MODBUS ATTACK DETECTION

USING KG AND GNN

Albany, April 29,2025

Course 538| Trinh Nguyen | tnguyen31@albany.edu

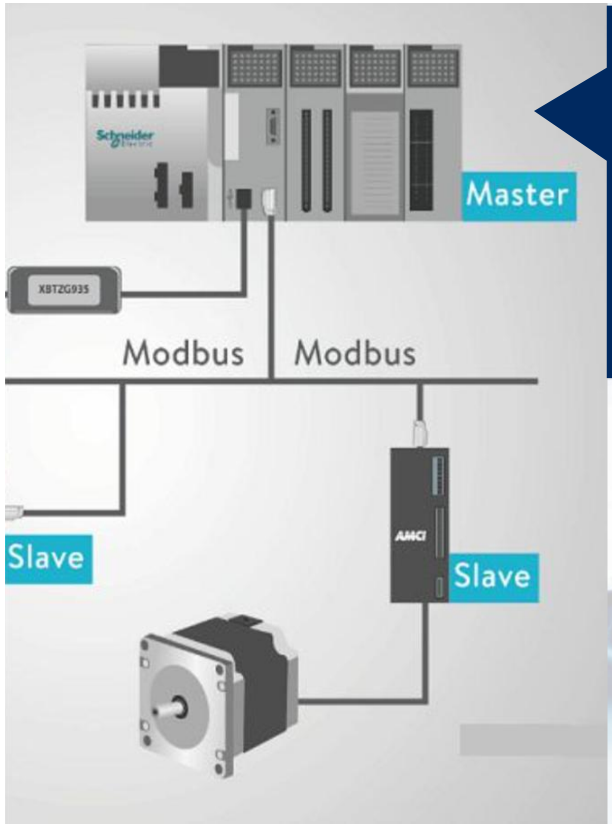
Contents:

01 Overview – Modbus in cybersecurity

Understand Modbus attack in industrial system

What is mod-bus?

- **Concept:** communication protocol used for transmitting information over networks between electronic devices, esp. in industrial automation systems (company set up, factory set up,..)
- **Types**: RTU, ASCII, TCP/IP
- **Operation model**: Master/Slave.
  One device (Master) requests data, others (Slaves) respond



A PLC Master sends a request via Modbus to a temperature sensor to get current readings

Modbus protocol lacks built-in security

Attack cause shutdowns, data manipulation or equipment damage

➔ **Detecting Modbus threats is critical**

02 Dataset Description

CIC APT IIoT Dataset 2024

- **Developed by:** Canadian Institute for Cybersecurity (CIC) and National Research Council Canada (NRC)
- **Data size:** (21,599,219 rows x 70 columns) ~ 10GB
- **Content:** nodes (IP, source) and edges (e.g., Protocol name, duration flows, WasGeneratedBy,..) with some NaNs due to varying types, label for the attack category
- **Files link:** [IIoT Dataset 2024 | Datasets | Research | Canadian Institute for Cybersecurity | UNB]

Phase 1: Normal data (12,062,396 rows, 70 columns)

ts	flow_duration	Header_Length	Source IP	Destination IP	Source Port	Destination Port	Protocol Type	Protocol_name	Duration	Rate	Srate	Fragments	Sequence number	Protocol Version	flow_idle_time	flow_active_time	label	subLabel	subLabelCat
1701426437	0	66	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	0	0	0	0	0	1701426437	0	0	0	0
1701426437	0.00211215	132	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	946.902359	946.902359	0	0	0	0.00211215	0.00211215	0	0	0
1701426437	0.00232816	198	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	1288.57266	1288.57266	0	0	0	0.000216007	0.002328157	0	0	0
1701426437	0.00432921	264	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	923.957264	923.957264	0	0	0	0.002001047	0.004329205	0	0	0
1701426437	0.00949502	330	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	526.591839	526.591839	0	0	0	0.005165815	0.00949502	0	0	0
1701426437	0.01160717	396	172.16.64.128	172.16.66.128	41750	502	6	TCP	64	516.921863	516.921863	0	0	0	0.00211215	0.01160717	0	0	0

Phase 2: Normal data + Attack data (9,536,823 rows, 70 columns)

ts	flow_duration	Header_Length	Source IP	Destination IP	Source Port	Destination Port	Protocol Type	Protocol_name	Duration	Rate	Srate	Fragments	Sequence number	Protocol Version	flow_idle_time	flow_active_time	label	subLabel	subLabelCat
1701581789	30.39080596	7160	172.16.63.128	172.16.65.128	41596	8888	6	TCP	64	0.954235963	0.954235963	0	0	0	30.28496003	30.39080596	1	discovery	permission groups discovery
1701581789	30.39291787	7226	172.16.63.128	172.16.65.128	41596	8888	6	TCP	64	0.987072058	0.987072058	0	0	0	0.002111912	30.39291787	1	discovery	permission groups discovery
1701581789	30.39357495	7292	172.16.65.128	172.16.63.128	8888	41596	6	TCP	64	1.019952409	1.019952409	0	0	0	0.000657082	30.39357495	1	discovery	permission groups discovery
1701581789	30.39568686	7358	172.16.65.128	172.16.63.128	8888	41596	6	TCP	64	1.052780947	1.052780947	0	0	0	0.002111912	30.39568686	1	discovery	permission groups discovery
1701581820	61.11016893	7424	172.16.63.128	172.16.65.128	41596	8888	6	TCP	64	0.540008326	0.540008326	0	0	0	30.71448207	61.11016893	1	discovery	permission groups discovery
1701581820	61.11228085	7490	172.16.63.128	172.16.65.128	41596	8888	6	TCP	64	0.556352987	0.556352987	0	0	0	0.002111912	61.11228085	1	discovery	permission groups discovery

03 Data preprocessing

CIC APT IIoT Dataset 2024

Platform: Google Colab

- Merge two phases (Using power query to reduce data size)
- Check & Handle text data types
- Identify & create files for nodes - edge - label
- Using SMOTE to handle imbalance
- Save file and complete data prep. Prepare to load into model

```
edge_feat = np.load("edge_feat_scaled.npy")
label_bi = np.load("label_bi.npy")
label_mul = np.load('label_mul.npy')

num_edges = len(edge_feat)
train_val, test = train_test_split(np.arange(num_edges), test_size=5000, stratify=label_bi)
train_before, val = train_test_split(train_val, test_size=6004, stratify=label_bi[train_val])

print("Before undersampling: {}".format(Counter(label_bi[train_before])))

# Apply SMOTE on the training set
smote = SMOTE(random_state=42)

# Need to reshape train_before so SMOTE sees a 2D array
train_before_feat = edge_feat[train_before]  # Features for SMOTE
train_before_labels = label_bi[train_before] # Labels for SMOTE

train_feat_resampled, y_smote = smote.fit_resample(train_before_feat, train_before_labels)

print("After SMOTE oversampling: {}".format(Counter(y_smote)))
```

```
Before undersampling: Counter({np.int8(0): 89106, np.int8(1): 894})
After SMOTE oversampling: Counter({np.int8(0): 89106, np.int8(1): 89106})
```

04 Model implementation

CIC APT IIoT Dataset 2024

GNN
Graph Neural Network

- **What:** A model process data structured as graphs (nodes connected by edges)

- **How:** learns node representations by aggregating information from neighboring nodes in a graph repeatedly

- Use a message-passing mechanism to Propagates node features through the graph

GraphSAGE
Graph Sample and Aggregation

- **What:** A type of GNN, sampling a fixed-size neighborhood around each node

- **How:** During each layer, it samples a fixed number of neighbors, aggregates their features, and then updates the node's representation

GAT
Graph Attention Network

- **What:** A type of GNN, uses attention mechanisms to learn the importance of neighbors

- **How:** not aggregate neighbor features, it assigns **weight** to them

➔ Selective learning

Multi head GAGNN
Graph Attention GNN

- **What:** A type of GNN, uses multiple attention mechanisms in parallel

- **How:** each head learn an attention pattern

➔ Aggregate/average output to form a node representation

04 Model implementation

Graph neural network (GNN)

Step 1: Load data & libraries

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from tqdm import tqdm

# ---------------------------
# Load Data
# ---------------------------
edge_feat = np.load("edge_feat_rus.npy", allow_pickle=True)
adj = np.load("adj_rus (1).npy", allow_pickle=True)
label = np.load("label_bi_rus (1).npy")
```

- Load the relevant libraries to work with model

- Load the files for nodes, edge feature, label that was preprocessed earlier
- Ensure the data shape is preserved

04 Model implementation

Graph neural network

Step 2: Tensor transformation & data sampling

```python
# Ensure edge_feat contains only numerical values
edge_feat = edge_feat.astype(np.float32)
# Reduce Dataset Size: Sample 5% of Data
subset_size = 0.05
sample_indices, _ = train_test_split(np.arange(len(label)),
                                     train_size=subset_size, stratify=label, random_state=42)
# Apply sampling
edge_feat = edge_feat[sample_indices]
label = label[sample_indices]

# Find unique nodes that remain in the sampled dataset
valid_nodes = set(np.unique(edge_feat))  # Nodes that exist in feature matrix
node_mapping = {node: i for i, node in enumerate(valid_nodes)}

# Re-map adjacency list indices to match the reduced dataset
adj_remapped = []
for src, dst in adj[sample_indices]:
    if src in node_mapping and dst in node_mapping:  # Only keep valid edges
        adj_remapped.append([node_mapping[src], node_mapping[dst]])

# Convert adj to NumPy array and PyTorch tensor
adj_numeric = np.array(adj_remapped, dtype=np.int64)
adj_numeric = torch.tensor(adj_numeric, dtype=torch.long)

# Convert Labels to One-Hot Encoding
num_classes = int(label.max()) + 1  # Get number of unique classes
label_one_hot = np.eye(num_classes)[label.astype(int)]
label = torch.tensor(label_one_hot, dtype=torch.float32)

# Convert Node Features to PyTorch Tensor
edge_feat = torch.tensor(edge_feat, dtype=torch.float32)
```

Data sample and apply the sampling across edge and label

Remap the node (source & destination) after resampling to ensure the current node is valid

Converts edge features, adjacency list, and labels into PyTorch tensors for model training.

04 Model implementation

Graph neural network

Step 3: Define the model

```
# GNN Model Definition
# ---------------------------
class GNN(nn.Module):
    def __init__(self, in_feats, hidden_feats, out_feats, dropout=0.5):
        super(GNN, self).__init__()
        self.conv1 = nn.Linear(in_feats, hidden_feats)
        self.conv2 = nn.Linear(hidden_feats, out_feats)
        self.dropout = dropout

    def forward(self, x, edge_index): #x is node feature, edge_index is edge list
        h = self.conv1(x)
        h = F.relu(h)

        # Fix: Ensure edge_index is 2D
        if edge_index.dim() == 1:
            edge_index = edge_index.view(-1, 2)  # Reshape into (num_edges, 2)

        edge_src, edge_dst = edge_index[:, 0], edge_index[:, 1]

        # Fix: Ensure valid_mask is never empty
        valid_mask = (edge_dst < h.shape[0]) & (edge_src < h.shape[0])  # Ensure indices are valid

        if valid_mask.sum() == 0:  # If all edges are invalid, return h unchanged
            return torch.sigmoid(self.conv2(F.dropout(h, p=self.dropout, training=self.training)))

        # Fix: Ensure index_add_ does not fail
        h_scatter = torch.zeros_like(h)
        h_scatter.index_add_(0, edge_dst[valid_mask], h[edge_src[valid_mask]])

        h = F.dropout(h_scatter, p=self.dropout, training=self.training)
        h = self.conv2(h)
        return torch.sigmoid(h)  # Sigmoid activation # output value in (0,1) range
```

Initialize the constructor __init__ with parameter like input feature, hidden feature, output feature, drop out.

Model has 2 layers:

- conv1: Transforms input node features into hidden features.
- conv2: Transforms hidden features into output features.

Forward method to decide how input flow through the graph

04 Model implementation

Graph neural network

Step 4: Initialize the model

```
# Initialize Model, Loss, Optimizer
# ---------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_features = edge_feat.shape[1]
num_classes = label.shape[1]  # Correct way to get the number of labels

model = GNN(in_feats=num_features, hidden_feats=128, out_feats=num_classes).to(device)
criterion = nn.BCELoss()  # Binary Cross-Entropy Loss
optimizer = optim.Adam(model.parameters(), lr=0.003)

# ---------------------------
# Train-Test Split on Reduced Dataset
# ---------------------------
train_idx, test_idx = train_test_split(np.arange(len(label)), test_size=0.2, random_state=42)
train_idx, val_idx = train_test_split(train_idx, test_size=0.1, random_state=42)

# Move tensors to GPU if available
edge_feat, adj_numeric, label = edge_feat.to(device), adj_numeric.to(device), label.to(device)
```

Split data for training, validation and testing
80% training (take 10% as validation)
20% testing

04 Model implementation

Graph neural network

Step 5: Start the training loop

```
num_epochs = 10
batch_size = 64
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    # Use tqdm for batch progress tracking
    with tqdm(total=len(train_idx) // batch_size, desc=f"Epoch {epoch+1}/{num_epochs}", unit="batch") as pbar:
        for i in range(0, len(train_idx), batch_size):
            batch_idx = train_idx[i:i + batch_size]
            batch_feat = edge_feat  # Use full node feature matrix
            batch_adj = adj_numeric  # Adjacency is global, not batch-specific
            batch_label = label[batch_idx]

            optimizer.zero_grad()
            outputs = model(batch_feat, batch_adj) #running the model
            loss = criterion(outputs[batch_idx], batch_label)  # Select only batch outputs
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

            # Compute Accuracy
            predicted_labels = (outputs[batch_idx] > 0.5).float()  # Convert to binary labels
            correct_predictions += (predicted_labels == batch_label).sum().item()
            total_samples += batch_label.numel()  # Total number of elements in labels

            # Update progress bar
            pbar.set_postfix(loss=f"{loss.item():.4f}")
            pbar.update(1)

    # Compute and print epoch accuracy
    epoch_accuracy = correct_predictions / total_samples
    print(f"Epoch {epoch + 1}/{num_epochs}, Avg Loss: {epoch_loss / len(train_idx):.6f}, Accuracy: {epoch_accuracy:.4f}")
```

```
Epoch 1/10: 107batch [00:03, 34.73batch/s, loss=0.0107]
Epoch 1/10, Avg Loss: 0.002509, Accuracy: 0.9662
Epoch 2/10: 107batch [00:03, 34.70batch/s, loss=0.0111]
Epoch 2/10, Avg Loss: 0.001096, Accuracy: 0.9906
Epoch 3/10: 107batch [00:02, 40.25batch/s, loss=0.0072]
Epoch 3/10, Avg Loss: 0.001029, Accuracy: 0.9908
Epoch 4/10: 107batch [00:02, 40.48batch/s, loss=0.0061]
Epoch 4/10, Avg Loss: 0.001005, Accuracy: 0.9909
Epoch 5/10: 107batch [00:02, 40.17batch/s, loss=0.0073]
Epoch 5/10, Avg Loss: 0.000939, Accuracy: 0.9910
Epoch 6/10: 107batch [00:03, 32.94batch/s, loss=0.0058]
Epoch 6/10, Avg Loss: 0.000930, Accuracy: 0.9908
Epoch 7/10: 107batch [00:02, 37.94batch/s, loss=0.0086]
Epoch 7/10, Avg Loss: 0.000982, Accuracy: 0.9910
Epoch 8/10: 107batch [00:02, 39.56batch/s, loss=0.0082]
Epoch 8/10, Avg Loss: 0.000917, Accuracy: 0.9910
Epoch 9/10: 107batch [00:02, 40.43batch/s, loss=0.0049]
Epoch 9/10, Avg Loss: 0.000901, Accuracy: 0.9910
Epoch 10/10: 107batch [00:02, 37.00batch/s, loss=0.0070]
```

Epoch 10/10:
Avg Loss: 0.000906,
Accuracy: 0.9910

04 Model implementation

Graph neural network

Step 5: Validation and testing

```python
from sklearn.metrics import accuracy_score, f1_score, classification_report,
                            cohen_kappa_score, roc_auc_score
model.eval()
with torch.no_grad(): #no backpropagation turn off all memory for updating
    # Validation
    val_outputs = model(edge_feat, adj_numeric) #invoke the def forward function
    val_outputs = val_outputs[val_idx]  # Select only validation samples
    val_labels = label[val_idx]

    val_loss = criterion(val_outputs, val_labels)  # Compute validation loss

    val_outputs_binary = (val_outputs > 0.5).float()  # Threshold for metrics

    # Test
    test_outputs = model(edge_feat, adj_numeric)
    test_outputs = test_outputs[test_idx]  # Select only test samples
    test_labels = label[test_idx]

    test_loss = criterion(test_outputs, test_labels)  # Compute test loss

    test_outputs_binary = (test_outputs > 0.5).float()  # Threshold for metrics
    #float convert the boolean, True become 1, False become 0
# --------------------------
# Validation Metrics
# --------------------------
val_true = val_labels.cpu().numpy()
val_pred = val_outputs_binary.cpu().numpy()
val_outputs_proba = val_outputs.cpu().numpy()

val_accuracy = accuracy_score(val_true.flatten(), val_pred.flatten())
val_kappa = cohen_kappa_score(val_true.flatten(), val_pred.flatten())
val_auc = roc_auc_score(val_true.flatten(), val_outputs_proba.flatten())
```

```python
# Test Metrics
# --------------------------
test_true = test_labels.cpu().numpy()
test_pred = test_outputs_binary.cpu().numpy()
test_outputs_proba = test_outputs.cpu().numpy()

test_accuracy = accuracy_score(test_true.flatten(), test_pred.flatten())
test_kappa = cohen_kappa_score(test_true.flatten(), test_pred.flatten())
test_auc = roc_auc_score(test_true.flatten(), test_outputs_proba.flatten())

# --------------------------
# Print Results
# --------------------------
print("\n--- Validation Set ---")
print(f"Validation Loss: {val_loss.item():.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")
print(f"Validation Cohen's Kappa: {val_kappa:.4f}")
print(f"Validation AUC: {val_auc:.4f}")

print("\n--- Test Set ---")
print(f"Test Loss: {test_loss.item():.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Cohen's Kappa: {test_kappa:.4f}")
print(f"Test AUC: {test_auc:.4f}")

print("\nTest Classification Report:")
print(classification_report(test_true, test_pred, zero_division=0))
```

05 Overall result

Graph neural network

```
--- Validation Set ---
Validation Loss: 0.0523
Validation Accuracy: 0.9934
Validation Cohen's Kappa: 0.9868
Validation AUC: 0.9904

--- Test Set ---
Test Loss: 0.0509
Test Accuracy: 0.9926
Test Cohen's Kappa: 0.9852
Test AUC: 0.9931

Test Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1022
           1       0.99      0.99      0.99       870

   micro avg       0.99      0.99      0.99      1892
   macro avg       0.99      0.99      0.99      1892
weighted avg       0.99      0.99      0.99      1892
 samples avg       0.99      0.99      0.99      1892
```

The model shows excellent performance with high accuracy **(99.34% validation, 99.26% test**), low loss, as reflected in high Cohen's Kappa and AUC scores.

The classification report indicates balanced and strong performance across both classes, with precision, recall, and **F1-score near 0.99** for both validation and test sets.

CLASSIFIER	TRAINING ACCURACY	VALIDATION ACCURACY	TEST ACCURACY	F1 Measure	KAPPA	ROC area
GNN Model	0.9910	0.9934	0.9926	0.9900	0.9852	0.9931
New Method I (GAT)	0.9910	0.9934	0.9926	0.9900	0.9852	0.9932
New Method II (GraphSAGE)	0.9910	0.9934	0.9926	0.9900	0.9852	0.9927
New Method III (Multi Head GA GNN)	0.9908	0.9934	0.9926	0.9900	0.9852	0.9908

06

Discussion

Limitation, learning & future work

Limitation	Big dataset make it impossible to load the data, if yes, it also cause run time crash Reduced data set is not representative enough, insufficient for best learning
Learning	Understand the dataset first Identify class imbalance if any (which cause the model to overfit) Try out different scenario (Ex: use smote/under-sampling) to test feasibility
Future work	Continue to try the full dataset on strong system to check the result, extend further with classification on multi label. ➔ Timely detecting intrusions will help to minimize security risks

THANK YOU!