GR-244 ENCRYPTED MALICIOUS TRAFFIC DETECTION USING MACHING LEARNING

Abstract

Computer network system administrators need to inspect and analyze network traffic and detect malicious communications, monitor system performance, and provide operational services. However, identifying threats contained within encrypted network traffic, which has become increasingly prevalent, poses a unique set of challenges. It is imperative to monitor this traffic or threats and malware but do so in a way that maintains privacy. This project aims to develop a machine learning-based system that can accurately detect malware communication in this setting.

Introduction

We were fortunate to enough to have been matched with Dr. Liang Zhoa to work on a web development project to inspect and analyze network traffic and detect malicious communications, monitor system performance, and provide operational services. It is imperative to monitor this traffic or threats while maintaining privacy. This project's goal was to develop a machine learning-based system that accurately detect malware communications.

.

Editing Content w/ Python

```
#Importing Datasets
malicious_dataset = pd.read_csv('malicious_flows.csv')
benign_dataset = pd.read_csv('sample_benign_flows.csv')

# Removing duplicated rows from benign_dataset (5300 rows removed)
benign_dataset = benign_dataset[benign_dataset.duplicated(keep=False) == False]

# Combining both datasets together
all_flows = pd.concat([malicious_dataset, benign_dataset])

# Reducing the size of the dataset to reduce the amount of time taken in training model
reduced_dataset = all_flows.sample(25000)

#removing nan values
df = reduced_dataset.drop(reduced_dataset.columns[np.isnan(reduced_dataset).any()], axi

reduced_y=df['isMalware']
reduced_x=df.drop(['isMalware'], axis=1)

#Showing information
reduced_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25000 entries, 11055 to 10194
Columns: 584 entries, Src_Port to 99
dtypes: float64(576), int64(1971)
memory usage: 96.3 MB
```

```
# Splitting datasets into training and test data
x_train, x_test, y_train, y_test = train_test_split(reduced_x, reduced_y, test_size=0.2

# Training a random forest classifier
rf_clf=RandomForestClassifier(max_depth=100)
rf_clf.fit(x_train, y_train)
rf_prediction = rf_clf.predict(x_test)
rf_score=accuracy_score(y_test, rf_prediction)
print('Random Forest Classifier Accuracy score:', accuracy_score(y_test, rf_prediction
```

```
Random Forest Classifier Accuracy score:  0.9984
```

```
#function to plot most import features of random forest model
def plot_feature_importance(importance,names,model_type):

    #Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sn.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    plt.ylim(0, 50)
    #Add chart labels
    plt.title(model_type + 'FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
```

```
plot_feature_importance(rf_clf.feature_importances_,x_train.columns,'RANDOM FOREST')
```

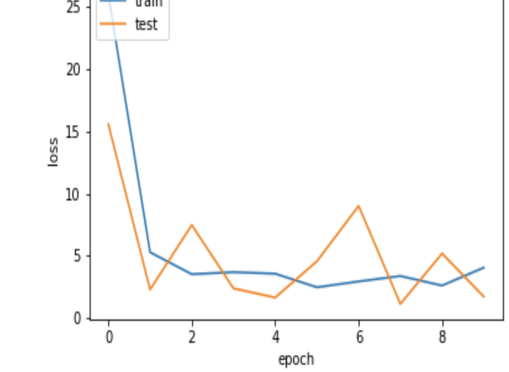
Technologies

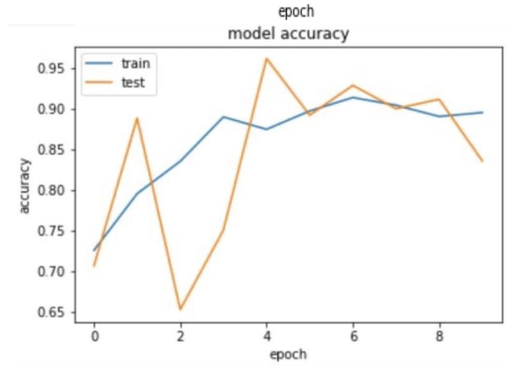
Results

Models	Accuracy Score
Decision Tree	0.9996
Logistic Regression	0.939
Random Forest	0.9998
Neural Network	0.9998

Predicted

Actual	0	1
0	2609	217
1	98	2076





Conclusions

The final version of the product was delivered on November 17th, 2022, which includes a simple user interface that accurately displays the accuracy and loss from our train and test data. Although there were some complications and issues that led this project off track in the early stages, the client was overall satisfied by the product and all the requirements initially stated by the client were successfully met.
Based upon our observations, Neural Network and Random Forest are most accurate.

Acknowledgments

1. Dr. Jack Zheng (Advisor)
2. Dr. Lian Zhoa (Client)

Contact Information

https://malicioustraffic.wixsite.com/malicious-traffic-de

References

Diong, W. (2020, August 14). *Walterdiong/TLS-malware-detection-with-machine-learning: Leveraging machine learning to detect TLS based malware in encrypted traffic without decryption*. GitHub. Retrieved November 14, 2022, from https://github.com/WalterDiong/TLS-Malware-Detection-with-Machine-Learning

Johnson, Daniel. "Artificial Neural Network Tutorial with Tensorflow Ann Examples." *Guru99*, 17 Sept. 2022, https://www.guru99.com/artificial-neural-network-tutorial.html.

KENNESAW STATE UNIVERSITY
COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING

Author(s)-
Megen Cochran, Skylar Story, Tatiana Brown, Simisola Babtunde