README.txt

This project consists of several folders and files designed to capture network traffic (via eBPF/XDP), process it, detect potential anomalies, and optionally train an AI model to improve detection. Below, you'll find a description of the main folders and files, as well as basic execution instructions.

Project Structure

```
├── AI_training
│   ├── archive
│   │   └── ... (CSV and training data)
│   ├── incremental_model.joblib      (Pre-trained default model)
│   ├── scaler.joblib                 (Pre-trained StandardScaler)
│   ├── training.py                   (Main script for incremental 
training)
│   └── ...
├── kernel_space
│   └── packet_capture.c              (eBPF/XDP code for capturing 
and mapping traffic)
├── user_space
│   ├── anomaly_detector.py           (Main anomaly detection 
script)
│   └── arithmetic_compression.py     (Arithmetic compression class 
for anomalous flows)
└── ...
```

File Descriptions

1. kernel_space/packet_capture.c

- Contains the eBPF/XDP program that hooks into the network interface.
- Captures packets and updates runtime flow data structures (BPF maps).
- Uses per-CPU hashing to manage flow states efficiently.

2. user_space/anomaly_detector.py

- Main file for running anomaly detection.
- Loads the C code (`packet_capture.c`) and attaches it to the network interface using XDP.
- Defines `ctypes` structures (e.g., `FlowKey`, `FlowData`) and maps flows (`flows`, `exported_flows`).
- Exports inactive flows or flows exceeding activity timeouts, sending them through the AI model.
- If a flow is considered anomalous, it is compressed and stored in a binary file.

3. user_space/arithmetic_compression.py

- Contains the `AdaptiveArithmeticCodingFlows` class, implementing arithmetic compression logic for flows.
- Serializes `FlowKey` and `FlowData` to compress and store only anomalous data.

4. AI_training/training.py

- Script for incremental training of the AI model.
- Loads labeled traffic samples from `archive/` CSV files, cleans the data, and extracts relevant columns.
- Scales features (via StandardScaler) and trains the model incrementally (`SGDClassifier`).
- Generates/updates `incremental_model.joblib` and `scaler.joblib`.

5. AI_training/incremental_model.joblib and AI_training/scaler.joblib

- Generated files from the training process:
  - `incremental_model.joblib`: Trained classifier model.
  - `scaler.joblib`: Feature scaler (StandardScaler).

How to Run the Project

Prerequisites

- Install BCC or libbpf (depending on your eBPF setup).
- Python 3 and libraries listed in the code (e.g., pandas, scikit-learn, numpy, joblib, bcc, etc.).
- Sufficient privileges to attach an XDP program to the network interface (e.g., via `sudo`).

Initial Training (Optional)

If you wish to retrain the model from scratch (or add more data to the training set), navigate to the `AI_training` folder or the project root and execute:

```
python3 AI_training/training.py
```

This script will look for CSV files in `AI_training/archive/` and generate/update `incremental_model.joblib` and `scaler.joblib`.

Running Anomaly Detection

To start real-time anomaly detection, execute:

```
python3 user_space/anomaly_detector.py
```

- The script will prompt for a sampling rate percentage (0-100).
- Example: Enter `30` to analyze only 30% of the traffic (randomly sampled) while discarding the rest.

- The eBPF/XDP program `packet_capture.c` will be attached to the `enp0s8` network interface (you can modify the interface in the script if needed).
- The script monitors flows and reports anomalies with the message `ANOMALY DETECTED`. Anomalous flows are compressed and saved in a binary file (`compressed_flows.dat`).

Stopping the Script

To stop `anomaly_detector.py`, press `Ctrl + C`. The program will detach the XDP hook from the network interface and exit.

Customization and Notes

- The `user_space` folder contains the main scripts for detection and compression.
- The `kernel_space` folder stores the XDP/eBPF code.
- The `AI_training` folder is used for (re)training and storing the model.

Changing the Network Interface

To use a different network interface (e.g., `eth0` instead of `enp0s8`), edit `anomaly_detector.py` where the XDP program is attached:

```
b.attach_xdp(dev="enp0s8", fn=fn_capture_packet, flags=0)
```

Replace `enp0s8` with the desired interface.