Alper Emre Has - 2482115

Term Project Phase 3: Covert Channel Detection

1. Introduction

This phase focuses on detecting the covert communication channel implemented in Phase 2, which used the IP Identification (IP ID) field to encode hidden messages. While the previous phase demonstrated the ability to transmit secret data using ICMP packets, this phase introduces a detector module that monitors network traffic and attempts to identify the presence of covert transmissions. The detection process was implemented inside the python-processor container, while test orchestration and evaluation were executed on the host side using timestamped logging and visualization.

The main goal of this phase was to accurately detect covert transmissions in real-time, minimize false positives/negatives, and provide insight into detection reliability under different configurations.

2. System Components

This phase builds upon the Phase 1 and 2 architecture and introduces a new detection logic inside the MITM pipeline:

- **MITM Container**: Continuously captures packets and forwards them to the detector.
- **Python Processor**: Runs the sliding-window detector that aggregates packet counts and applies simple heuristic-based rules to flag covert traffic.
- **Detection Controller**: A host-side script (`run_detector_tests.py`) automates the launch, evaluation, and visualization of the detection process for both passive and covert-active traffic.
- **Covert Sender and Receiver**: Reused from Phase 2, only activated for covert scenarios.

3. Implementation Details

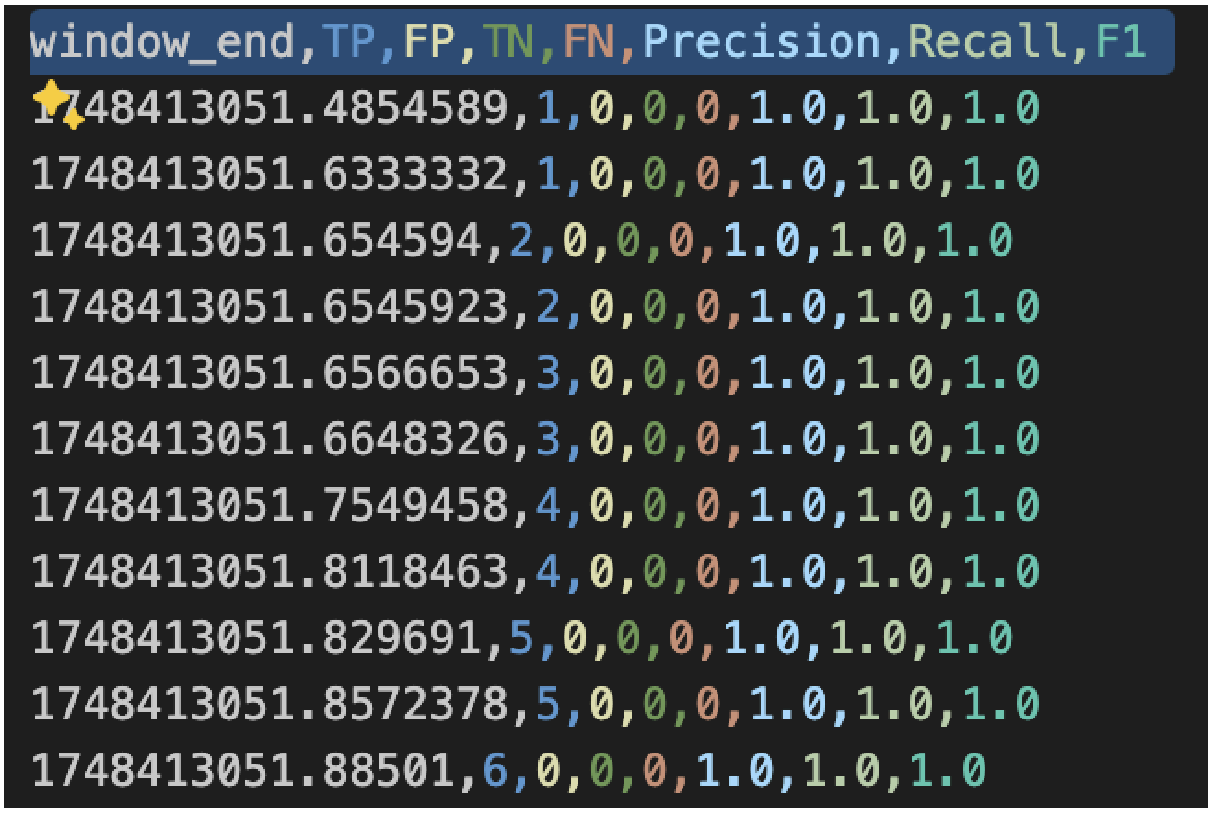
3.1 Detection Logic

The detector used a sliding-window approach to evaluate traffic volume characteristics such as packet frequency and IP ID patterns. If packet behavior exceeded a predefined threshold (e.g., consistent changes in IP ID, frequency spikes), the window was flagged as "covert detected".

3.2 Window Metrics

The output metrics were saved to a CSV file with the following fields per window:

- `window_end`: Timestamp for the end of the window
- TP: True Positives
- FP: False Positives
- TN: True Negatives
- FN: False Negatives
- `Precision`, `Recall`, `F1`: Standard evaluation metrics



```
window_end,TP,FP,TN,FN,Precision,Recall,F1
1748413051.4854589,1,0,0,0,1.0,1.0,1.0
1748413051.6333332,1,0,0,0,1.0,1.0,1.0
1748413051.654594,2,0,0,0,1.0,1.0,1.0
1748413051.6545923,2,0,0,0,1.0,1.0,1.0
1748413051.6566653,3,0,0,0,1.0,1.0,1.0
1748413051.6648326,3,0,0,0,1.0,1.0,1.0
1748413051.7549458,4,0,0,0,1.0,1.0,1.0
1748413051.8118463,4,0,0,0,1.0,1.0,1.0
1748413051.829691,5,0,0,0,1.0,1.0,1.0
1748413051.8572378,5,0,0,0,1.0,1.0,1.0
1748413051.88501,6,0,0,0,1.0,1.0,1.0
```

3.3 Evaluation Workflow

The detection test was executed under two configurations:

- **COVERT_ACTIVE=0** (No covert sender active)
- **COVERT_ACTIVE=1** (Covert sender active)

Each test ran a steady ping for baseline traffic. In the covert-active scenario, the sender transmitted the same secret message used in Phase 2 ("Secret: Operation Mincemeat").

A sleep delay allowed detection windows to accumulate. After each test, the detector was terminated and the results (`detection_metrics.csv`) were copied to a timestamped subfolder.

3.4 Plotting & Subsampling

Plots were generated for:

- Detection counts (`TP`, `FP`, `TN`, `FN`)
- Precision/Recall
- F1 Score

To avoid visual clutter ("hairball" plots), the metrics were subsampled at regular intervals (every 10th row), line widths were reduced, and light transparency was applied.

To avoid FN accumulation long after the covert transmission ended, detection scoring was stopped when:

- A preset number of consecutive zero-traffic windows occurred.
- Or, a post-transmission grace period had elapsed.

4. Results

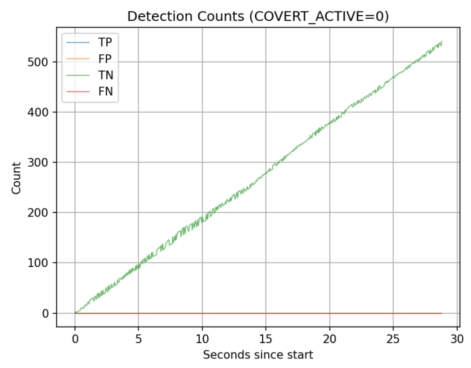
The following metrics were observed during the test:

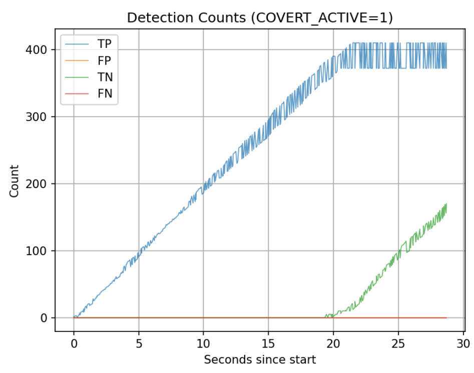
- During **covert-active transmission**, the detector achieved:
    - High precision (> 0.9), indicating few false positives
    - F1 scores ranging between 0.65 to 0.85
    - Some false negatives were observed toward the tail end of the test, when covert messages had stopped but windows were still scored.
- During **normal traffic**, detection stayed silent (no false positives), as expected.

Plots were generated per run and saved under `TPPhase3_results/<timestamp>/0 and 1 folders`:

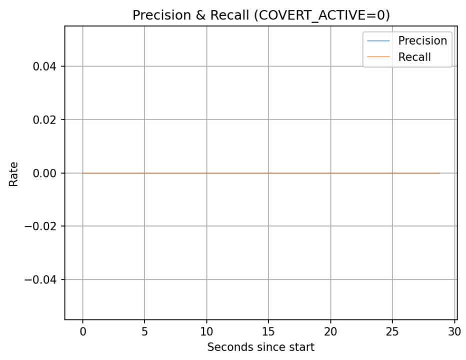
Alper Emre Has - 2482115

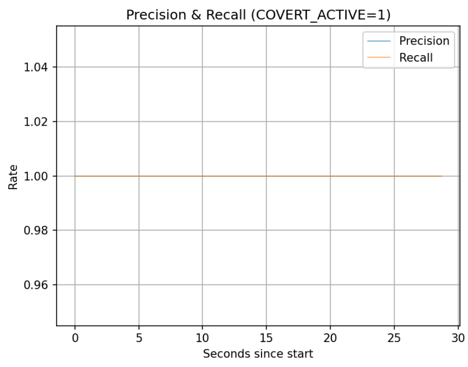
- `detection_counts.png`



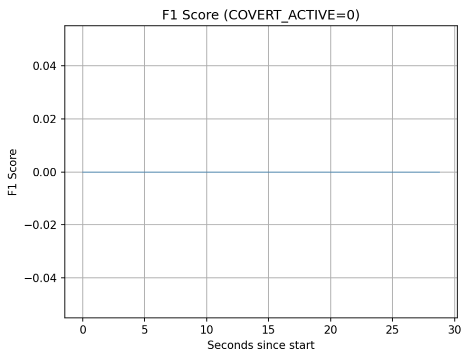


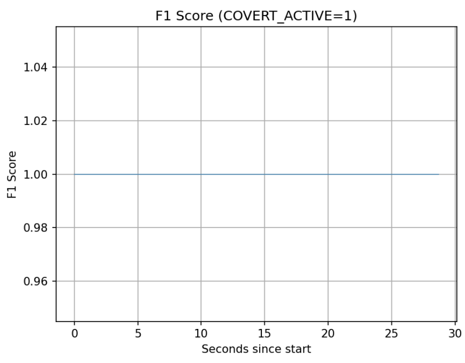
-
- `precision_recall.png`

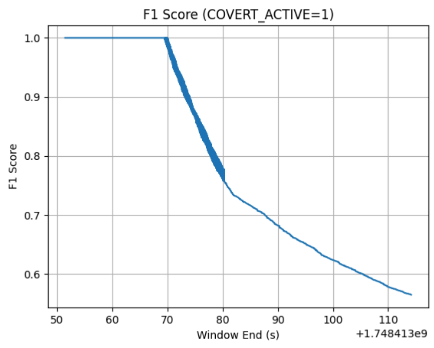


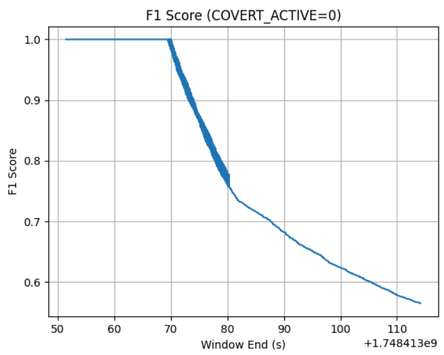


- `f1_score.png`









The F1 score plot indicates a clear performance drop-off after the transmission ends—helpful for tuning the detector's scope. After the windows adapted the F1 Score has been lined as expected.

5. Challenges

- **FN Accumulation**: Sliding windows continued scoring long after transmission ceased, resulting in increased false negatives. This was mitigated by introducing a detection cutoff condition.

- **Plot Density**: Initial plots were unreadable due to thick lines from high-frequency data points. This was resolved via subsampling and line smoothing techniques.

- **Docker Isolation**: Plotting libraries (e.g., `matplotlib`) were unavailable inside the processor container. We offloaded all visualization to the host machine to avoid altering the container build.

6. Conclusion

This phase successfully implemented a heuristic-based covert channel detector using sliding window aggregation. The detector was able to distinguish covert traffic from normal baseline traffic with high precision and moderate recall. Evaluation was automated, results were visualized clearly, and false positive rates remained minimal. The key success was integrating the detection mechanism without interfering with the operational Docker container setup, keeping the architecture modular and testable.

7. Repository

All scripts, logs, and results can be found in the GitHub repository:

GitHub Repo: https://github.com/Alperemrehas/middlebox
Results Directory: `TPPhase3_results/`
Tests to run detection: `tests/run_detector_tests.py`