',
 'float-to-double',
 'float-to-int',
 'float-to-long',
 'goto',
 'goto/16']
```

Sometimes majority of malware apps does not follow the coding guidence so they place most of information static in application. The main scope of malware apps to damage the target because of that they did not focus on removing static strings from the apps.

function to list all the static strings are defined in android app.

In [148]:

```
# genrating all strings
for datarow in data:
    ALL_STRINGS += list(datarow['Strings'].keys())

ALL_STRINGS = sorted(list(set(ALL_STRINGS)))
print('TOTAL STRINGS ' + str(len(ALL_STRINGS)))

print('Printing some examples of static strings defined')
ALL_STRINGS[500:510]
```

```
TOTAL STRINGS 197549
Printing some examples of static strings defined
```

Out[148]:

```
['#publicit\\u00e9',
 '#publicite',
 '#px',
 '#read(byte[]) returned invalid result:',
 '#recent-check',
 '#root',
 '#sdk_version_',
 '#search_input{',
 '#search_input{color: #6c6c6c;}',
 '#search_input{color:#666666;}']
```

function to generate string vector matric from all the above strings

In [149]:

```
def generate_opstrings_vector(dictOpStrings, ALL_STRINGS):
    opstring_vec = []
    for opstring in ALL_STRINGS:
        try:
            opstring_vec.append(dictOpStrings[opstring])
        except Exception as err:
            opstring_vec.append(0)
    return opstring_vec
```

Finally, We are merging the all processed and proper features to create one dataframe so that we can easily see or reuse it.

Creating dataset for all the data available from data json and storing it into matrix form.

Following method also may take some time as it also loops through all APK's feature and stores into dictionary and then into dataframe.

In [150]:

```
pdArr = []
scan_columns = []
appendCols = True
for i in range(len(data)):
    datarow = data[i]
    permission_vec = get_permission_matrix(datarow['Permissions'])
    opcode_vec = generate_opcode_vector(datarow['Opcodes'], ALL_OPECODES)
    opstring_vec = generate_opstrings_vector(datarow['Strings'], ALL_STRINGS)

    #Creating each apk's dictionary and appending into array.
    dictVector = {}
    dictVector['Permissions'] = permission_vec
    dictVector['Opcodes'] = opcode_vec
    dictVector['Strings'] = opstring_vec
    dictVector['VersionCode'] = datarow['VersionCode']
    dictVector['isMalware'] = int(datarow['malware'])


    #appending virustotal features
    vt_json = json.load(open('VT_ANALYSIS/test.json'))  # to append key value

    try:
        #appedning virus info cols dynamically
        for scanInfo in vt_json['scans']:
            dictVector[str(scanInfo)] = int(datarow[scanInfo])
            if appendCols: scan_columns.append(str(scanInfo))

        dictVector['vt_total'] = datarow['vt_total']
        dictVector['vt_positives'] = datarow['vt_positives']
    except Exception as err:
        #appedning virus info for not available
        for scanInfo in vt_json['scans']:
            dictVector[str(scanInfo)] = int(datarow['malware'])

        dictVector['vt_total'] = 0
        dictVector['vt_positives'] = 0
        print(err)

    scan_columns.append('Permissions')
    scan_columns.append('Opcodes')
    scan_columns.append('Strings')
    scan_columns.append('VersionCode')
    scan_columns.append('vt_total')
    scan_columns.append('vt_positives')
    scan_columns.append('isMalware')

    pdArr.append(dictVector)

df = pd.DataFrame(pdArr, columns=list(set(scan_columns)), dtype=np.float32)
# df.to_csv('dataframe.csv', index=None, header=True) stores large size of csv
df.head()
```

```
'WhiteArmor'
'TheHacker'
'TrendMicro-HouseCall'
'TheHacker'
'TheHacker'

'TheHacker'
```

```
'TrendMicro'
'TheHacker'
'TheHacker'
'TheHacker'
'Webroot'
'TheHacker'
'Bkav'
'TrendMicro-HouseCall'
'TheHacker'
'WhiteArmor'
'GData'
```

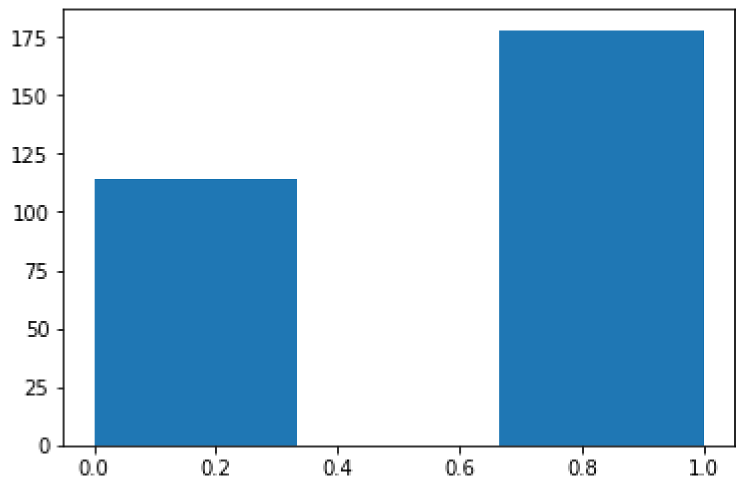
As we can see that, We have all the features are presented in visualize form to evaluate it in better way.

In [151]:

```
plt.hist(df['isMalware'], bins=3)
```

Out[151]:

```
(array([114.,   0., 178.]),
 array([0.        , 0.33333333, 0.66666667, 1.        ]),
 <a list of 3 Patch objects>)
```



In [152]:

```
print('From above chart')
print('We have '+str(len(df[df['isMalware'] == 0]))+ ' benign and ' +str(len(df[df[
```

```
From above chart
We have 114 benign and 178 malware APKs
```

Now we done with the feature engineering, now we have to combine them all.

Creating label **y** from the **'isMalware'** column as it is out target label for classifier.

In [153]:

```
y = np.asarray(list(df['isMalware']))
del df['isMalware']
```

Now we have to combine individual matrix of "Permissions", "Opcodes" and "Strings" into X. But lets create Xp, Xo and Xs seprately and combine them later.

In [154]:

```
Xp = np.asarray(list(df['Permissions']))
Xo = np.asarray(list(df['Opcodes']))
Xs = np.asarray(list(df['Strings']))
```

Creating features from the columns 'Permissions', 'Opcodes', 'Strings', 'VersionCode' and storing it into X

In [155]:

```

X = np.concatenate((Xp, Xo), 1)
X = np.concatenate((X, Xs), 1)
```

Remove the above defined columns as we already saved the features in Xp, Xo and Xs.

In [156]:

```
del df['Permissions'], df['Opcodes'], df['Strings']
```

Let's check the remaining columns

In [157]:

```
df.values.shape
```

Out[157]:

```
(292, 61)
```

So we have still 61 columns left to be added into feature matrix. Let's see which columns yet pending.

In [158]:

```
df.head()
```

Out[158]:

	McAfee	Panda	VBA32	F-Secure	K7AntiVirus	Kingsoft	TrendMicro-HouseCall	TheHacker	McAfee-GW-Edition	E
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
4	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 61 columns

Now combining the all above matrices into out final feature matrix X.

In [159]:

```
X = np.concatenate((X, df.values), 1)
```

Feature Vector is Ready !!

Splitting the data into training/testing dataset is one of the techniques that used to check that our dataset is working proper. It also used to reduce overfitting and underfitting related problems.

The most used trade off is the **70%** (training data) and **30%** (testing data)

so splitting the data into training and testing dataset for making the dataset as robust as possible

In [160]:

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_sc
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.3, random_state=0
```

In [161]:

```
print('length of training dataset '+ str(len(yTrain)) + ' & length of testing datas
```

```
length of training dataset 204 & length of testing dataset 88
```

Feeding these feature and labels in SVC classifier

In [162]:

```
svc = SVC(C=1.0, kernel='linear', probability=True, gamma=0.1, tol=0.001)
```

In [163]:

```
svc.fit(xTrain, yTrain)
score = str(svc.score(xTest, yTest))
print('SVC Score ' + score)
```

```
SVC Score 0.9545454545454546
```

Lets evaluate result report of our SVC Classifier

In [164]:

```
y_pred = svc.predict(xTest)
from sklearn import metrics

svc_report = metrics.classification_report(yTest, y_pred, target_names=['benign', '
print(svc_report)
```

```
             precision    recall  f1-score   support

     benign       1.00      0.89      0.94        38
    malware       0.93      1.00      0.96        50

avg / total       0.96      0.95      0.95        88
```

So we have out classification report that predicts 100% accuracy. Howerver another approch of testing our classifier is plotting confusion matrix. Which will give you the exact idea about how our algo works in perspective of True Positive & False Positive.

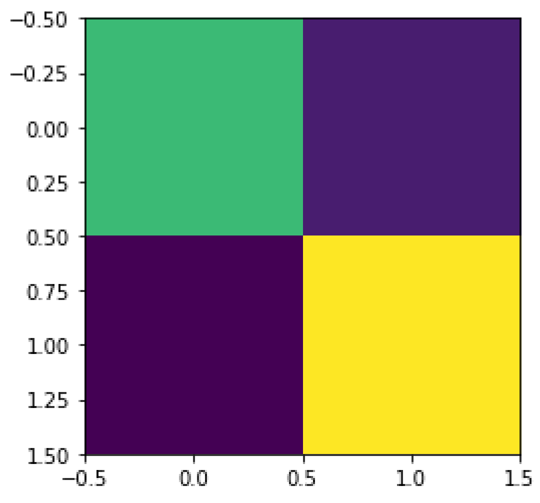
In [165]:

```
# Plot the confusion matrix
cm = metrics.confusion_matrix(yTest, y_pred)
print(cm)
plt.imshow(cm)
```

```
[[34  4]
 [ 0 50]]
```

Out[165]:

```
<matplotlib.image.AxesImage at 0x7fb7540e8898>
```



So our algo. works quite good in this report also as it classifies 34 out of 38 of benign and 4 misclassified and 50 out of 50 of malware.

In [166]:

```
clf = AdaBoostClassifier(n_estimators=100, random_state=0)
clf.fit(xTrain, yTrain)
score = str(clf.score(xTest, yTest))
print('AdaBoostClassifier Score ' + score)
```

```
AdaBoostClassifier Score 0.9886363636363636
```

Lets evaluate result report of our AdaBoostClassifier

In [167]:

```
y_pred = clf.predict(xTest)
ada_report = metrics.classification_report(yTest, y_pred, target_names=['benign', '
print(ada_report)
```

```
             precision    recall  f1-score   support

     benign       0.97      1.00      0.99        38
    malware       1.00      0.98      0.99        50

avg / total       0.99      0.99      0.99        88
```

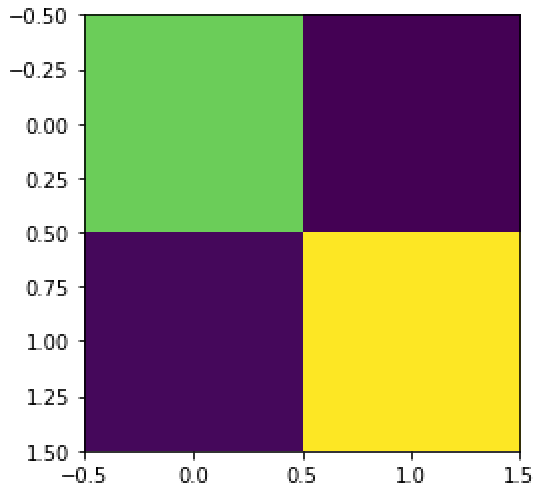
In [168]:

```
# Plot the confusion matrix
cm = metrics.confusion_matrix(yTest, y_pred)
print(cm)
plt.imshow(cm)
```

```
[[38  0]
 [ 1 49]]
```

Out[168]:

```
<matplotlib.image.AxesImage at 0x7fb753e2ae80>
```



In [169]:

```
rclf = DecisionTreeClassifier(max_depth=20, random_state=0)
rclf.fit(xTrain, yTrain)
score = str(rclf.score(xTest, yTest))
print('DecisionTreeClassifier Score ' + score)
```

```
DecisionTreeClassifier Score 0.9886363636363636
```

Lets evaluate result report of our DecisionTreeClassifier

In [170]:

```
y_pred = rclf.predict(xTest)
dt_report = metrics.classification_report(yTest, y_pred, target_names=['benign', 'm
print(dt_report)
```

```
             precision    recall  f1-score   support

     benign       0.97      1.00      0.99        38
    malware       1.00      0.98      0.99        50

avg / total       0.99      0.99      0.99        88
```

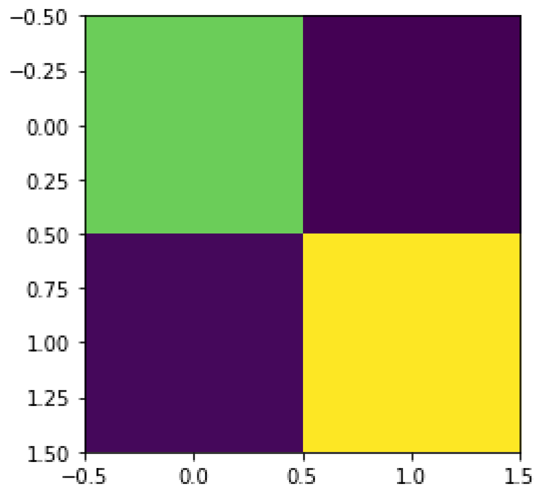
In [171]:

```
# Plot the confusion matrix
cm = metrics.confusion_matrix(yTest, y_pred)
print(cm)
plt.imshow(cm)
```

```
[[38  0]
 [ 1 49]]
```

Out[171]:

```
<matplotlib.image.AxesImage at 0x7fb754911438>
```



Now let combine all the classifiers into VotingClassifier for robustness in the production to detect malware apk correctly.

In [172]:

```
eclf2 = VotingClassifier(estimators=[('svc', svc), ('adaboost', clf), ('rf', rclf)]
eclf2.fit(xTrain, yTrain)
score = str(eclf2.score(xTest, yTest))
print('VotingClassifier Score ' + score)
```

```
VotingClassifier Score 1.0

/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:
151: DeprecationWarning: The truth value of an empty array is ambiguou
s. Returning False, but in future this will result in an error. Use `a
rray.size > 0` to check that an array is not empty.
  if diff:
```

Lets evaluate result report of our VotingClassifier

In [173]:

```
y_pred = eclf2.predict(xTest)
vote_report = metrics.classification_report(yTest, y_pred, target_names=['benign',
print(vote_report)
```

```
             precision    recall  f1-score   support

     benign       1.00      1.00      1.00        38
    malware       1.00      1.00      1.00        50

avg / total       1.00      1.00      1.00        88


/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:
151: DeprecationWarning: The truth value of an empty array is ambiguou
s. Returning False, but in future this will result in an error. Use `a
rray.size > 0` to check that an array is not empty.
  if diff:
```

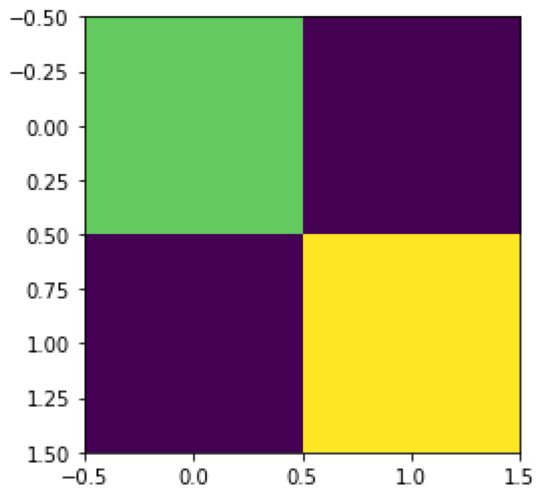
In [174]:

```
# Plot the confusion matrix
cm = metrics.confusion_matrix(yTest, y_pred)
print(cm)
plt.imshow(cm)
```

```
[[38  0]
 [ 0 50]]
```

Out[174]:

```
<matplotlib.image.AxesImage at 0x7fb7539d3278>
```



The confusion matrix above also says that by combining all the 3 classifiers it classifies 38 out of 38 of benign and 50 out of 50 of malware and achievieng 100% accuracy.

finally storing the trained classifier into file storage.

In [175]:

```
import pickle

# save the model to disk
filename = 'classifier.dat'
pickle.dump(eclf2, open(filename, 'wb'))
```