# Host Based Intrusion Detection Systems : Techiniques, Datasets & Challenges

## Introduction

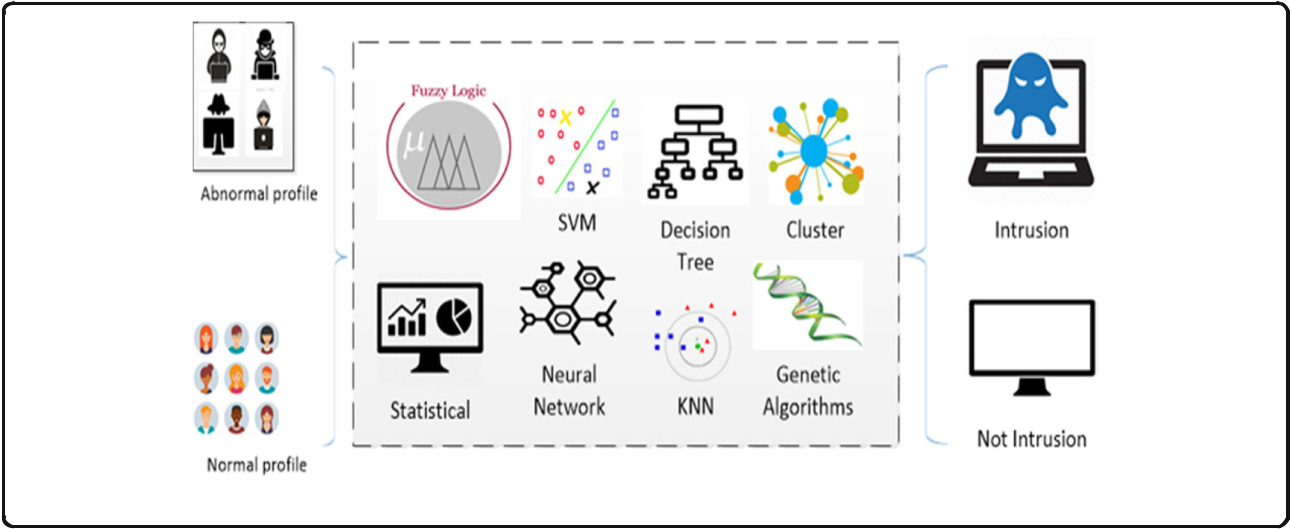
The aim of an IDS is to identify different kinds of malware as early as possible, which cannot be achieved by a traditional firewall. With the increasing volume of computer malware, the development of improved IDSs has become extremely important.

Table -1 Comparison of IDS technology types based on their positioning within the computer system

|  |  |  |  |
| --- | --- | --- | --- |
|  | Advantages | Disadvantages | Data source |
|  |  |  |  |
| Technology HIDS | • HIDS can check end-to-end encrypted | • Delays in reporting attacks | • Audits records, log files, Application Program |
|  | communications behaviour. | • Consumes host resources | Interface (API), rule patterns, system calls. |
|  | • No extra hardware required. | • Needs to be installed on each |  |
|  | • Detects intrusions by checking hosts file | host. |  |
|  | system, system calls or network events. | • It can monitor attacks only on |  |
|  | • Every packet is reassembled | the machine where it is installed. |  |
|  | • Looks at the entire item, not streams only |  |  |
| NIDS | •Detects attacks by checking network | •Challenge is to identify attacks | •Simple Network Management Protocol |
|  | packets. | from encrypted traffic. | (SNMP) |
|  | •Not required to install on each host. | •Dedicated hardware is required. | •Network packets (TCP/UDP/ICMP), |
|  | •Can check various hosts at the same | •It supports only identification of | •Management Information Base (MIB) |
|  | period. | network attacks. | •Router NetFlow records |
|  | •Capable of detecting the broadest ranges | •Difficult to analysis high-speed |  |
|  | of network protocols | network. |  |
|  |  | •The most serious threat is the |  |
|  |  | insider attack. |  |
|  |  |  |  |

The focus of our analysis will be to build an **HIDS system using systemcall data** and machine learning techniques.

We will explore Supervised learning-based IDS techniques to detect intrusions by using labeled training data. A supervised learning approach usually consists of two stages, namely training and testing. In the training stage, relevant fea-tures and classes are identified and then the algorithm learns from these data samples. In supervised learning IDS, each record is a pair, containing a network or host data source and an associated output value (i.e., label), namely intrusion or normal. Next, feature selection can be applied for eliminating unnecessary features. Using the training data for selected features, a supervised learning technique is then used to train a classifier to learn the inherent relationship that exists between the input data and the labelled output value. The resultant classifier then becomes a model which, given a set of feature values, predicts the class to which the input data might belong.



## Performance metrics for HIDS

HIDS are typically evaluated based on the following standard performance measures for a two-class classifier:

Table-2 Confusion Matrix for IDS System

Actual Class Predicted Class

|  |  |  |
| --- | --- | --- |
| Class | Normal | Attack |
| Normal | True negative (TN) | False Positive (FP) |
| Attack | False Negative (FN) | True positive (TP) |
|  |  |  |

**True Positive Rate (TPR):** It is calculated as the ratio between the number of correctly predicted attacks and the total number of attacks. If all intrusions are detected then the TPR is 1 which is extremely rare for an IDS. TPR is also called a Detection Rate (DR) or the Sensitivity. The TPR can be expressed mathematically as TPR = TP /TP + FN

**False Positive Rate (FPR):** It is calculated as the ratio between the number of normal instances incorrectly classified as an attack and the total number of normal instances.

FPR= FP/FP+TN

**False Negative Rate (FNR):** False negative means when a detector fails to identify an anomaly and classifies it as normal. The FNR can be expressed mathematically as:

FNR = FN/FN +TP

**Classification rate (CR) or Accuracy:** The CR measures how accurate the IDS is in detecting normal or anomalous traffic behavior. It is described as the percentage of all those correctly predicted instances to all instances:

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

**Receiver Operating Characteristic (ROC) curve:** ROC has FPR on the x-axis and TPR on the y-axis.A test with perfect discrimination (no overlap in the two distributions) has a ROC curve that passes through the upper left corner (100% sensitivity, 100% specificity).

## Dataset Details

ADFA-LD dataset created by Australian Defence Force Academy created for evaluation of system-call-based HIDS

<https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/>

Table-3 Number of system calls traces in different categories of AFDA-LD and AFDA-WD

|  |  |  |
| --- | --- | --- |
| ADFA- LD |  |  |
|  |  |  |
| Dataset | Traces | System Calls |
| Training data | 833 | 308,077 |
| Validation data | 4372 | 2,122,085 |
| Attack data | 746 | 317,388 |
| Total | 5951 | 2,747,550 |

Table-4 ADFA-LD attack class

|  |  |  |  |
| --- | --- | --- | --- |
| Attack | Payload | Vector | Count |
|  |  |  |  |
| Hydra-FTP | Password brute force | FTP by Hydra | 162 |
| Hydra-SSH | Password brute force | SSH Hydra | 176 |
| Adduser | Add new super user | Client-side poisoned executable | 91 |
| Java-Meterpreter | Java based Meterpreter | TIkiWiki vulnerability exploit | 124 |
| Meterpreter | Linux Meterpreter Payload | Client side poisoned executable | 75 |
| Webshell | C100 Webshell | PHP remote file inclusion vulnerability | 118 |
|  |  |  |  |

## Machine Learning Techinques Applied

### Data Preprocessing

1) Split the Attack data of each category (Hydra-FTP, Hydra-SSH, Adduser, Java-Meterpreter, Meterpreter and Webshell ) into 70% training data and 30 % test data. For instance there are are 10 folders in “Adduser” attack. Therefore, 7 of these folders are to be used for training and 3 folders are to be used for testing.

2) For the Normal data, files in “Training\_Data\_Master” folder are to be used as training data and files in “Validation\_Data\_Master” folder are to be used as test data.

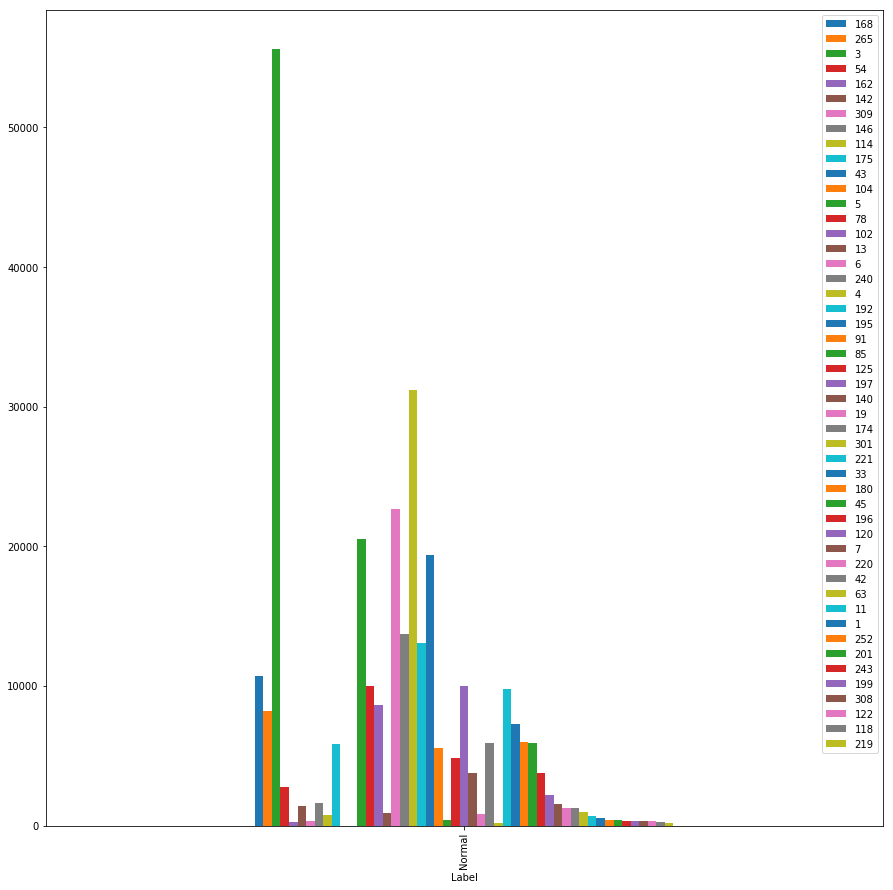
3) Write a python script to find the frequency of occurences of all unique 1-grams, 2-grams and 3-grams system call sequences in the training data for both Attack data (across all categories of attack) and Normal data.

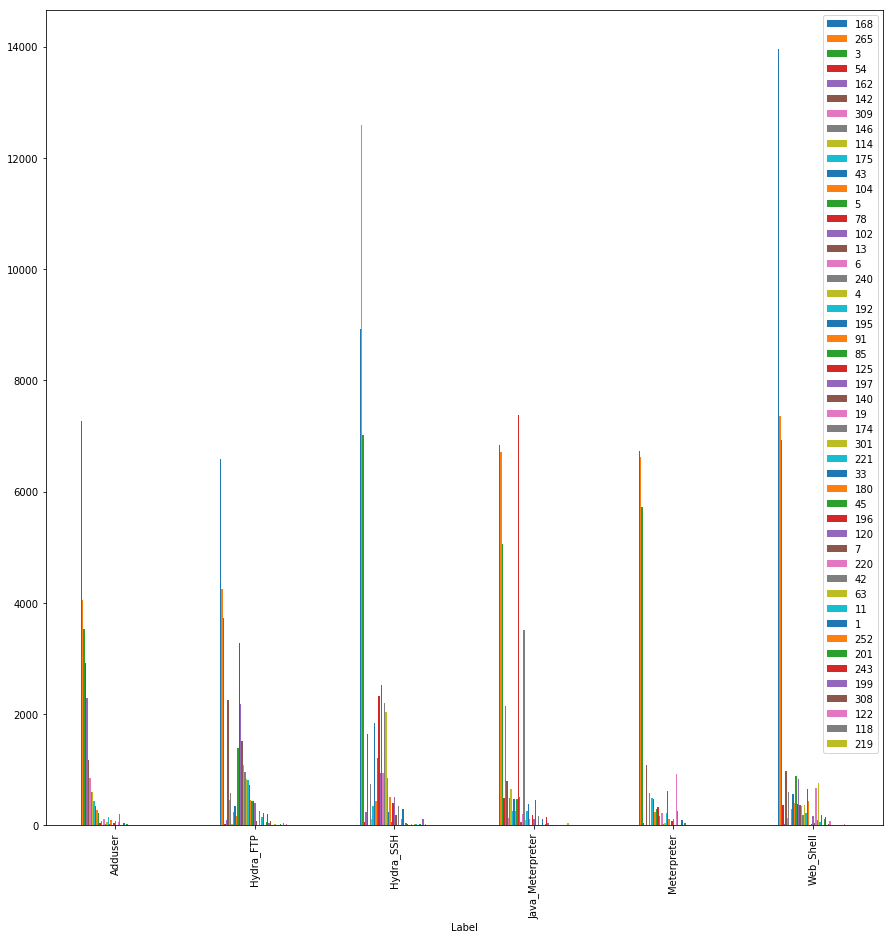
4) Perform the same task on files in the “Training\_Data\_Master” to obtain all the unique 1- grams, 2-grams and 3-grams.

5) Once we have obtained the frequencies of all the unique n-grams terms in the training data, use the top 30% n-grams terms with the highest frequency to create a data set.

6) Apply the same procedure to generate the test dataset from the test files of the attack data (for all attack types) and the normal files in the “Validation\_Data\_Master” using the top 30% 3-grams terms with highest frequencies obtained during the training phase. The classifier model developed during the training phase will finally be validated on the Test dataset.

### EDA : Top 50 n-grams for 1/2/3 were plotted

For Normal System Call 1-gram distribution is seen in this graph



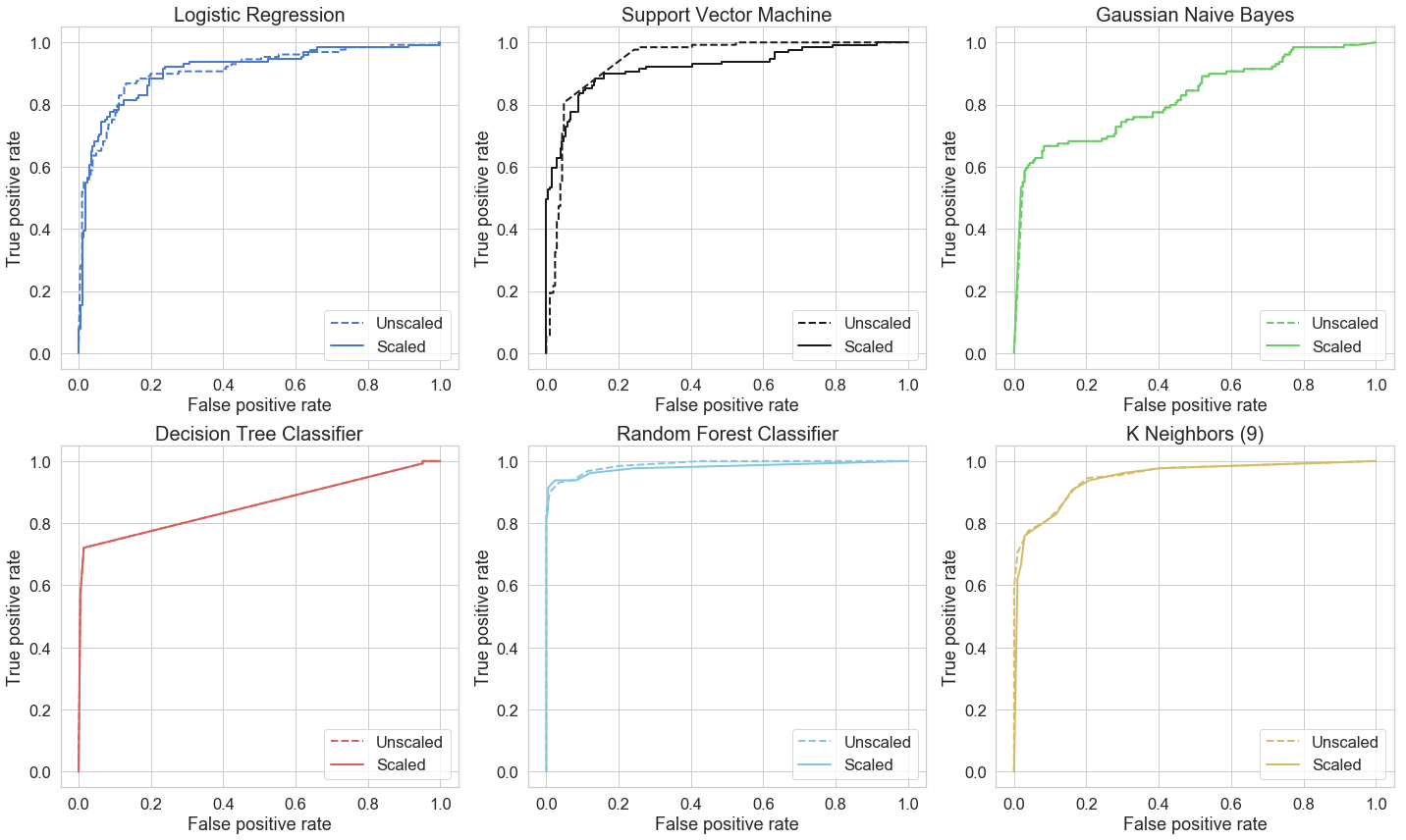
For Attacks System Call 1-gram distribution is seen in this graph

### Model Comparison

### Supervised Models

On the preprossed data supervised learning modelling was done the results observed are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accuracy** | **Precision** | **Recall** | **F1** |
| **LogisticReg** | 0.844776 | 0.834783 | 0.744186 | 0.786885 |
| **SVMC** | 0.683582 | 0.925926 | 0.193798 | 0.320513 |
| **GaussNB** | 0.811940 | 0.934211 | 0.550388 | 0.692683 |
| **DecisionTree** | 0.886567 | 0.978947 | 0.720930 | 0.826667 |
| **RandomForest** | 0.958209 | 0.983193 | 0.875969 | 0.930041 |
| **kNN9** | 0.865672 | 0.818182 | 0.837209 | 0.827586 |



We further improved Random Forest Model using Grid Search validation & obtained following results:

|  |
| --- |
| Final Model Parameters:  {'bootstrap': True,  'class\_weight': None,  'criterion': 'gini',  'max\_depth': 80,  'max\_features': 3,  'max\_leaf\_nodes': None,  'min\_impurity\_decrease': 0.0,  'min\_impurity\_split': None,  'min\_samples\_leaf': 3,  'min\_samples\_split': 8,  'min\_weight\_fraction\_leaf': 0.0,  'n\_estimators': 300,  'n\_jobs': None,  'oob\_score': False,  'random\_state': 42,  'verbose': 0,  'warm\_start': False}  Accuracy: 0.91 |

### Deep Learning Models - Deep Autoencoder

We used Deep Autoencoder model to evaluate how Deep Learning techiques will perform on this dataset.

Building the model Our Autoencoder uses 4 fully connected layers with 14, 7, 7 and 29 neurons respectively. The first two layers are used for our encoder, the last two go for the decoder. Additionally, L1 regularization will be used during training :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_6 (InputLayer) (None, 87) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_22 (Dense) (None, 87) 7656

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_23 (Dense) (None, 43) 3784

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_24 (Dense) (None, 43) 1892

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_25 (Dense) (None, 87) 3828

=================================================================

Total params: 17,160

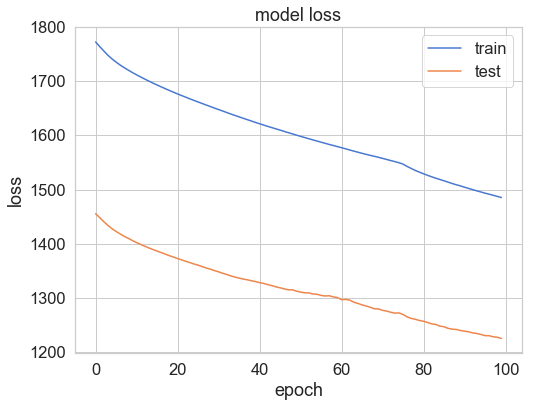
Trainable params: 17,160

Non-trainable params: 0

Epoch 100/100

662/662 [==============================] - 0s 115us/step - loss: 1485.2451 - acc: 0.6360 - val\_loss: 1225.0467 - val\_acc: 0.6157

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



## 

threshold = 0.2

## 

Obtained accuracy with Deep Autoencoder **63%.**

We've created a very simple Deep Autoencoder in Keras that can reconstruct what normal system call looks like. Initially we gave a lot of one-class examples (normal systemcalls) to a model and it learned (somewhat) how to discriminate whether or not new examples belong to that same class. Our model seems to catch a lot of the attack cases.

### LSTM For SystemCall Prediction

We also attempted to use simple LSTM for system call predictions, the results are summarized below.

Data dimention : (20450(n\_samples), 19 (time-step), 341(unique calls))

ADFA Dataset Preprocessing:

1) The system call language model estimates the probability distribution of the next call in a sequence given the sequence of previous calls.

2) We assume that the host system generates a finite number of system calls.

3) We index each system call by using an integer starting from 1 and denote the fixed set of all possible system calls in the system as S = {1, · · · , K}. Let x = x1x2 · · · xl(xi ∈ S) denote a sequence of l system calls.

LSTM Based Model :

1) At the Input Layer, the call at each time step xi is fed into the model in the form of one-hot encoding,

in other words, a K dimensional vector with all elements zero except position xi.

2) At the Embedding Layer\*, incoming calls are embedded to continuous space by multiplying embedding matrix W,

which should be learned.

3) At the Hidden Layer\*, the LSTM unit has an internal state, and this state is updated recurrently at each time step.

4) At the Output Layer, a softmax activation function is used to produce the estimation of normalized probability values of possible calls coming next in the sequence.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_28 (LSTM) (None, 100) 176800

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_16 (Dense) (None, 341) 34441

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_10 (Activation) (None, 341) 0

=================================================================

Total params: 211,241

Trainable params: 211,241

Non-trainable params: 0

Epoch 10/10

20298/20298 [==============================] - 19s 928us/step - loss: 0.0049 - acc: 0.9983 - val\_loss: 0.0083 - val\_acc: 0.9978

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## LSTM Binary Classifier

## Model Summary

* 1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_41 (LSTM) (None, 13) 780

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_59 (Dense) (None, 1) 14

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_60 (Dense) (None, 1) 2

=================================================================

Total params: 796

Trainable params: 796

Non-trainable params: 0

**Model Performance Summary/Plots**

precision recall f1-score support

0 0.91 0.94 0.92 39123

1 0.20 0.15 0.17 4111

micro avg 0.86 0.86 0.86 43234

macro avg 0.56 0.54 0.55 43234

weighted avg 0.85 0.86 0.85 43234

**Test Data performace**

Accuracy : 0.862261

Precision : 0.880766

Recall : 0.862261

F1 score : 0.871119

ROC AUC : 0.601735

## 

## 

## Conclusion

HIDS is complex dataset ,pre-processing step is crucial to understand and then use various ML & DL to build models for same. Though ADFA-LD dataset contains many new attacks, it is not adequate.As normal activities are frequently changing and may not remain effective over time, there exists a need for newer and more comprehensive datasets that contain wide-spectrum of malware activities.

## References

LSTM-BASED SYSTEM-CALL LANGUAGE MODELING AND ROBUST ENSEMBLE METHOD FOR DESIGNING HOST-BASED INTRUSION DETECTION SYSTEMS <https://arxiv.org/pdf/1611.01726.pdf>

Idea: char-based system call

<https://github.com/karpathy/char-rnn> -char NN

padding the sequence

<https://stackoverflow.com/questions/42002717/how-should-we-pad-text-sequence-in-keras-using-pad-sequences>

<https://github.com/fchollet/keras/issues/1641>

loss functions : [https://keras.io/losses/#categorical\_crossentropy](https://keras.io/losses/" \l "categorical_crossentropy)