Lightweight Wi-Fi Management-Frame IDS

A lightweight Intrusion Detection System (IDS) for **Wi-Fi management-frame attacks**, focusing on:

- Deauthentication floods
- Probe-request floods
- Beacon / channel anomalies

The IDS runs on a **Raspberry Pi** (or any Linux host with a monitor-mode Wi-Fi NIC), passively sniffs 802.11 management traffic, and raises alerts using a combination of:

- **Rule-based thresholds** (per sender, per BSSID, per channel)
- **Statistical anomaly detection** (z-scores against learned baselines)

It supports both **offline pcap analysis** and **live monitoring** with a real-time **web dashboard**.

Features

- **Management-Frame Focused Detection**
    - Detects abnormal patterns in:
        - Deauthentication frames
        - Probe-request frames
        - Beacon activity (e.g., spikes on specific channels)
- **Hybrid Detection Engine**
    - Sliding-window counts with configurable thresholds
    - Global z-score anomaly detection for deauth / probe / beacon streams
    - Detection of per-sender and per-BSSID deauth bursts
- **Offline and Live Modes**
    - Offline analysis of `.pcap` / `.pcapng` files
    - Live sniffing from a **monitor-mode interface** (e.g. `wlan0mon` on Pi)
- **Real-Time Dashboard**
    - FastAPI backend + HTML/JS frontend
    - Live time-series graph of deauth / probe / beacon rates
    - Status bar with severity levels (Secure / Anomalous / Intrusion)
    - Alert table with severity tags and details
- **Configurable & Calibratable**
    - YAML configuration for thresholds, window sizes, anomaly settings
    - `calibrate_config.py` to derive thresholds and baselines from:
        - Normal traffic captures (pcaps or metrics CSV)
        - Optional attack captures (deauth/probe floods)

    - Optional alerts JSONL (to filter out suspicious windows during learning)

- **CLI Tools**
    - Colorful terminal mode for live debugging and pcap replay
    - Metrics and alerts exported to CSV / JSONL for later analysis

---

Tech Stack

Languages & Frameworks

- Python 3
- FastAPI (REST API backend)
- Uvicorn (ASGI server)
- HTML/CSS/JavaScript (frontend)
- Chart.js (time-series plots)

Libraries

- PyShark (wrapper over `tshark` for packet capture)
- `dataclasses`, `collections.deque`, `enum`, `math`, `statistics`
- `pyyaml` for YAML config handling
- `argparse`, `csv`, `json` for CLI and data I/O

Tools

- `tshark` / Wireshark (underlying packet engine)
- `aircrack-ng`, `aireplay-ng`, `mdk4` (used only for testing/attacks in our experiments)

System Requirements

Hardware

- **IDS Host**
    - Raspberry Pi (e.g., Pi 4) or any Linux machine
    - **Wi-Fi adapter with monitor-mode support** (USB NIC recommended on Pi)
- **Attacker Device (for testing)**
    - Separate laptop with Wi-Fi injection support (for deauth / probe / beacon floods)
- **Victim Clients**
    - Laptops / smartphones connected to the test Wi-Fi network

Software

- Linux (Ubuntu / Raspberry Pi OS or similar)
- Python 3.9+ recommended
- `tshark` installed and accessible in `$PATH`

- Ability to put Wi-Fi NIC into monitor mode
  - e.g. via `iw`, `airmon-ng`, or NetworkManager configuration

Permissions

- Live capture typically requires **root** or `sudo`
- Depending on distribution, `tshark` may also require elevated permissions

---

Setup Instructions

1. Clone the Repository

```
git clone https://github.com/jash-j3/wireless_lightweight_ids.git
cd wireless_lightweight_ids
```

2. Create and Activate Virtual Environment

```
python3 -m venv .venv
source .venv/bin/activate
```

3. Install Python Dependencies

Can be done via `requirements.txt` or manually.

Using `requirements.txt`:

```
pip install -r requirements.txt
```

Or manually:

```
pip install pyshark fastapi uvicorn pyyaml
```

4. Install `tshark`

On Debian/Ubuntu/Raspberry Pi OS:

```
sudo apt-get update
sudo apt-get install tshark
```

Ensure `tshark` works:

```
tshark -v
```

5. Configure Wi-Fi Interface in Monitor Mode

Can use the `monitor_mode.sh` script or manually set monitor mode.

Using the script:

```
sudo bash monitor_mode.sh start
```

Using `airmon-ng`:

```
sudo airmon-ng start <wlan0>
# Typically creates wlan0mon
```

Note the interface name (e.g. `wlan0mon`) for use with the IDS.

6. Configuration File

Check or edit `config/config.yaml`:

- Detection thresholds:
  - `thresholds.deauth_per_sec`
  - `thresholds.deauth_per_sec_bssid`
  - `thresholds.probe_req_per_sec`
  - `thresholds.beacon_per_sec`
- Time windows:
  - `windows_sec.deauth`, `probe_req`, `beacon`, `stats_interval`, etc.
- Anomaly settings:
  - `anomaly.use_zscore`
  - `anomaly.z_threshold`

You can later generate a calibrated config using `calibrate_config.py` (see below).

Usage

1. Offline Analysis (pcap / pcapng)

Analyze management traffic in a capture file:

```

```

```
python3 src/ids_offline.py \
  --pcap data/pcaps/test.pcapng \
  --print-metrics-head 5
```

Outputs:

- Alerts in JSONL: `data/alerts/offline_alerts.jsonl`
- Metrics in CSV: `reports/offline_metrics.csv`
- Console summary line with number of packets, events, windows, alerts.

Generate plots using `src/plot_offline_metrics.py`:

```
python3 src/plot_offline_results.py \
  --metrics reports/offline_metrics.csv \
  --alerts data/alerts/offline_alerts.jsonl \
  --out-dir reports/plots
```

---

2. Live IDS (CLI View)

Run live IDS from a monitor interface:

```
sudo python3 src/ids_live.py \
  --iface wlan0mon \
  --print-events \
  --summary-every 5
```

Or replay an existing pcap as if it were live:

```
python3 src/ids_live.py \
  --pcap data/pcaps/test.pcapng \
  --print-events \
  --summary-every 2
```

Options:

- `--display-filter "wlan.fc.type==0"` (default: management frames)
- `--alerts-out data/alerts/live_alerts.jsonl` to persist alerts
- `--no-color` to disable ANSI color in the terminal

---

3. Web Dashboard (FastAPI + Frontend)

Start the Dashboard Server

Live capture mode:

```
sudo python3 src/ids_dashboard_server.py \
  --iface wlan0mon \
  --config config/config.yaml \
```

PCAP replay mode (for demo / testing):

```
python3 src/ids_dashboard_server.py \
  --pcap data/pcaps/test.pcapng \
  --config config/config.yaml \
```

The server will:

- Start capturing and feeding packets into the IDS engine.
- Expose REST API endpoints (see API section below).
- Continuously write:
  - Metrics CSV (e.g. `reports/dashboard_metrics.csv`)
  - Alerts JSONL (e.g. `data/alerts/dashboard_alerts.jsonl`)

View Dashboard

Open the frontend in your browser:

```
src/templates/index.html
```

This HTML file polls `http://localhost:8000/api/state` by default and displays:

- Cards with current deauth / probe / beacon rates
- Real-time line chart for management-frame traffic
- Security status (Secure / Anomalous / Intrusion) with color changes
- Alerts table (latest events, severity tags, detail)

(If the API is on a different host/port, adjust `API_URL` in `index.html`.)

---

4. Calibration (Automatic Threshold & Baseline Tuning)

Use `calibrate_config.py` to tune thresholds and baselines based on recorded traffic.

Example 1: Only Normal Traffic (pcaps)

```

```

```
python3 src/calibrate_config.py \
  --config-in  config/config.yaml \
  --config-out config/config_calibrated.yaml \
  --normal-pcap data/pcaps/normal1.pcapng \
  --normal-pcap data/pcaps/normal2.pcapng
```

Example 2: Normal Metrics + Alerts (from dashboard runs)

```
python3 src/calibrate_config.py \
  --config-in  config/config.yaml \
  --config-out config/config_calibrated.yaml \
  --normal-metrics reports/dashboard_metrics_normal.csv \
  --normal-alerts-jsonl data/alerts/dashboard_normal.jsonl
```

Example 3: Include Attack Metrics

```
python3 src/calibrate_config.py \
  --config-in  config/config.yaml \
  --config-out config/config_calibrated.yaml \
  --normal-metrics reports/dashboard_metrics_normal.csv \
  --deauth-attack-metrics reports/dashboard_metrics_deauth.csv \
  --probe-attack-metrics  reports/dashboard_metrics_probe.csv
```

The script computes:

- Normal means and standard deviations for each stream
- Static thresholds separating normal from attack windows (where possible)
- A global z-threshold with a target false-positive rate

It writes out `config_calibrated.yaml`, which you can then use as the main config.

Project Structure

```
.
├── config
│   ├── config.yaml                # Base IDS configuration (thresholds,
windows, anomaly settings)
│   └── config_calibrated.yaml     # Auto-calibrated configuration
(generated by calibrate_config.py)
│
├── data
│   ├── alerts
│   │   ├── dashboard_alerts.jsonl  # Alerts emitted during dashboard /
live runs
```

```
│   │   └── offline_alerts.jsonl    # Alerts emitted during offline pcap
analysis
│   └── pcaps                      # Sample capture files used for
testing/evaluation
│       ├── Deauth.pcap
│       ├── Disass.pcap
│       ├── Evil_Twin.pcap
│       ├── (Re)Assoc.pcap
│       ├── Rogue_AP.pcap
│       ├── test1.pcapng
│       ├── test2.pcapng
│       ├── test3_deauth_card.pcapng
│       └── test.pcapng
│
├── reports
│   ├── dashboard_metrics.csv       # Metrics exported from dashboard /
live runs
│   ├── offline_metrics.csv         # Metrics exported from offline IDS
runs
│   └── plots                       # Helper plots generated from metrics
│       ├── alerts_timeline.png
│       ├── beacon.png
│       ├── deauth.png
│       ├── evil_twin_signal.png
│       ├── probe.png
│       ├── top_bssid_deauth.png
│       ├── top_sender_beacon.png
│       ├── top_sender_deauth.png
│       └── top_sender_probe.png
│
├── src
│   ├── ids_offline.py              # Core IDS engine + offline pcap/pcapng
analysis
│   ├── ids_live.py                 # CLI for live monitoring or pcap
replay (terminal view)
│   ├── ids_dashboard_server.py     # FastAPI server + capture thread +
JSON API for dashboard
│   ├── calibrate_config.py         # Script to calibrate
thresholds/baselines from captures/metrics
│   ├── plot_offline_results.py     # Script to generate plots from
offline_metrics.csv
│   └── templates                   # Dashboard frontend assets (served by
FastAPI)
│       ├── index.html              # Dashboard UI (status, charts, alerts
table)
│       └── chart.js                # Chart.js bundle (local copy)
│
├── monitor_mode.sh                 # Helper script to put Wi-Fi interface
into monitor mode
├── requirements.txt                # Python dependencies
├── LICENSE                         # Project license
├── README.md                       # Project documentation
└── structure.cpp                   # Initial C++ design sketch for IDS
data structures (reference only)
```

API Details

`GET /api/state`

Returns the current IDS state for the dashboard.

Request:

```
GET /api/state HTTP/1.1
Host: localhost:8000
```

Typical Response JSON:

```
{
  "ok": true,
  "now": 1700000000.123,
  "mode": "detection",
  "metrics": {
    "ts_from": 1700000000.0,
    "ts_to": 1700000001.0,
    "deauth": 5,
    "probe": 12,
    "beacon": 80,
    "top_sender_deauth": 5,
    "top_bssid_deauth": 5,
    "top_channel": 6,
    "top_channel_beacon_count": 80,
    "top_channel_deauth_count": 5
  },
  "alerts": [
    {
      "ts_from": 1700000000.0,
      "ts_to": 1700000001.0,
      "kind": "THRESH_DEAUTH_PER_SENDER",
      "details": {
        "sender_mac": "AA:BB:CC:DD:EE:FF",
        "count_in_window": 5,
        "threshold": 3
      }
    }
  ],
  "learn_stats": {
    "windows_total": 100,
    "windows_used_for_learning": 80,
    "windows_rejected_suspicious": 20
  }
}
```

Fields can be extended; the important ones for the dashboard are:

- `metrics.deauth/probe/beacon` (global current rates)
- `metrics.top_sender_deauth`, `metrics.top_bssid_deauth`
- `metrics.top_channel` and channel-level counts
- `alerts`: type and details of recent alerts
- `mode`: `"learning"` or `"detection"`

`GET /api/mode`

Returns the current IDS mode.

```
{ "mode": "detection" }
```

`POST /api/mode`

Sets the IDS mode.

Request:

```
POST /api/mode HTTP/1.1
Content-Type: application/json

{ "mode": "learning" }
```

Response:

```
{ "ok": true, "mode": "learning" }
```

This is used by the dashboard toggle to switch between learning and detection modes.

Demo Video & Final Presentation

- **Demo Video:**
  - Video Link
- **Final Presentation Slides:**
  - Slides Link

Contributors

- **Jash Jhatakia**
- **Advait Jain**
- **Parth Kansagra**