Kubernetes Intrusion Detection System

1. Introduction

This document provides technical details for the neural network-based intrusion detection system (IDS) designed for Kubernetes environments. The system uses deep learning to analyze network traffic patterns and identify potential security threats in containerized applications.

2. Dataset Analysis

2.1 Dataset Overview

The Kube-IDS0 dataset consists of two main components:

1. **Bank of Anthos (BoA) Dataset**
   - Path: `/kaggle/input/kube-ids0/boa_dataset/processed/boa_dataset_ml_ready_frontend_microservice.csv`
   - Contains network traffic from a simulated banking application
   - Includes both normal operations and simulated attack traffic
2. **DVWA Dataset**
   - Path: `/kaggle/input/kube-ids0/dvwa_dataset/processed/dvwa_dataset_ml_ready.csv`
   - Derived from the Damn Vulnerable Web Application
   - Contains deliberately vulnerable web application traffic

2.2 Feature Description

The datasets contain various network traffic features including:

- Source and destination IP addresses (encoded)
- Port information
- Protocol details
- Traffic volume metrics
- Connection duration
- TCP flags
- Inter-arrival time statistics

The target variable is the 'label' column which classifies traffic as normal or identifies specific attack types.

3. Model Architecture

3.1 Neural Network Structure

The model is implemented as a sequential neural network with the following components:

```
def create_model(trial):
    model = Sequential()
    model.add(Dense(trial.suggest_int("units1", 64, 256), activation='relu',
input_shape=(X_train.shape[1],)))
    model.add(Dropout(trial.suggest_float("dropout1", 0.1, 0.5)))
    model.add(Dense(trial.suggest_int("units2", 32, 128), activation='relu'))
    model.add(Dropout(trial.suggest_float("dropout2", 0.1, 0.5)))
    model.add(Dense(len(np.unique(y)), activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model
```

3.2 Optimal Hyperparameters

After optimization with Optuna, the best hyperparameters were determined to be:

- First hidden layer: 230 units
- First dropout rate: 0.159
- Second hidden layer: 89 units
- Second dropout rate: 0.197
- Batch size: 43

These parameters yielded the highest validation accuracy during hyperparameter optimization.

4. Data Preprocessing

4.1 Data Cleaning

```
# Handle missing values
df.fillna(0, inplace=True)
```

Missing values in the dataset are replaced with zeros.

4.2 Feature Engineering

The main preprocessing steps include:

1. Label encoding for categorical labels
2. `label_encoder = LabelEncoder()`
3. `df['label'] = label_encoder.fit_transform(df['label'])`
4. Feature standardization
5. `scaler = StandardScaler()`

```
6. X_train = scaler.fit_transform(X_train)
7. X_test = scaler.transform(X_test)
```

4.3 Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

The data is split with 80% for training and 20% for testing, with a fixed random seed to ensure reproducibility.

5. Training Process

5.1 Hyperparameter Optimization

Optuna is used for hyperparameter tuning with the following search space:

- First layer units: 64-256
- Second layer units: 32-128
- Dropout rates: 0.1-0.5
- Batch size: 16-64

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=20)
```

5.2 Training Configuration

The final model is trained with:

- Optimizer: Adam
- Loss function: Sparse Categorical Crossentropy
- Metrics: Accuracy
- Epochs: 50
- Validation split: 20% of training data

6. Model Evaluation

The final model achieved 100% accuracy on the test set, indicating excellent performance in detecting intrusion patterns.

```
Final Model Accuracy: 1.00
```

This suggests the model can effectively distinguish between normal traffic and various attack types in Kubernetes environments.

7. Model Persistence

The trained model is saved in H5 format:

```
final_model.save("models/neural_network_model.h5")
```

8. Implementation and Deployment

8.1 Model Integration

To integrate the model into a Kubernetes security monitoring system:

1. Deploy as a sidecar container that analyzes network traffic
2. Connect to Kubernetes network interfaces using packet capture tools
3. Preprocess captured traffic in real-time
4. Make predictions and alert on suspicious activity

8.2 Production Considerations

- **Performance**: The model is lightweight enough to run in resource-constrained environments
- **Monitoring**: Implement performance and drift monitoring
- **Retraining**: Schedule periodic retraining as attack patterns evolve
- **Alerts**: Connect to notification systems for security alerts

9. Limitations and Future Work

9.1 Limitations

- The dataset may not capture all possible attack vectors in real-world environments
- Deep learning models require retraining as new attack patterns emerge
- The model doesn't provide intrinsic explainability for its decisions

9.2 Future Enhancements

- Implement explainability techniques like SHAP or LIME
- Explore unsupervised anomaly detection for zero-day attacks
- Add recurrent layers to capture temporal patterns in traffic
- Create ensemble methods combining multiple detection techniques

10. References

1. Kube-IDS0 Dataset: https://www.kaggle.com/datasets/redamorsli/kube-ids0
2. TensorFlow Documentation: https://www.tensorflow.org/api_docs
3. Optuna Hyperparameter Optimization: https://optuna.org/
4. Kubernetes Security: https://kubernetes.io/docs/concepts/security/

Submitted By:

Team Name: codersincrocs
(PES University)

- Surya P S
- Sri Vidya
- Siri N Shetty
- Suprith S