Network Intrusion Detection System (NIDS) with Machine Learning

```
import numpy as np
import pandas as pd
import random
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.svm import OneClassSVM
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from scapy.all import sniff, IP, TCP
import threading
import time
from collections import defaultdict
import warnings
warnings.filterwarnings('ignore')

# Set seed for reproducibility
random.seed(42)
np.random.seed(42)

# ====================== Data Processing Module ======================
class DataProcessor:
    def __init__(self):
        self.scaler = StandardScaler()
        self.encoder = LabelEncoder()
        self.feature_names = [
            'duration', 'protocol type', 'src bytes', 'dst bytes',
            'wrong_fragment', 'urgent', 'count', 'srv_count',
            'dst_host_count', 'dst_host_srv_count'
        ]

    def preprocess(self, data):
        data['protocol_type'] =
self.encoder.fit_transform(data['protocol_type'])
        X = data[self.feature_names]
        X = self.scaler.fit_transform(X)
        return X, data['label']

    def preprocess_single_packet(self, packet_data):
        # Create a DataFrame from the single packet data
        packet_df = pd.DataFrame([packet_data])
        # Encode protocol_type
```

```
        packet_df['protocol_type'] =
self.encoder.transform(packet_df['protocol_type'])
        # Select the features
        X = packet_df[self.feature_names]
        # Scale the features
        X = self.scaler.transform(X)
        return X


# ====================== ML Models Module ======================
class IntrusionDetectionModels:
    def __init__(self):
        self.models = {
            'Random Forest': RandomForestClassifier(n_estimators=100,
random_state=42),
            # 'One-Class SVM': OneClassSVM(nu=0.1), # Removed from
supervised training loop
            'Isolation Forest': IsolationForest(contamination=0.1,
random_state=42),
            'Neural Network': MLPClassifier(hidden_layer_sizes=(50, 25),
max_iter=500, solver='adam', random_state=42)
        }
        self.best_model = None

    def train(self, X_train, y_train):
        results = {}
        for name, model in self.models.items():
            model.fit(X_train, y_train)
            # Only calculate score for models that have the score method
(supervised models)
            if hasattr(model, 'score'):
                score = model.score(X_train, y_train)
                results[name] = score
                print(f"{name} trained with accuracy: {score:.2f}")
            else:
                print(f"{name} trained (unsupervised model)")

        # Select the best model only from those with scores
        scored_results = {k: v for k, v in results.items() if k in
self.models and hasattr(self.models[k], 'score')}
        if scored_results:
            self.best_model = max(scored_results, key=scored_results.get)
            print(f"\nSelected best model: {self.best_model}")
            return self.models[self.best model]
        else:
```

```
            print("\nNo supervised models trained or evaluated.")
            # Handle the case where no supervised models are available for
selection
            # For now, you might want to return a default or raise an
error
            return None # Or handle appropriately based on your system
design

# ====================== Network Monitor Module ======================
class NetworkMonitor:
    def __init__(self):
        self.traffic data = []
        self.lock = threading.Lock()
        self.attack types = {
            'normal': 0,
            'ddos': 1,
            'port_scan': 2,
            'bruteforce': 3,
            'sql_injection': 4
        }

    def packet_handler(self, packet):
        if IP in packet and TCP in packet:
            features = {
                'duration': packet.time,
                'protocol_type': 'tcp',
                'src bytes': len(packet[IP].payload),
                'dst_bytes': len(packet[TCP].payload),
                'wrong fragment': 0,
                'urgent': packet[TCP].urgptr,
                'count': 1,
                'srv_count': 1,
                'dst_host_count': 1,
                'dst_host_srv_count': 1,
                'label': 'normal',
                'src_ip': packet[IP].src
            }
            if len(packet[TCP].payload) > 1000:
                features['label'] = 'ddos'
            elif packet[TCP].flags == 2 and packet[TCP].dport < 1024:
                features['label'] = 'port_scan'
            with self.lock:
                self.traffic_data.append(features)

    def start_capture(self, timeout=30):
```

```
        print(f"Starting network capture for {timeout} seconds...")
        sniff_thread = threading.Thread(
            target=lambda: sniff(prn=self.packet_handler, timeout=timeout)
        )
        sniff_thread.start()
        return sniff_thread

# ====================== Response System Module ======================
class ResponseSystem:
    def __init__(self):
        self.response_actions = {
            'ddos': self.block ip,
            'port_scan': self.alert_admin,
            'bruteforce': self.change password,
            'sql_injection': self.block_and_log,
            'default': self.log only
        }
        self.incident_log = []

    def execute_response(self, attack_type, details):
        action = self.response_actions.get(attack_type,
self.response_actions['default'])
        action(details)

    def block_ip(self, details):
        print(f"🚨 BLOCKING IP {details['src_ip']} for DDoS attack")
        self.incident_log.append({'timestamp': time.time(), 'action':
'blocked_ip', 'details': details})

    def alert_admin(self, details):
        print(f"⚠ ALERT: Port scan detected from {details['src ip']}")
        self.incident_log.append({'timestamp': time.time(), 'action':
'admin alert', 'details': details})

    def change password(self, details):
        print(f"🔐 Changing passwords for {details['target_service']}")

    def block_and_log(self, details):
        print(f"🛡 SQL Injection attempt blocked from
{details['src_ip']}")

    def log_only(self, details):
        print(f"📝 Logging suspicious activity from {details['src ip']}")

# ====================== Visualization Module ======================
```

```
class Visualization:
    @staticmethod
    def plot_confusion_matrix(y_true, y_pred):
        cm = confusion_matrix(y_true, y_pred)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt='d')
        plt.title('Confusion Matrix')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.show()

    @staticmethod
    def generate_report(y_true, y_pred):
        print(classification report(y true, y pred))

# ==================== Main System ====================
class NIDS:
    def __init__(self):
        self.data_processor = DataProcessor()
        self.ml_models = IntrusionDetectionModels()
        self.monitor = NetworkMonitor()
        self.response = ResponseSystem()
        self.visualization = Visualization()
        self.trained_model = None

    def train_model(self, dataset_path):
        print("Loading training data...")
        data = pd.read_csv(dataset_path)
        X, y = self.data processor.preprocess(data)
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
        print("\nTraining models...")
        self.trained_model = self.ml_models.train(X_train, y_train)

        if self.trained_model:
            print("\nEvaluating model...")
            y_pred = self.trained_model.predict(X_test)
            self.visualization.generate_report(y_test, y_pred)
            self.visualization.plot_confusion_matrix(y_test, y_pred)
        else:
            print("No supervised model was trained to evaluate.")


    def run realtime detection(self):
        print("\nStarting real-time monitoring...")
```

```
        if not self.trained_model:
            print("No supervised model trained. Cannot perform real-time
detection.")
            return

        capture_thread = self.monitor.start_capture(timeout=30)
        while capture_thread.is_alive():
            time.sleep(5)
            if self.monitor.traffic_data:
                latest_data_point = self.monitor.traffic_data[-1]
                # Preprocess the single packet data
                latest_features =
self.data_processor.preprocess_single_packet(latest_data_point)

                # Dummy prediction for demonstration - replace with actual
prediction
                predicted_label =
self.trained_model.predict(latest_features)[0]

                if predicted_label != 'normal':
                    self.response.execute_response(
                        predicted_label,
                        {'src_ip': latest_data_point['src_ip'],
'target_service': 'web'}
                    )
        print("\nMonitoring completed")

# ====================== Execution ======================
if __name__ == "__main__":
    nids = NIDS()

    # Generate deterministic demo data
    demo_data = pd.DataFrame([{
        'duration': 0.5, 'protocol_type': 'tcp', 'src_bytes': 500,
        'dst_bytes': 300, 'wrong_fragment': 0, 'urgent': 0,
        'count': 1, 'srv_count': 1, 'dst_host_count': 1,
        'dst_host_srv_count': 1, 'label': 'normal'
    } for _ in range(1000)])

    for attack in ['ddos', 'port scan', 'bruteforce']:
        demo_data = pd.concat([
            demo_data,
            pd.DataFrame([{
                'duration': np.random.uniform(1, 5),
                'protocol_type': 'tcp',
```

```
                'src_bytes': np.random.randint(1000, 10000),
                'dst_bytes': np.random.randint(1000, 10000),
                'wrong_fragment': np.random.randint(0, 3),
                'urgent': 0,
                'count': np.random.randint(5, 20),
                'srv_count': np.random.randint(5, 20),
                'dst_host_count': np.random.randint(5, 20),
                'dst_host_srv_count': np.random.randint(5, 20),
                'label': attack
            } for _ in range(200)])
        ])

    demo_data.to_csv('demo_dataset.csv', index=False)
    nids.train_model('demo_dataset.csv')
    nids.run_realtime_detection()
```

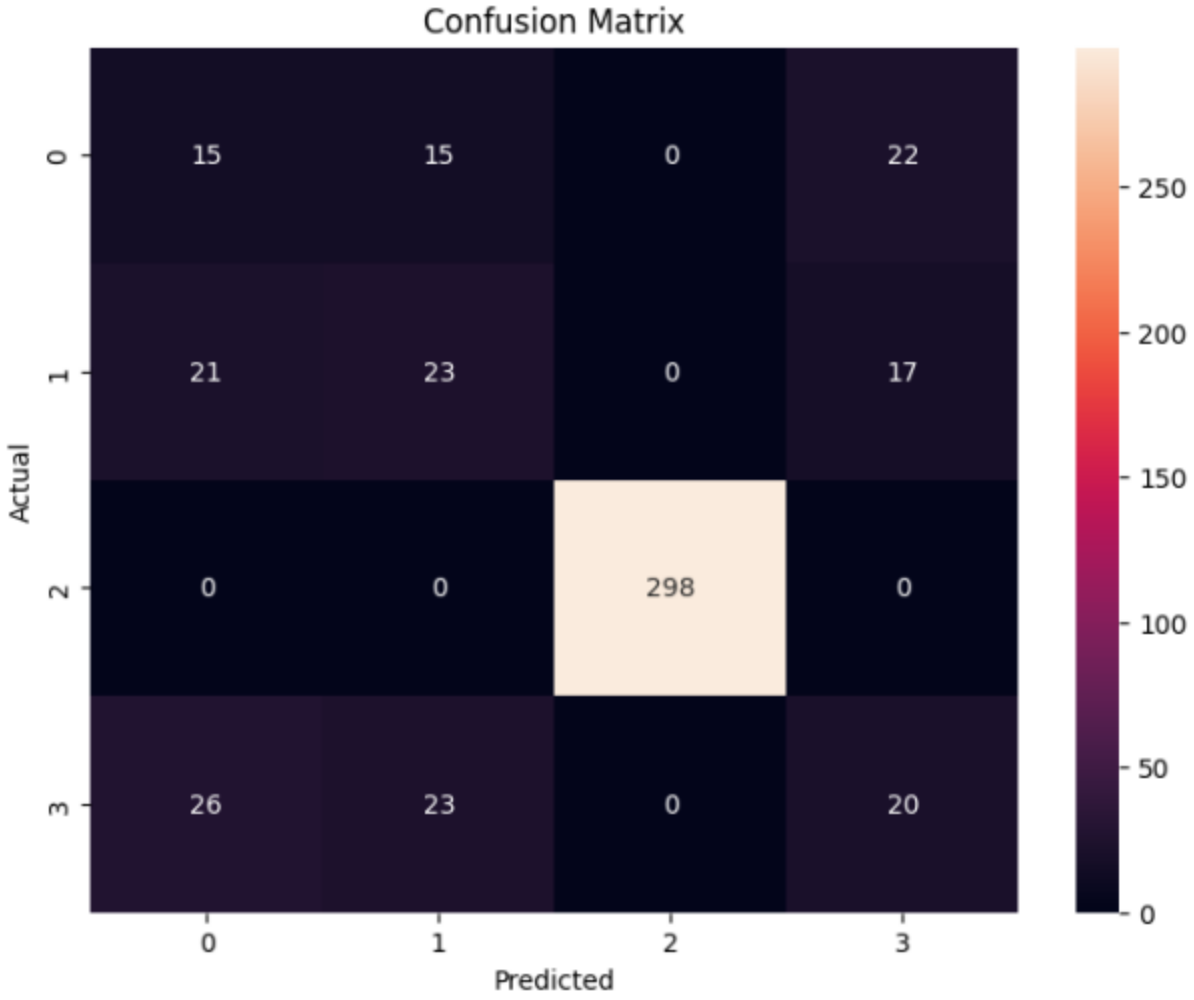
```
Loading training data...

Training models...
Random Forest trained with accuracy: 1.00
Isolation Forest trained (unsupervised model)
Neural Network trained with accuracy: 0.93

Selected best model: Random Forest

Evaluating model...
```

	precision	recall	f1-score	support
bruteforce	0.26	0.35	0.30	52
ddos	0.26	0.30	0.28	61
normal	1.00	1.00	1.00	298
port_scan	0.33	0.20	0.25	69
accuracy			0.72	480
macro avg	0.46	0.46	0.46	480
weighted avg	0.73	0.72	0.72	480



```
Starting real-time monitoring...
Starting network capture for 30 seconds...

Monitoring completed
```