Attention Based Host Intrusion Detection System

Irtesam Mahmud Khan
ID: 0421052039
Department of CSE, BUET
Dhaka, Bangladesh
irtesam.m.khan@gmail.com

***Abstract*—With the development of deep learning, various method have been adopted in Host Intrusion Detection System or HIDS. However, the traditional methods of HIDS have been proven to be vulnerable to higher number of false alarm. In this study, we have proposed a novel hierarchical attention based deep learning method of detection intrusion on a host. We have evaluated our model on ADLF-LD, which is a collection of a trace data of Linux system calls. We have tuned our model's hyper parameters to produce the optimum result, and our method has successfully outperforms the existing methods.**

***Index Terms*—HIDS, Attention, Neural Networks**

I. Introduction

The host-based intrusion detection system (HIDS) has been gaining attention in the community of cybersecurity for over two decades. Compared with the network-based intrusion detection system (NIDS), HIDS has the superiorities of fine granularity and the ability to detect internal attacks. HIDS analyzes auditing data from operating systems, whereas NIDS analyzes data from network traffic. System-call-based HIDS is about analyzing collected Linux system call traces [10]. HIDS monitors activities such as system or shell logs within hosts to discover unauthorized behaviors. HIDS can perform various kinds of data mining methods such as artificial neural networks on host auditing data to discover attacks. In Unix like operating systems, system calls are referred when a kernel service from the operating system is requested by a running process. System calls are significant interactions between programs and the system kernel. System-call-based HIDS has gained attention in the past 20 years because of the increasing number of attacks focusing on Linux servers. System-call based HIDS has been developed for intrusion detection in virtual hosts and embedded platforms such as smartphones [10]. A system-call based HIDS monitors real-time system call traces to detect abnormal system call sequences. System-call based HIDS can trigger alarms when abnormal system call traces are detected from normal traces. By using system calls as the input data, an HIDS can manipulate the most original information of an operating system [10]. Analyzing system-call traces which is invoked by the running programs to discriminate normal and abnormal behaviors was first introduced by Forrest et al. [7], [11]. Due to the complexity and diversity of user behavior, the previous abnormal intrusion detection method for constructing normal behavior patterns is very unstable. Then in their subsequent works, they defined normal profile using short sequences of system calls [11]. Kim et al. [8] proposed an ensemble method combining a number of LSTM classifiers to detect malicious system call sequences. Chawla et al. [3] proposed a combined architecture of CNN and RNN with pretraining of the CNN layer to reduce the training time significantly. Shaohua et al. [11] introduced a sequence to sequence model to predict future system calls and used the generated sequences to detect attack sequences. Diep et al. [6] proposed a combination of multi-channel CNN and LSTM which can predict attack sequences with remarkable precision. Shuaichuang et al. [13] proposed an attention-LSTM model which enables them to give varying weights to the system calls based on their relative importance for predicting correctly the label of the sequence. Recent advancement in attention mechanism has improved the performance of NLP models significantly. But we see very few efforts made to tackle the problem of HIDS by using the attention mechanism. Attention enables us to apply different weights to different inputs at each time step for predicting outputs. Our objective of this work is to propose a hierarchical attention based GRU model that can efficiently detect attack system calls based on the relative importance of the system calls. We have splitted the system call sequences into sentences and applied attention firstly on the system calls, secondly on the sentences to extract their relative importance with regard to predicting the label of the sequence. Results show that our method can efficiently detect the attack system calls and at the same time, does not generate too may false positives. Our proposed approach will help detect malicious system calls in the host machine. Before that, a dataset must be provided containing the possible attacks and normal sequences that occur in the host machine. Getting trained on that dataset, our approach will be able to detect any upcoming attacks and warn the owner accordingly. The unique features of our study are:

- Applying a new method Hierarchical Attention to detect intrusion
- Training the model on benchmark dataset for comparison with other existing methods
- Using Attention mechanism for increasing explainability

II. Background

In this section first we describe a brief technical overview of host intrusion detection systems and associated methods. After that, we describe some related works with their limitations. Finally, we give a overview of how we plan the remedy of those limitations and hence derive the novelty of our work.

A. Technical Background

In this subsection, first, we define the anomaly based host intrusion detection system and after that, we go through a brief overview of various deep learning networks that are widely used in detection intrusion in host based system. The following subsection describes some relevant existing works on host intrusion detection and analyze their limitations and our plan to resolve them.

1) *Anomaly based Host Intrusion Detection (HIDS)* An Intrusion Detection System (IDS) refers to hardware/software platform to monitor activities to detect malicious signs. There can be two types of IDS: i) Host based IDS (HIDS) and ii) Network based IDS (NIDS). An anomaly based HIDS is an intrusion detection system which detects any kind of computer intrusions and misuse by monitoring the execution activity of the system and classifying them as either normal or anomalous. There are multiple approaches of detecting anomalies and the most recent and widely adopted approach is to use a deep learning based techniques and train the system to with pre-labeled normal and anomalous activities so that it can classify any forthcoming system activity. The upcoming portion of the subsection the deep learning techniques that have been used in the state-of-the-art HIDS models.
2) *Recurrent Neural Networks(RNN):* Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. The typical structure of RNN is shown in Figure 1. In order to use system call sequence to detect anomalies,RNN needs to be trained with the training dataset. Each system call can be considered as a word and a system call sequence can be considered as a sentence. In this way, the intrusion detection problem can be solved through the techniques of Natural Language Processing (NLP) and RNNs can be a very good candidate in this regards since it can process the input keeping the model size constant Computation and most importantly, it takes into account historical information which is mandatory for NLP.

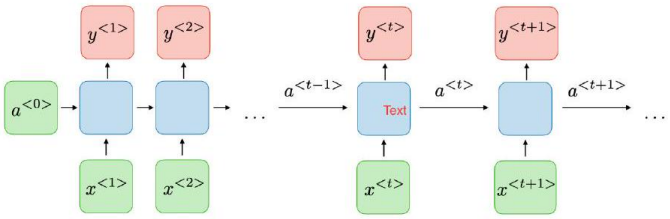

Fig. 1. Architecture of a typical RNN

3) *LSTM and GRU:* RNNs suffer from short term memory. If the input is too long, gradient values vanish during back propagation and as a result update of weights becomes insignificant. This is called Vanishing gradient problem. To overcome this issue, LSTM was created as a solution. In Fig. 2, architecture of an LSTM is shown. We can see that an LSTM has three types of gates. Their details are described as follows.

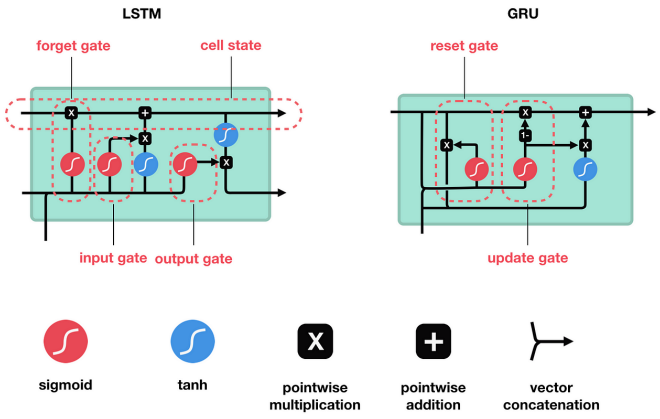

Fig. 2. Architecture of LSTM and GRU

a) **Forget Gate** This gate decides which information to throw away or kept. This gate is responsible for carrying important information from past states and pass it to the next states.
b) **Input Gate** This gate takes the current input and previous hidden state as input and decides what information to be kept.
c) **Output Gate** This gate decidese what the next hidden state should be based on the current input and previous hidden states.

The architecture of GRU is shown in Fig. 2. It is a simpler version of LSTM. It gets rid of cell states and uses hidden states to pass information. It has only two gates.

a) **Update Gate** It acts similar to the forget and input gates of an LSTM. It decides what information to throw away and what new information to add.
b) **Reset Gate** It is another gate to decide how much past information to forget.

4) *Attention Mechanism* The attention mechanism emerged as an improvement over the encoder decoder-based neural machine translation system in natural language processing (NLP). Bahdanau et al. [1] proposed the first attention model in 2015. Before that, LSTMs/GRUs were being used in encoder-decoder models but their performance were not satisfactory when input sentences were large. Also, another problem was that there was no way to give more importance to some of the input words than others while translating the sentence. Fig. 3 demostrates the attention mechanism proposed by Bahdanau et al. [1]. In their work, they used bidirectional LSTM layer, that is why we see two sets of hidden states, one for the forward direction and the other for

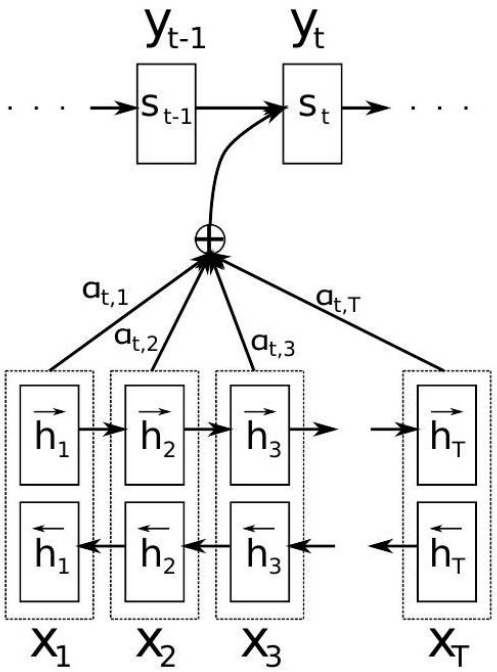

Fig. 3. First Attention Model

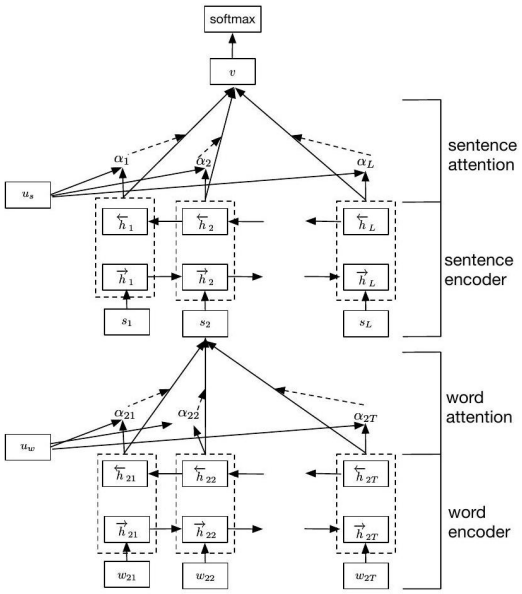

Fig. 4. Hierarchical Attention Model

the backward direction. All the hidden states are used to compute a context vector which is then used to predict the output at current time step. Each hidden state is given a weight which represents the relative importance of that input for predicting the current output.

Hierarchical attention was first proposed by Yang et al. [14]. Their proposed model is demonstrated in Fig. 4. At first phase, words are fed into a bidirectional GRU layer and the outputs of the GRU layer are fed into an attention layer to compute the relative importance of words inside a sentence. At the second phase, sentences are fed into another bidirectional GRU layer, and the outputs of the GRU layer is are fed into another attention layer. This layer gives us the relative importance of each of the sentence of a document which is dependent on the importances of the words of that sentence computed in the first phase.

B. Existing Works

With the current popularity of deep learning, there have been a good number of deep learning based host intrusion detection systems. The performance of any deep learning based model largely depends on the availability of suitable datasets. During decade of 1990, Knowledge Discovery in Databases (KDDCup1999, KDDCup98) were largely used in the evaluation of any intusion detection system. However, in 2013, Creech et al. [4] claimed that these datasets currently sufficiently represent relevant architecture or contemporary attack protocols, and therefore, evaluating any intrusion detection system using these datasets does not provide an effective performance metric, and contributes to erroneous efficacy claims. Creech et al. in his article, designed a new contemporary dataset for evaluating host based intrusion detection systems, and his ADFA Linux Data (ADFA-LD) [5] is mainly a collection of Linux system call sequences of both normal and malicious categories. Kim et al. [8] proposed a system call language-modeling approach for designing an anomaly based host intrusion detection systems. They also designed a novel ensemble method by blending multiple thresholding classifiers into a single one in order to overcome the issue of high false alarm rates which is very common in conventional methods. There are some significant advantage of such system-call language modeling such as it can learn the semantic meaning and interactions of each system call that the traditional methods cannot consider. For benchmarking, they used both the ADLF-LD an KDD98 datasets. They changed the LSTM layer parameters and designed three independent language models; i) one layer with 200 cells, ii) one layer with 400 cells, and iii) two layers with 400 cells. In their experiment, the final one provided the best result. They further improved the performance by using their ensemble method (AUC 0.928) than that of the averaging (AUC 0.890) and the voting (AUC 0.859) ensemble methods. But they did not provide any detailed explanation about how they computed their ROC curve, what their training set was and on which data they computed their model's performance. Chawla et al. [3] proposed an anomaly based intrusion detection system based on a combined architecture of RNN and CNN. They used Gated Recurrent Units (GRU) instead of the normal LSTM networks and provided a reduced training time. For evaluating their approach, they built five independent models; i) one layer with 200 GRU units ii) one layer with 200 LSTM units iii) Six layered 1D CNN with 200 GRU units iv) Seven layered 1D CNN with 500 GRU units v) Eight layered 1D CNN with 600

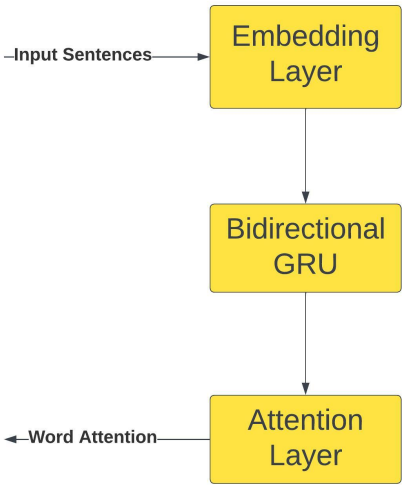

Fig. 5. Block Diagram of the word attention model

GRU units. They train each of the models with 833 training sequences of ADLF-LD dataset. The final model provided the best performance with AUC value of 0.81. Their significant contribution was to reduce the training time. Their CNN-GRU model needed only 10 training epochs to reach the convergence while the LSTM model needed 100 epochs resulting into even a lower AUC value of 0.74. Shaohua et al. [11] introduced a sequence-to-sequence model using RNN. Their prediction model predicts the upcoming system call sequence and this sequence will be used to predict any attack. They run their experiments on ADFA-LD dataset and demonstrated convincing performance in anomaly prediction. They have proved that that their model is very capable of predicting the most likely system call sequence with a high accuracy. They evaluated their predicted sequence using various classifiers: CNNs, RNNs, SVM and Random Forest and they have gained very promising AUC values varying between 0.91 to 0.95. There has been a very recent work of Shuaichuang et al. [13] where they have proposed an Attention-LSTM based network intrusion detection system. The use of attention mechanism has enabled them to pay attention to special features of the data. They compared the performance of their model with the conventional CNN and RNN and showed that their model improves the accuracy and precision rate of network intrusion detection but also decreases the false alarm rate. Diep et al. [6] proposed a combinational model of multichannel CNN and bidirectinal LSTM to detect anomaly in host-based intrusion detection systems. They performed 10- fold cross-validation on ADFA-LD dataset and achieved an accuracy of 97

III. Methods

In this section, we explain our approach towards detecting malicious system call sequences.

A. Sytem Architecture

The model architecture is shown in Fig. 5 and Fig. 6. Fig. 6 is our proposed model and it has a block named Word Attention Model which is elaborated in Fig. 5. The Word Attention Model in Fig. 5 takes sentences as input and Embedding layer converts each word of the sentence into a real-valued vector. These vectors are fed into the Bidirectional GRU layer. Output of the GRU layer is given as input to the attention layer which produces attention values of the words inside the sentence.

Our proposed model in Fig. 6 takes system call sequences as input and applies a Time Distributed layer on the input. Time Distributed layer is a keras wrapper which allows Keras to apply a layer to each temporal slice of an input. In our case, the time distributed layer is allowing us to apply the Word Attention Model to every sentence of any system call sequence. After the Word Attention Model, we again apply bi-directional GRU layer and attention layer. After that, a dense layer with 1 neuron is placed which will give us the probability of the system call sequence of being malicious.

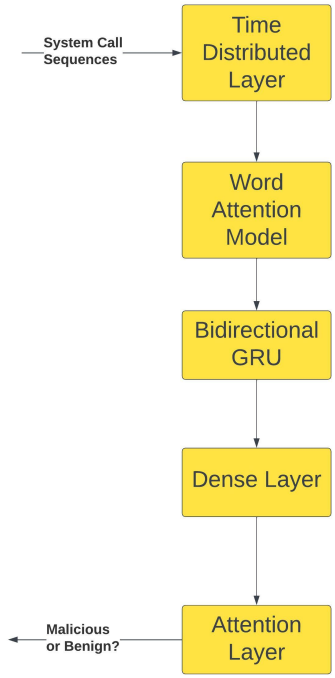

Fig. 6. Model

B. Workflow

Fig. 7 demonstrates the flow chart of our proposed approach. The details are described as follows.

Tokenizing Sytem Calls In the dataset that we are using, the system calls are represented as integers. But numbering doesn't start from 1. Also, numbering is not sequential. Tokenizing means to convert each number into a sequential number starting from 1. This helps us

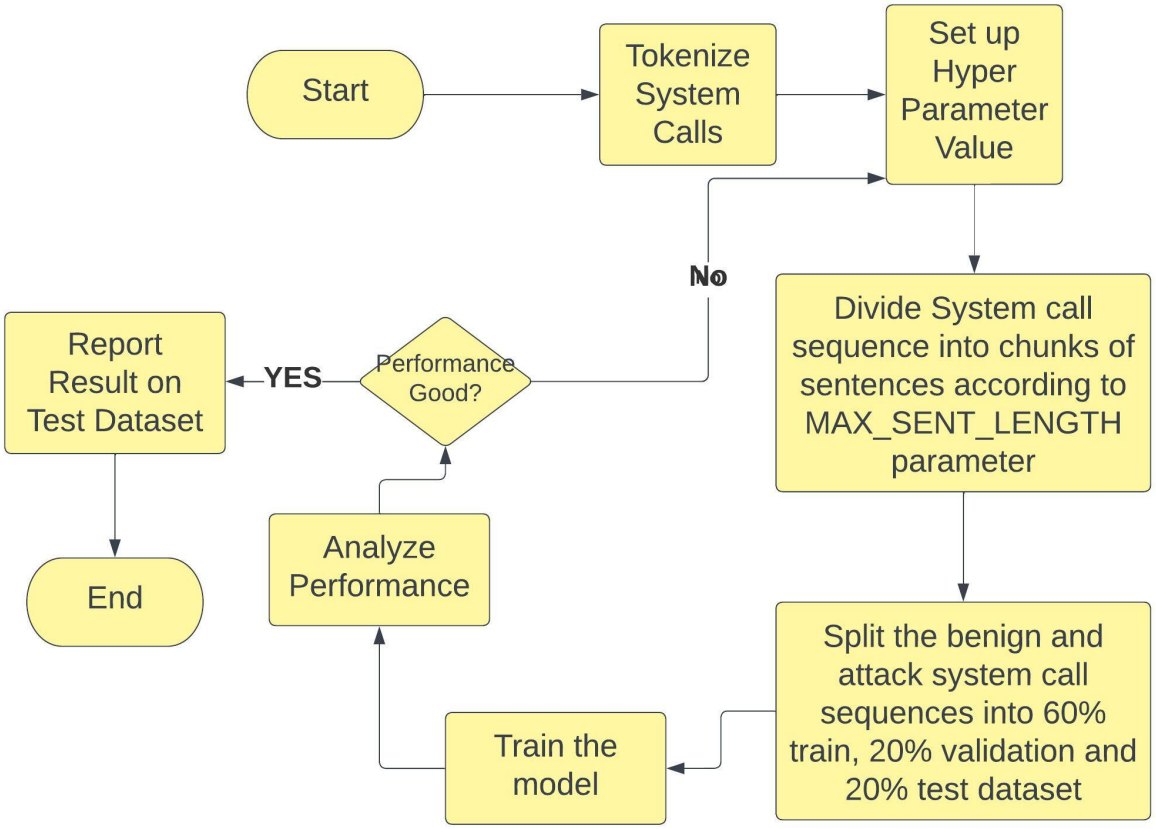

Fig. 7. Workflow

to efficiently prepare the input data that will be fed to the model.

Tuning hyper parameters There are a number of hyper parameters in our model. We have to set up their values before starting the training phase.

Dividing each sequence into chunks of sentences As we will be applying a hierarchical attention model on the system call sequences where each sequence has a label, these sequences need to be split into sentences so that two level of attention layers can be applied on them. To do that, we need to decide what should be the number of words in each of the sentences. We call this hyper parameter MAX SENT LENGTH, i.e., maximum length of any sentence.

Splitting the dataset We split the dataset into 60% training, 20% validation and 20% test dataset. While splitting, we ensure to maintain the proportion of benign and malicious system calls.

Training the model The model is trained with the training dataset. The procedure is elaborated later.

Analyzing performance The trained model is used to predict labels of system call sequences of the validation dataset. Hyper parameters are tuned based on the performance and the whole process from Dividing each sequence into chunks of sentences is repeated.

Reporting result on test set After obtaining a model performing well in the validation dataset, this model is used to predict labels of the instances in the test dataset and its results are reported.

IV. Results

We first describe different parts of the results and then show the numeric values in this section.

A. Dataset

The effectiveness of any machine learning model significantly depends on the availability of datasets. In general, the more data we can provide to a machine learning model, the more the model can learn the pattern of data and improve performance. In case of intrusion detection system, it is highly expected that the dataset should contain the contemporary attacks so that the model can be more practical. All the intrusion detection systems designed between 1990 and 2010 used the datasets from Knowledge Discovery in Databases (KDD98 [9] KDD99 [12]). However, with the introduction of new novel attacks and also due to natural againg process, these datasets have lost most of their relevance in intrusion detection system. In 2013, Creech [4], [5] brought up the

limitations of KDD datasets and generated a new dataset called ADFA-LD, mainly targetting the development of efficeint host intrusion detection system. This ADFA-LD dataset contains traces of system calls of Linux based operating system Ubuntu 11.04. The traces have been captured during the execution of normal processes and also during the exposure to some common known attacks. The ADFA-LD datast contains:

- 833 normal system call traces for training
- 4372 normal system call traces for validation
- 746 system call traces for six attack vectors. Table

Table I shows the description of the attacks. The patterns of the attacks of this ADFA-LD dataset is more aligned with the modern day attacks, which makes it more consistent with the actual network environment. Moreover, being a large scaled dataset, it is more suitable to train a deep learning model properly. For simplicity, we label the dataset into two different classes: i) We label the attack sequences as the Positive class, and ii) The normal sequences as the Negative class. Table II shows how we split our dataset for training, validation and testing.

B. Performance Metrics

Some terminologies related to performance metrics are discussed below:

True Positives (TP) True positives are the cases when the actual class of the data point was 1 (True) and the predicted is also 1(True).

True Negatives (TN) True negatives are the cases when the actual class of the data point was 0 (False) and the predicted is also 0 (False).

False Positives (FP) False positives are the cases when the actual class of the data point was 0 (False) and the predicted is 1 (True).

False Negatives (FN)

False negatives are the cases when the actual class of the data point was 1 (True) and the predicted is 0 (False). We want the model to give 0 False Positives and 0 False Negatives. But that is not the case in real life as any model will not be 100% accurate most of the times.

Metrics to evaluate performance of any machine learning model:

- **Accuracy:** Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made. Accuracy is a good measure when the target variable classes in the data are nearly balanced.

$$\text{Accuracy } = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision } = \frac{TP}{TP + FP}$$

- **Recall or Sensitivity:** Recall is a measure that tells us what proportion of samples that are actually positive have been predicted by the algorithm as positive.

$$\text{Recall } = \frac{TP}{TP + FN}$$

- **Specificity:** Specificity is a measure that tells us what proportion of samples that are actually negative have been predicted as negative. Specificity is the exact opposite of Recall.

$$\text{Specificity } = \frac{TN}{TN + FP}$$

- **AUC - ROC Curve:** AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents the area under the ROC curve. It tells how much model is sapable af distinguishing between classes. Higher the AUC, better the model is at predicting 0_s as 0_s and 1 s as 1 s.

$$TPR = \text{ Recall/Sensitivity } = \frac{TP}{TP + FN}$$

$$FPR = 1 - \text{ Specificity } = \frac{FP}{TN + FP}$$

Fig. 8 shows a sample ROC curve. In this figure, TPR is on y-axis and FPR is on x-axis. The green curve represents the TPR vs. FPR values of the model at different thresholds. We assume that the model is giving a probability value as output for each input. Threshold denotes the value above which all probabilities are considered as positive and vice versa. An ideal model should produce probability value 0 for all negative instances and 1 for all positive instances. Hence, the ROC curve of an ideal model should be the red line in Fig. 8 and the area under the ROC curve, i.e., AUC will be 1.

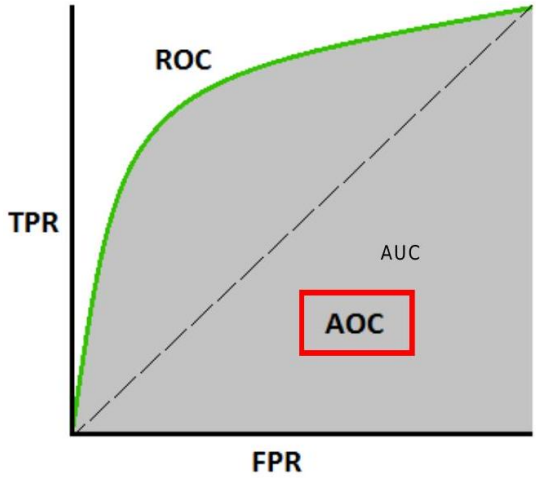

Fig. 8. Sample ROC curve

C. Hyper-parameters

There are a number of hyper-parameters in our approach. These parameters are listed as follows.

1) MAX SENT_LENGTH: It denotes the number of system calls in a sentence. In our hierarchical model, we are splitting each system call sequence into several sentences according to MAX_SENT_LENGTH and applying two layers of attention,

Payload/Effect	Vector	Traces Count
Password bruteforce	FTP by Hydra	162
Password bruteforce	SSH by Hydra	176
Add new superuser	Client side poisoned executable	91
Java Based Meterpreter	Tiki Wiki vulnerability exploit	124
Linux Meterpreter Payload	Client side poisoned executable	75
C100 Webshell	PHP Remote File Inclusion vulnerability	118

TABLE I
DETAIL DESCRIPTION OF SIX ATTACK VECTORS IN ADFA-LD DATASET

Type	# of $(+)$ samples	# of $(-)$ samples	Percentage
Training	448	3122	60
Validation	149	1041	20
Test	149	1041	20

TABLE II
SPLITTING THE DATASET

first on the system calls, second on the sentences. As its value increases, number of sentences from a sequence will decrease and training time will decrease eventually. But second level attention will become less effective.

2) Embedding Dimension: As we are using an embedding layer inside the model, we need to set the embedding dimension. It denotes the dimension of the real-valued vector that each system call is mapped onto. The more we increase its value, model will be more able to generate representative vector for each system call, but training time will increase proportionally.

3) GRU_UNITS: As We are using Bi-directional GRU layer in the model, we need to decide the number of GRU units in the layer. Increasing number of GRU units may perform better but with an overhead of extra training time.

4) Learning rate: It is a significant parameter. Large value of learning rate means the model takes big jumps towards minimum loss and as a result, it can diverge. Small value of learning rate will result in slow training of the model.

We use cyclical learning rate policy to determine optimum learning rate. We train the model in two phases. In the first phase, the learning rate is increased exponentially per epoch and loss at each epoch is recorded. These loss values are plotted. In the second phase, we train the network with the learning rate found in the first phase.

5) Batch Size: It denotes the number of instances to be processed after which the network will update gradients via back propagation with respect to the accumulated loss. Small value of batch size will enable the network to learn more about noisy instances, but frequent back propagation will increase the training time significantly.

D. Training and Validation

We use Keras python library on top of Tensorflow. Before training begins, we need to decide the number of epochs to train the network. Higher number of epochs results in overfitting of the model and lower number of epochs causes underfitting of the model. To tackle this problem, we use ModelCheckPoint from the Keras library. We define two callbacks, one monitors the loss in validation set after each epoch and saves the model whenever it sees a lower loss than what it has already seen, another one monitors the accuracy in validation set after each epoch and saves the model whenever it sees an accuracy grater than what it has already seen. We set number of epochs to 600 .

As there are less number of positive samples compared to the number of negative samples, the dataset is imbalanced. If this imbalance is not handled, the model will not be able to learn to predict the minority class's instances correctly. We use a keras parameter named class_weight to tackle the imbalance of the dataset. We set the value of class_weight to 7 .

We set batch size $= 32$ [2], embedding dimension $= 16$, GRU units $= 32$ initially and varied MAX_SENT_LENGTH from 5 to 30 . The obtained results on validation set are listed in Table III.

We then increase the batch size to 64 and do not alter any other hyper-parameter's values. The obtained results on validation set are listed in Table IV.

E. Testing our Model

We can observe from Tables IV and V that batch size $= 64$ has not degraded the performance but reduced training time significantly. From Table V, we see the model is performing satisfactorily with MAX_SENT_LENGTH $= 15$. So, we report the performance of the model on the test dataset in Table V.

MAX_SENT_LENGTH	Accuracy	AUC	Positive predictive value	Sensitivity	Negative predictive value	Specificity	F1	Training time per epoch
5	0.95	0.95	0.75	0.84	0.98	0.96	0.79	160 s
10	0.96	0.97	0.84	0.83	0.98	0.98	0.83	85 s
15	0.95	0.97	0.8	0.86	0.98	0.97	0.83	62 s
20	0.95	0.96	0.75	0.84	0.98	0.96	0.79	50 s
25	0.92	0.95	0.62	0.91	0.98	0.92	0.74	43 s
30	0.94	0.96	0.68	0.95	0.99	0.94	0.79	43 s

TABLE III
EXPERIMENTAL RESULTS WITH INITIAL VALUES OF HYPER PARAMETERS AND BATCH SIZE = 32

MAX_SENT_LENGTH	Accuracy	AUC	Positive predictive value	Sensitivity	Negative predictive value	Specificity	F1	Training time per epoch
5	0.96	0.96	0.81	0.85	0.98	0.97	0.83	84 s
10	0.94	0.96	0.7	0.91	0.99	0.94	0.79	47 s
15	0.96	0.97	0.8	0.89	0.98	0.97	0.84	34 s
20	0.94	0.96	0.7	0.89	0.98	0.94	0.78	28 s
25	0.96	0.97	0.81	0.86	0.98	0.97	0.83	25 s
30	0.91	0.95	0.59	0.9	0.98	0.91	0.71	22 s

TABLE IV
EXPERIMENTAL RESULTS WITH INITIAL VALUES OF HYPER PARAMETERS AND BATCH SIZE = 64

MAX_SENT_LENGTH	Accuracy	AUC	Positive predictive value	Sensitivity	Negative predictive value	Specificity	F1
15	0.94	0.96	0.74	0.84	0.98	0.96	0.79

TABLE V
EXPERIMENTAL RESULTS ON TEST SET

	Accuracy	AUC	Positive predictive value	Sensitivity	Negative predictive value	Specificity	F1
Kim et al. [8]	-	0.928	-	-	-	-	-
Chawla et al. [3]	-	0.81	-	-	-	-	-
Shaohua et al. [11]	-	0.95	-	-	-	-	-
Diep et al. [6]	0.97	-	0.97	-	0.97	0.97	0.97
This study	0.94	0.96	0.74	0.84	0.98	0.96	0.79

TABLE VI
COMPARATIVE ANALYSIS

We observe that performance of the model for the positive class is comparatively worse. This is because of the fact that positive class is the minority class and has only 12.5% of the total number of samples. Hence, a few wrong predictions are affecting the precision and recall greatly. From Table V, we see the positive predictive value and Sensitivity values are 0.74 and 0.84 respectively. It means the model is correctly predicting 125 positive samples among 149 positive samples in the test set and it is giving false positive alarm for 44 samples.

F. Comparative Analysis

Comparative results are shown in Table VI. As we can see, most existing works have reported the AUC value only. They have usually provided the ROC curve and the AUC value as an evaluation metric. Only Diep et al., provided values for some other metrics. However, they have not provided AUC score. However, we can see that our AUC score is better or equal to all other methods. Most of these previous works have not published their code. As a result this is not possible to reproduce them to evaluate with our model.

We evaluate our model with the work of Shaohua et al., in Fig. 9 with respect to ROC curve. We can observe both method has similar ROC curve.

V. CONCLUSION

In this study, we proposed a deep learning model based HIDS. The results indicate our model outperforming some other state of the art methods in terms of AUC. Although most of the other models have not reported different evaluation metrics. As a result, full comparison was not possible. We tried to use another dataset, namely KDD98. But could not work that out so far. In the future, we plan to introduce more explainability to the model. It already have better explainability because of the Attention mechanism. We also want to test

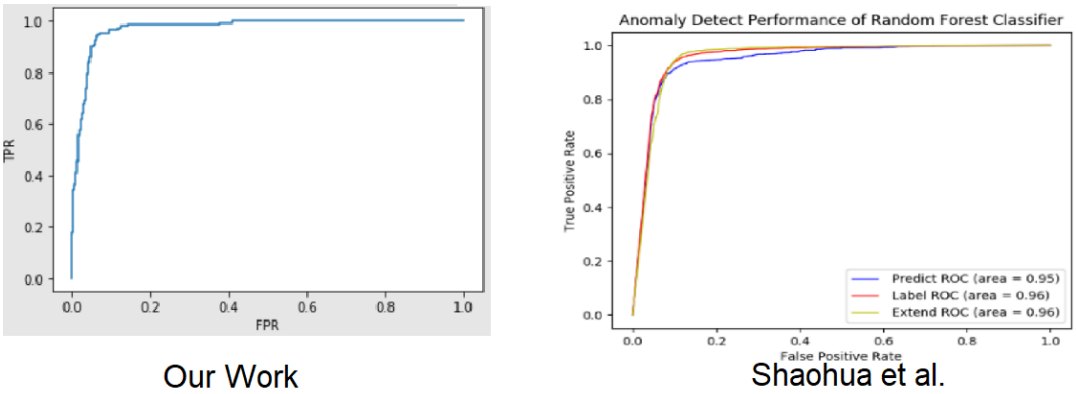

Fig. 9. Comparison between ROC curves

these on more datasets to understand how the model performs there

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478, 2012.

[3] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. Host based intrusion detection system with combined cnn/rnn model. In *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18*, pages 149–158. Springer, 2019.

[4] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492. IEEE, 2013.

[5] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4):807–819, 2013.

[6] Nguyen Ngoc Diep, Nguyen Thi Thanh Thuy, and Pham Hoang Duy. Combination of multi-channel cnn and bilstm for host-based intrusion detection. *Southeast Asian Journal of Sciences*, 6(2):147–159, 2018.

[7] Stephanie Forrest, Steven Hofmeyr, and Anil Somayaji. The evolution of system-call monitoring. In *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, December 2008.

[8] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon. Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. *arXiv preprint arXiv:1611.01726*, 2016.

[9] Richard P Lippmann, David J Fried, Isaac Graf, Joshua W Haines, Kristopher R Kendall, David McClung, Dan Weber, Seth E Webster, Dan Wyschogrod, Robert K Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26. IEEE, 2000.

[10] Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. Host-based intrusion detection system with system calls. *ACM Computing Surveys*, 51(5):1–36, November 2018.

[11] Shaohua Lv, Jian Wang, Yinqi Yang, and Jiqiang Liu. Intrusion prediction with system-call sequence-to-sequence model. *IEEE Access*, 6:71413–71421, 2018.

[12] Atilla Özgür and Hamit Erdem. A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015. 2016.

[13] Shuaichuang Yang, Minsheng Tan, Shiying Xia, and Fangju Liu. A method of intrusion detection based on attention-LSTM neural network. In *Proceedings of the 2020 5th International Conference on Machine Learning Technologies*. ACM, June 2020.

[14] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.