Malicious URL Detection

Ashish Kumar
Data Science and Engineering
Indian Institute of Science, Education and Reasearch
Bhauri, Madhaya Pradesh - 462066
ashish21@iiserb.ac.in

July 22, 2024

Abstract

On the internet we search thousands of websites. That means the search engines show thousands of websites(malicious or benign), how to know that the website we are clicking is malicious or benign? This research paper introduces a new, effective method for detecting these malicious URLs using a model called CharBERT, which understands both the small parts of words (subwords) and individual characters.Also they have develop three keys modules (1) Hierarchical Feature Extraction,(2) Layer-Aware Attention,(3) Spatial Pyramid Pooling.

1 Introduction

Malicious URLs are increasingly used for phishing attacks, with a 102% rise recently. Traditional methods like blacklists, heuristic, and rule-based detection struggle to keep up with evolving threats. Machine learning (ML) and deep learning (DL) offer better detection by analyzing URL patterns and automatically learning from data. Convolutional Neural Networks (CNNs) and models like BERT have shown promise. The proposed method, PLMMFA (Pre-trained Language Model-guided Multi-Level Feature Attention network), uses CharBERT and includes three key components to enhance malicious URL detection.

2 Dataset

In this project I have been working on the kaggle(binary) dataset using 632,503 samples (equally divided between malicious and benign URLs),csv format dataset and using 80% for training and 20% for testing.After training 80% dataset of kaggle(binary) dataset I am testing with different type of dataset like GramBeddings Dataset 800,000 samples (400,000 malicious, 400,000 benign),Mendeley

Dataset 1,561,934 samples (1,526,619 benign, 35,315 malicious) both have divide the datset in training and test datset for GramBedding Train 640,000(benign=320,002,malicious=319,997),Test: 159997(benign=79999,malicious=79998) and for Mendeley Test:360,000 ,Train:1,201,934.

2.1 datasets features

In the dataset it contain only two features,one is URLs (benign and malicious) and second feature is label (0 for benign and 1 for malicious) now using this features the preprocessing is creating the following features:

- `input_ids`: List to store input token IDs.
- `input_types`: List to store segment IDs.
- `input_masks`: List to store attention masks.
- `label`: List to store labels (0 for benign, 1 for malicious).
- `urltype`: Type of URL (0 for benign, 1 for malicious).
- `end_ids`: Charbert Input
- `start_ids`: Charbert Input
- `char_ids`: Charbert Input

3 Algorthim

3.1 Backbone Network

Character Embedding involves splitting the input sentence into subwords and characters, embedding each character, and using BiGRU to form token-level embeddings from the first and last characters of each token. Merging and Splitting Representations transform and combine token and character embeddings using a CNN layer and split them back with a fully connected layer, aided by a residual connection to retain original details. Finally, the embeddings are updated and normalized, enhancing token representation by combining rich information from characters and subwords.

3.2 Hierarchical Feature Extraction

Merge and Stack Outputs:Combine word-level and character-level outputs using a one-dimensional convolution, then stack them along a new dimension to create a tensor of rank (N, H, W, C).
Permute the Tensor:Swap dimensions using a permutation matrix, resulting in a tensor of rank (H, N, W, C) for better feature alignment.
Prepare for Attention Module:The permuted tensor is now well-aligned and ready for the attention module to focus on important features.

3.3 Layer-Aware Attention

Layer-Aware Attention helps the model prioritize features from different layers of CharBERT by assigning weights to them. Features are extracted from all layers and processed through average and max pooling, followed by MLP processing with shared weights and ReLU activation. The results are combined and normalized using a sigmoid function to create an attention map. This map is multiplied with the original feature map, emphasizing important features and enhancing the model's performance by leveraging a rich set of features.

3.4 Spatial Pyramid Pooling

Spatial Pyramid Pooling starts with weighted feature maps and applies three pyramid levels to create different scales of sub-regions. At each pyramid level, a sliding window of size $[a/n]$ and stride $[b/n]$ is used to cover the feature map. Pooling operations (e.g., max or average pooling) are performed on each sub-region to summarize the features within it.

3.5 Dimension analysis

- Initial CharBERT Output: (64, 200, 768) per layer.
- After Feature Extraction: (12, 64 , 200, 768).
- Attention Maps: (12, 1, 1).
- Final Output: Binary classification

4 Experiments on BERT Model

In this experiment, I have configure a neural network model with key hyperparameters. The model uses a hidden size of 768 and includes 12 attention heads and 12 hidden layers. Dropout probabilities for attention and hidden layers are set at 0.1, and the activation function used is GELU. The model supports a maximum of 512 position embeddings and an intermediate layer size of 3072. It is not set up as a decoder and doesn't output attention scores or hidden states by default. The vocabulary size is 28,996, with mappings defined for two labels (0 and 1). The configuration also includes details like the initializer range (0.02) and epsilon for layer normalization (1e-12). And the hyperparameters are Batch size: 64, Optimizer: AdamW, Learning rate: 2e-5, Weight decay: 1e-4, Dropout rate: 0.1, Training duration: 3 epochs and model selection based on the best validation loss during training.

4.1 experiment on kaggle(binary) dataset

In this section, I trained a Kaggle binary dataset, using 80% for training and 20% for testing. The dataset contains 632,503 samples, equally divided between

malicious and benign URLs.After training I tested with different type of dataset like GramBeddings and Mendeley.

4.1.1 results

The results of above experiments are as follows:

- input_ids_train.shape:(1012006,200)
- input_types_train.shape:(1012006,200)
- input_masks_train.shape:(1012006,200)
- `y_train.shape`:(1012006,1)
- input_ids_test.shape:(253002,200)
- input_types_test.shape:(253002,200)
- input_masks_test.shape:(253002,200)
- `y_test.shape`:(253002,1)

Epoch 1:
Test set: Average loss: 0.6934, Accuracy: 49.78%, Precision:49.79%, Recall: 7.23%, F1:12.63%.
Epoch 2:
Test set: Average loss: 0.6932, Accuracy: 50.02%, Precision:50.15%, Recall: 72.22%, F1:59.19%.
Epoch 3:
Test set: Average loss: 0.6932, Accuracy: 50.13%, Precision:50.19%, Recall: 84.63%, F1:63.01%.

4.1.2 Testing on different datasets:

GramBeddings:
Average loss: 0.6939, Accuracy: 45.48%, Precision:47.25%, Recall: 77.71%, F1:58.77%.
Mendeley:
Average loss: 0.6989, Accuracy: 13.33%, Precision:2.23%, Recall: 88.34%, F1:4.34%.
Kaggle(binary):
Average loss: 0.6934, Accuracy: 48.92%, Precision:49.36%, Recall: 83.37%, F1:62.01%.

4.2 experiment on kaggle(multi-class) dataset

In this section, I trained a Kaggle multi-class dataset, using 80% for training and 20% for testing.Dataset contain benign 428079, defacement 95306, phishing 94086, malicious 23645 samples.

4.2.1 results

The results of above experiments are as follows:

- input_ids_train.shape:(820656, 200)
- input_types_train.shape:(820656, 200)
- input_masks_train.shape:(820656, 200)
- `y_train.shape`:(820656, 1)
- input_ids_test.shape:(205164, 200)
- input_types_test.shape:(205164, 200)
- input_masks_test.shape:(205164, 200)
- `y_test.shape`:(205164, 1)

Epoch 1:
Test set: Average loss: 0.6932, Accuracy: 50.12%, Precision: 50.16%, Recall: 55.84%, F1: 52.85%
Epoch 2:
Test set: Average loss: 0.6932, Accuracy: 50.05%, Precision: 50.08%, Recall: 76.37%, F1: 60.49%
Epoch 3:
Test set: Average loss: 0.6932, Accuracy: 50.09%, Precision: 50.05%, Recall: 80.37%, F1: 64.89%

4.2.2 Testing on different datasets:

Kaggle(binary):
Average loss: 0.6931, Accuracy: 50.73%, Precision:50.61%, Recall: 60.37%, F1:55.06%.
GramBeddings:
Average loss: 0.6934, Accuracy: 48.73%, Precision:48.94%, Recall: 58.67%, F1:53.37%.
Mendeley:
Average loss: 0.6949, Accuracy: 38.93%, Precision:2.28%, Recall: 63.11%, F1:4.40%.

5 Experiment on CHARBERT Model

In this section all hyperparameter are same as BERT model, and also during data preprocessing it is using char_ids, start_ids, end_ids.

5.1 experiment on GramBeddings dataset

In this section I have trained GramBeddings dataset, using 80% and 20% for testing. dataset for training contains 640000 samples and it contain benign 320002 malicious 319997.

5.1.1 results

The results of above experiment are as follows:]

- `Length of input_ids`: 1279998
- `Length of input_types`: 1279998
- `Length of input_masks`: 1279998
- `Length of char_ids`: 1279998
- `Length of start_ids`: 1279998
- `Length of end_ids`: 1279998
- `Length of labels`: 1279998
- input_ids_train.shape:(1023998, 200)
- input_types_train.shape:(1023998, 200)
- input_masks_train.shape:(1023998, 200)
- char_ids_train.shape:(1023998, 200)
- start_ids_train.shape:(1023998, 200)
- end_ids_train.shape:(1023998, 200)
- **`y_train.shape`:(1023998, 1)**
- input_ids_test.shape:(256000, 200)
- input_types_test.shape:(256000, 200)
- input_masks_test.shape:(256000, 200)
- char_ids_test.shape:(256000, 200)
- start_ids_test.shape:(256000, 200)

- end_ids_test.shape:(256000, 200)
- `y_test.shape`**:(256000, 1)**

Epoch 1:
Test set: Average loss: 0.0759, Accuracy: 97.24%, Precision: 97.50%, Recall: 96.97%, F1: 97.23%.
Epoch 2:
Test set: Average loss: 0.0577, Accuracy: 97.94%, Precision: 98.49%, Recall: 97.37%, F1: 97.93%.
Epoch 3:
Test set: Average loss: 0.0539, Accuracy: 98.13%, Precision: 99.42%, Recall: 96.82%, F1: 98.11%.
Epoch 4:
Test set: Average loss: 0.0402, Accuracy: 98.61%, Precision: 98.85%, Recall: 98.36%, F1: 98.61%.
Epoch 5: Test set: Average loss: 0.0364, Accuracy: 98.86%, Precision: 99.24%, Recall: 98.48%, F1: 98.86%.

5.1.2 Epoch vs Loss

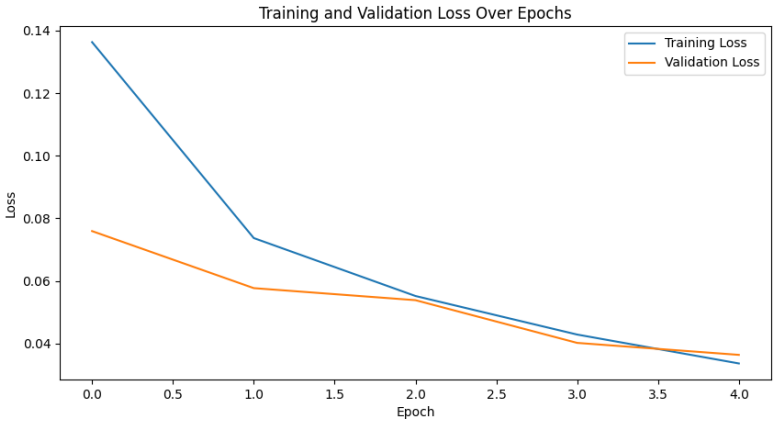

Figure 1: Epoch vs Loss graph

5.1.3 Testing

In this section I have test GramBeddings test dataset which contain 159997 samples and in it contains benign 79999 and malicious 79998.

- Length of input_ids: 159997

- Length of input_types: 159997
- Length of input_masks: 159997
- Length of char_ids: 159997
- Length of start_ids: 159997
- Length of end_ids: 159997
- Length of label: 159997

Test set: Average loss: 0.0745, Accuracy: 97.88%, Precision: 98.36%, Recall: 97.37%, F1: 97.87%.

5.1.4 ROC_curve

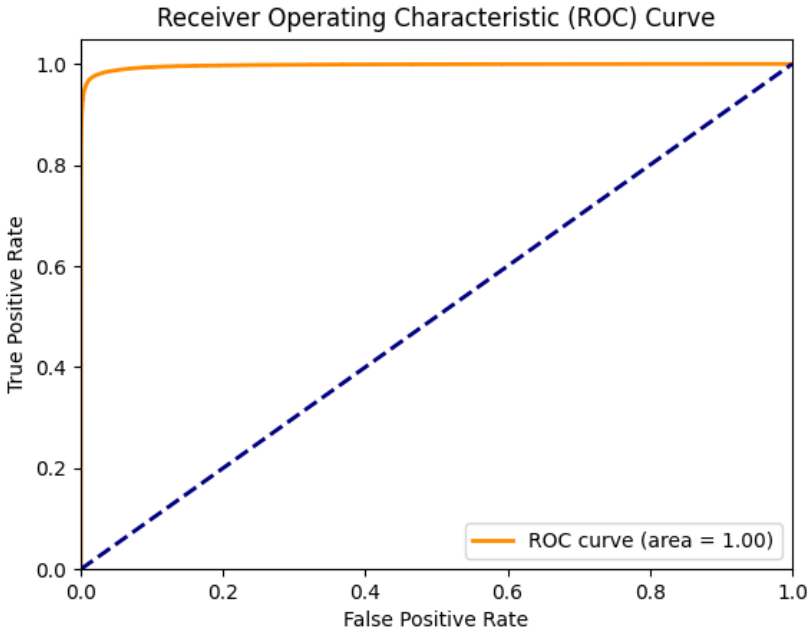

Figure 2: ROC_curve

6 Usefuls LINKS

1 Base paper REPORT

2 All 5 epoch Training history

7 Reference

W. Ma, Y. Cui, C. Si, T. Liu, S. Wang, and G. Hu, “Charbert: characteraware pre-trained language model,” arXiv preprint arXiv:2011.01513, 2020.

Ruitong Liu, Yanbin Wang, Haitao Xu, Zhan Qin, Yiwei Liu, Zheng Cao, ”Malicious URL Detection via Pretrained Language Model Guided Multi-Level Feature Attention Network”, arXiv preprint arXiv:2311.12372