Report for the Course on Cyber-Physical Systems and IoT Security

Title of the reference paper: Fingerprinting Electronic Control Units for Vehicle Intrusion Detection

Name of who is involved in the project: Mostafizor Rahman (2141829)

Source code: https://github.com/MostafizorRahman/can-bus-mid-project/tree/main

Abstract

The objective of this project is to address the increasing cybersecurity challenges in Controller Area Network (CAN) transit systems by creating a machine learning-based intrusion detection framework. The initiative distinguishes between normal and intrusive traffic by fingerprinting Electronic Control Units (ECUs) on the basis of timing and frequency patterns. Using features such as id_count and timestamp_diff, a dataset containing CAN bus message records was analyzed. An array of machine learning models, including Logistic Regression, SVM, Gradient Boosting, and Random Forest, were assessed. The Random Forest model's robustness for intrusion detection was demonstrated by its greatest ROC-AUC score of 83%.

Introduction

The communication systems of vehicles are becoming increasingly vulnerable to cybersecurity threats as they become more interconnected. The CAN bus, which is extensively employed in automotive networks, is susceptible to attacks such as spoofing, inundation, and replay due to its inherent lack of security measures. The objective of this project is to improve the security of CAN buses by utilizing machine learning to identify anomalies that suggest intrusions.

Objectives

This project aims to:

- Create a system that is based on machine learning in order to identify any intrusions that may occur within the CAN bus network of linked vehicles.

- When doing ECU fingerprinting, it is important to make use of timing (timestamp_diff) and frequency (id_count) aspects in order to discern between normal traffic and malicious traffic.

- The performance of a number of different machine learning models, such as Random Forest, Gradient Boosting, Support Vector Machine (SVM), and Logistic Regression, should be evaluated.

- Designing and implementing the predictive model, including feature engineering, model training and performance evaluation.

- Evaluating the efficacy and efficiency of the developed model using the appropriate evaluation metrics, such as F1 score, recall, and accuracy in conjunction with numerous comparisons to other methods.

- With the use of hyperparameter tuning, optimize the performance of the model that has the best overall performance.

- By analyzing and comparing all four models using methodologies will determine which model is the most effective

System Setup

Environment and Tools Used:

- **Programming Language: Python**
- **Libraries:**
  - pandas, numpy for data manipulation.
  - matplotlib, seaborn for visualizations.
  - scikit-learn for machine learning model implementation and evaluation.
- **Hardware:** Standard laptop with a multi-core processor and 8GB RAM

Dataset:

- **Source:** The dataset, downloaded from Kaggle, consisting of message logs with the following fields:
  - TS (Timestamp): Time of message transmission.
  - ID1 (CAN Identifier): Message identifier.
  - DLC fields (Payload): Data transmitted in the CAN message.
  - target: Label indicating normal (0) or intrusive traffic (1-3).

- **Dataset Properties:**
  - Balanced class distribution with equal instances for each target class (0, 1, 2, 3).
  - Total size: Approximately 8.5 million rows.

Experiments

Methodology:

1. **Feature Engineering:**
   - Derived features include:
     - timestamp_diff: Time difference between consecutive messages.
     - id_count: Frequency of each message identifier.
   - Features normalized using StandardScaler.

2. **Model Training:**
   - Models Trained:
     - Random Forest
     - Gradient Boosting
     - Support Vector Machine (SVM)
     - Logistic Regression
   - Hyperparameter Tuning:
     - Random Forest optimized using RandomizedSearchCV.
     - Parameters tuned: n_estimators, max_depth, and min_samples_split.

3. **Evaluation Metrics**:
   - Accuracy
   - Precision
   - Recall
   - F1-Score
   - ROC-AUC

Key Results:

Random Forest:

- Accuracy: 58%
- ROC-AUC: 83%
- **Confusion Matrix Insights:**
  - Strong performance for Class 0 and Class 3.
  - Misclassifications observed for Class 2 due to feature overlaps**.**

Gradient Boosting:

- Accuracy: 55%
- ROC-AUC: 80%
- **Confusion Matrix Insights:**
  - Better class balance than Random Forest but lower overall precision and recall.

SVM:

- Accuracy: 51%
- ROC-AUC: 74%
- **Observation:**
  - Struggled with large-scale data due to computational inefficiency.

Logistic Regression:

- Accuracy: 49%
- ROC-AUC: 76%
- **Observation:**
  - Performed poorly due to non-linear data patterns.

Results and Discussion

Figures and Tables:

1. **Feature Distribution:**
   - The distribution of timestamp_diff shows a sharp peak around zero, indicating consistent message intervals.

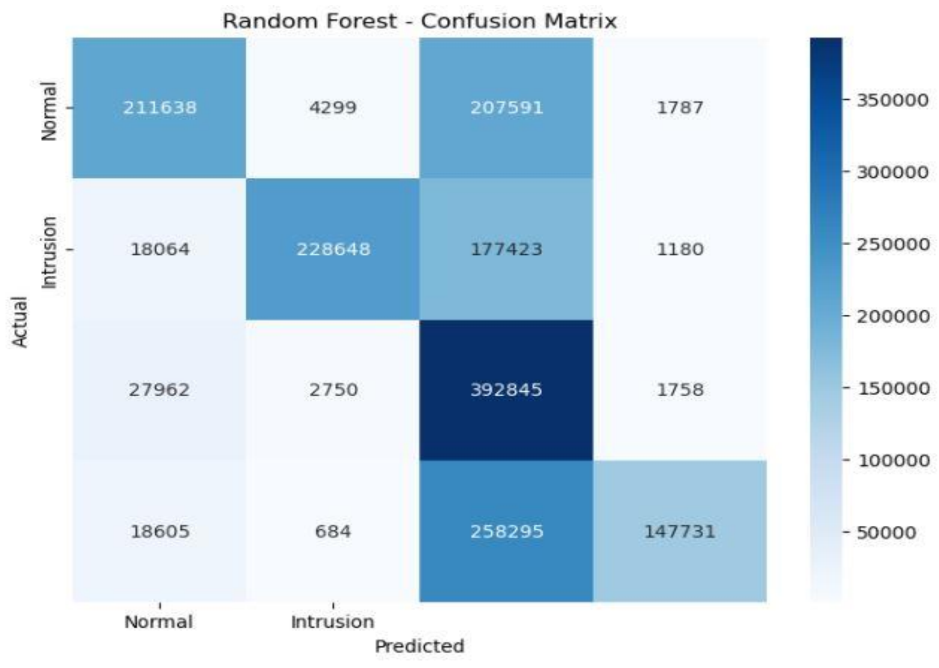
- Outliers in the distribution likely correspond to anomalies or attacks.

2. **Model Comparison:**

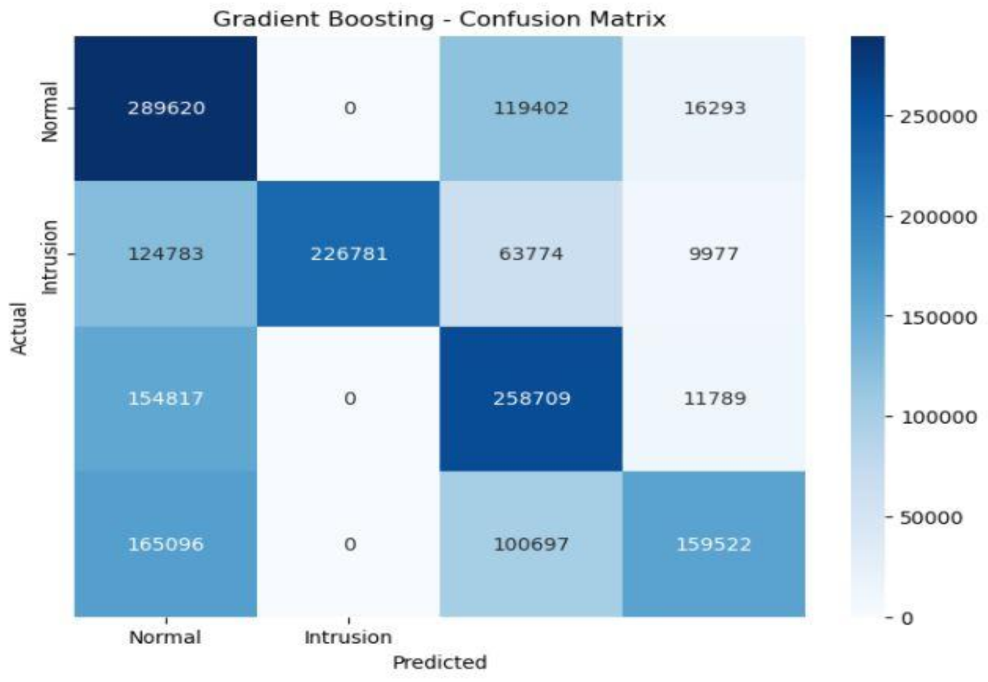
Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Random Forest	58%	77%	58%	59%	83%
Gradient Boosting	55%	67%	55%	56%	80%
SVM	51%	76%	51%	51%	74%
Logistic Regression	49%	65%	49%	49%	76%

3. **Confusion Matrices:**

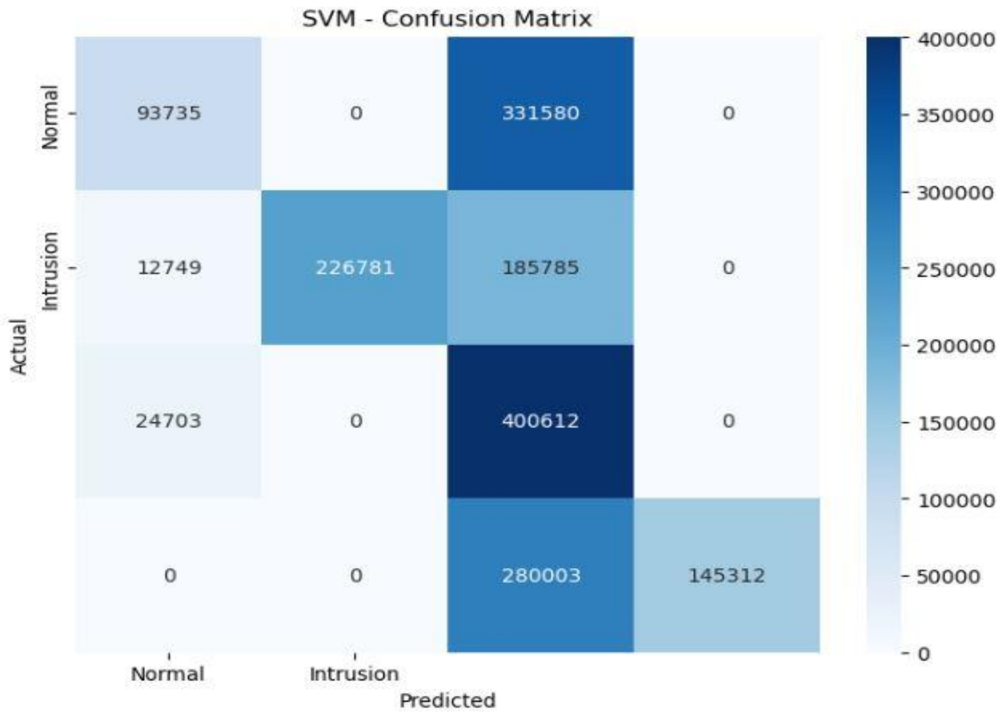
- **Random Forest Confusion Matrix:**



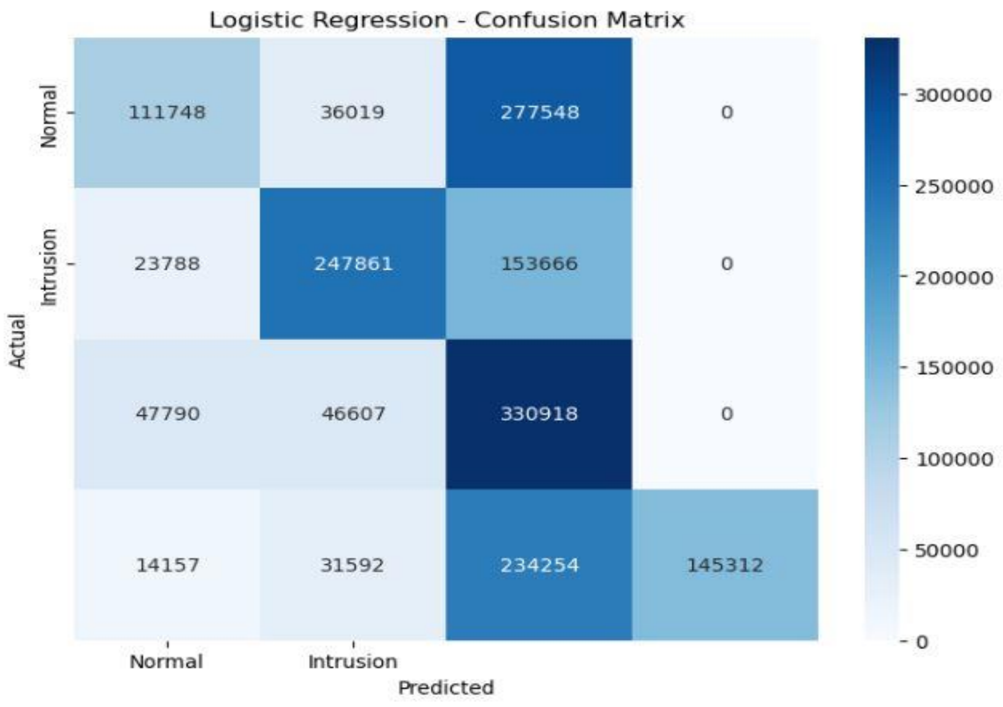
- **Gradient Boosting Confusion Matrix:**



- **SVM Confusion Matrix:**

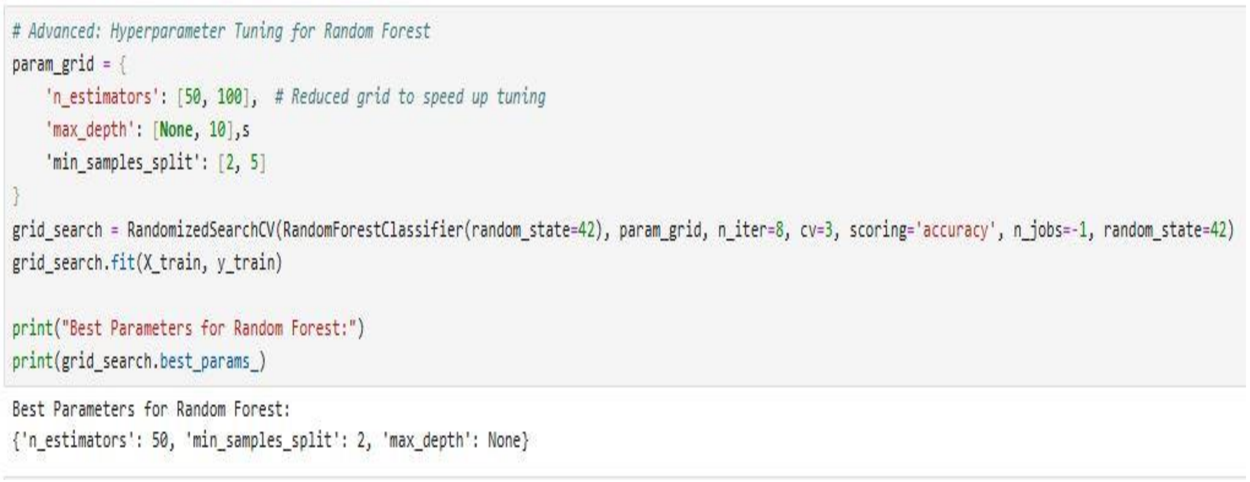


- **Logistic Regression Confusion Matrix:**



4. **Hyperparameter Tuning:**

- **Best Parameters for Random Forest:**
    - n_estimators: 50
    - max_depth: None
    - min_samples_split: 2



```
# Advanced: Hyperparameter Tuning for Random Forest
param_grid = {
    'n_estimators': [50, 100],  # Reduced grid to speed up tuning
    'max_depth': [None, 10],s
    'min_samples_split': [2, 5]
}
grid_search = RandomizedSearchCV(RandomForestClassifier(random_state=42), param_grid, n_iter=8, cv=3, scoring='accuracy', n_jobs=-1, random_state=42)
grid_search.fit(X_train, y_train)

print("Best Parameters for Random Forest:")
print(grid_search.best_params_)
```

```
Best Parameters for Random Forest:
{'n_estimators': 50, 'min_samples_split': 2, 'max_depth': None}
```

Discussion:

1. **Model Performance:**
    - Random Forest outperformed other models with the highest ROC-AUC score of 83%.
    - Gradient Boosting provided more balanced predictions but at the cost of slightly lower accuracy.
    - SVM and Logistic Regression failed to handle the complexity of the dataset effectively.
2. **Challenges:**
    - Misclassifications in Class 2 suggest overlapping patterns in the feature space.
    - Feature engineering focused on timing and frequency; additional features like payload entropy could improve separability.
3. **Recommendations for Improvement:**
    - Introduce advanced sequence-based models like LSTMs or Transformers.
    - Expand the parameter grid for more exhaustive hyperparameter tuning.

Conclusion

1. **Key Findings:**
    - Random Forest performed the best, highlighting the utility of ensemble approaches for intrusion detection.
    - Feature engineering with timestamp_diff and id_count was effective, however further improvement is required for improved class separation.
2. **Future Work:**
    - Explore deep learning models for capturing temporal patterns.
    - Implement the system in a real-time environment to evaluate scalability and latency.