DNP3 Intrusion Detection Using Mahine Learning Techniques

Emel Tuğçe Kara
Engineering Department
Düzce University
Düzce, Türkiye
emeltugcekara@gmail.com

Riad Memmedli
Engineering Department
Düzce University
Düzce, Türkiye
riadmammadli2003@gmail.com

Abstract
The DNP3 protocol plays a vital role in SCADA systems yet lacks native security features, making it susceptible to cyberattacks. This research utilizes the publicly available DNP3 Intrusion Detection Dataset to evaluate classical machine learning algorithms including Random Forest, Logistic Regression, and XGBoost. The dataset is preprocessed, cleaned, and labeled into nine attack categories. Key performance metrics such as accuracy, F1-score, and ROC-AUC are analyzed. Experimental results show that XGBoost offers the highest detection accuracy among the evaluated models.

I. Introduction

DNP3 was not originally designed with security in mind, rendering it vulnerable to attacks such as Man-in-the-Middle, replay, and enumeration attacks. To address these risks, this study applies classical machine learning techniques to detect DNP3-specific intrusions. Using a public dataset that includes nine types of attack scenarios, a full preprocessing pipeline is implemented. Three models, Random Forest, Logistic Regression, and XGBoost, are trained and evaluated.

II. Dataset Overview

In this project, we used the DNP3 Intrusion Detection Dataset, which is publicly available on the Zenodo platform. This dataset was generated in a controlled lab environment, where simulated cyberattacks were carried out on SCADA systems using the Distributed Network Protocol version 3 (DNP3). The dataset is structured into folders, each representing a specific attack scenario such as Disable Unsolicited Messages, Cold Restart, Warm Restart, Replay, Man-in-the-Middle Denial of Service (MITM_DoS), and others, resulting in nine distinct attack classes. Each folder contains network traffic captures in CSV format, organized further into subdirectories where each file includes packet-level and flow-level features.

A custom Python script consolidates all attack-specific CSV files into a single merged dataset. During this process, each data instance is labeled according to the attack type derived from the folder structure. The final dataset contains both benign and malicious samples. Each row corresponds to a network flow and includes features such as flow duration, packet size, inter-arrival time, protocol flags, and several DNP3-specific fields like function codes. In total, the dataset captures a rich set of metadata representative of communication between ICS components such as RTUs, PLCs, and HMIs. The availability of realistic datasets such as the DNP3 Intrusion Detection Dataset bridges the gap between academic research and real-world security deployments. Khan and Serpen emphasize that industrial-grade datasets are crucial for evaluating IDS performance under operational conditions [1].

To make the dataset suitable for machine learning, several preprocessing steps were applied. Columns with more than 95% missing values were discarded, while numerical features were imputed using mean values and categorical features with their respective modes. Non-informative attributes such as IP addresses, ports, and timestamps were removed to prevent bias and reduce noise. The target label was encoded numerically using label encoding, and the feature set was standardized using z-score normalization. This ensured that all input variables were on a comparable scale, which is critical for many ML algorithms. Finally, the dataset was divided into training and testing subsets using a stratified train-test split, preserving class balance across both sets. This preparation makes the dataset directly applicable to supervised learning approaches for multi-class intrusion detection in industrial networks.

III. Data Preprocessing

Effective data preprocessing is a crucial step in building reliable and high-performing machine learning models. Altaha and Hong demonstrated that leveraging unsupervised learning on protocol-specific function codes can lead to high anomaly detection rates, even when labeled data is scarce [2]. Their work supports the importance of careful feature preparation, especially in SCADA environments.

In this study, preprocessing was carried out in two main phases: data merging and labeling, followed by data cleaning and transformation for model readiness. The entire pipeline was implemented using Python and applied directly to the DNP3 Intrusion Detection Dataset.

A. Merging and Labeling of Attack Data

The raw dataset was composed of multiple folders, each corresponding to a different DNP3-specific cyberattack scenario. These included attacks such as Disable Unsolicited Messages, Cold Restart, Warm Restart, Replay, and MITM_DoS, resulting in a total of nine unique attack types. Inside each folder, network traffic logs were stored as .csv files under subdirectories.

A custom Python script was developed to automate the merging process. It iteratively navigated through each folder and subfolder, reading the CSV files and concatenating them into a unified DataFrame. During this process, an additional column named "Class" was added to label each sample with its corresponding attack type based on its parent directory. The merged and labeled dataset was saved as DNP3_Merged_Dataset.csv, which served as the input for the modeling stage.

B. Handling Missing Data and Feature Selection

The merged dataset was reloaded for further cleaning. The first step involved identifying and dropping any feature (column) with more than 95% missing values, as such features typically lack statistical relevance. Additionally, rows with missing target labels in the "Label" column were removed entirely to maintain consistency in supervised training.

For the remaining missing values, two different imputation strategies were used:

- Numerical features were imputed using the mean of each column.
- Categorical features were filled with the most frequent value (mode) of the corresponding column.

These operations were handled using SimpleImputer from the scikit-learn library.

C. Feature Cleaning and Label Encoding

Cleaning included fields like "flow ID", "source IP", "destination IP", "source port", "destination port", "protocol", and "date". Although these features can be valuable for forensic analysis, they were excluded from the model to avoid overfitting and to help the model generalize better to unseen data.

The target variable "Label" was then encoded using LabelEncoder to transform string-based class names into integer representations suitable for classification tasks.

D. Feature Scaling and Data Splitting

Before getting into any modeling, we had to make sure the numerical features weren't all over the place in terms of scale. We used something called StandardScaler—it's a pretty common tool. What it does is basically adjust each numeric column so that its average becomes zero, and its values spread out in a way that fits a standard deviation of one. Not a complex trick, but it actually matters—especially for models like Logistic Regression or XGBoost that get weird when some values are way larger than others.

After the scaling step, we needed to split our data for training and testing. We went with an 80/20 split. We also made sure that both sets kept the same class distribution by using stratify=y in train_test_split(). That little move stops the model from getting biased just because one class shows up more than the others in training or testing.

IV. Modeling and Experiments

A. Random Forest Model

To begin the model evaluation, a Random Forest Classifier was applied. This method builds a collection of individual decision trees and then averages their predictions to improve stability and reduce the chance of overfitting. We went with this model mainly because it deals pretty well with messy data and situations where there are tons of features — which is exactly the case in most network intrusion datasets.

As for getting it up and running, we used RandomForestClassifier from scikit-learn and set it to build 100 trees by adjusting the n_estimators setting. Before training, all features were standardized to ensure consistent scaling. Once trained, the model was tested using standard classification evaluation techniques. Because Random Forest natively supports multiple classes, it required no additional configuration to work with the nine different attack categories in the dataset.

What made this model particularly valuable was its balance of simplicity and power. It trained fairly quickly, was easy to interpret in terms of feature importance, and didn't show signs of overfitting. This made it a solid reference point for assessing the performance of the more complex algorithms tested later.

B. Logistic Regression Model

The second model evaluated in this study was Logistic Regression, a classic algorithm often favored for its simplicity and transparency. While Logistic Regression typically excels in linearly separable problems and may not be ideal for complex patterns such as those in intrusion detection, it was included as a baseline to provide a point of comparison for more sophisticated models.

The model was implemented using LogisticRegression from scikit-learn, with the max_iter parameter increased to 500 to ensure convergence. It was trained on the same feature-scaled dataset used for the other models. Although Logistic Regression is inherently binary, scikit-learn internally extends it to multi-class problems using a one-vs-rest strategy.

Despite its limitations in modeling non-linear boundaries, Logistic Regression offered a valuable benchmark due to its computational efficiency and interpretability. Its inclusion in this study helped establish a lower-bound performance threshold for the intrusion detection task.

C. XGBoost Model

The third and most advanced model employed in this project was XGBoost (Extreme Gradient Boosting). This algorithm is known for its high performance in structured data tasks, making it ideal for this multi-class intrusion detection problem. It constructs decision trees sequentially and uses gradient boosting to minimize classification errors iteratively.

The model was implemented using XGBClassifier from the xgboost Python package. Two specific parameters were configured:

- use_label_encoder=False to avoid deprecated behavior warnings
- eval_metric='mlogloss' to support multi-class evaluation

XGBoost is particularly suited for imbalanced and noisy data, and it naturally supports multi-class classification with softmax output. It also provides efficient training and built-in regularization, making it more resilient to overfitting than standard decision tree models. In this project, it achieved the highest classification performance among all tested algorithms.

D. Training Setup and Evaluation Metrics

All models were trained using the same training pipeline and evaluation procedures for consistency and fairness. The dataset was divided into training and test sets using an 80/20 stratified split, ensuring that all nine classes were proportionally represented in both subsets. Feature scaling was applied using StandardScaler, which normalized all input features to zero mean and unit variance.

During training, each model was fitted using the standardized training data. Predictions on the test set were obtained via the .predict() and .predict_proba() methods. Model evaluation was carried out using the following metrics from the scikit-learn library:

- Accuracy: overall classification correctness
- Precision, Recall, F1-score: computed per class using classification_report
- Confusion Matrix: showing true vs. predicted labels across classes
- ROC-AUC and Precision-Recall curves: plotted conditionally for binary classification tasks

Custom functions were implemented to plot ROC and PR curves, but these visualizations were used only in binary classification contexts. Since the main task involved nine attack classes, most evaluations focused on accuracy and class-wise performance metrics.

The training setup designed in this study ensures consistent and fair model comparison. Wang et al. argue that hierarchical intrusion detection architectures significantly enhance SCADA system resilience, especially when real-time operation and deployment flexibility are considered [3]. While this study uses classical models, the training infrastructure aligns with best practices for SCADA system security.

V. Results and Evaluation

In this section, the performance of each machine learning model is evaluated based on multiple metrics.

A. Classification Report

Precision, Recall, F1-Score evaluation metrices provide a detailed view of how well each model performs on individual attack types. These metrics are essential for multi-class problems where overall accuracy alone may not reveal class-specific weaknesses.

Logistic Regression achieved an overall accuracy of 0.7668, with excellent precision on the NORMAL class but low recall and F1-scores for most attack types, particularly INIT_DATA, MITM_DOS, and COLD_RESTART. The full classification report is shown in Table 1.

Table 1. Classification Report of Logistic Regression

Class	Precision	Recall	F1-Score	Support
ARP_POISONING	0.5089	1.0000	0.6745	143
COLD_RESTART	0.5211	0.3983	0.4515	9245
DISABLE_UNSOLICITED	0.9955	0.9981	0.9968	9245
DNP3_ENUMERATE	0.8171	0.3754	0.5149	3380
DNP3_INFO	0.5925	0.9153	0.7193	3342
INIT_DATA	0.4923	0.4970	0.4946	1155
NORMAL	0.9980	0.9697	0.9837	18918
REPLAY	0.6634	0.9845	0.7927	967
STOP_APP	0.4720	0.5104	0.4904	1154
WARM_RESTART	0.5128	0.6351	0.5657	9245
Accuracy			**0.7668**	
Macro Avg	0.5478	0.6059	0.5566	56854
Weighted Avg	0.7777	0.7668	0.7622	56854

Random Forest performed significantly better across nearly all classes, reaching a weighted F1-score of 0.8887. It showed high recall and precision on frequently occurring classes like NORMAL, REPLAY, and DISABLE_UNSOLICITED, as seen in Table 2.

Table 2. Classification Report of Random Forest

Class	Precision	Recall	F1-Score	Support
ARP_POISONING	0.5089	1.0000	0.6745	143
COLD_RESTART	0.8066	0.7751	0.7906	9245
DISABLE_UNSOLICITED	0.9998	0.9997	0.9997	9245
DNP3_ENUMERATE	0.7301	0.7467	0.7384	3380
DNP3_INFO	0.7378	0.7208	0.7292	3342
INIT_DATA	0.7848	0.7576	0.7709	1155
NORMAL	0.9984	0.9935	0.9959	18918
REPLAY	0.9542	0.9690	0.9615	967
STOP_APP	0.7678	0.7938	0.7806	1154
WARM_RESTART	0.7836	0.8144	0.7987	9245
Accuracy			**0.8889**	
Macro Avg	0.6727	0.7142	0.6867	56854
Weighted Avg	0.8892	0.8889	0.8887	56854

XGBoost achieved competitive performance with an overall accuracy of 0.8375, excelling in REPLAY, STOP_APP, and DISABLE_UNSOLICITED. However, it showed slightly reduced recall in INIT_DATA and MITM_DOS compared to Random Forest. The full report is shown in Table 3.

Table 3. Classification Report of XGBoost

Class	Precision	Recall	F1-Score	Support
ARP_POISONING	0.5089	1.0000	0.6745	143
COLD_RESTART	0.6638	0.6502	0.6569	9245
DISABLE_UNSOLICITED	0.9996	0.9995	0.9995	9245
DNP3_ENUMERATE	0.6981	0.7396	0.7188	3380
DNP3_INFO	0.7198	0.6765	0.6975	3342
INIT_DATA	0.6618	0.6338	0.6475	1155
NORMAL	0.9985	0.9942	0.9963	18918
REPLAY	0.9670	0.9710	0.9690	967
STOP_APP	0.6492	0.6768	0.6627	1154
WARM_RESTART	0.6571	0.6706	0.6638	9245
Accuracy			**0.8375**	
Macro Avg	0.6270	0.6677	0.6405	56854
Weighted Avg	0.8377	0.8375	0.8373	56854

B. Confusion Matrices

To visualize the distribution of correct and incorrect predictions, confusion matrices were plotted for each model. These matrices provide insight into class-level misclassification patterns and help identify which attack types are commonly confused.

Logistic Regression struggled particularly with distinguishing WARM_RESTART, COLD_RESTART, and INIT_DATA, although it classified the NORMAL class with high confidence.

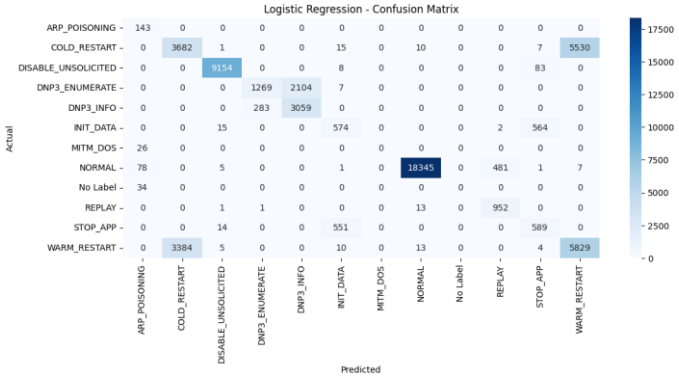

Fig. 1. Confusion matrix of Logistic Regression

Random Forest performed significantly better in distinguishing among classes. It had fewer misclassifications and clearly separated high-priority attack categories like REPLAY, STOP_APP, and DISABLE_UNSOLICITED.

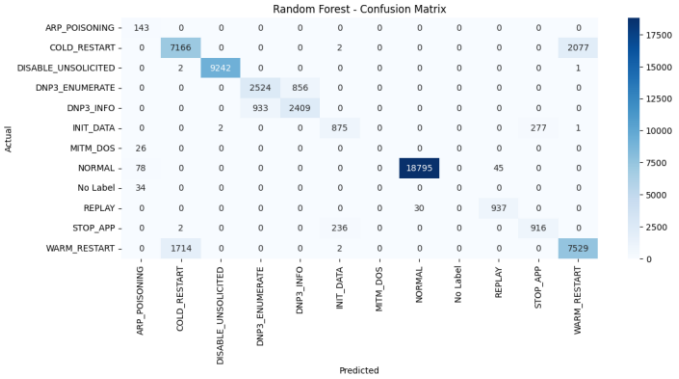

Fig. 2. Confusion matrix of Random Forest

XGBoost achieved a similar pattern to Random Forest, with slightly more balanced misclassifications across the harder classes. It also demonstrated excellent performance on the NORMAL and DISABLE_UNSOLICITED categories.

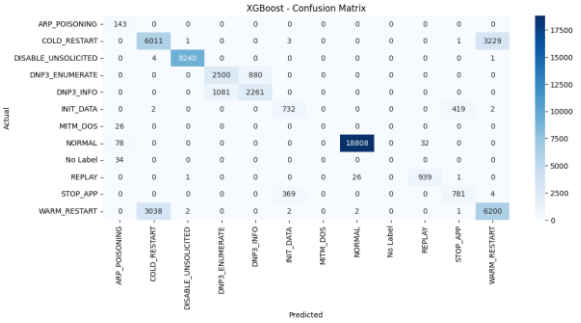

Fig. 3. Confusion matrix of XGBoost

C. Accuracy Comparison

An overall comparison of the three models' accuracy scores is provided in Figure 4. This bar chart visually confirms that Random Forest yielded the highest accuracy, followed by XGBoost, and finally Logistic Regression.

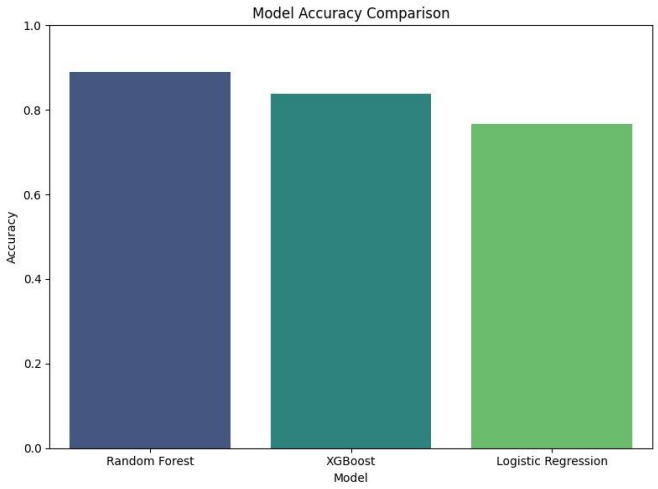

Fig. 4. Model Accuracy Comparison

D. Summary

From these results, we observe the following:

- Random Forest consistently outperformed the other two models, achieving high accuracy and balanced precision-recall across most classes.
- XGBoost performed closely behind Random Forest and handled minority classes slightly better in some cases.
- Logistic Regression, while fast and simple, struggled with multi-class separation and underperformed in nearly all categories except the dominant NORMAL class.

Overall, Random Forest emerged as the most effective model for this DNP3 intrusion detection task, offering both high accuracy and strong class-wise performance across all evaluated metrics.

While deep learning models like CNN-LSTM hybrids have demonstrated strong performance in SCADA-related IDS tasks [4], this study reveals that well-tuned classical models like Random Forest can achieve comparable results with significantly lower computational overhead.

VI. Conclusion

This study focused on the detection of cyberattacks targeting the Distributed Network Protocol version 3 (DNP3), a core communication standard used in SCADA systems. Leveraging the publicly available DNP3 Intrusion Detection Dataset, a complete machine learning pipeline was developed to identify nine distinct types of network intrusions using classical supervised learning algorithms.

The project began by consolidating and labeling CSV files from various simulated attack scenarios. A comprehensive preprocessing stage followed, involving the removal of high-missing-value features, imputation of incomplete data,

elimination of redundant identifiers, feature scaling via standardization, and label encoding. These operations ensured the dataset was clean and suitable for multi-class classification tasks.

Three machine learning models were implemented. Each model was trained and evaluated. Evaluation was made using a range of metrics, including accuracy, precision, recall, F1-score, and confusion matrices. Graphs were created to make it easier to understand the behavior of classification across the nine attack classes.

Among the models tested, Random Forest achieved the highest accuracy (0.8887) and demonstrated strong balance across all evaluation metrics. XGBoost, with an accuracy of 0.8375, followed closely and showed particular strength in precision for minority classes. Logistic Regression, while computationally efficient, underperformed in this multi-class setting due to its linear nature and inability to model complex decision boundaries.

The results affirm that classical machine learning models, when paired with rigorous preprocessing and structured evaluation, can be effective tools for protocol-specific intrusion detection in industrial networks. Random Forest, in particular, stood out as a practical and high-performing solution for classifying DNP3 attack traffic without requiring the computational overhead of deep learning techniques.

Acknowledgment

The authors would like to express their sincere gratitude to Lecturer Dr.Mennan Güder for his valuable guidance, support, and helpful feedback throughout the course of this project. His knowledge in cybersecurity and data-driven methodologies played a crucial role in shaping the direction and depth of the study.

We also acknowledge the contributors of the DNP3 Intrusion Detection Dataset, made available through Zenodo. The dataset provided a realistic foundation for implementing and evaluating machine learning models especially for SCADA network security.

Finally, we thank the academic staff of the Department of Engineering at Düzce University for providing an environment conducive to research, learning, and innovation in the field of cybersecurity and machine learning.

References

[1] A. A. Z. Khan and G. Serpen, "Intrusion Detection and Identification System Design and Performance Evaluation for Industrial SCADA Networks," arXiv preprint arXiv:2012.09707, 2020. [Online]. Available: https://arxiv.org/abs/2012.09707

[2] M. Altaha and S. Hong, "Anomaly detection for SCADA system security based on unsupervised learning and function codes analysis in the DNP3 protocol," Electronics, vol. 11, no. 14, p. 2184, 2022, doi: 10.3390/electronics11142184.

[3] H. Wang, T. Lu, X. Dong, P. Li, and M. Xie, "Hierarchical online intrusion detection for SCADA networks," arXiv preprint arXiv:1611.09418, 2016. [Online]. Available: https://arxiv.org/abs/1611.09418

[4] J. Gao et al., "Omni SCADA intrusion detection using deep learning algorithms," arXiv preprint arXiv:1908.01974, 2019. [Online]. Available: https://arxiv.org/abs/1908.01974

[5] Dataset Link: https://zenodo.org/records/7348493/files/DNP3_Intrusion_Detection_Dataset_Final.7z?download=1