DDoS Attack Detection Using Machine Learning (XG Boost Algorithm)

BITS Pilani
Pilani Campus

Submitted By:

- Rakshita Goel(2022B1A71108P)
- Arnav Dham (2022A7PS1182P)

INTRODUCTION

- Distributed Denial-of-Service (DDoS) attacks have grown in scale and complexity, disrupting businesses, government services, and critical infrastructure.
- Attackers exploit botnets to send overwhelming amounts of traffic, rendering servers unresponsive.
- With the increasing adoption of cloud computing, IoT, and edge networks, traditional security solutions like firewalls and intrusion detection systems struggle to handle large-scale attacks.

DDOS ATTACK:

- A cyber attack where multiple compromised computers flood a target system with excessive requests, leading to service disruption.
- Unlike traditional Denial-of-Service (DoS), DDoS uses multiple sources, making mitigation more difficult.

XGBoost Algorithm:

- XGBoost is a robust machine learning model known for its efficiency, speed, and accuracy in detecting anomalies in network traffic.
- It can effectively differentiate between normal and malicious traffic, even in large-scale network environments.

Related Work

1. Statistical Methods

- Analyzes traffic patterns (packet rates, entropy) to detect anomalies.
- Xiao et al.'s r-polling reduces training data while maintaining accuracy.
- Entropy-based detection struggles with stealthy threats.
- Prone to false positives, requiring extensive baseline analysis.

2. ML Techniques

- Traditional ML: CKNN improves accuracy but lacks scalability; SVM-CSOACN enhances classification but is resource-intensive; ANN handles non-linear patterns but demands high computation.
- Ensemble ML: Random Forest is fast but has more false positives; GBDT achieves 97.69% accuracy but requires longer training.
- Balances accuracy, false positives, and efficiency based on security needs.

3. XG Boost Work

- Chen et al.'s XGBoost achieved 98.53% accuracy and lowest false positive rate (0.008).
- Used POX, Mininet, and OpenMP for real-time SDN detection.
- Highly scalable with dataset expansion (400K → 4M).
- Limitations: outdated dataset, focus on flooding attacks, lacks real-world complexity.

Data Collection and Pre-processing

Dataset Used: CICDDoS2019 and BCCC-Mal-NetMem-2025

- Collected by the Canadian Institute for Cybersecurity.
- Contains attack and benign traffic with labeled data for ML training.
- Focused on SYN flood attacks in this research.

```python
# Identify benign and attack samples
benign_df = df[df['Label'] == 'Benign']
attack_df = df[df['Label'] != 'Benign']

# Number of benign samples
benign_count = len(benign_df)

# To get a 9:1 ratio, we need attack samples = 9 * benign_count
attack_count_needed = 9 * benign_count

# Downsample the attack class to match the required count
attack_df = attack_df.sample(n=attack_count_needed, random_state=42)

# Combine the balanced attack and benign data
balanced_df = pd.concat([benign_df, attack_df])

# Shuffle the dataset
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)

# Save the balanced dataset
balanced_df.to_csv("01-12-balanced_dataset.csv", index=False)

print("Class balance after resampling: {balanced_df['Label'].value_counts()}")
```

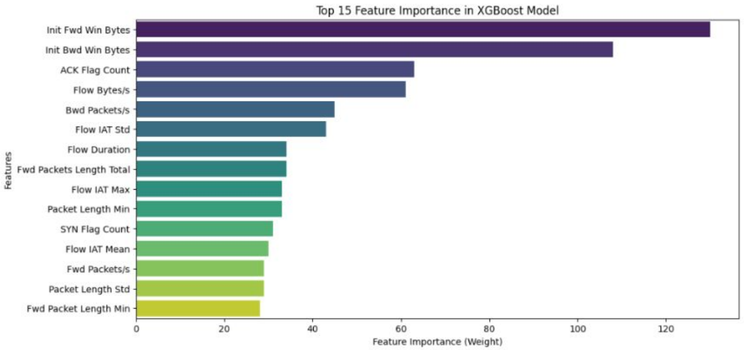
Preprocessing Steps

- Data Cleaning: Removed duplicate and missing entries.
- Feature Standardization: Normalized values to maintain consistency.
- Handling Class Imbalance: Applied undersampling to balance attack-to-benign ratio (9:1).

```python
import pandas as pd

# Load the balanced dataset from V1
df = pd.read_csv("01-12-balanced_dataset.csv")

# Remove duplicate records
df = df.drop_duplicates()

# Drop columns where all values are missing
df = df.dropna(axis=1, how='all')

# Convert all numeric columns properly
for col in df.columns:
    if df[col].dtype == 'object':  # If column is a string
        try:
            df[col] = pd.to_numeric(df[col])  # Convert to numeric if possible
        except ValueError:
            pass  # Ignore errors and keep as string

# Drop rows with missing values
df = df.dropna()

# Handle infinite values by replacing them with NaN
df.replace([float('inf'), float('-inf')], float('nan'), inplace=True)

# Optionally, drop rows with NaN values after replacing infinities
df = df.dropna()

# Save as CSV
df.to_csv("01-12-cleaned_dataset.csv", index=False)

print("Data cleaning completed. File saved as 01-12-cleaned_dataset.csv")
```

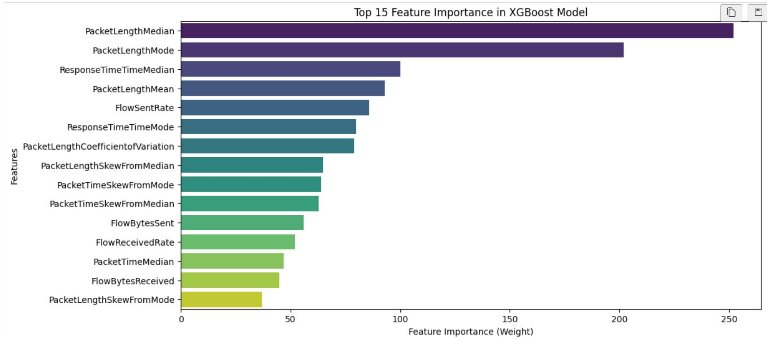
Feature Selection

initial_xgb.get_booster()



- initial_xgb is an XGBoost model trained using XGBClassifier or XGBRegressor.
- .get_booster() retrieves the underlying Booster object, which represents the trained XGBoost model.

get_score(importance_type="weight")



- get_score() extracts feature importance scores from the booster.
- Feature importance is measured by the number of times a feature is used in splits across all trees in the model.
- This returns a dictionary where:
    - Keys = feature names (e.g., 'Flow Bytes/s', 'Fwd Packets/s').
    - Values = importance scores (e.g., 15, 23, indicating how many times the feature was used in splits).

Hyper Parameter Tuning

Hyperparameters in XGBoost

XGBoost has several important hyperparameters that influence its performance:

After performing GridSearchCV, the optimal hyperparameters were:

- **learning_rate**: Controls the step size taken by the optimizer. - 0.2
- **n_estimators**: Determines the number of boosting trees. -100
- **max_depth**: Sets the maximum depth of each tree. - 3
- **subsample**: Defines the percentage of rows used for each tree construction. -0.8
- **colsample_bytree**: Determines the fraction of features used per tree. - 0.8

```python
# Define hyperparameter grid
param_grid = {
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1, 0.2],
    "n_estimators": [100, 200, 300],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0],
}

# Perform GridSearchCV with feature-selected dataset
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
grid_search = GridSearchCV(
    estimator=xgb_clf,
    param_grid=param_grid,
    scoring="roc_auc",
    cv=5,
    n_jobs=-1,
    verbose=2,
)

grid_search.fit(X_train_selected, y_train)

# Get the best model
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
```

Accuracy and Results

Results on Testing dataset(1)

Accuracy: 0.9977949283351709

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	1.00	374
1	1.00	1.00	1.00	533
accuracy			1.00	907

ROC-AUC: 0.9985753127790431

```python
df_train = pd.read_csv(r"C:\Users\Arnav Dham\OneDrive\Desktop\SOP\Dataset\Syn-training.csv")

X_combined = df_train.drop("Label", axis=1)
y_combined = LabelEncoder().fit_transform(df_train["Label"])  # Benign=0, Syn=1

# Feature selection on combined data
initial_xgb = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
initial_xgb.fit(X_combined, y_combined)
feature_importance = initial_xgb.get_booster().get_score(importance_type="weight")
selected_features = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)[:15]  #
selected_features = [f[0] for f in selected_features]

# Prepare training data with selected features
X_train = df_train[selected_features]
y_train = LabelEncoder().fit_transform(df_train["Label"])

# Train the model
model = XGBClassifier(
    use_label_encoder=False,
    eval_metric="logloss",
    scale_pos_weight=1.6,  # Prioritize SYN
    max_depth=3,
    learning_rate=0.2,
    n_estimators=100,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=0.1, #reduce parameters
    reg_lambda=2.0 #reduce overfitting
)
model.fit(X_train, y_train, verbose=True)

df_syn_testing = pd.read_csv(r"C:\Users\Arnav Dham\OneDrive\Desktop\SOP\Dataset\syn-testing.csv")

# Prepare syn-testing data with the same selected features
X_syn_testing = df_syn_testing[selected_features]
y_syn_testing = LabelEncoder().fit_transform(df_syn_testing["Label"])  # Benign=0, Syn=1

# Predict on syn-testing
y_pred_proba = model.predict_proba(X_syn_testing)[:, 1]
y_pred_default = model.predict(X_syn_testing)

# Evaluate on syn-testing
print("Results on syn-testing.csv")
print("Accuracy:", (y_pred_default == y_syn_testing).mean())
print("Classification Report:\n", classification_report(y_syn_testing, y_pred_default))
print("Confusion Matrix:\n", confusion_matrix(y_syn_testing, y_pred_default))
print("ROC-AUC:", roc_auc_score(y_syn_testing, y_pred_proba))
```

Accuracy and Results

Results on 20% Test Split(dataset 1)

Accuracy: 0.9995024168325277

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5407
1	1.00	1.00	1.00	8661
accuracy			1.00	14068

ROC-AUC: 0.99999

```python
# Split into 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train with selected features
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

# Train the final model
model = XGBClassifier(
    use_label_encoder=False,
    eval_metric="logloss",
    scale_pos_weight=1.6,
    max_depth=3,
    learning_rate=0.2,
    n_estimators=100,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=0.1,
    reg_lambda=2.0
)
model.fit(X_train_selected, y_train, verbose=True)

# Predict on test set
y_pred_proba = model.predict_proba(X_test_selected)[:, 1]
y_pred_default = model.predict(X_test_selected)

# Evaluate on test set
print("Results on 20% Test Split:")
print("Accuracy:", (y_pred_default == y_test).mean())
print("Classification Report:\n", classification_report(y_test, y_pred_default))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_default))
print("ROC-AUC:", roc_auc_score(y_test, y_pred_proba))
```

Accuracy and Results

Results on 15%(dataset 2)

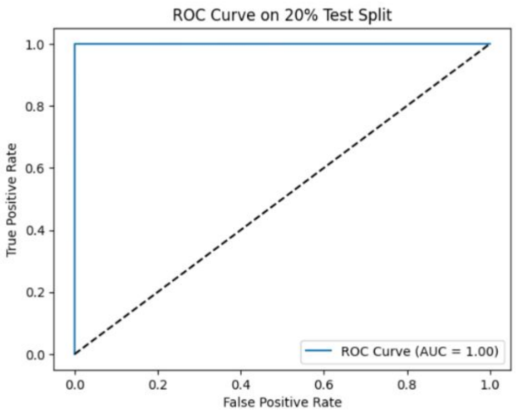
- 60% dataset used for training
- 25% for feature selection
- 15% for testing

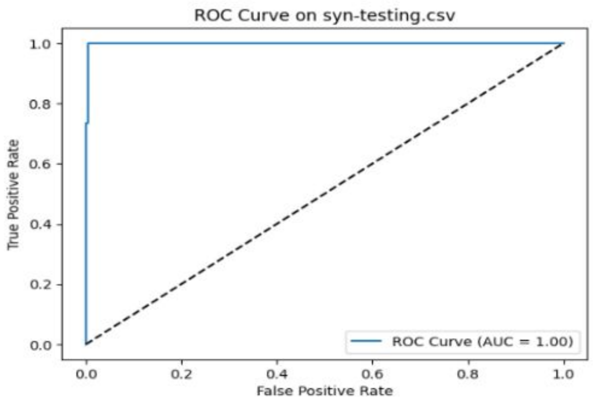
```
Evaluation on 15% syn-testing set:
Accuracy: 0.9997357669871338
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     72955
           1       1.00      1.00      1.00     74642

    accuracy                           1.00    147597
   macro avg       1.00      1.00      1.00    147597
weighted avg       1.00      1.00      1.00    147597

Confusion Matrix:
 [[72928    27]
 [   12 74630]]
ROC-AUC: 0.9999991049502092
```

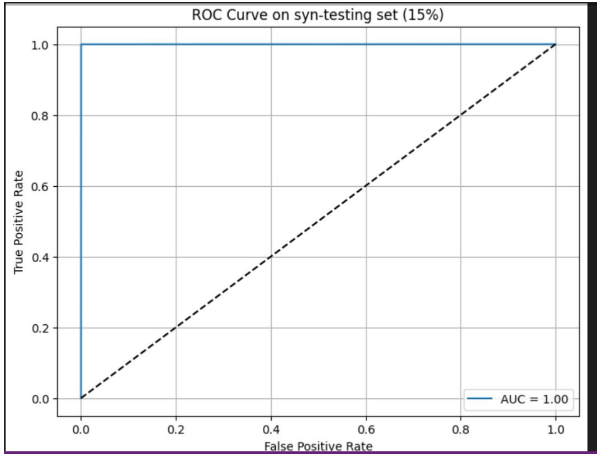
```python
# Step 1: Load your final dataset
df = pd.read_csv(r"C:\Users\PAVILION\Downloads\cleaned_dataset2.csv")

# Step 2: Split the dataset for training and evaluation
df_train = df.sample(frac=0.6, random_state=42)  # 60% for training
df_temp = df.drop(df_train.index)
df_test_sampled = df_temp.sample(frac=0.25, random_state=42)  # 25% of total (40% * 0.625)
df_syn_testing = df_temp.drop(df_test_sampled.index)  # Remaining 15%

# Step 3: Combine df_train + df_test_sampled for feature selection
df_combined = pd.concat([df_train, df_test_sampled])
X_combined = df_combined.drop("Label", axis=1)
y_combined = LabelEncoder().fit_transform(df_combined["Label"])

# Step 4: Feature selection
initial_xgb = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
initial_xgb.fit(X_combined, y_combined)
feature_importance = initial_xgb.get_booster().get_score(importance_type="weight")
selected_features = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)[:15]
selected_features = [f[0] for f in selected_features]

# Step 5: Prepare final training data with selected features
X_train = df_train[selected_features]
y_train = LabelEncoder().fit_transform(df_train["Label"])
```

Accuracy and Results

ROC Curves







Accuracy and Results

5 Fold validation for dataset 2

Fold	Accuracy	ROC-AUC	SYN Recall
1	0.9997	1.0000	0.9998
2	0.9998	1.0000	0.9999
3	0.9998	1.0000	0.9998
4	0.9998	1.0000	0.9999
5	0.9996	1.0000	0.9999
Average	0.9998	1.0000	0.9999

```python
# Encode target
X = df.drop("Label", axis=1)
y = LabelEncoder().fit_transform(df["Label"])

# Set up Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracies = []
roc_aucs = []
recalls = []

print("\n🔁 5-Fold Stratified Cross-Validation Results:\n")

for fold, (train_index, test_index) in enumerate(skf.split(X, y), 1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model_kf = XGBClassifier(
        use_label_encoder=False,
        eval_metric="logloss",
        scale_pos_weight=1.6,
        max_depth=3,
        learning_rate=0.2,
        n_estimators=100,
        subsample=0.8,
        colsample_bytree=0.8,
        reg_alpha=0.1,
        reg_lambda=2.0
    )

    model_kf.fit(X_train, y_train)
    y_pred = model_kf.predict(X_test)
    y_proba = model_kf.predict_proba(X_test)[:, 1]

    acc = (y_pred == y_test).mean()
    roc_auc = roc_auc_score(y_test, y_proba)
    recall_syn = classification_report(y_test, y_pred, output_dict=True)["1"]["recall"]

    accuracies.append(acc)
    roc_aucs.append(roc_auc)
    recalls.append(recall_syn)

    print(f"Fold {fold}: Accuracy = {acc:.4f}, ROC-AUC = {roc_auc:.4f}, SYN Recall = {recall_syn:.4f}")
```

Thankyou!!