ANDROID MALWARE DETECTION

ABSTRACT

- Most of the anti-malware Software are using Unique Identifier which is ineffective to detect advanced unknown malwares. In this paper, we study How Machine Learning algorithms are effectively detecting Malware/Benign apps based on the Permissions and API Call Sequences, Where we will be using bagging, boosting and SVM to detect Malware.

INTRODUCTION

- Android is a mobile operating system designed for smartphones and tablets.
- Android's kernel is being developed upon the Linux kernel. Different Devices uses Different Kernels versions like 4.14, 4.19 or 5.4 of the Linux kernel.
- Android Mobile malware is **Malicious Software** specially being written to attack mobile devices. These types of malware Targets Mobile’s operating systems and mobile phone technology, it is a potential threat to consumer devices. Mobile malware becomes more challenging for the cyber security industry as Malwares are becoming more powerful and attacks more frequently.
- Android Mobile malware developers are called cybercriminals. They could have many objectives behind developing this kind of Malware. Usually they demand money to release the data they steals from the consumer.

TYPES OF MALWARES?

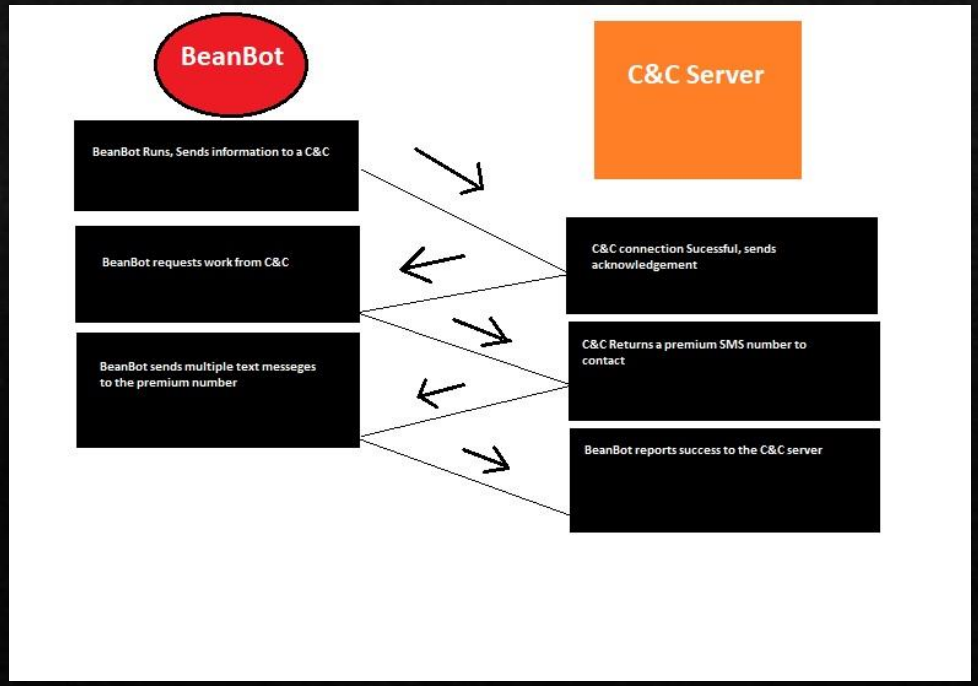
- The Most common malwares types are viruses, worms, mobile bots, mobile phishing attacks, ransomware, spyware and Trojans. Some of them may be the combination of more then one type.
- worm is a type of malware that infects other devices also while remains active on primarily infected device. Worms typically being transmitted through SMS or MMS.
- mobile bot runs as Default apps on consumer device once it’s being installed and It gets all the access to the particular device and the content of it.
- Ransomware malware uses encryption to lock the data on the infected device itself and demands the payments before the data is decrypted and allow consumer to access the data again.
- Spyware attacks email accounts, notes , colander apps and other app where personal data can be found and the data is being sent to it’s remote server.
- Trojan Malware typically uses some other apps to insert into the consumer devices, it deactivates all the other application the device and also can deactivate consumer device itself, once user open or execute it. Most effective Trojans are Banking Trojans, which can target both international and regional banks using fake apps.

MALWARE FAMILIES ?

Family	Type	Category	Aliases	Summary
AccuTrack	Monitoring-Tool	Riskware	AccuTrack.A	AccuTrack.A tracks down the GPS location of the device on which it was installed on.
AdSMS	Trojan	Malware	AdSMS.A	AdSMS.A is distributed via a link embedded in a fake SMS; on clicking the link, the trojan's Android Package (APK) file is downloaded and installed onto the mobile device.
BeanBot	Trojan	Malware	Trojan:Android/BeanBot.A	BeanBot.A forwards device's data to a remote server and sends out premium-rate SMS messages from the infected device.
Cawitt	Trojan	Malware	Trojan:Android/Cawitt.A	Cawitt.A operates silently in the background, gathering device information which it later forwards to a remote server.
DroidDream	Trojan	Malware	Trojan:Android/DroidDream	Trojans in the DroidDream family collects data from an infected device, and later forwards the data to a remote server.
RootSmart/Bmaster	Trojan-Downloader	Malware	Trojan-Downloader:Android/RootSmart, Trojan-Downloader:Android/RootSmart.A	Trojan-Downloader:Android/RootSmart forwards device details to a remote server, and downloads and installs additional applications onto the compromised device.

HOW DIFRRENT FAMILY BUT SAME TYPE OF MALWARE WORKS?

- BeanBot Malware Family:
  1. BeanBot Malware is a type of Trojan, which transport identifiable information from infected deice to C&C (Command and Control) server and also sends unwanted text messages in the background that leads to excessive phone bills.
  2. BeanBot include many packages, which are as follow:



- BeanBot usually disguised with some fake legit application license. It uses repackage version of legitimate Android Applications and rewrite small portion of the original Apps source code to use the framework to get information about app and it's fooling the license code by calling the original framework call while it changes certain fields in the returned object.

- AdSMS

  - This is also a Trojan type Malware for the Android devices using Android version >2.1. It registers to a service provider's phone number and send messages to those phone without consent of the victim.
  - Once it's been installed to Mobile Devices, the packages doesn't show any icons. It can be seen running on the device.
  - It kills applications on the victim devices
  - Sends SMS without consent of the victim uses IMEI number of victim's device.
  - Contact with remote server to download configurations.

```
com.qihoo360.mobilesafe
com.tencent.qqpimsecure
com.anguanjia.safe
```

```
http://adsms.[CENSORED].cn/Submit.aspx?ver=1.4&sys=SDKLEVEL
  &imei=IMEI&ua=PHONEMODEL&pro=100,1000
```

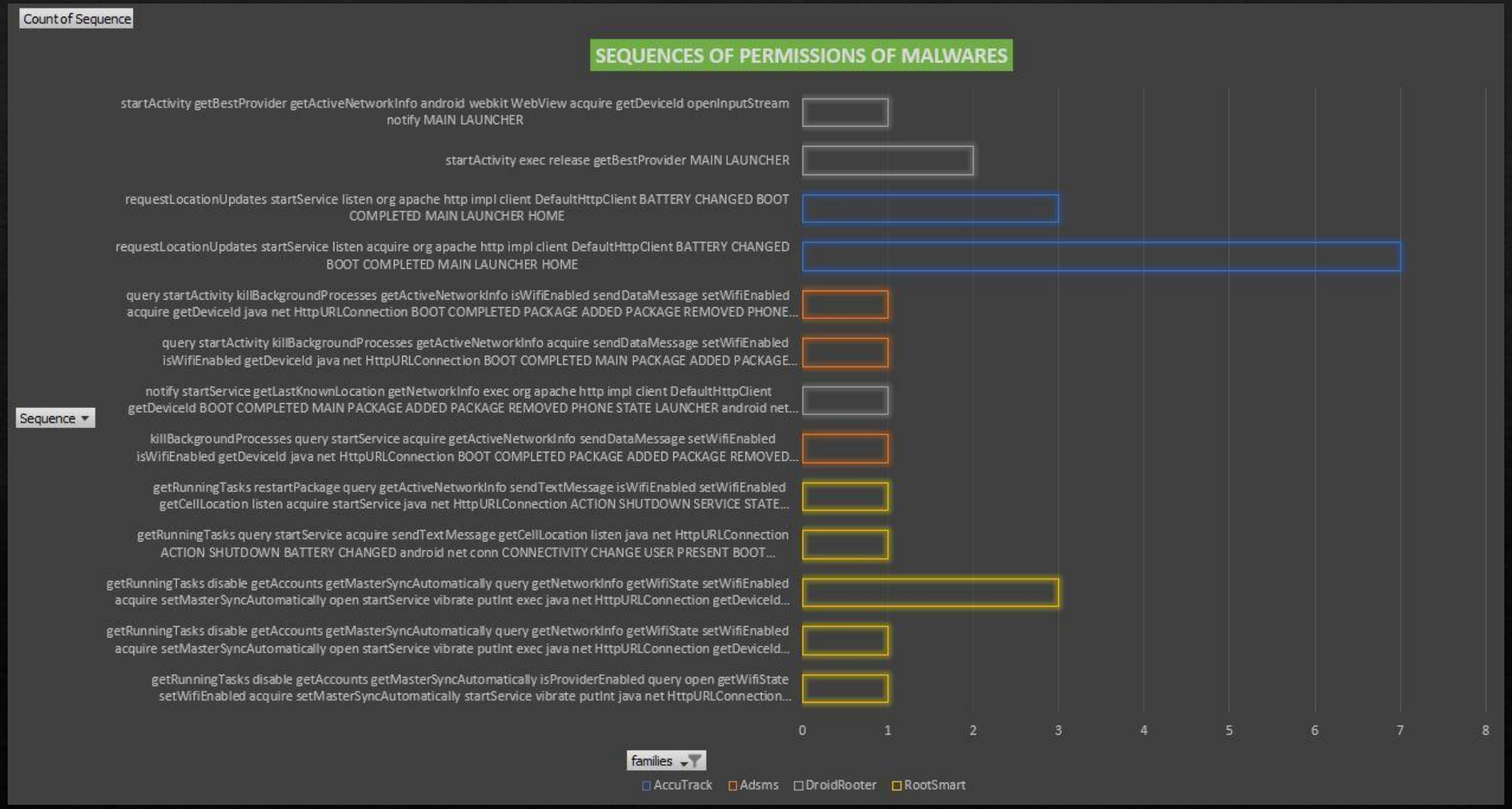
Configuration that AdSMS Downloads

```
<cmdsystem>
        <mobile>PHONE NUMBER</mobile>
        <regport>PHONE NUMBER TO REGISTER</regport>
        <actport>PHONE NUMBERS</actport>
        <mode>time</mode>
        <pbsendport>1062, 1065, 1066</pbsendport>
        <killprocess>APP TO KILL</killprocess>
        <killinstall>APPLICATION IDs</killinstall>
        <killuninst>APPLICATION IDs</killuninst>
</cmdsystem>
<cmdupdate>
  <version></version>
  <verurl>URL TO DOWNLOAD UPDATE</verurl>
</cmdupdate>
<channel>
        <spid></spid>
        <sptype>SMS or MMS</sptype>
        <spdir>BODY of SMS to send to SPPORT</spdir>
        <spport>PHONE NUMBER</spport>
        <spredir>PHONE NUMBER</spredir>
        <sptimes>INTEGER</sptimes>
        <rekeyword></rekeyword>
        <report></report>
</channel>
<cmdpush>
        <pushmode></pushmode>
        <pushlist>COMMA SEPARATED LIST</pushlist>
        <pushurl></pushurl>
        <pushbody></pushbody>
</cmdpush>
<cmdSepCnl>
        <keyword></keyword>
        <length></length>
        <kfconfirm>PHONE NUMBER to SEND SMS CONFIRMATION</kfconfirm>
        <spid></spid>
</cmdSepCnl>
```

- RootSmart
  - It's a type of Adware, Adware are use to pop up advertisement randomly on the screen.
  - It also Classify as a Trojan Malware type.
  - It is capable of remote access connection handling using Dos to capture keyboard entries.
  - It Attacks Android Devices running Android version = 2.3.4 or greater.
  - Once installed, the Fcbakeluncher activity starts and start the service of malicious activities on the victim's device.
  - It Process various Actions like:
    1. marks the malware as started.
    2. set an alarm to check the phone is rooted.
    3. Shutdown.
    4. start a thread named l that locates a local directory named "shells".
    5. starts a thread named u that finished the installation.
    6. checks the status of the application.
    7. start a thread named i, that downloads a file named shells.zip from a remote server.
    8. retrieve device's IMEI and posts the information to a remote server.
    9. starts a thread named t that will download a package from a remote server and check its MD5.

- All the files downloaded from remote server.
- host is encrypted and hidden.
- Decrypted url: `go.[CENSORED].com`
- The file is downloaded from : `go.[CENSORED].com/androidService/resources/commons/shells.zip`

WHAT ARE USER PERMISSIONS SEQUENCES FOR DIFFERENT MALWARES?



SIMILARITIES AND DIFERENCE BETWEEN MALWARE FAMILIES?

- Similarities:
  1. Most of the Malware samples has used same permissions.
  2. Most Malware Samples are attacking internal storage.
  3. Malware Samples connects with remote server.
  4. Most of the Malware tries to still data from the device.
- Differences:
  1. Most of the Malwares has different intent to attack Devics.

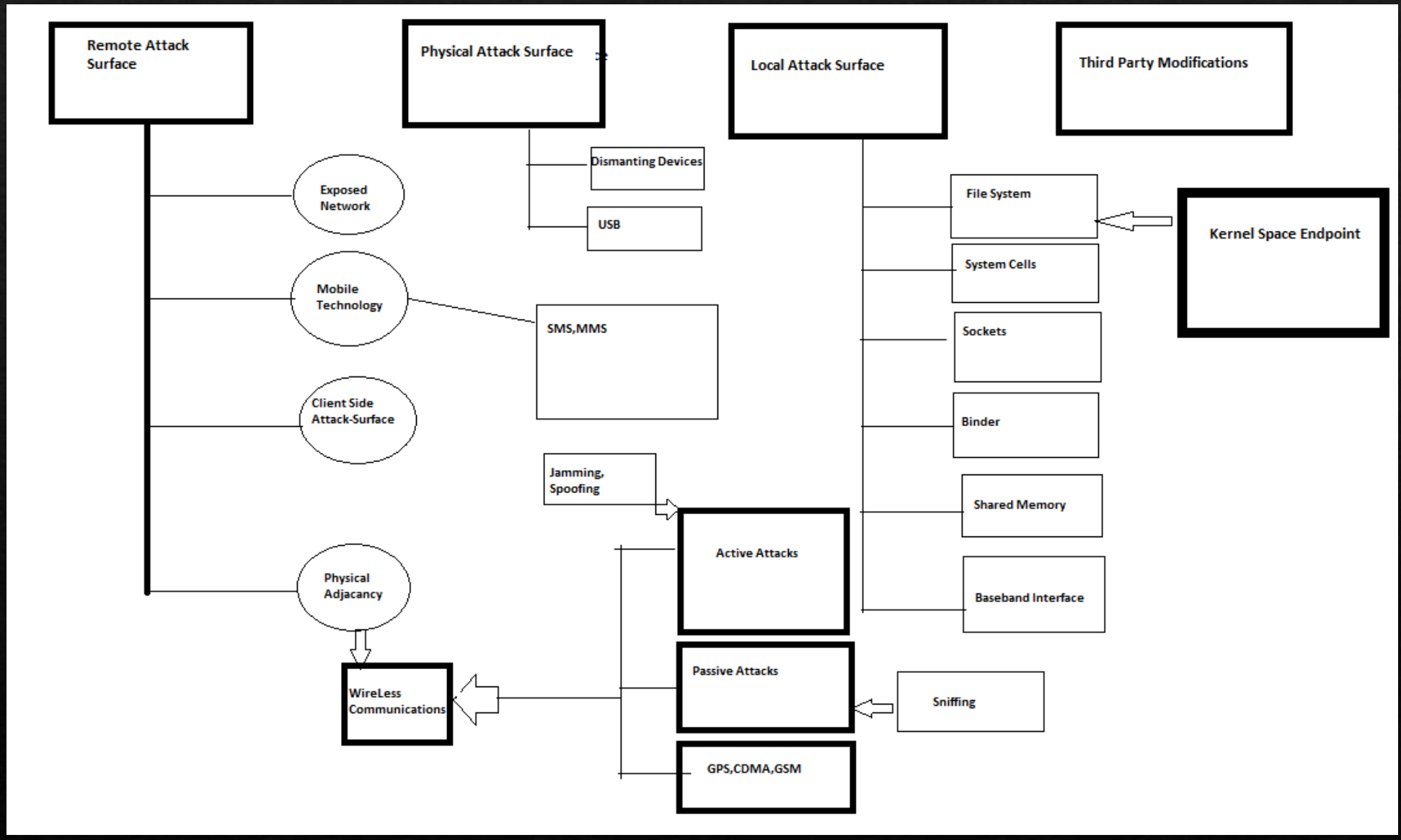
How are Malicious activities and their critical API call and Permission sequence pattern?

Sequence	families
startService notify exec getDeviceId java net HttpURLConnection BOOT COMPLETED MAIN LAUNCHER	ExploitLinuxLotoor
exec startActivity notify query getLastKnownLocation getConnectionInfo android webkit WebView release getAllBookmarks getDeviceId getNetworkInfo BOOT COMPLETED MAIN HOME LAUNCHER	Plankton
getRunningTasks start notify query startActivity getActiveNetworkInfo getConnectionInfo android webkit WebView setWifiEnabled getCellLocation exec BATTERY CHANGED ACTION LAUNCHER BOOT COMPLETED MAIN SIG STR	DroidKungFu
startActivity getNetworkInfo openConnection ACTION POWER CONNECTED INPUT METHOD CHANGED MAIN UMS CONNECTED android net conn CONNECTIVITY CHANGE UMS DISCONNECTED USER PRESENT LAUNCHER android net conn MEDIA NOFS android net	BaseBridge
restartPackage sendTextMessage getSubscriberId java net HttpURLConnection MAIN LAUNCHER	MobileTx
notify startActivity java net HttpURLConnection getDeviceId BOOT COMPLETED MAIN PHONE STATE LAUNCHER	FakeInstaller
notify query startActivity getActiveNetworkInfo getWifiState setWifiEnabled getBestProvider exec java net HttpURLConnection BATTERY CHANGED ACTION LAUNCHER EDIT MAIN SIG STR VIEW DEFAULT BOOT COMPLETED	DroidKungFu
query exec startActivity getActiveNetworkInfo getDeviceId android webkit WebView MAIN LAUNCHER	DroidKungFu
android webkit WebView BOOT COMPLETED MAIN HOME LAUNCHER	Opfake
startActivity sendTextMessage getDeviceId android webkit WebView BOOT COMPLETED MAIN PHONE STATE USER PRESENT LAUNCHER	Opfake
getAccounts getRunningTasks killBackgroundProcesses stop notify query startActivity sendTextMessage getActiveNetworkInfo isWifiEnabled exec java net HttpURLConnection getDeviceId CheckinHub BROWSABLE MAIN PACKAGE ADDED PACKAGE REMOVED	GinMaster
notify sendBroadcast setWallpaper getActiveNetworkInfo getDeviceId exec BOOT COMPLETED MAIN LAUNCHER	GinMaster
notify startActivity setWallpaper getNetworkInfo exec org apache http impl client DefaultHttpClient BOOT COMPLETED MAIN LAUNCHER	GinMaster
notify setComponentEnabledSetting query getLastKnownLocation getActiveNetworkInfo getConnectionInfo startService openStream acquire BOOT COMPLETED PACKAGE ADDED MAIN PACKAGE REMOVED PACKAGE REPLACED INFO LAUNCHER	Plankton
sendTextMessage MAIN LAUNCHER	Jifake
sendBroadcast putInt sendMultipartTextMessage listen BOOT COMPLETED	MTracker
getRunningTasks notify query startService getActiveNetworkInfo getConnectionInfo exec getDeviceId BOOT COMPLETED MAIN default	SMSZombie
notify query getActiveNetworkInfo startService getLastKnownLocation openStream acquire getAccounts getAllBookmarks sendTextMessage getDeviceId BOOT COMPLETED HOME com google android c dm intent RECEIVE com google android c dm intent REGIST	Plankton
startService getBestProvider acquire sendTextMessage getDeviceId android webkit WebView query BOOT COMPLETED MAIN LAUNCHER	Iconosys
query getLastKnownLocation start listen sendBroadcast java net HttpURLConnection MAIN PHONE STATE LAUNCHER	Fidall
notify sendBroadcast acquire sendTextMessage org apache http impl client DefaultHttpClient BOOT COMPLETED MAIN HOME LAUNCHER com google android c dm intent RECEIVE com google android c dm intent REGISTRATION DATA SMS RECEIVED	FakeInstaller
notify query startService getAllNetworkInfo java net HttpURLConnection getSubscriberId BOOT COMPLETED INPUT METHOD CHANGED ACTION POWER CONNECTED android net conn CONNECTIVITY CHANGE MAIN USER PRESENT LAUNCHER	BaseBridge
query startActivity getAllNetworkInfo sendTextMessage java net HttpURLConnection getDeviceId MAIN LAUNCHER	BaseBridge
startService android webkit WebView getCellLocation notify getActiveNetworkInfo getSubscriberId BOOT COMPLETED MAIN HOME LAUNCHER	Opfake
start getAccounts vibrate requestLocationUpdates org apache http impl client DefaultHttpClient getDeviceId MAIN LAUNCHER	FakeTimer
query getActiveNetworkInfo vibrate getDeviceId java net HttpURLConnection startActivity MAIN PACKAGE ADDED LAUNCHER	GinMaster
getRunningTasks notify query startActivity getActiveNetworkInfo getConnectionInfo exec setWifiEnabled getCellLocation android webkit WebView BATTERY CHANGED ACTION BOOT COMPLETED MAIN SIG STR LAUNCHER	DroidKungFu
startActivity sendTextMessage android webkit WebView BOOT COMPLETED MAIN QUICKBOOT POWERON LAUNCHER	Opfake
restartPackage openOutputStream startActivity getConnectionInfo open requestLocationUpdates getLine Number java net HttpURLConnection MAIN DEFAULT LAUNCHER	JS/Exploit-DynSrc
startActivity onGeolocationPermissionsShowPrompt notify query getNetworkInfo startUsingNetworkFeature release sendTextMessage getDeviceId android webkit WebView BOOT COMPLETED MAIN LAUNCHER	Kmin
startActivity onGeolocationPermissionsShowPrompt notify query getNetworkInfo startUsingNetworkFeature release sendTextMessage getDeviceId android webkit WebView BOOT COMPLETED MAIN LAUNCHER	Kmin
notify startService getDeviceId java net Socket BOOT COMPLETED MAIN LAUNCHER	Luckycat
requestLocationUpdates startService listen org apache http impl client DefaultHttpClient BATTERY CHANGED BOOT COMPLETED MAIN LAUNCHER HOME	AccuTrack
notify query startService getNetworkInfo getLastKnownLocation getDeviceId openStream acquire BOOT COMPLETED	Plankton

HOW DOSE MALWARE WORKS?

- Malware usually attacks a device by fooling users into installing an application or clicking a link. When the user clicks or install the apps, the code are being executed and after that Malware works like:
- Duplicate or copy itself in different parts of the file system
- Installing unwanted software and slowing the system down considerably.
- locking access to Data, Apps or even the Device itself and forcing the user to make a payment to get access to data.
- Capture and attack essential systems of a device and rendering inappropriately.
- Most common trigger to execute is a click, typically on a link. Description for this kind of links are mostly like ‘Your System is Compromised’ or ‘Click to Win Prize’ etc.
- Malwares can be insert into Device by fake Legitimate applications like unzip file, pdf converter etc.

HOW DOSE MALWARE WORKS?



OBFUSCATION TECHNIQUES

- What is Obfuscation ?
- Code Obfuscation is a process where executables are being modified and it terminate hacking. It modifies metadata but output remain same. Basically it Manages to reverse engineer almost all the codes. Literally now days, with no time and effort all the codes can be reversed engineered using decompiles.

HOW MACHINE LEARNING HELPS TO DETECT MALWARE?

- Machine Learning uses algorithm to see patterns in the data to predict correctly. Basically it learns the pattern from the existing data and predicts upon some unknow data and by using this process a model is formed with the set of mathematical principals. Here we are predicting or detecting Malware/benign apps using Machine Learning.
- Supervised learning
- Supervised Machine Learning is a process of Machine Learning where we have both Independent and dependent(target) Variables and using this process we can solve both regression and classification problem statements.
- Unsupervised learning
- One of the machine learning process is unsupervised Machine learning. Unlike Supervised Machine Learning we don't Have any Target Variables to predict. We will only have independent features and using unsupervised machine learning we can predict clusters.
- Deep learning
- Deep Learning is an High-Level Machine Learning Process where we use neural networks (Just like how Human brain works) to solve more complicated problem statements like computer visions, object detections, speech detections etc.

DATA-SET FOR THE ANALYSIS?

- Here we get 3 DATA-SETS.
  1. Benign_Data.

     Benign Data-Set contains MD5 ids of Bengin Apps, API Calls, Intent, Permissions.

  1. Malicious_Data.

     Malicious Data-Set also contains MD5 ids of Bengin Apps, API Calls, Intent, Permissions.

  1. Family_Data.

     Family Data contains all MD5 Ids and Malware Family Names.

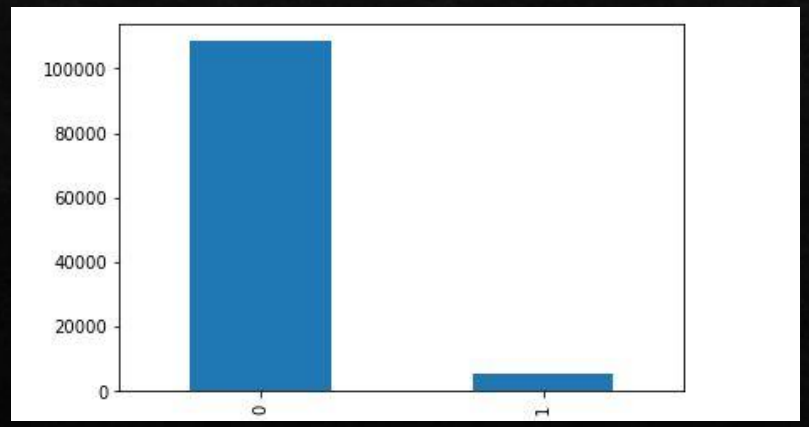
HOW TO APROCH MALWARE DETECTION USING ML?

- Importing Datasets
- Cleaning Datasets
- Merging and Labeling
- Feature Engineering
- Feature Extractions
- Data Preparation for Machine Learning
- Splitting Data into Test and Train sets
- Perform Machine Learning Algorithm to Solve Problem Statements

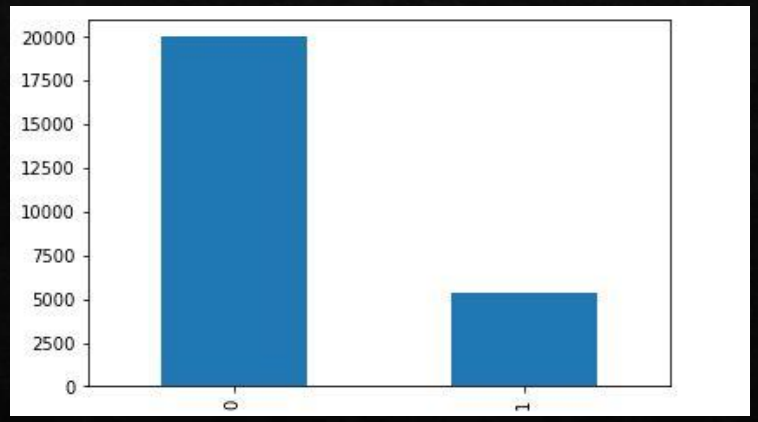
WHAT TYPE OF FEATURE ENGINEERING IS PERFORMED ON THE DATA SET AND WHY?

- There are 168 Number of Malware families presents in the Data-et.
- Problem Statement is to detect Malware/Benign. That's why I grouped all the Malware together and Assign 0 to them and Grouped all the Benign Together and assign 1 to them.
- Then I Checked the data by Plotting in a Bar chart and Find out that the data was imbalanced. That's Why I had to Perform Under Sampling Technique to Make The Data Balanced.

IMBALANCED DATA



BALANCED DATA



WHICH FEATURE EXTRACION IS USED ?

- Lemmatizations and CountVectorizer (Bag of Words) is used to Extract Features Out of The Datasets.

```
In [46]: Malware_Data.drop(columns='3', axis=1, inplace=True)

In [47]: Malware_Data['0'].str.lower()
         Malware_Data['1'].str.lower()
         Malware_Data['2'].str.lower()

Out[47]: 0                                    call_phone internet
         1                          access_network_state internet
         2                                              internet
         3        access_network_state internet write_external_s...
         4        access_coarse_location hardware_test internet ...
                                       ...
         25345                                         send_sms
         25346    disable_keyguard read_logs access_network_stat...
         25347    access_coarse_location access_fine_location ac...
         25348              access_network_state wake_lock internet
         25349    access_fine_location receive_boot_completed ac...
         Name: 2, Length: 25350, dtype: object

In [48]: Para = []
         for row in range(0,len(Malware_Data.index)):
             Para.append(' '.join(str(i) for i in Malware_Data.iloc[row, 0:2]))

In [49]: len(Para)

Out[49]: 25350
```

Performing Lemitization and Bag_of_Words for feature extraction

```
In [50]: Lemit = WordNetLemmatizer()

In [53]: Permission_Words = []
         corp = []
         for i in range(0, len(Para)):
             Malware_Permissions = re.sub('[^a-zA-Z]',' ', Para[i])
             Malware_Permissions = Malware_Permissions.split()
             Malware_Permissions = [Lemit.lemmatize(Word) for Word in Malware_Permissions if not Word in set(stopwords.words('english'))]
             Malware_Permissions = ' '.join(Malware_Permissions)
             corp.append(Malware_Permissions)
```

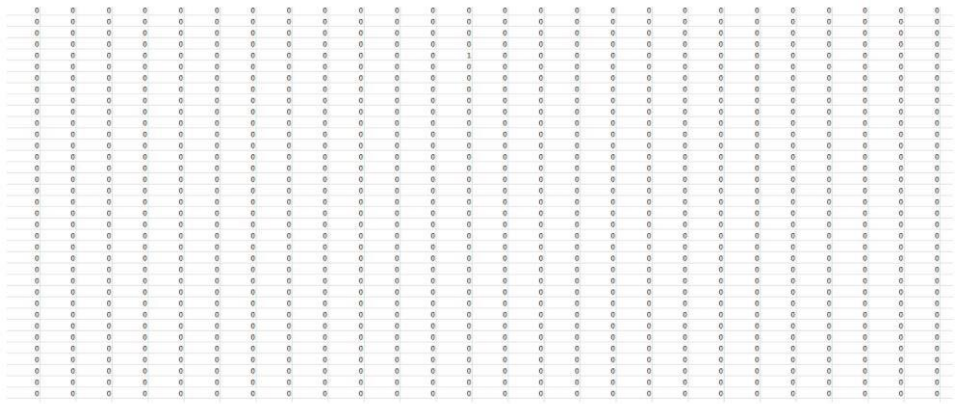
```
In [54]: corp[1]

Out[54]: 'startActivity getBestProvider start getActiveNetworkInfo android webkit WebView MAIN LAUNCHER'

In [54]: from sklearn.feature_extraction.text import CountVectorizer
         Cv= CountVectorizer(max_features= 4000,ngram_range=(1,1))

In [55]: X_all = Cv.fit_transform(corp).toarray()
```

Data for ML look like image below



WHY FETURE EXTRACTION IS NEEDED AND HOW MANY FEATURES ARE BEING EXTRACTED?

- Data-Set Contains Texts but Machine Learning Algorithm Doesn't except Inputs as Text that's Why Feature Extraction is Important, Because by applying Bag of Words I Can use meaningful words as Features.
- I've Got 1919 number of Features By applying Features Extraction.

```
In [57]: X_all.shape
Out[57]: (25350, 1919)
```

WHICH MACHINE LEARING ALGORITHM TO USE TO DETECT MALWARE?

- Bagging Techniques(Random Forest)

Hyper Parameter Tuning

Applying Machine Learning Algorithm

Performing Random Forest

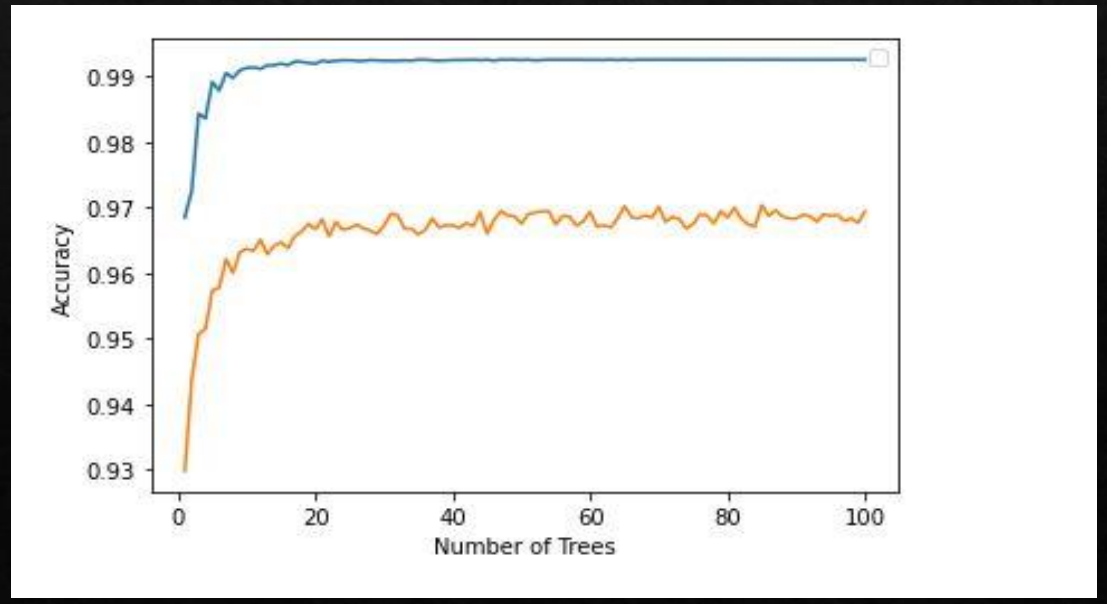
HyperParameter Tuning for getting optimal number of trees needed for this perticuler dataset

In [69]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
Estimators = np.arange(1,101,1)
Test_Score = []
Train_Score = []

for i in Estimators:
    Rf = RandomForestClassifier(n_estimators= i, criterion='gini')
    LRF = Rf.fit(X_Train, y_Train)
    Train_Prediction= LRF.predict(X_Train)
    Test_Prediction = LRF.predict(X_Test)
    F1_Train = metrics.f1_score(y_Train, Train_Prediction,average='weighted')
    F1_Test =  metrics.f1_score(y_Test, Test_Prediction,average='weighted')
    Test_Score.append(F1_Test)
    Train_Score.append(F1_Train)

from matplotlib.legend_handler import HandlerLine2D
import matplotlib.pyplot as Ploting
fig, Est_Sbplt = Ploting.subplots()
Est_Sbplt.plot(Estimators,Train_Score)
Est_Sbplt.plot(Estimators, Test_Score)
Est_Sbplt.set_xlabel('Number of Trees')
Est_Sbplt.set_ylabel('Accuracy')
Est_Sbplt.legend()
```

Optimal number of trees by Accuracy



Creating Model Using RandomForest

```
FRF = RandomForestClassifier(n_estimators= 20 , criterion='gini')
L_FRF = FRF.fit(X_Train, y_Train)
RF_PREDICTION = L_FRF.predict(X_Test)

print(metrics.classification_report(y_Test, RF_PREDICTION))
print(metrics.confusion_matrix(RF_PREDICTION,y_Test))
print('Accuracy Of Randomforest Model', round(metrics.f1_score(y_Test, RF_PREDICTION,average='weighted'),2))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6004
           1       0.97      0.87      0.92      1601

    accuracy                           0.97      7605
   macro avg       0.97      0.93      0.95      7605
weighted avg       0.97      0.97      0.97      7605

[[5956  201]
 [  48 1400]]
Accuracy Of Randomforest Model 0.97
```

◈ Using Boosting Techniques

Hyper Parameter Tuning

PERFORMING XGBoost Classifier

Hyper Parameter Tuning

```
params  = {'max_depth':[1,3,5,7,9,11,13,15],
           'learning_rate': [0.05,0.1,0.15,0.2,0.25,0.3,0.35],
           'gamma':[0.0,0.1,0.2,0.3,0.5,0.6,0.7,0.8,0.9],
          }
```

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
XGB_Clf = XGBClassifier()
randomSearch = RandomizedSearchCV(XGB_Clf,param_distributions= params, n_iter=10,scoring='f1_weighted',n_jobs=-1,cv =10,verbose=3
randomSearch.fit(X_PCA,Y_all)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 20.3min finished

[23:21:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.

RandomizedSearchCV(cv=10,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,
                                           n_estimators=100...
                                           num_parallel_tree=None,
                                           random_state=None, reg_alpha=None,
                                           reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   scoring='f1_weighted', verbose=3)
```

```
randomSearch.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.3, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=11,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Creating Model using XGBoost

```
from xgboost import XGBClassifier
XGB_Clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.3, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=11,
              min_child_weight=1, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)

XGB_Fits = XGB_Clf.fit(X_Train, y_Train)
XGB_Predictions = XGB_Fits.predict(X_Test)

from sklearn import metrics
print(metrics.classification_report(y_Test,XGB_Predictions))
print(metrics.confusion_matrix(XGB_Predictions,y_Test))

print('Accuracy Score is', metrics.f1_score(y_Test, XGB_Predictions,average='weighted' ))

from sklearn.model_selection import cross_val_score
Accuracy_Scores = cross_val_score(XGB_Clf, X_PCA,Y_all, cv =10, scoring='f1_weighted')
print('Cross Validation XGBOOST Accuarcy Scores', Accuracy_Scores)
print('Cross Validation Final XGBOOST Accuracy Score is', round(Accuracy_Scores.mean(),2))
```

```
[23:24:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6004
           1       0.95      0.90      0.93      1601

    accuracy                           0.97      7605
   macro avg       0.96      0.95      0.95      7605
weighted avg       0.97      0.97      0.97      7605

[[5933  155]
 [  71 1446]]
Accuracy Score is 0.9699856308567625
```

- Using SVM

Performing SVM

```
from sklearn.svm import SVC
Svm_Classifier = SVC(kernel='linear')
Svm_Fit = Svm_Classifier.fit(X_Train,y_Train)
```

```
from sklearn import metrics
Predict = Svm_Classifier.predict(X_Test)
```

```
print('Accuracy of Svm is', metrics.f1_score(y_Test,Predict,average='weighted'))
```

```
Accuracy of Svm is 0.9402421584434013
```

WHAT ARE THE OBSTACLES TO FORM THESE MODELS?

- During Feature Extractions I have got 1919 number of features. So I was needed to reduce dimensions to have a proper dimension for the input to get a good accuracy figure. That's why I performed PCA.

Applying PCA for Dimention Reduction

```
%matplotlib inline
from sklearn.decomposition import PCA
pca = PCA(n_components=1919)
pca_Fit = pca.fit(X_all)
Reduced_X  = pca_Fit.transform(X_all)
```

```
Varience_Explained = pca.explained_variance_ratio_
print(np.round(Varience_Explained,2))
```

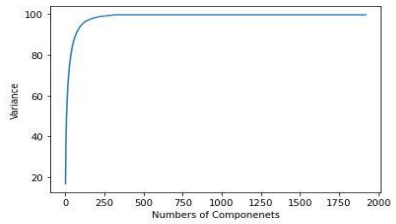
```
[0.17 0.08 0.07 ... 0.   0.   0.  ]
```

```
Cumulative_Varience = np.cumsum(np.round(pca.explained_variance_ratio_, decimals = 4)*100)
print(np.round(Cumulative_Varience,2))
```

```
[16.79 25.08 31.93 ... 99.57 99.57 99.57]
```

```
import matplotlib.pyplot as Ploting
Ploting.plot(range(1,1920), Cumulative_Varience)
Ploting.xlabel('Numbers of Componenets')
Ploting.ylabel('Variance')
```

```
Text(0, 0.5, 'Variance')
```




```
Pca_Final = PCA(n_components=120)
Pca_Fit = Pca_Final.fit(X_all)
Reduced_X_Final = Pca_Fit.transform(X_all)
```

```
Pca_Colums_Name = []
for i in range(0,120):
    Pca_Colums_Name.append(str(i))
```

```
Data_PCA = pd.DataFrame(Reduced_X_Final, columns=Pca_Colums_Name)
Data_PCA.head(n=5)
```

HOW PCA WORKS?

- Main Idea behind PCA is to reduce dimensions of a particular dataset.
- 1st step of PCA should be standardize the range of the continuous variables, that all the variables can contributes equally to the analysis.
- The motive behind the standardization of continuous variables is to terminate the possibilities of dominations of higher variance variables over lower variance variables.
- Mathematically it can be done by using Z-Score

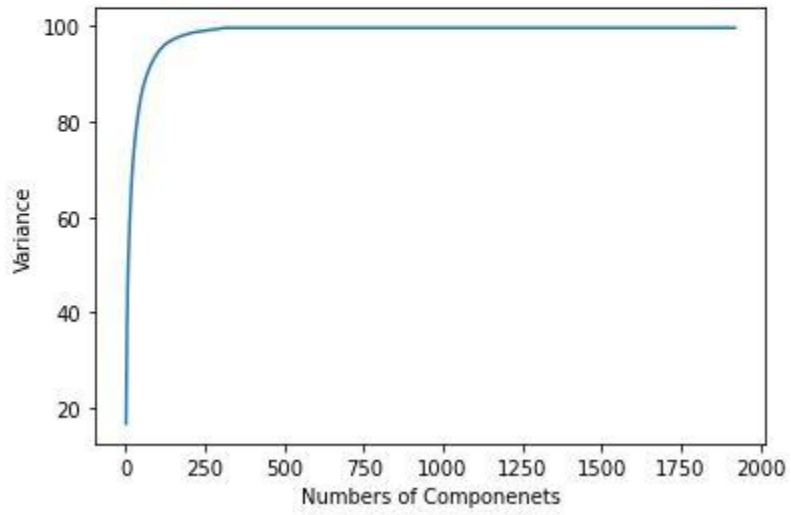
$$z = \frac{x - \bar{x}}{\sigma(x)}$$ [x = value, $\bar{x}$ = mean(x), $\sigma(x)$ = standard deviation (x)]

- Then matrix of covariance is being computed.
- Then Eigenvalues are computed.
- Then Eigenvalues of the Covariance Matrix identify Principal Components.

In this particular Analysis, i don't have any continuous variable instead we have all categorical Variables in a form of one hot encoded.
I've got 1919 features and after applying PCA we got 120 Principle Components explaining more Then 95 % of variance of the data and that's how applying PCA effectively reduces dimensions.

```
import matplotlib.pyplot as Ploting
Ploting.plot(range(1,1920), Cumulative_Varience)
Ploting.xlabel('Numbers of Componenets')
Ploting.ylabel('Variance')
```

```
Text(0, 0.5, 'Variance')
```



HOW RANDOM FOREST CLASSIFYING MALWARE?

- Random forest is a Supervised Machine Learning. It uses bagging techniques to train the data and it uses Decision Tree algorithm.
- It is used to solve classification Regression both problem statement.
- For detecting Malware it takes ‘k’ number of features from ‘m’ number of features where k<<m and form a decision tree and it repeats the process until reach ultimate number of iterations.
- Above we can see we uses n_estimators = 20 (n_estimators means number of trees). As after applying PCA my features reduced to 120, Random forest takes ‘k’ number of features from m= 120 features at a time to form a tree and these process repetes until it reach ultimate number of n_estimaters which is 20.
- Now after forming 20 trees it gets 20 different output for a particular sample in the dataset and then it goes with majority (just like voting) and gives output.
- Consider my output is 1 for a particular sample that means out of 20 at least 11 decision tree predicted it as 1. That’s why it is called bagging.

HOW XGBOOST CLASSIFIES MALWARE?

- XGBOOST is also a supervised Machine Learning Technique. It's also used in both classification and Regression use Cases.
- In Malware detection I have got X_all as independent features and Y_all as Dependent features (lebels).
- Unlike Randomforest, XGBoost dosent form full depth of decision trees, it can from trees with max_depth between 3-12.
- XGBoost tries to reduce the residuals by forming trees and this process goes until residuals becomes really low or reaches number of iterations.
- In this particular case of Malware Detection My XGBoost are forming 100 trees with max_depth 11 in order to classifie Malware.

WHICH MACHINE LEARING GIVES THE HIGHER ACCURACY?

- For Detecting MALWARE I've used 3 algorithm, which are as follow:
  - Random Forest Classifier (Gives 97 % Accuracy).

```
RFF = RandomForestClassifier(n_estimators= 20 , criterion='gini')
L_FRF = FRF.fit(X_Train, y_Train)
RF_PREDICTION = L_FRF.predict(X_Test)

print(metrics.classification_report(y_Test, RF_PREDICTION))
print(metrics.confusion_matrix(RF_PREDICTION,y_Test))
print('Accuracy Of Randomforest Model', round(metrics.f1_score(y_Test, RF_PREDICTION,average='weighted'),2))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6004
           1       0.97      0.87      0.92      1601

    accuracy                           0.97      7605
   macro avg       0.97      0.93      0.95      7605
weighted avg       0.97      0.97      0.97      7605

[[5956  201]
 [  48 1400]]
Accuracy Of Randomforest Model 0.97
```

  - XGBoost Classifier (Gives 97% Accuracy).

```
from xgboost import XGBClassifier
XGB_Clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.3, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=11,
              min_child_weight=1, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)

XGB_Fits = XGB_Clf.fit(X_Train, y_Train)
XGB_Predictions = XGB_Fits.predict(X_Test)

from sklearn import metrics
print(metrics.classification_report(y_Test,XGB_Predictions))
print(metrics.confusion_matrix(XGB_Predictions,y_Test))

print('Accuracy Score is', metrics.f1_score(y_Test, XGB_Predictions,average='weighted' ))

from sklearn.model_selection import cross_val_score
Accuracy_Scores = cross_val_score(XGB_Clf, X_PCA,Y_all, cv =10, scoring='f1_weighted')
print('Cross Validation XGBOOST Accuarcy Scores', Accuracy_Scores)
print('Cross Validation Final XGBOOST Accuracy Score is', round(Accuracy_Scores.mean(),2))
```

```
[23:24:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6004
           1       0.95      0.90      0.93      1601

    accuracy                           0.97      7605
   macro avg       0.96      0.95      0.95      7605
weighted avg       0.97      0.97      0.97      7605

[[5933  155]
 [  71 1446]]
Accuracy Score is 0.9699856308567625
```

  - SVM(Support Vector Machine) (Gives 94 % Accuracy).

Performing SVM

```
from sklearn.svm import SVC
Svm_Classifier = SVC(kernel='linear')
Svm_Fit = Svm_Classifier.fit(X_Train,y_Train)
```

```
from sklearn import metrics
Predict = Svm_Classifier.predict(X_Test)
```

```
print('Accuracy of Svm is', metrics.f1_score(y_Test,Predict,average='weighted'))
```

```
Accuracy of Svm is 0.9402421584434013
```

DISCUSSIONS

- Android Malwares are typically disguised consumer to insert into victim's Devices.
- It attacks important and most Signiant systems on device to paralyze it.
- Can be Disguised as Fake Legit Android Softwares.
- It works without consent of victim.
- It sends data to it's Remote server.
- By the Analysis in the Malware detection, we got some information and sequences of the Malware like I mentioned above. In order to get it in a prepare the data for Machine Learning inputs, I was needed to perform Text Mining ,PCA and different Algorithms to form a Model that can detect Malware/Benign.
- I was needed to perform Text Mining because Machine Learning Model doesn't accept texts to be input, it required Numbers that it can find a pattern inside.
- I was needed to perform PCA because the number of feature has huge.
- Out of 3 models I find XGBoost is more effective, However Random forest Model has given same accuracy as XGBoost but XGBoost works faster then the Random forest Model.

COCLUSIONS

- To avoid Malware to trick, consumers needs to be aware of any illegitimate urls or free software.
- In this Analysis we have detected malwares with 97% accuracy.
- Malware Detections Techniques can be varied upon the datasets we get.
- Ensemble Technique works really good for particular Analysis and Model forming.