Overview

You've been hired as a Data Engineer at a cybersecurity company that monitors network traffic for DDoS attacks. Your task is to build a data pipeline that can process network traffic logs in both real-time (streaming) and batch modes to detect potential security threats.

The Challenge

A major client has been experiencing intermittent DDoS attacks and needs both:

- Real-time alerting when an attack is happening
- Historical analysis to understand attack patterns

Your pipeline should demonstrate how you would handle this in a production environment.

Dataset

We're providing you with:

- network_traffic.csv - 2 hours of network traffic data (~100-200MB)
- generate_ddos_dataset.py - Script to generate additional data if needed
- Contains both normal traffic and various DDoS attack types (SYN flood, UDP flood, HTTP flood, amplification attacks)

Data Characteristics

The dataset intentionally includes real-world data quality issues:

- Missing values (timestamps, IPs)
- Malformed IP addresses
- Corrupted fields
- Duplicate records
- Outliers and anomalies

Requirements

Core Requirements (Must Have)

- Stream Processing
    - Simulate real-time processing of network events
    - Detect potential DDoS attacks as they occur
    - Calculate metrics using sliding windows (e.g., requests per IP per minute)
    - Handle any Data Cleaning / Normalization needed
    - Demonstrate stateful stream processing
- Batch Processing
    - Perform historical analysis on the full dataset
    - Generate aggregate statistics and attack patterns
    - Identify top attackers and attack types

- Data Storage
  - Save processed data to a persistent store (database, data lake, or files)
  - Design a schema that supports both real-time queries and historical analysis
- Alerting Logic
  - Implement rules to detect DDoS attacks
  - Generate alerts when thresholds are exceeded
  - Consider different attack patterns (volume-based, pattern-based)

Bonus Points

- Multiple Technologies: Use both Spark and Flink (or another streaming framework, Spark Streaming OK)
- ML Data Preparation: Show how you would clean and build datasets specifically for AI/ML training
  - Feature engineering for ML models
  - Handle class imbalance (rare attacks vs normal traffic)
  - Create train/test/validation splits
  - Generate labeled datasets with proper data quality for model training
- ML Implementation (Extra Bonus): Actually train an anomaly detection model
  - Use any approach (supervised, unsupervised, or semi-supervised)
  - Show model evaluation metrics
  - Demonstrate how it would integrate with streaming pipeline
  - Compare ML detection vs rule-based detection
- Performance Optimization: Demonstrate optimization techniques
- Visualization: Dashboard or monitoring display - Grafana
- Production Considerations: Error handling, logging, monitoring, scalability discussion

Technical Choices

You're free to use any of the following environments:

- Option 1: Google Colab
  - Free, no setup required
  - Limited to Python/PySpark
  - Good for demonstrating concepts
- Option 2: Databricks Community Edition
  - Free tier available
  - Full Spark environment
  - Closest to production
- Option 3: Docker Compose
  - Full control over environment
  - Can include Kafka, Flink, Spark, databases
  - Most realistic but requires more setup
- Option 4: Your Own Solution
  - Use any tools/platforms you prefer
  - Explain your choices

Deliverables

- Code Repository

    - All source code
    - Clear file organization
    - README with setup instructions
- Documentation
    - Explanation of your approach
    - Architecture decisions and trade-offs
    - How to run your solution
    - Sample outputs/results
- Analysis Results
    - Screenshots or output showing:
        - Stream processing detecting attacks in real-time
        - Batch analysis results
        - Data quality metrics
        - Performance metrics (throughput, latency)
- Discussion (in README or separate document)
    - How would you scale this for production?
    - What monitoring would you add?
    - How would you handle late-arriving data?
    - Cost/performance trade-offs of your approach

Example Attack Detection Rules

Consider implementing detection for:

- High request rate from single IP (>100 requests/minute)
- Port scanning (single IP hitting many ports)
- Sudden traffic spikes (10x normal volume)
- Slowloris attacks (many incomplete connections)

ML Feature Ideas (Bonus)

If attempting the ML bonus, consider engineering features like:

- Time-based: Requests per second/minute/hour, time since last request
- Statistical: Rolling mean/std of packet sizes, entropy of ports accessed
- Behavioral: New IP (never seen before), unusual port combinations
- Network: Packet size distributions, protocol ratios, flag patterns
- Derived: Rate of change, acceleration of request frequency

Questions?

If you have questions about the requirements, make reasonable assumptions and document them. We're more interested in your approach and thinking than perfect adherence to specifications.

Submission

Please submit:

- GitHub repository link (preferred) or zip file

- Any additional documentation
- Instructions to run your solution

Good luck!

Starter Code

```
#!/usr/bin/env python3
"""
Run this script locally to generate a CSV file for your DDoS detection
assessment.
Usage: python generate_ddos_dataset.py

This will create 'network_traffic.csv' (~100-200MB) with 2 hours of messy
data.
"""

import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import random
import hashlib
import os

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

print("Starting DDoS Dataset Generation...")
print("=" * 60)

# Configuration
HOURS_OF_DATA = 2  # 2 hours is good for Colab
NORMAL_RATE = 500  # events per minute
OUTPUT_FILE = "network_traffic.csv"

# Time range
start_time = datetime.now() - timedelta(hours=HOURS_OF_DATA)

# IP pools
normal_ips = [
    f"192.168.{random.randint(1,10)}.{random.randint(1,254)}" for _ in
range(100)
]
server_ips = [f"10.0.1.{i}" for i in range(1, 11)]
botnet_ips = [
    f"{random.randint(1,223)}.{random.randint(0,255)}.
{random.randint(0,255)}.{random.randint(1,254)}"
    for _ in range(500)
]

# Services and ports
```

```
services = {
    80: "HTTP",
    443: "HTTPS",
    22: "SSH",
    21: "FTP",
    3306: "MySQL",
    5432: "PostgreSQL",
    53: "DNS",
    3389: "RDP",
    8080: "HTTP-ALT",
    8443: "HTTPS-ALT",
}

# Attack schedule (minute_start, minute_end, attack_type,
intensity_multiplier)
attack_schedule = [
    (15, 25, "syn_flood", 50),  # Minutes 15-25: SYN flood
    (40, 50, "http_flood", 30),  # Minutes 40-50: HTTP flood
    (70, 80, "udp_flood", 40),  # Minutes 70-80: UDP flood
    (95, 105, "amplification", 35),  # Minutes 95-105: DNS amplification
]

print(f"Generating {HOURS_OF_DATA} hours of network traffic")
print(f"Normal rate: {NORMAL_RATE} events/minute")
print(f"Attack windows: {len(attack_schedule)}")
print()

all_records = []
total_minutes = HOURS_OF_DATA * 60

for minute in range(total_minutes):
    if minute % 30 == 0:
        print(f"   Processing minute {minute}/{total_minutes}...")

    current_time = start_time + timedelta(minutes=minute)

    # Check if we're in an attack window
    is_attack = False
    attack_type = None
    intensity = 1

    for start, end, atype, mult in attack_schedule:
        if start <= minute < end:
            is_attack = True
            attack_type = atype
            intensity = mult
            break

    # Generate events for this minute
    if is_attack:
        num_events = NORMAL_RATE * intensity
    else:
        num_events = int(NORMAL_RATE * random.uniform(0.8, 1.2))
```

```
    for _ in range(num_events):
        # Timestamp with some spread
        ts = current_time + timedelta(seconds=random.uniform(0, 60))

        # Source IP (botnet during attacks)
        if is_attack and random.random() < 0.8:
            source_ip = random.choice(botnet_ips)
        else:
            source_ip = random.choice(normal_ips)

        # Destination
        dest_ip = random.choice(server_ips)

        # Port selection
        if attack_type == "http_flood":
            dest_port = random.choice([80, 443, 8080])
        elif attack_type == "syn_flood":
            dest_port = random.choice([80, 443, 22, 3306])
        elif attack_type == "udp_flood":
            dest_port = random.randint(1, 65535)
        elif attack_type == "amplification":
            dest_port = random.randint(1024, 65535)
            source_port = 53  # DNS
        else:
            dest_port = random.choices(
                list(services.keys()), weights=[30, 35, 5, 2, 3, 3, 5, 2,
10, 5]
            )[0]
            source_port = random.randint(1024, 65535)

        # Protocol
        if attack_type == "udp_flood" or attack_type == "amplification":
            protocol = "UDP"
        elif dest_port in [53]:
            protocol = random.choice(["TCP", "UDP"])
        else:
            protocol = "TCP"

        # Packet size
        if attack_type == "syn_flood":
            packet_size = random.randint(40, 60)
        elif attack_type == "amplification":
            packet_size = random.randint(2000, 4000)  # Amplified
        elif attack_type == "http_flood":
            packet_size = random.randint(500, 1500)
        else:
            packet_size = int(np.random.lognormal(6, 1.5))
            packet_size = max(20, min(65535, packet_size))

        # Response time
        if is_attack:
            response_time = (
                random.uniform(100, 5000) if attack_type != "syn_flood"
else None
```

```
            )
        else:
            response_time = np.random.exponential(50)

        # Build record
        record = {
            "timestamp": ts.isoformat(),
            "source_ip": source_ip,
            "dest_ip": dest_ip,
            "source_port": source_port
            if "source_port" in locals()
            else random.randint(1024, 65535),
            "dest_port": dest_port,
            "protocol": protocol,
            "packet_size": packet_size,
            "ttl": random.choice([64, 128, 255])
            if not is_attack
            else random.randint(1, 255),
            "flags": "SYN"
            if attack_type == "syn_flood"
            else random.choice(["SYN", "ACK", "PSH,ACK", "FIN,ACK", ""]),
            "response_time_ms": round(response_time, 3) if response_time
else None,
            "country": random.choice(
                ["US", "CN", "RU", "DE", "FR", "GB", "JP", "IN", None, ""]
            ),
            "service": services.get(dest_port, "UNKNOWN"),
            "bytes_sent": packet_size,
            "bytes_received": 0
            if attack_type == "syn_flood"
            else int(packet_size * random.uniform(0.5, 10)),
            "packets_sent": 1,
            "packets_received": 0
            if attack_type == "syn_flood"
            else random.randint(1, 5),
            "session_id": hashlib.md5(
                f"{source_ip}{dest_ip}{random.randint(1,100000)}".encode()
            ).hexdigest()[:8]
            if not is_attack
            else None,
            "is_attack": is_attack,
            "attack_type": attack_type,
        }

        # Add HTTP fields for web traffic
        if dest_port in [80, 443, 8080, 8443]:
            record["http_method"] = random.choice(
                ["GET", "POST", "PUT", "DELETE", None]
            )
            record["uri_path"] = random.choice(
                ["/", "/api/v1/users", "/login", "/search", None, ""]
            )
            record["status_code"] = random.choice([200, 404, 500, 503,
None])
```

```
            record["user_agent"] = random.choice(
                [
                    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
                    "python-requests/2.28.1",
                    "bot",
                    None,
                    "",
                ]
            )
        else:
            record["http_method"] = None
            record["uri_path"] = None
            record["status_code"] = None
            record["user_agent"] = None

        all_records.append(record)

print(f"\n✅ Generated {len(all_records):,} raw records")

# Add data quality issues
print("🔧 Adding realistic data quality issues...")

# Randomly remove some critical fields (2% of records)
for i in random.sample(range(len(all_records)), int(len(all_records) *
0.02)):
    if random.random() < 0.5:
        all_records[i]["source_ip"] = None
    else:
        all_records[i]["timestamp"] = None

# Randomly remove non-critical fields (10% of records)
non_critical = ["user_agent", "country", "uri_path", "http_method"]
for i in random.sample(range(len(all_records)), int(len(all_records) *
0.1)):
    field = random.choice(non_critical)
    if field in all_records[i]:
        all_records[i][field] = None

# Add some malformed IPs (1% of records)
for i in random.sample(range(len(all_records)), int(len(all_records) *
0.01)):
    all_records[i]["source_ip"] = random.choice(
        ["999.999.999.999", "192.168.1", "0.0.0.0", "corrupted",
"192.168.1.256"]
    )

# Add some corrupted timestamps (0.5% of records)
for i in random.sample(range(len(all_records)), int(len(all_records) *
0.005)):
    all_records[i]["timestamp"] = random.choice(
        ["2024-13-45 25:99:99", "INVALID_TIME", "", "0000-00-00 00:00:00"]
    )

# Add duplicates (1% of records)
```

```
num_duplicates = int(len(all_records) * 0.01)
for _ in range(num_duplicates):
    original = random.choice(all_records)
    all_records.append(original.copy())

# Add outliers (0.5% of records)
for i in random.sample(range(len(all_records)), int(len(all_records) *
0.005)):
    all_records[i]["packet_size"] = random.choice([-1, 999999, 0])
    all_records[i]["response_time_ms"] = random.choice([-100, 1000000,
None])

print("📝 Converting to DataFrame...")
df = pd.DataFrame(all_records)

# Shuffle for realism
df = df.sample(frac=1).reset_index(drop=True)

# Save to CSV
print(f"💾 Saving to {OUTPUT_FILE}...")
df.to_csv(OUTPUT_FILE, index=False)

# Calculate file size
file_size_mb = os.path.getsize(OUTPUT_FILE) / (1024 * 1024)

# Print summary statistics
print()
print("=" * 60)
print("✅ DATASET GENERATION COMPLETE!")
print("=" * 60)
```