A Project Report

On

BadUSB Device Detection Using Deep Neural Network

BY

Name of the student ID.no

KATHAN NAIK 2020A7PS2082H

Under the supervision of

Dr.Paresh Saxena

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS OF CS F366: Laboratory Project

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN) HYDERABAD CAMPUS

(MAR 2024)

ACKNOWLEDGMENTS

I am incredibly grateful to Dr. Paresh Saxena and Mr. Chillara Anil Kumar for giving me the opportunity to contribute to this project. Working on this project has been an invaluable experience, allowing me to delve into the intricate realms of both security and deep learning. I have learned so much and grown both personally and professionally through this endeavor. Thank you for entrusting me with this opportunity.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

CERTIFICATE

This is to certify that the project report entitled "BadUSB Device Detection Using DNN" submitted KATHAN NAIK (ID No. 2019A7PS2082) in partial fulfillment of the requirements of the course CS F366, Laboratory Project Course, embodies the work done by them under my supervision and guidance.

Date: 18-5-2024 (Paresh Saxena) BITS- Pilani, Hyderabad Campus

ABSTRACT

The USB protocol is widely used due to its plug-and-play functionality and device support. However, this popularity has led to a major security concern: the implicit trust in USB devices, which can be exploited by malicious devices. Current research uses supervised learning to detect such devices, but our study shows weaknesses in these models against data poisoning attacks. Previous work on this project has proposed and demonstrated a sophisticated adversarial data poisoning attack that can manipulate these models to misclassify an attack device as benign. By generating keystroke data using a keystroke attack device and modifying the initial training data, we achieved a significant decrease in detection accuracy, highlighting the need for more robust detection strategies against evolving malicious USB devices. Deep Neural Networks can help develop more capable detection mechanisms. Deep Convolutional Network and Generative Adversarial Network can be applied in this problem. The state of the art technologies that utilize the Transformers architecture can also be utilized for this problem. Transfer learning of models like CLIP and BERT can be of great use in this problem. The aim of this project is to leverage these techniques to develop a more robust mechanism against this type of data poisoning attacks.

TABLE OF CONTENTS

1. Introduction of BadUSB Devices

BadUSB is a term used to describe a class of malicious USB devices that exploit the trust typically placed in USB peripherals by computers and other devices. These devices can impersonate legitimate USB devices such as keyboards, mice, or storage drives, but they contain malicious firmware or hardware that can be used to perform a variety of attacks.
One of the key characteristics of BadUSB attacks is their ability to be "reprogrammed" after they have been manufactured, allowing them to change their functionality to suit different attack scenarios. For example, a USB device could initially present itself as a keyboard to the host computer and then, once it has gained access, reprogram itself to act as a network adapter to exfiltrate data or perform other malicious activities. BadUSB attacks can be used for a variety of purposes, including Installation of malicious script, data theft of sensitive data, denial of service attack or botnet creation.

BadUSB devices that pretend to be Human Interface Devices (HID) are particularly deceptive and dangerous. Human Interface Devices include devices like keyboards, mice, and game controllers, which are typically considered benign and are trusted by operating systems without much scrutiny. A BadUSB device that masquerades as an HID can exploit this trust to carry out various malicious actions, often without the user's knowledge.
Here's how it works:

Impersonation: The BadUSB device presents itself to the computer as a HID, such as a keyboard or a mouse. This allows it to interact with the computer as if it were a legitimate input device.

Malicious Commands: Once connected, the BadUSB device can send malicious commands to the computer. For example, it can simulate keystrokes to type commands or execute scripts, effectively taking control of the computer.

Payload Delivery: The HID impersonation can be used to deliver malware payloads. For instance, the device can act as a keyboard and type out a series of commands to download and execute malware from the internet.

Data Exfiltration: HID impersonation can also be used for data exfiltration. The device can simulate the copying of sensitive files or the typing of sensitive information and send it to the attacker-controlled device.

2. Feature Abstraction from USB Traffic

As part of the Previous Work carried out in this project, a huge amount of data was collected from users. The focus of this project has been on Keyboard pretending BadUSB devices. Many users are given the task of typing paragraphs, command lines and python programs in text editors. While doing this task their USB Traffic was recorded. The recorded traffic from Wireshark is then used to abstract useful features for input to the ML/DL model. This abstraction of features is done using a Python script.

Here is the list of 29 abstracted features:

Packet length based features: These types of features do not depend on the time and largely depend on the number and size of forward and backward packets. Many of these features are dependent on the USB protocol.

1 tot_fw_pk: Total packets in the forward direction
2 tot_bw_pk; Total packets in the backward direction
3 tot_l_fw_pkt: Total size of packet in forward direction
4 fw_pkt_l_max: Maximum size of packet in forward direction
5 fw_pkt_l_min: Minimum size of packet in forward direction
6 fw_pkt_l_avg: Average size of packet in forward direction
7 fw_pkt_l_mean; Mean size of packet in forward direction
8 fw_pkt_l_std: Standard deviation of packet size in forward direction
9 bw_pkt_l_ max: Maximum size of packet in backward direction
10 bw_pkt_l_ min: Minimum size of packet in backward direction
11 bw_pkt_l_mean: Mean size of packet in backward direction
12 bw_pkt_l_std: Standard deviation of packet size in backward direction
13 pkt_size_avg: Average size of packet

Time based features: These features depend on time. These features are influenced the most by the typing behavior of benign users.

14 fl_dur: Flow duration
15 fw_fl_byt_s: Flow byte rate, i.e., the number of bytes transferred per second in forward direction
16 bw_fl_byt_ s: Flow byte rate, i.e., the number of bytes transferred per second in backward direction

17 bw_fl_pkt_ s: flow packets rate, i.e., the number of packets transferred per second in backward direction
18 fw_iat_tot: Sum of the inter arrival times between consecutive packets in the forward direction
19 fw_iat_avg: Average of the inter arrival times between consecutive packets sent in the forward direction
20 fw_iat_std: Std.dev of the inter arrival times between consecutive packets sent in the forward direction
21 fw_iat_max: Maximum of the inter arrival times between consecutive packets sent in the forward direction
22 fw_iat_min: Minimum of the inter arrival times between consecutive packets sent in the forward direction
23 bw_iat_tot: Sum of the inter arrival times between consecutive packets in the backward direction
24 bw_iat_avg: Average of the inter arrival times between consecutive packets sent in the backward direction
25 bw_iat_std: Std.dev of the inter arrival times between consecutive packets sent in the backward direction
26 bw_iat_max: Maximum of the inter arrival times between consecutive packets sent in the backward direction
27 bw_iat_min: Minimum of the inter arrival times between consecutive packets sent in the backward direction
28 fw_pkt_s: Number of forward packets per second
29 bw_pkt_s: Number of backward packets per second set record

3. GANs for BadUSB Device Detection

Generative Adversarial Networks is a technique where two adversarial models, one of which is generative model and the other one is discriminator, is made to compete with each other in order to enhance both generative and discriminative capabilities.

3.1 Introduction to GANs

Introduced by Ian Goodfellow and his colleagues in 2014. GANs are designed to generate new data instances that resemble your training data. They consist of two neural networks, the generator and the discriminator, which are trained simultaneously through a competitive process.

For GANs implementation there are 5 major components that need to be defined.

Generator: The generator takes random noise as input and generates data samples. It learns to map this noise to output data that resembles your training data. For example, in image generation, the generator takes random noise vectors and outputs images.

Discriminator: The discriminator is like a binary classifier that receives both real data samples from your training set and fake data samples from the generator. It learns to distinguish between real and fake data. The discriminator's goal is to correctly classify real data as real (label 1) and fake data as fake (label 0).

Training Process: During training, the generator and discriminator are trained alternately in a competitive manner. The generator tries to produce data samples that are increasingly indistinguishable from real data, while the discriminator tries to improve its ability to differentiate between real and fake data. Defining the right batch size is crucial for successful training of models.

Loss Function: The training process is guided by a loss function. Both the generator and discriminator loss function will drive the back propagation and adjusting of weights in adversarial neural networks. There are multiple loss functions that can be applied to a GANs setup, and choosing the right function for a specific use case is crucial.

Equilibrium / Convergence: Ideally, the GAN reaches an equilibrium where the generator produces high-quality, realistic samples, and the discriminator is unable to distinguish between real and fake data. Training GANs for optimal number of epochs is important. Otherwise GANs can face mode collapse.

Another usefulness of GANs is for data augmentation. Data augmentation using GANs involves generating synthetic data samples that are similar to the original data. This technique can be useful when the original dataset is limited in size or diversity. By training a GAN on the original dataset, the generator learns to create new data samples that resemble the original data, while the discriminator learns to distinguish between real and fake data. The synthetic data samples generated by the GAN can then be added to the original dataset to create an augmented dataset, which can help improve the performance of a machine learning model, especially in tasks like image and video processing. However, generating high-quality synthetic data can be challenging, and the performance of the augmented dataset depends on the quality of the GAN training.

3.2 GANs for Cyber Security

GANs is used in Cyber Security for two major purposes. Both the use cases are discussed below.

1) **Data Augmentation to enrich the Dataset:** As discussed earlier GANs can be used to augment any kind of data. This can lead to creation of better datasets of different kinds of attacks and attack signatures, specifically when it comes to the field of cyber security.

2) **Develop a powerful Discriminator for IDS:** Discriminator trained by adversarial training can be used as Intrusion Detection System (IDS) for different purposes like Malware detection, DDoS detection, Trojan detection or any other kind of attack including BadUSB Attacks.

This IEEE Survey Paper lists all the popular publications where use of GANs for security purposes is proposed. Covering details of this survey paper is beyond the capacity of this report but I have found this paper extremely useful and I advise reader to go through this paper for better understanding.

3.3 Augmentation of Benign Data

We have used GANs first for augmenting the benign data. The best results obtained by us are with the following architecture and hyper parameters.

Generator: The Generator is having noise of dimension 3 and output of dimension 14. There are 2 hidden layers in the generator with first having 240 neurons with ‘relu’ activation and second with 480 neurons with ‘linear’ activation. There is a high dropout of 0.5 after both layers.

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(240, activation='relu', input_shape=(noise_dim,)))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(480, activation='linear'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(14, activation='relu'))

    return model
```

Discriminator: The discriminator is having 120 and 60 neutrons in first and second hidden layers respectively with ‘relu’ activation. This is followed by sigmoid activation in the output layer.

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(120, activation='relu', input_shape=(14,)))
    model.add(layers.Dense(60, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))

    return model
```

Training Process: The best training output is obtained by training discriminator with 8 batch size and generator with 1 generated output. This is done to establish stochastic gradient descent in the generator.

Loss Function: Both the discriminator and generator uses cross-entropy for loss calculation as below.

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Equilibrium / Convergence: By carefully observing the loss at each level the equilibrium is observed at 20 epochs.

The evaluation of this augmented data is done by using Machine Learning models to distinguish between original benign data and augmented benign data. Below you can see the table for obtained results.

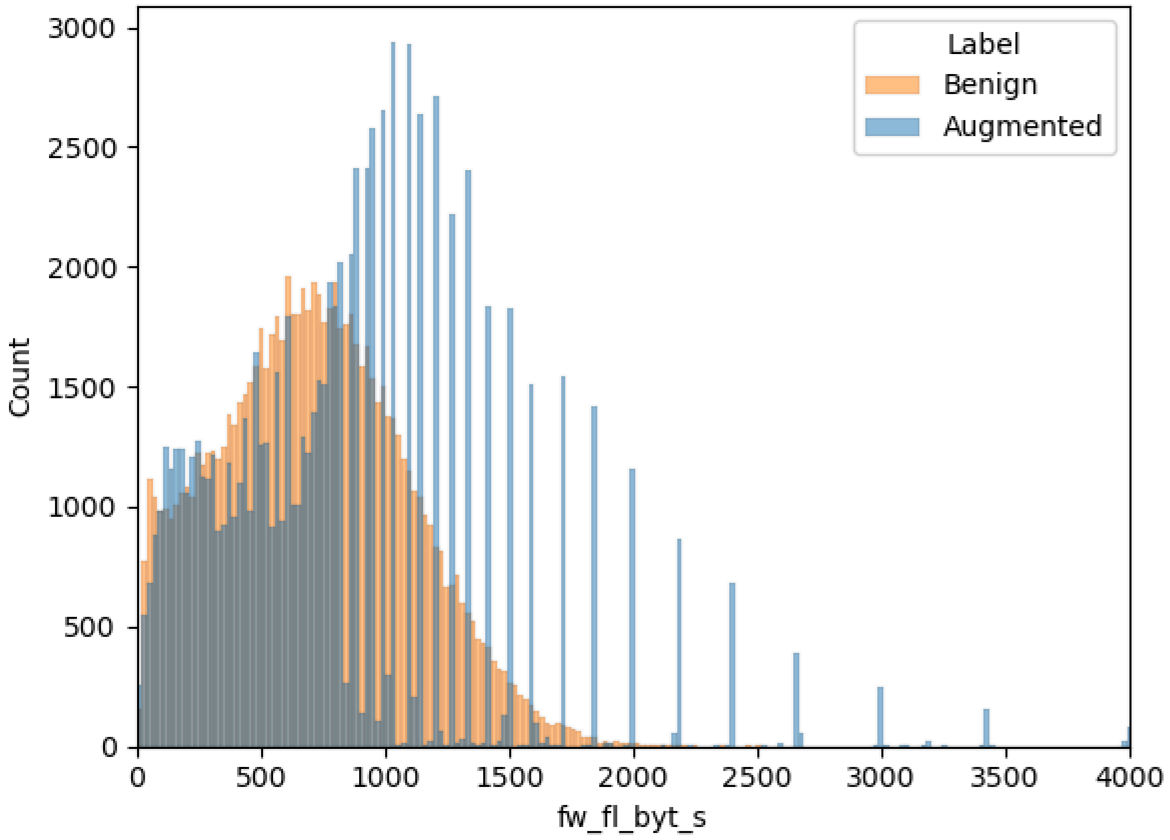
Decision Tree	60%
Naive Bayes	53%
MLP	91%
SVM	54%

Here you can also see the feature importance score of decision tree classification and distribution of both augmented and benign data on these features with high feature importance score.

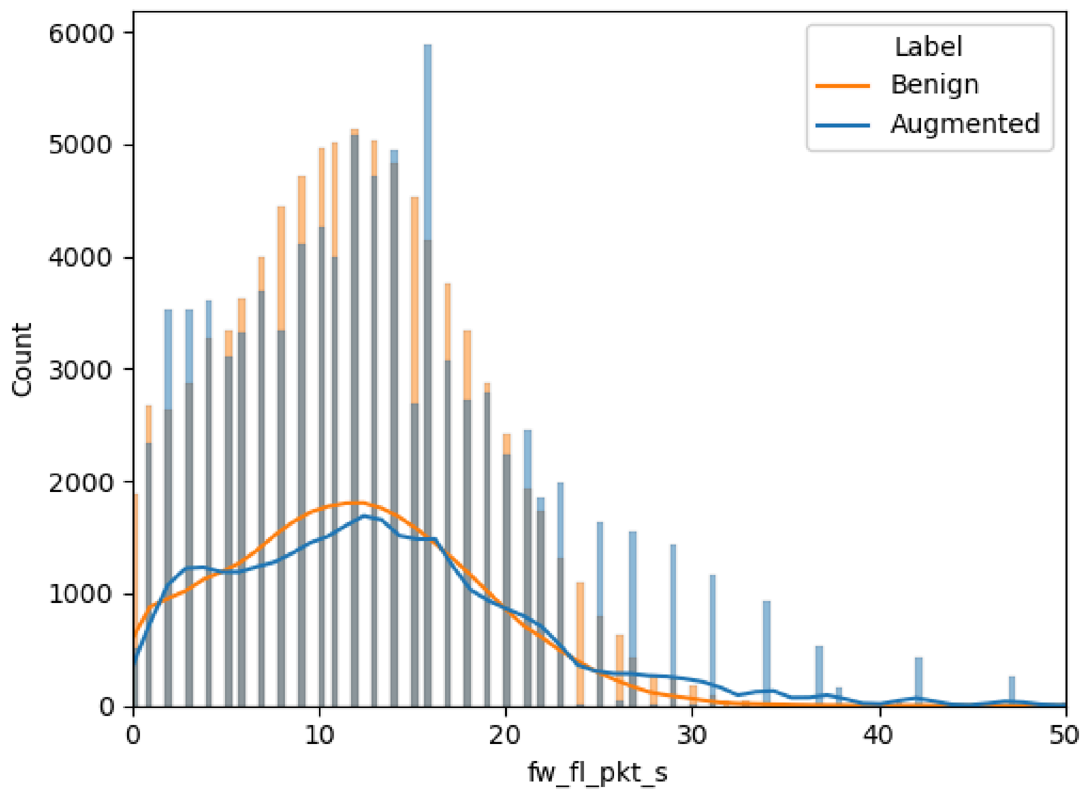
The feature with the highest feature importance score is ‘fw_fl_byt_s’ followed by ‘bw_iat_max’. All other features except below have 0 feature importance score.

```
fw_fl_byt_s: 0.7267577663762137
bw_fl_byt_s: 0.02949680510114947
bw_iat_max: 0.20711024735504083
bw_pkt_s: 0.03663518116759613
```

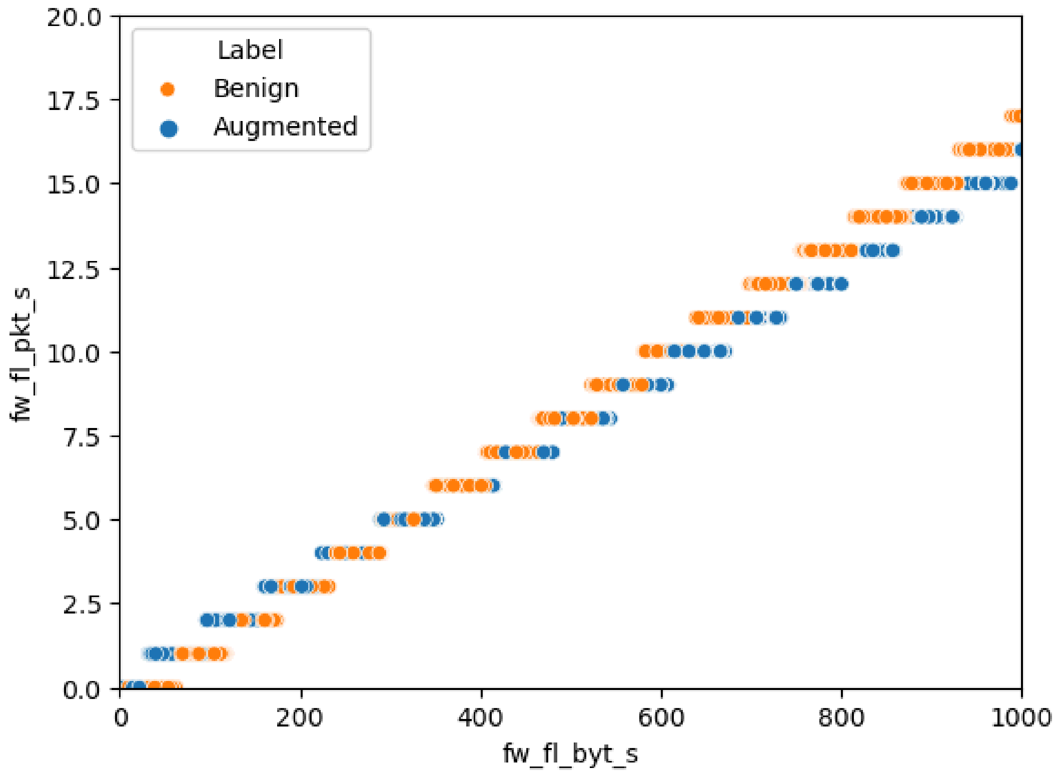
You can clearly see uneven distribution in real and augmented benign data of feature with highest importance score here. Improving consistency in distribution further in these features will increase the augmented data quality even further and we will be able to achieve low distinguishing accuracy even in models like MLP.



Other important features have more than similar distribution like below.



Label
Benign
Augmented
Count
fw_fl_pkt_s
6000
5000
4000
3000
2000
1000
0
10
20
30
40
50



Label
Benign
Augmented
fw_fl_pkt_s
fw_fl_byt_s
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000

3.4 Improvement in Accuracy due to GANs Data

Data augmentation using GANs enhances the accuracy of machine learning models by increasing data volume, balancing datasets, improving generalization, and enriching feature learning. This approach leverages the strengths of GANs to create realistic and diverse synthetic data, providing a more robust training set for the target model.

The Bening data augmented by the GANs has significantly improved accuracy of the machine learning models that we were using. Given below is the table of % improvement in accuracy due to addition of GANs augmented data in the training process.

ML models	AA2_Accuracy (GANs data)	AA2_accuracy (published work)	Change in % accuracy
DT	76.1	65.58	12.93
kNN	77.63	62.91	14.66
SVM	77.38	61.02	16.36
NB	71.345	53.58	17.76
MLP	77.59	64.93	7.17
RF	82.46	76.84	5.62
XGB	79.42	65.57	13.85

These results show promising improvement in accuracy and prove usefulness of data augmentation using GANs in such use cases.

4. Time Series Classification

As the USB Traffic is a time series data the classification that is used specifically for time series data classification can also be used for this classification.

4.1 Introduction to Time Series Classification

Time series classification is a type of machine learning problem where the objective is to classify a sequence of data points, typically measured at successive points in time, into distinct categories. Unlike traditional classification tasks where the input features are static, time series classification involves inputs that are ordered and often correlated over time. This temporal aspect makes the problem unique and often requires specialized techniques and models.

Such specialized models can be Convolutional Neural Networks, Recurrent Neural Networks like LSTM, Transformers or LLMs. Next we will talk about how we tried one possible convolutional ensemble model for our use case.

4.2 InceptionTime: Finding AlexNet for time series classification

We have implemented the Inception Time paper for our implementation because it highlights the advancements in time series classification (TSC) through deep learning. TSC involves categorizing time series data, and while traditional algorithms like HIVE-COTE have achieved high accuracy, they suffer from high training time complexity, making them impractical for many real-world datasets. In contrast, deep learning offers both high accuracy and scalability. The authors introduce InceptionTime, an ensemble of deep Convolutional Neural Network models based on the Inception-v4 architecture. Their experiments demonstrate that InceptionTime matches HIVE-COTE in accuracy while being significantly more scalable, capable of training on 1,500 time series in one hour and 8 million time series in 13 hours, a scale unattainable by HIVE-COTE.

The architecture of an Inception network classifier in the described study incorporates several innovative features to enhance its performance and scalability for time series classification. Here's a breakdown of its composition:

Residual Blocks: The Inception network uses two different residual blocks. In contrast, ResNet typically uses three residual blocks. Each residual block in the Inception network comprises three Inception modules instead of traditional fully convolutional layers.

Inception Modules: Each Inception module is a complex layer designed to handle multiple convolutional operations simultaneously. These modules allow the network to learn different levels of abstraction and features from the input data.

Shortcut Connections: The input of each residual block is passed through a shortcut linear connection. This connection directly adds the input to the output of the block, effectively helping to mitigate the vanishing gradient problem by maintaining the gradient flow throughout the network.

Global Average Pooling (GAP) Layer: After the residual blocks, the network includes a GAP layer. The GAP layer averages the output multivariate time series over the entire time dimension, reducing the data to a more manageable form while preserving the essential features.

Fully-Connected Softmax Layer: The final layer is a traditional fully-connected softmax layer. This layer contains a number of neurons equal to the number of classes in the dataset. It outputs the probabilities for each class, enabling the classification decision.

Stacking of Inception Modules: The architecture consists of six different Inception modules stacked sequentially. This stacking allows the network to progressively extract more complex and abstract features from the input time series data.

Given below is the architecture details of this ensemble convolutional model generated by summary() function.

```
________________________________________________________________________________
_____
 Layer (type)                Output Shape         Param #     Connected to
================================================================================
=========
 input_1 (InputLayer)         [(None, 29, 1)]      0           []

 conv1d (Conv1D)              (None, 15, 64)       512         ['input_1[0][0]']

 max_pooling1d (MaxPooling1D)   (None, 8, 64)        0           ['conv1d[0][0]']

 conv1d_2 (Conv1D)             (None, 8, 64)        4160        ['max_pooling1d[0][0]']
```

```
conv1d_4 (Conv1D)              (None, 8, 64)        4160        ['max_pooling1d[0][0]']

max_pooling1d_1 (MaxPooling1D)  (None, 8, 64)       0           ['max_pooling1d[0][0]']

conv1d_1 (Conv1D)              (None, 8, 64)        4160        ['max_pooling1d[0][0]']

conv1d_3 (Conv1D)              (None, 8, 64)        12352       ['conv1d_2[0][0]']

conv1d_5 (Conv1D)              (None, 8, 64)        20544       ['conv1d_4[0][0]']

conv1d_6 (Conv1D)              (None, 8, 64)        4160        ['max_pooling1d_1[0][0]']

concatenate (Concatenate)      (None, 8, 256)       0           ['conv1d_1[0][0]',
                                                                 'conv1d_3[0][0]',
                                                                 'conv1d_5[0][0]',
                                                                 'conv1d_6[0][0]']

conv1d_8 (Conv1D)              (None, 8, 64)        16448       ['concatenate[0][0]']

conv1d_10 (Conv1D)             (None, 8, 64)        16448       ['concatenate[0][0]']

max_pooling1d_2 (MaxPooling1D)  (None, 8, 256)      0           ['concatenate[0][0]']

conv1d_7 (Conv1D)              (None, 8, 64)        16448       ['concatenate[0][0]']

conv1d_9 (Conv1D)              (None, 8, 64)        12352       ['conv1d_8[0][0]']

conv1d_11 (Conv1D)             (None, 8, 64)        20544       ['conv1d_10[0][0]']

conv1d_12 (Conv1D)             (None, 8, 64)        16448       ['max_pooling1d_2[0][0]']

conv1d_13 (Conv1D)             (None, 8, 256)       16640       ['max_pooling1d[0][0]']

concatenate_1 (Concatenate)    (None, 8, 256)       0           ['conv1d_7[0][0]',
                                                                 'conv1d_9[0][0]',
                                                                 'conv1d_11[0][0]',
                                                                 'conv1d_12[0][0]']

add (Add)                      (None, 8, 256)       0           ['conv1d_13[0][0]',
                                                                 'concatenate_1[0][0]']

re_lu (ReLU)                   (None, 8, 256)       0           ['add[0][0]']

conv1d_15 (Conv1D)             (None, 8, 64)        16448       ['re_lu[0][0]']

conv1d_17 (Conv1D)             (None, 8, 64)        16448       ['re_lu[0][0]']

max_pooling1d_3 (MaxPooling1D)  (None, 8, 256)      0           ['re_lu[0][0]']

conv1d_14 (Conv1D)             (None, 8, 64)        16448       ['re_lu[0][0]']

conv1d_16 (Conv1D)             (None, 8, 64)        12352       ['conv1d_15[0][0]']

conv1d_18 (Conv1D)             (None, 8, 64)        20544       ['conv1d_17[0][0]']
```

```
 conv1d_19 (Conv1D)            (None, 8, 64)       16448      ['max_pooling1d_3[0][0]']

 concatenate_2 (Concatenate)    (None, 8, 256)       0          ['conv1d_14[0][0]',
                                                   'conv1d_16[0][0]',
                                                   'conv1d_18[0][0]',
                                                   'conv1d_19[0][0]']

 conv1d_21 (Conv1D)            (None, 8, 64)       16448      ['concatenate_2[0][0]']

 conv1d_23 (Conv1D)            (None, 8, 64)       16448      ['concatenate_2[0][0]']

 max_pooling1d_4 (MaxPooling1D)  (None, 8, 256)      0          ['concatenate_2[0][0]']

 conv1d_20 (Conv1D)            (None, 8, 64)       16448      ['concatenate_2[0][0]']

 conv1d_22 (Conv1D)            (None, 8, 64)       12352      ['conv1d_21[0][0]']


conv1d_24 (Conv1D)            (None, 8, 64)       20544      ['conv1d_23[0][0]']

 conv1d_25 (Conv1D)            (None, 8, 64)       16448      ['max_pooling1d_4[0][0]']

 concatenate_3 (Concatenate)    (None, 8, 256)       0          ['conv1d_20[0][0]',
                                                   'conv1d_22[0][0]',
                                                   'conv1d_24[0][0]',
                                                   'conv1d_25[0][0]']

 add_1 (Add)                  (None, 8, 256)       0          ['re_lu[0][0]',
                                                   'concatenate_3[0][0]']

 re_lu_1 (ReLU)                (None, 8, 256)       0          ['add_1[0][0]']

 global_average_pooling1d (Glob  (None, 256)         0          ['re_lu_1[0][0]']
 alAveragePooling1D)

 dense_5 (Dense)               (None, 1)            257        ['global_average_pooling1d[0][0]'
                                                  ]

==========================================================================================
=========
Total params: 363,009
Trainable params: 363,009
Non-trainable params: 0
```

With this model we were able to achieve accuracy between 65-70% which was not sufficient. So in the next step we implemented pre-trained LLM models.

4.3 BERT and CLIP: State of the art models

BERT (Bidirectional Encoder Representations from Transformers) and CLIP (Contrastive Language-Image Pre-training) are considered state-of-the-art models in their respective domains due to their innovative architectures, training methods, and exceptional performance across various tasks.

4.3.1 BERT

Traditional models like RNNs and earlier Transformers processed text either left-to-right or right-to-left. BERT, however, uses a bidirectional approach, allowing it to consider the context from both directions simultaneously. This bidirectional understanding enables BERT to capture the meaning of words in their full context, leading to better comprehension of the nuances and relationships within the text.

BERT leverages the Transformer architecture, which relies on self-attention mechanisms. This allows the model to weigh the importance of different words in a sentence when making predictions. Transformers can process entire sequences of text in parallel, leading to faster training times and better scalability compared to RNN-based models.

4.3.2 CLIP

CLIP is designed to understand and link visual and textual information. It learns to associate images and their corresponding textual descriptions, enabling it to perform tasks that require understanding both modalities. This multimodal approach is crucial for tasks like image captioning, visual question answering, and zero-shot image classification.

CLIP uses a contrastive learning approach during pre-training, where it learns to distinguish between matching and non-matching image-text pairs. By training on a large and diverse dataset of images and their descriptions, CLIP develops a rich, joint embedding space for both images and text, allowing for flexible and powerful cross-modal retrieval and classification. Similar to BERT, the clip also utilizes Transformers.

These models are developed by industry leading researchers and companies like Google and are trained on huge amounts of data. Such training on billions of data samples requires extensive computational resources. But this training or learning can be leveraged for our use case. Where the existing weights of models are used and then the model is further fine tuned on a new dataset. This is called transfer learning and it is a widely used practice.

Using BERT and CLIP we were able to operate above 90% accuracy and a good balance of both precision and recall. These models seem to have an edge on performance of other models and approaches due to their huge training and utilization of transformer architecture.

Conclusion

Deep Neural Network can be used to create a robust intrusion detection system for sophisticated BadUSB attacks. GANs can be an extremely useful technique for such a use case where data can be augmented for better defender training or even making more sophisticated attackers. Some promising success has been achieved by using GANs for augmentation of benign data. Other approaches like Convolutional Networks can also be used for this purpose but more novel approaches like using transformer based models that are used for time-series classification can be of extreme use here. Such models like BERT and CLIP are able to achieve more than 90% accuracy and thus can be useful for the development of a powerful defender.

References

1) Deceiving supervised machine learning models via adversarial data poisoning attacks: a case study with USB keyboards.
Anil Kumar Chillara, Paresh Saxena, Rajib Ranjan Maiti, Manik Gupta, Raghu Kondapalli, Zhichao Zhang & Krishnakumar Kesavan
[https://link.springer.com/article/10.1007/s10207-024-00834-y]

2) A Comprehensive Survey of Generative Adversarial Networks (GANs) in Cybersecurity Intrusion Detection.

AERYN DUNMORE (Graduate Student Member, IEEE), JULIAN JANG-JACCARD, FARIZA SABRINA (Member, IEEE) AND JIN KWAK

[https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10187144]

3) Ismail Fawaz, H., Lucas, B., Forestier, G. *et al.* InceptionTime: Finding AlexNet for time series classification. *Data Min Knowl Disc* **34**, 1936–1962 (2020).
[https://doi.org/10.1007/s10618-020-00710-y]

4) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova
Google AI Language